A. D. MCCXXIV

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

DEPENDABILITY EVALUATION OF MOBILE DISTRIBUTED SYSTEMS VIA FIELD FAILURE DATA ANALYSIS

MARCELLO CINQUE

Tesi di Dottorato di Ricerca

Novembre 2006

Dipartimento di Informatica e Sistemistica

via Claudio, 21- I-80125 Napoli - *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

A. D. MCCXXIV

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

DEPENDABILITY EVALUATION OF MOBILE DISTRIBUTED SYSTEMS VIA FIELD FAILURE DATA ANALYSIS

MARCELLO CINQUE

Tesi di Dottorato di Ricerca

(XIX Ciclo)

Novembre 2006

Il Tutore

Prof. Domenico Cotroneo

Il Coordinatore del Dottorato

Prof. Luigi P. Cordella

Dipartimento di Informatica e Sistemistica

via Claudio, 21- I-80125 Napoli - *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

DEPENDABILITY EVALUATION OF MOBILE DISTRIBUTED SYSTEMS VIA FIELD FAILURE DATA ANALYSIS

By
Marcello Cinque

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
"FEDERICO II" UNIVERSITY OF NAPLES
VIA CLAUDIO 21, 80125 – NAPOLI, ITALY
NOVEMBER 2006

"The important thing is not to stop questioning"
Albert Einstein

Table of Contents

List of Tables

List of Figures

Acknowledgements

"Proprio alla fine degli studi mi trovai sperduto tra infiniti dubbi ed errori. Mi sembrava di aver studiato solo per scoprire quanto fossi ignorante"

E se Cartesio aveva questa sensazione, figuriamoci io! E' proprio vero. Più si conosce e più ci si rende conto che c'è ancora molto da conoscere. Ma forse il bello è proprio questo. Se avessimo già tutte le risposte, la nostra esistenza sarebbe quanto meno noiosa.

E' con questo spirito che si conclude un altro importante capitolo della mia vita. E ora, nell'attesa di precipitarmi nell'ignoto di una nuova avventura, non mi resta che ringraziare tutti coloro che hanno creduto in questa mia folle corsa verso la consapevolezza dell'ignoranza. O meglio, la conoscenza. E' a tutti voi che dedico il frutto dei miei sforzi. L'elenco è lungo, e se dimentico qualcuno, non abbiatene. In fondo sono sempre stato un pò distratto.

Alla mia famiglia. Per altri tre lunghi anni avete sopportato la mia assenza e i miei malumori. Ma avete reso grande la mia soddisfazione. Grazie a voi, alla vostra fiducia, al vostro sostegno, ogni mio più piccolo passo è diventato un successo. Grazie ed infinitamente grazie. Per sempre.

Ad Armando. Allo specchio della mia coscienza. All'altra faccia della mia stessa medaglia. Al fratello che non ho mai avuto. Quello che sono lo devo a te. Quindi non prendertela se qualche volta sbaglio, in fondo è anche un pò colpa tua! E' vero, questi anni ci hanno allontanato un pò. Abbiamo dovuto imparare a camminare da soli, a perderci e a ritrovarci. Ma ci ha sorretto l'esperienza di una vita trascorsa insieme. E sono sicuro che allo stesso modo sapremo sorreggerci nelle incognite degli anni a venire. Grazie Arma, soprattutto per quello che diventeremo.

Ai miei amici di sempre, alle persone care. Ancora una volta avete dovuto sopportare la mia deliberata "volontà di stressarmi". Rassegnatevi, sarà sempre così. E' un morbo inguaribile. Grazie a Maxone, a Luxone, a Zio Brin (con Vittoria e la piccola Erika), a Stepon, Andrea il nano, Massimo capikki... grazie ragazzi. Grazie anche a Roby, persona

speciale, mio angelo custode (ma anche diavolo tentatore) nell'esperienza d'oltreoceano, e non solo. E un grazie enorme a te, Milù. Come te, nessuna è riuscita a sostenermi, a comprendermi, a farmi sentire importante. Grazie.

A tutta la famiglia "allargata". Grazie ai cugini, agli zii, alle zie. Siete delle persone eccezionali. La vostra stima è davvero importante per me, e così come mi ha aiutato finora, mi aiuterà a tenere duro e andare dritto per la mia strada.

A Domenico. In un'unica persona mio tutor, "padrone", e amico. La tua fiducia e il tuo sostegno sono stati fondamentali. Così come i tuoi insegnamenti, le belle serate passate insieme, e le lunghe giornate in treno a lavorare... fino ad arrivare alla stesura di questa tesi, che riassume anche i tuoi di sforzi. Infinite grazie a te, e al prof. Stefano Russo per avermi fatto vivere un'esperienza davvero indimenticabile. Irripetibile.

Ai ragazzi del laboratorio. Alla vostra compagnia, alle innumerevoli ore di fatica e risate trascorse assieme. A Gene, o generazio, o generaptor, o generantropo... grande compagno di viaggi, e nostro unico rettore. Con immenso rispetto. A Tatore, amico di sventure pre e post laurea, impareggiabile faticatore ma allo stesso tempo fautore di risate e coffee break. Devo ammettere che quando non ci sei, il laboratorio si intristisce un pò. A Pavelo, Gabry, Hulio, Fabio il biondo, Carmine. Grazie ragazzi. Senza il vostro aiuto, molto di quanto è riassunto qui di seguito non sarebbe stato possibile.

Napoli, Italy — Marcello
November 30, 2006

Introduction

Recent decades have been characterized by an explosive growth in the deployment of small, mobile devices, ranging from personal digital assistant to smart phones and wireless sensors. At the same time, more and more wireless communication technologies have been delivered, such as the IEEE 802.11, Bluetooth, and cellular networks. This scenario has led to the definition of the mobile computing paradigm, enabled by the so-called Mobile Distributed Systems (MDSs).

As these systems become increasingly complex, their reliability begins to decrease due to complex interactions between software modules and multiple applications. The sophisticated software that is taking on more duties, can contain bugs, can be difficult to comprehend and analyze, and ages in quality over time. As a consequence, consumer tolerance for application crashes and malfunctions is decreasing, and many consumers may consider switching brands if their current device does not perform to expectations. Hence, the dependability of these devices is directly related with the business opportunity. At the same time, more and more critical-data-driven applications have been introduced on the mobile computing market. Phone-based banking, ticket booking, and e-trading are examples of applications where the user expects correct behavior despite accidental errors and/or malicious tampering with the device. Dependability will become even more critical as new applications emerge for mobile distributed systems, e.g., robot control [91, 62, 57], traffic control [4], telemedicine [7, 9], and video-surveillance [73]. In such scenarios, a failure affecting the application could result in a significant loss or hazard, e.g., a robot performing

uncontrolled actions.

A significant research quest is to evaluate the dependability levels that can be achieved by these system in order to i) assess whether today's mobile distributed systems are adequate enough to satisfy the requirements of highly dependable wireless-based systems, and ii) evaluate how these systems meet dependability needs for the consumer-electronic mass-market, such as, achieving reasonably low failure rates at a reasonable cost [93].

Despite these concerns, very few studies have looked into the dependability of mobile distributed systems. As a result, there is little understanding of how and why these systems fail or the methods/techniques needed to meet consumer expectations of device robustness and reliability. The need to comprehend how these system fail cannot be understated. Today, no real numbers are available, neither published nor communicated by manufacturers, able to give an order of magnitude estimate of failure rate experienced by mobile distributed systems' users. It becomes thus hard to propose novel solutions for fault tolerance or new models for fault forecasting if one has no notion of the actual failure behavior of these systems.

Learning about the real failure behavior of these systems would be a major step forward. A well established methodology to evaluate the dependability of operational systems and to identify their dependability bottlenecks is represented by field failure data analysis (FFDA).

The effort profused in this dissertation deals with the field failure data analysis of mobile distributed systems. To simplify the work, such analysis addresses separately the two main components that build a MDS, i.e., mobile devices and wireless communication means, trying to face the new issues which arise when applying the FFDA methodology to this new class of systems. These issues are mainly related to the specificity of mobile devices, to the absence of publicly available data sources, and to the lack of previous experiences. In particular, for wireless communication means, it is needed to deal with the following challenging questions:

- *Which data source can be adopted?* Or, in other terms, which resources need to be monitored to gain the needed understanding on wireless technologies failures?

- *Can we use idle workloads?* Wireless communications are used in a spot way. Hence, the mere adoption of idle workloads (i.e., the normal load under which the system operates) may not guarantee that continuous time dependability measures can be properly estimated.

The situation gets even worse in the case of mobile devices. In particular, the following open issues arise:

- *Where do we have to start from?* There is no prior experience on the FFDA of mobile devices. In particular, the failure modes are still unknown. This knowledge is a prerequisite to understand what type of information is to be gathered.

- *How to collect the failure data?* Differently from traditional systems, no standard techniques have been defined so far to perform an on-line logging of the activity of mobile devices.

This dissertation tries to give answers to the above mentioned questions, by proposing two FFDA campaigns on MDSs, one addressing a wireless communication technology, and another related to mobile devices. Specifically, the former proposes a detailed FFDA on Bluetooth Personal Area Networks (PANs), whereas the second addresses smart phone devices equipped with the Symbian OS. Both Bluetooth and the Symbian had been chosen for their widespread use in today's MDSs, hence they are representative of a wide class of today's MDSs. The proposed findings are the result of a three years experience, and partially extend previously published results, as [27][29][3].

As far as the Bluetooth campaign is concerned, multiple failure data sources and automated workloads are adopted. The use of multiple data sources enables to monitor the

behavior of the thorough set of components which build the Bluetooth protocol stack. On the other hand, the adoption of automated workload is needed to measure continuous time dependability attributes and to gather a richer set of information in a relatively short period of time. The experiments provide useful insights which are then used to characterize failures distribution and to improve the dependability level of Bluetooth PANs. More specifically, the novel contribution of the work is threefold. First, a detailed failure model of Bluetooth PANs is defined. Second, the self-robustness of Bluetooth wireless channels with respect to faults affecting the radio channels is characterized. Third, failure masking and recovery actions are proposes, able to improve the dependability level of Bluetooth PANs by orders of magnitude.

As for the smart phone campaign, the starting point of the study has been an analysis of failure reports submitted by mobile phones' users on publicly available web forums. Based on this experience, a failure logger has been developed and deployed on actual smart phones, which is able to capture all the failure information of interest. Results from the experimental campaign are totally new to the FFDA research community and allow to quantify the dependability of today's commercial smart phones, and to put the evidence on the more common causes lying behind the failures.

The rest of the dissertation is organized as follows.

Chapter 1 provides the needed background on mobile distributed systems and on the dependability of computer systems, then it reviews the state of art of mobile distributed systems' dependability.

A description of the field failure data analysis methodology is given in chapter 2. The chapter also proposes a framework to analyze and confront the related work in the area,

and concludes with a detailed comparison of the most significant FFDA campaigns which have been conducted over the last three decades.

Chapter 3 is dedicated to the FFDA study of Bluetooth PANs. After giving the necessary background on the Bluetooth technology, the chapter outlines the data collection methodology which has been adopted, included the description of the workloads that have been used to exercise the network. The details on the key findings, along with the evaluation of the obtained dependability improvement is given at the end of the chapter.

The Symbian smart phones FFDA is discussed in chapter 4. Similarly to chapter 4, it first provides the needed background on the Symbian OS, then it discussed the preliminary results gained from the analysis of freely available failure reports posted by mobile phones users. Moved from these results, the chapter describes the used data collection methodology, discusses the design of the failure logger, and details the results of the analysis performed by means of the logger.

The dissertation concludes with final remarks and the indication of the lessons learned. It is also evidenced the need for a unified methodology of FFDA studies, able to overcome the diversity problems arising from the conduction of several different studies from different actors. Such a methodology could facilitate the communication between the FFDA research community, enabling a simpler comparison between future studies and enriching their credibility.

Since we cannot know all that there is to be known about anything, we ought to know a little about everything.

Blaise Pascal

Chapter 1

Dependability of Mobile Distributed Systems

Today's advantages in mobile computing hardware, as well as wireless networking, deliver more and more complex mobile computing platforms, which today encompass a variety of systems, each one characterized by specific kinds of mobile terminals and communication protocols. The wide spread use of these mobile computing platforms is leading to a growing interest of dependability issues. This chapter introduces separately the notions of mobile distributed systems and computer systems dependability. Then it reviews the recent experience on Mobile Distributed Systems dependability, motivating the need for more experimental research.

1.1 Mobile Distributed Systems, and Mobile Computing

The recent evolution in device miniaturization and wireless communication technologies makes it possible to integrate small and portable computing devices into distributed systems, leading to the notion of Mobile Distributed Systems (MDSs). Mobile devices include laptop computers, handheld devices such as personal digital assistants (PDAs) and mobile phones, wearable devices, and wireless sensors. The word "mobile" derives from the ability of these devices to connect conveniently to networks in different places, making the mobile computing paradigm possible. Mobile computing, is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment [33].

In mobile computing, users who are away from their home intranet are still provided with access to resources via the devices they carry with them. For this reason, is crucial for these devices to be connected trough some form of wireless media. To this purpose, several wireless technologies have been proposed recently. Examples are Bluetooth [12], IEEE 802.11 (Wi-Fi [49]) and IrDA (Infrared Data Association [77]) for short ranges communication, and 2.5G, 3G cellular for wider coverages.

A MDS can thus be defined as a computer system which enables the mobile computing paradigm and that is composed by two essential elements:

- a set of mobile devices;
- one or more wireless communication means.

Several instances of MDSs exist, that can be differentiated with respect to the wireless means they adopt. Generally speaking, we can distinguish infrastructure-based MDSs, where the connectivity is assured by means of special-purpose devices, called access points (APs), and infrastructure-less MDSs, where the communication takes place via only multi-hop wireless links between mobile devices. Infrastructure-based MDSs adhere to the nomadic computing paradigm [59], i.e., a special typology of mobile computing where mobile devices are nomads in wireless domains served by means of APs. Examples are IEEE 802.11 networks, Bluetooth Personal Area Networks [11], and cellular networks. On the other hand, examples of infrastructure-less MDSs are mobile-ad-hoc networks (MANETs) [92], and wireless sensor networks (WSNs) [2].

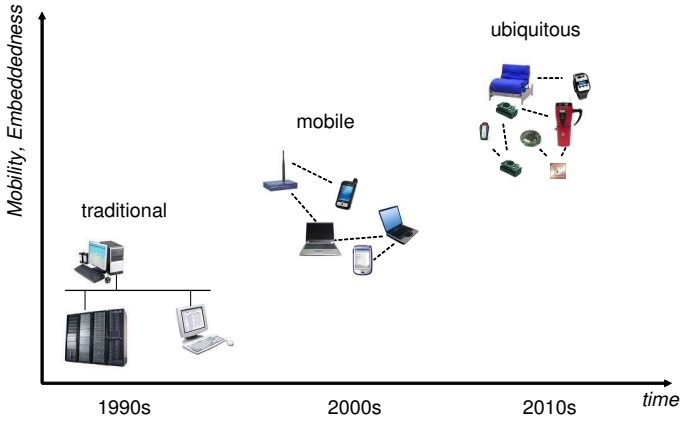


Figure 1.1: Distributed systems evolution with respect to time, mobility, and integration with the physical world (embeddedness)

Mobile computing is slightly evolving towards the ubiquitous computing vision. Ubiquitous computing [111] is the harnessing of many small, cheap computational devices that are present in users' physical environments, including the home, office, and elsewhere. They will be embedded in walls, chairs, clothing, light switches, cars - in everything. The term "ubiquitous" is intended to suggest that such small devices will be so pervasive in everyday objects that they will eventually become transparent to the user, and completely integrated in the environment. The presence of computers everywhere only becomes useful when they can communicate with one other. Hence, even in this case, it is essential to provide these devices with wireless communication potentialities.

Figure 1.1 envision a graphic representation of the present and possible future evolution of distributed systems, from traditional fixed systems and mobile systems to future ubiquitous systems.

The race towards such innovation, pushes more and more mobile and embedded devices on the market, with novel hardware platforms, communication technologies, and operating

systems. Under tremendous market pressure and continuously shrinking time to market, manufactures often cannot afford long comprehensive testing of the devices. As a result, new (or enhanced) devices are often tested by the consumers, who start to be more concerned about dependability issues. This may represent an issue for the fulfillment of the so-called "everyday dependability" requirements (meeting consumers needs with reasonable low failure rates at a reasonable cost [93]). More importantly, failure events may become unacceptable as the use of MDSs is more often hypothesized into business- and mission-critical scenarios.

Despite these concerns, very few studies have looked into the dependability of mobile distributed systems. As a consequence, there is little understanding of how and why mobile devices and/or wireless communication infrastructures fail or the methods/techniques needed to meet consumer expectations of robustness and reliability.

1.2 Basic Notions of Dependability

The effort on the definition of the basic concepts and terminology for computer systems dependability dates back to 1980, when a joint committee on "Fundamental Concepts and Terminology" was formed by the Technical Committee on Fault-Tolerant Computing of the IEEE Computer Society and the IFIP Working Group 10.4 "Dependable Computing and Fault Tolerance". A synthesis of this work was presented at FTCS-15 in 1985 [65], where computer system dependability was defined as *the quality of the delivered service such that reliance can justifiably be placed on this service.* This notion has evolved over the years.

Recent efforts from the same community define the dependability as *the ability to avoid service failures that are more frequent and more severe than is acceptable* [1]. This last definition has been introduced since it does not stress the need for justification of reliance.

The dependability is a composed quality attribute, that encompasses the following sub-attributes:

- **Availability**: readiness for correct service;
- **Reliability**: continuity of correct service;
- **Safety**: absence of catastrophic consequences on the user(s) and the environment;
- **Confidentiality**: absence of improper system alterations;
- **Maintainability**: ability to undergo modifications and repairs.

Recently, the notion of *everyday dependability* [93] has emerged as a new challenge for the computer system dependability community. While this community has traditionally faced important research problems related to business- and mission-critical systems, which risk catastrophic failures, the widespread use of consumer electronics and resource-constrained mobile devices poses new dependability requirements even for commercial applications.

Everyday systems must be sufficiently dependable for the needs of everyday people. They must thus provide cost-effective service with reasonable amounts of human attention. Dependability for these everyday needs arises from matching dependability levels to actual needs, achieving reasonably low failure rates at reasonable cost, providing understandable mechanisms to recognize and deal with failure, and enabling creation of individually-tailored

systems and configurations from available resources.

1.2.1 Threats

The causes that lead a system to deliver an incorrect service, i.e., a service deviating from its function, are manifold and can manifest at any phase of its life-cycle. Hardware faults and design errors are just an example of the possible sources of failure.

These causes, along with the manifestation of incorrect service, are recognized in the literature as dependability threats, and are commonly categorized as *failures*, *errors*, and *faults* [1].

A **failure** is an event that occurs when the delivered service deviates from correct service. A service fails either because it does not comply with the functional specification, or because this specification did not adequately describe the system function. A service failure is a transition from correct service to incorrect service, i.e., to not implementing the system function. The period of delivery of incorrect service is a service outage. The transition from incorrect service to correct service is a service recovery or repair. The deviation from correct service may assume different forms that are called *service failure modes* and are ranked according to *failure severities*.

An **error** can be regarded as the part of a system's total state that may lead to a failure. In other words, a failure occurs when the error causes the delivered service to deviate from correct service. The adjudged or hypothesized cause of an error is called a **fault**. Faults can be either internal or external of a system. Depending on their nature, faults can be

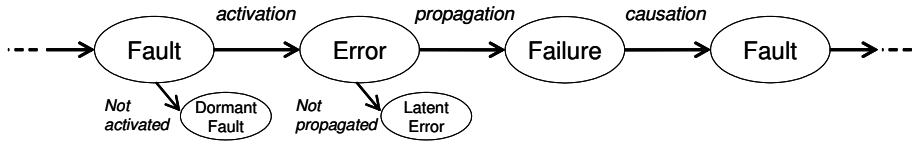


Figure 1.2: The chain of threats of faults, errors, and failures (adapted from [1])

classified as:

- *Development faults*: include all the internal faults which originate during the development phase of a system's hardware and software.
- *Physical faults*: include all the internal faults due to physical hardware damages or misbehaviors.
- *Interaction faults*: include all the external faults deriving from the interaction of a system with the external environment.

Failures, errors, and faults are related each other in the form of a *chain of threats* [1], as sketched in figure 1.2. A fault is *active* when it produces an error; otherwise, it is *dormant*. An active fault is either i) an internal fault that was previously dormant and that has been activated, or ii) an external fault. A failure occurs when an error is *propagated* to the service interface and causes the service delivered by the system to deviate from correct service. An error which does not lead the system to failure is said to be a *latent* error. A failure of a system component *causes* an internal fault of the system that contains such a component, or causes an external fault for the other system(s) that receive service from the given system.

1.2.2 Formalization and Measures

This section provides a more formal characterization of the dependability attributes of interest for this dissertation, along with the basic measures which are commonly used to quantify them.

Reliability

The reliability $R(t)$ of a system is the conditional probability of delivering a correct service in the interval $[0, t]$, given that the service was correct at the reference time 0 [100]:

$$R(0, t) = P(no\ failures\ in\ [0, t] | correct\ service\ in\ 0) \tag{1.1}$$

Let us call $F(t)$ the unreliability function, i.e., the cumulative distribution function of the failure time. The reliability function can thus be written as:

$$R(t)=1\text{-}F(t) \tag{1.2}$$

The reliability was the only dependability measure of interest to early designers of dependable computer systems. Since reliability is a function of the mission duration T, mean time to failure (MTTF) is often used as a single numeric indicator of system reliability [82]. In particular, the time to failure (TTF) of a system is defined as the interval of time between a system recovery and the consecutive failure, as evidenced in figure 1.3.

Another widely adopted measure of reliability is the failure rate, that is, the frequency with which a system fails. Failure rates can be expressed using any measure of time, but hours is the most common unit in practice. The Failures In Time (FIT) rate of a device is

the number of failures that can be expected in one billion (10^9) hours of operation. This term is used particularly by the semiconductor industry. Usually, the failure rate of a system is not constant during all the system life-time, but it follows the so-called bath-tube form, i.e., a system experiences a decreasing failure rate when it is firstly deployed, due to infant-mortality failures, then it follows a rather constant failure rate during the operational life, and, finally, it experiences an increasing failure rate at the end of its life, due to wear-out failures.

Maintainability

The maintainability, $M(t)$, is generally referred as the ability of a system to be *easily* repaired after the occurrence of a failure. A commonly adopted indicator for the maintainability is the mean time to recover (MTTR). In particular, the time to recover (TTR) can be defined as the time needed to perform a repair, that is, the interval of time between a failure and its consequent recovery, as shown in figure 1.3.

Availability

A system is said to be available at a the time t if it is able to provide a correct service at that instant of time. The availability can thus be thought as the expected value $E(A(t))$ of the following $A(t)$ function:

$$A(t) = \begin{cases} 1 & if\ proper\ service\ at\ t \\ 0 & otherwise \end{cases} \tag{1.3}$$

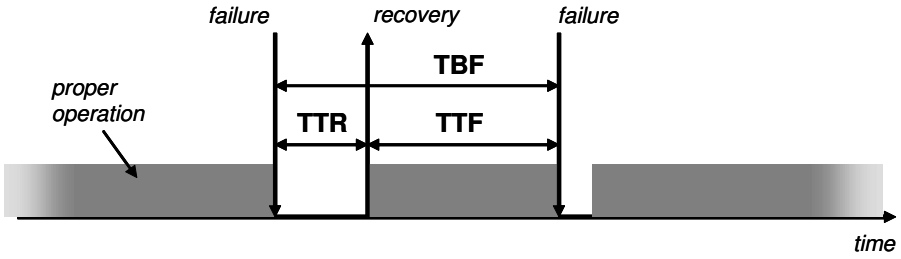


Figure 1.3: TTF, TTR and TBF

In other terms, the availability is the fraction of time that the system is operational. The measuring of the availability became important with the advent of time-sharing systems. These systems brought with it an issue for the continuity of computer service and thus minimizing the "down time" became a prime concern. Availability is a function not only of how rarely a system fails but also of how soon it can be repaired upon failure. Clearly, a synthetic availability indicator can be computed as:

$$Av = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} \tag{1.4}$$

where MTBF = MTTF + MTTR is the mean time between failures. As evidenced in figure 1.3, the time between failures (TBF) is the time interval between two consecutive failures. Obviously, this measure makes sense only for the so-called repairable systems.

When measuring MTTF and MTTR, it is important to characterize their variability, in terms of the standard deviation or in terms of the coefficient of variation. The coefficient of variation C_v is defined as the standard deviation divided by the mean. The advantage of using the C_v as a measure of variability, rather than the standard deviation, is that it is normalized by the expected value, and hence allows comparison of variability across distributions with different expected values.

1.2.3 Means

Over the course of the past 50 years many means have been developed to attain the various attributes of dependability. These means can be grouped into four major categories [1]:

- **Fault prevention**, to prevent the occurrence or introduction of faults. Fault prevention is enforced during the design phase of a system, both for software (e.g., information hiding, modularization, use of strongly-typed programming languages) and hardware (e.g., design rules).

- **Fault tolerance**, to avoid service failures in the presence of faults. It takes place during the operational life of the system. A widely used method of achieving fault tolerance is redundancy, either temporal or spatial. Temporal redundancy attempts to reestablish proper operation by bringing the system in a error-free state and by repeating the operation which caused the failure, while spatial redundancy exploits the computation performed by multiple system's replicas. The former is adequate for transient faults, whereas the latter can be effective only under the assumption that the replicas are not affected by the same permanent faults. This can be achieved through design diversity [5].

  Both temporal and spatial redundancy requires error detection and recovery techniques to be in place: upon error detection (i.e., the ability to identify that an error occurred in the system), a recovery action is performed. Such a recovery can assume the form of rollback (the system is brought back to a saved state that existed prior the

occurrence of the error; it needs to periodically save the system state, via checkpointing techniques [109]), rollforward (the system is brought to a new, error-free state), and compensation (a deep knowledge of the erroneous state is available to enable error to be masked). Fault masking, or simply masking, results from the systematic usage of compensation. Such masking will prevent completely failures from occurring.

The measure of effectiveness of any given fault tolerance technique is called its coverage, i.e, the percentage of the total number of failures that are successfully recovered by the fault tolerance mean.

- **Fault removal**, to reduce the number and severity of faults. The removal activity is usually performed during the verification and validation phases of the system development, by means of testing and/or fault injection [6]. However, fault removal can also be done during the operational phase, in terms of corrective and perfective maintenance.

- **Fault forecasting**, to estimate the present number, the future incidence, and the likely consequences of faults. Fault forecasting is conducted by performing an evaluation of the system behavior with respect to fault occurrence or activation. Evaluation has two aspects: qualitative, or ordinal, evaluation, that aims at identifying, classifying, and ranking the failure modes that would lead to system failures; and quantitative, or probabilistic, evaluation, that aims to evaluate in terms of probabilities the extent to which some of the attributes are satisfied; those attributes are then viewed as measures.

The quantitative evaluation can be performed at different phases of the system's life cycle: the design phase, the prototype phase and the operational phase [50]. In the design phase, the dependability can be evaluated via modeling and simulation, including simulated fault injection. The simulation can give immediate feedback to the designers who can timely improve the design. Simulation parameters are however based on past experiences on same systems, and these parameters can be often invalidated by changes in the technology. During the prototype phase, a prototype version of the system runs under controlled conditions. This activity can only study the effects of induced faults. Important measures, such as the mean time to failure, cannot be derived. It is carried out via controlled fault injection experiments, in order to evaluate the system resilience to software and/or hardware faults, including the coverage and recovery capabilities of the system. Finally, during the operational phase, field failure data analysis (FFDA) can be performed, aiming at measuring the dependability attributes of a system according to the failures that naturally manifest during system operation. As FFDA is the central topic of this dissertation, it is widely described in chapter 2.

1.3 Experiences on Mobile Distributed Systems' Dependability

Recent research efforts on MDSs have mainly addressed architectural and technological issues, such as the definition of ad-hoc routing protocols, mobility management middleware/architectures, and wireless communication protocols. However, as MDSs increase in popularity, more attention is being devoted to dependability issues. In the early 90s, typical failure modes for a wireless system were already clear [61], while at the beginning of this century has been recognized that "as wireless and mobile services grow, weaknesses in network infrastructures become clearer. Providers must now consider ways to decrease the number of network failures and to cope with failures when they do occur" [97].

In the following, exemplary experiences which have been recently matured on MDS's dependability are reported.

1.3.1 Failure modes

The typical failure modes of a MDS are known since many years. In [61], Vaidya et al. proposed the following three failure modes:

- node failures, i.e., a node of the MDS stops running;
- connectivity failures, that is, a node looses its connectivity to the wireless network or it is not able to connect to the network;
- packet losses, due to weak wireless links.

Since the focus in [61] was on cellular networks, connectivity failures were referred to as the unavailability of the connection to the access point. With the advent of MANETs and WSNs, connectivity failures broaden their scope, leading to *isolation* or *partitioning* failures, i.e., one or more nodes loosing their connectivity to the rest of the network due to failures of the nodes in the proximity. Examples of works dealing with isolation failures are [10], [81], and [17].

1.3.2 Fault tolerance techniques

MDSs' fault tolerance techniques can be regarded with respect to the failure class they address. Here, experiences on the tolerance of connectivity failures, access point failures, and node failures are reported.

As for connectivity failures, the contribution in [22] proposes an apporach of tolerating connectivity problems due to the existence of "shadow regions", in IEEE 802.11g wireless networks. Simply, a redundant access point (AP) is placed in the shadow region to serve the mobile stations which roam into that region. With numerical examples, authors show that the redundancy schemes demonstrate significant improvement in connection dependability over the scheme with no redundancy. Zandy et al. [114] present two systems, reliable sockets (rocks) and reliable packets (racks), that provide transparent network connection mobility using only user-level mechanisms. Each system can detect a connectivity failure within seconds of its occurrence, preserve the endpoint of a failed connection in a suspended state for an arbitrary period of time, and automatically reconnect, with correct recovery of

in-flight data.

A mean to address partitioning problems in MANETs is described in [81]. In particular, the approach tries to recover the disconnected portion of the MANET by deploying forwarding nodes. The forwarding nodes can automatically move to appropriate locations for interconnecting network partitions. The mechanism is distributed and self-organized and can be integrated with other routing protocols. Connectivity failures for nomadic wireless networks are also addressed by triggering handoff procedures as soon as a broken link event is recognized [8, 105].

Access point failures can be considered as more severe than connectivity failures, because, when an access point fails, all of the mobile stations connected to a wired network via the access point loose connectivity. A new fault-detection approach for access point failures is presented in [43], with reference to IEEE 802.11 wireless networks. The approach is based on the signal-to-noise ratio, and promises to be more effective than traditional heartbeats. Moreover, authors describe and compare three techniques to recover from access-point failures in 802.11 wireless networks, namely access point redundancy (similarly as [22]), overlapped coverage, and multiplexed links (multiple wireless links for each mobile device).

The tolerance of node failures have been in part addressed by means of checkpointing techniques. The scheme proposed in [83] adopts optimistic logging for checkpointing, since it exhibit lower failure-free operation and failure recovery costs compared to other logging schemes. In the proposed scheme, the task of logging is assigned to the APs to reduce the message overhead. A similar approach is suggested in [113], which is also able to tolerate

access point failures.

The problem of node failures becomes particularly significant in the context of infrastructure-less MDSs, since node failures may lead to partitioning failures. In particular, there is a need for fault tolerance in WSNs, due to the harsh environmental conditions in which such networks can be deployed. The work in [30] focuses on finding algorithms for collaborative target detection with wireless sensor networks that are efficient in terms of communication cost, precision, accuracy, and number of faulty sensors tolerable in the network. Strategies for node failures tolerance have also been proposed in [61], with particular emphasis on data management.

1.3.3 Dependability improvement

As a response to the solicitation given in [97] about the need to decrease the number of failures, there have been several attempts for improving the dependability of different aspects of MDSs.

Intelligent, goal-directed mobility algorithms for achieving desired topological characteristics is introduced in [17], with reference to MANETs. These algorithms can improve the connectivity, coverage, and diameter of a MANET, even when faults affect the nodes of the network. In [38] authors propose an adaptive hybrid ARQ/FEC scheme to enhance the data throughput over Bluetooth networks based on observed error rates. The scheme is demonstrated to outperform the standard Bluetooth schemes, in the presence of errors.

[60] is concerned with the choice of Cyclic redundancy codes (CRCs) suitable for resource-constrained embedded systems, equipped with 8-bit micro-controllers. Authors evaluate the options for speeding up CRC computations on such processors, and evaluate classes of CRC generator polynomials which have the same computational cost as 24- or 16-bit CRCs, but provide 32-bit CRC levels of error detection. Finally, they recommend good polynomials within those classes for data word lengths typical of embedded networking applications.

A mobility management solution that improves the connection availability is presented in [28]. In particular, a Last Second Soft Handoff scheme is proposed, able to minimize connection unavailabilities in spite of transient signal degradations and access point overloads. The scheme has been integrated into a mobility management architecture, which provides connection awareness by means of an API, named Nomadic Computing Sockets (NCSOCKS), which handles temporary disconnections similarly to the reliable sockets approach [114].

1.3.4 Dependability Modeling

The modeling approach has been adopted to address several dependability aspects of MDSs.

The work in [10] investigates the connectivity of MANETs in a log-normal shadow fading environment. Assuming a spatial Poisson distribution of the network nodes, authors derive a closed-form expression for the probability that a node is isolated. The same issue is addressed in [89], where authors also consider the mobile version of the problem, in which nodes are allowed to move during a time interval. In [86] the Stochastic Activity Network

modeling approach is adopted for evaluating the dependability of a General Packet Radio Service (GPRS) network under outage conditions. Stochastic Activity Networks have also been adopted in [14], where authors introduce a general framework that encompasses performance, dependability and mobility of software systems and that supports the specification and estimate of dependability measures.

1.3.5 Dependability Evaluation

Fault forecasting of MDSs is generally undertaken by means of modeling and simulation activities. Examples are all the works considered in the previous subsection. As a further example, Trivedi et al. [23] propose models to evaluate the survivability of a wireless network. Network survivability reflects the ability of a network to continue to function during and after failures. Authors perceive the network survivability as a composite measure consisting of both network failure duration and failure impact on the network. Assuming Markovian property for the system, this measure is obtained by solving a set of Markov models.

Limited experience has been matured on the field dependability evaluation of operational MDSs. This is in part due to the relative youth of these systems, but it is also a consequence of the difficulties which arise when applying the FFDA methodology to MDSs (see section 2.6). Some preliminary effort in this direction is represented by works such as [74] and [39]. The former proposes a FFDA for the base stations of a wireless telecommunication system, whereas the latter reports on a collection of user-perceived failure data from Bluetooth piconets. Further effort is represented by the material presented in this

dissertation, as described in chapters 3 and 4.

1.4 The Need for FFDA on Mobile Distributed Systems

The lack of experiences on the FFDA of MDSs motivates the need for more research providing deep understanding on MDSs' failure dynamics, measures, and underlying causes. Such understanding is useful to drive the design, and it is one of the key issues in the discussions of architecture and system structure of a product [25]. Hence, it cannot be understated. Today, if we were to ask for an order of magnitude estimate of failure rate experienced by MDSs users, we do not have any real numbers, neither published nor made available by manufacturers.

At the same time there is an increasing need for everyday dependability in the commercial arena. However, it is hard to realize a system or solution for fault tolerance, or to propose new models for fault forecasting if one has no notion of the actual failure behavior of these systems.

Learning about the real failure behavior of these system would be a major step forward. This task can be accomplished by proposing FFDA studies on MDSs. To simplify the work, such studies can address separately the two components that build a MDS, i.e., mobile devices and wireless communication means. Indeed, the two components are inherently different and may require different measurement strategies.

If you steal from one author it's plagiarism; if you steal from many it's research.

Wilson Mizner

Chapter 2

Field Failure Data Analysis: Methodology and Related Work

Field Failure Data Analysis (FFDA in the following) provide information that allows the effect of errors on system behavior to be understood. It provides accurate information on the system being observed, for the elaboration and validation of analytical models, and for the improvement of the development process. The collected data helps to explain and to characterize the system under study. Qualitative analysis of the failure, error and fault types observed in the field yields feedback to the development process and can thus contribute to improving the production process [26]. As stated in [50], "there is no better way to understand dependability characteristics of computer systems than by direct measurements and analysis".

This chapter discusses the principles of the FFDA methodology and provide a useful framework to evaluate and classify FFDA studies. Based on this framework, the related literature of the latest three decades is examined and compared. Finally, a discussion about open issues and challenges about the FFDA of MDSs concludes the chapter.

2.1 FFDA Objectives and Applicability

The Field Failure Data Analysis of a computer system embraces all fault forecasting techniques which are performed in the operational phase of the system's life time. This analysis aim at measuring dependability attributes of the actual and deployed system, under real workload conditions. By measuring it is meant to monitor and record natural occurring errors and failures while the normal system operation. In other words, the failing behavior is not forced or induced in the systems. The objective of a FFDA campaign mainly concerns

the detailed characterization of the actual dependability behavior of the operational system. More in detail, FFDA studies main objectives can be summarized as the following:

- identification of the classes of errors/failures as they manifest in the field, along with their relative severity and correlation among them. In other terms, FFDA is useful to derive the actual failure model of an operational system;
- analysis of failure and recovery times statistical distributions;
- correlation between failures and system workload;
- modeling of the failing behavior and recovery mechanisms, if any;
- identification of the root causes of outages, and indication of dependability bottlenecks;
- provision of figures useful to validate or to populate simulated failure models;
- derivation of general results which a crucial to guide research and development.

Although FFDA studies are useful for evaluating the real system, they are limited to manifested failures, such as the ones that can be traced. In addition, the particular conditions under which the system is observed can vary from an installation to another, thus casting doubts on the statistical validity of the results. It is worth noting that the analysis of data collected on a given system is hardly beneficial to the current version of the system. It can be instead useful for the successive generations of systems. Finally, FFDA studies may require a long period of observation of the target system, especially when the system is robust and failure events are rare.

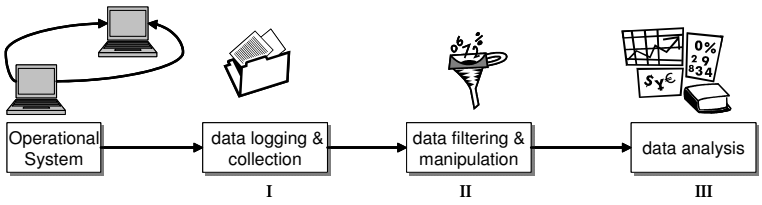


Figure 2.1: The FFDA methodology

To achieve statistical validity and to shorten the observation period, these studies should be carried out on more than one deployed system, each of them under different environmental conditions.

2.2 The FFDA Methodology

FFDA studies usually account three consecutive steps, as shown in figure 2.1: i) data logging and collection, where data are gathered from the actual system, ii) data filtering and manipulation, concerning the extraction of the information which are useful for the analysis, and iii) data analysis, that is the derivation of the intended results from the manipulated data. In this section, details about the best practice on each of these steps are presented.

2.2.1 Data Logging and Collection

Data logging and collection consists in the definition of *what* to collect and *how* to collect it. This require a preliminary study of the system, and its environment, in order to identify the technique that can be successfully used, or to decide whether it is necessary to develop an ad-hoc monitoring system. The choice of the appropriate technique also depends on the

purposes of the analysis.

Common techniques for data logging and collection are failure reports and event logging. Failure reports are generated by human operators, typically users or specialized maintenance staff. A report usually contains information such as the date and hour when the failure has occurred, a description of the observed failing behavior, the action taken by the operator to restore proper operation, the hardware/software module pinpointed as responsible of the failure, and, if possible, the root cause of the failure. The problem with this technique is that human operators are responsible for the detection of the failure, hence some failure may remain undetected. Moreover, the information contained in the report can vary from one operator to another, depending on his experiences and opinions. Recently, automated failure report systems have been proposed. An example is represented by the Microsoft's Corporate Error Reporting software. It creates a detailed report every time that an application crashes or hangs, or when the OS crashes. The report contains a snapshot of the computer's state during the crash. This information includes a list containing the name and time-stamp of binaries that were loaded in the computer's memory at the time of crash, as well as a brief stack trace. This information allows for a quick identification of the routine that caused the failure as well as the reason and cause for the failure.

Event logs are machine-generated. Data are logged by user and system applications and modules running on the machine, and contains information either about the regular execution or about erroneous behaviors. Hence, from these logs, it is possible to extract useful information about failures which occur on the system. An event log entry typically contains a time-stamp of the event and a description, along with the application/system

module that signaled the event. A limit of event logging is that the detection of a failure event depends on whether or not the application/system module logs that particular event. In other terms, not all the possible erroneous conditions are logged. Thus, with this technique it might be hard to pinpoint root causes of each failure event.

An example of an event logging system is the `syslogd` daemon for Unix operating systems. This background process records events generated by different local sources: kernel, system components (disk, network interfaces, memory), daemons and application processes that are configured to communicate with `syslogd`. Different event types with different severity levels are generally recorded. Severity levels are: 1) emergency, 2) alert, 3) critical, 4) error, 5) warning, 6) notice, 7) info, and 8) debug. Events are composed by a time-stamp, the host, the user, the process name and id, and the event description itself. The configuration file `/etc/syslog.conf` specifies the destination file of each event received by `syslogd`, depending on its severity level and its origin. A file of severity level X will contain entries which severity ranges from level 1 to level X. Hence, a debug log file holds all kinds of information, whereas an emergency log file only keeps panic information.

The same principles apply to the MS Windows operating systems. For NT and 2K, the event logger is implemented as a system service that runs in the background and waits for processes running on the local (or a remote) system to send it reports of events. Each event report is stored in a specific event log file. Three event log files are defined: the security log, for security information and auditing, the system log, for events logged by system modules, and the application log, for application related information. Events are composed by an event type (information, error, warning), a time-stamp, the source of the

event, the category, the event ID, the user, and the computer.

In some cases failure reports and event logs cannot be used, and ad-hoc monitoring systems need to be developed. First, these techniques are not available for any class of system. Examples are the Java virtual machine [32], and the Symbian OS (see chapter 4) where the native monitoring systems are neither present nor sufficient to conduct proper analysis. Second, these techniques may result inadequate for a certain class of studies. For instance, the work in [35] aims at characterizing the security behavior of a system under attack. Thus, the monitoring system had to take into account and collect the network traffic affecting the machines under attack.

Usually, FFDA studies adopt idle workloads, i.e., the normal load under which the system operates. However, there are cases where the system is solicited under automated workloads, i.e, applications running on the system to emulate the potential use of the system. Automated workloads are required in those cases where the system load is sporadic or highly intermittent and thus the analysis under the idle workload is not feasible. They may also be useful for interactive systems, which require human operators to use the system. The workload can in this case emulate the behavior of a human operator.

2.2.2 Data Filtering and Manipulation

Data filtering and manipulation consist in analyzing the collected data for correctness, consistency, and completeness. This concerns the filtering of invalid data and the coalescence of redundant or equivalent data. This is especially true when event logs are used. Logs,

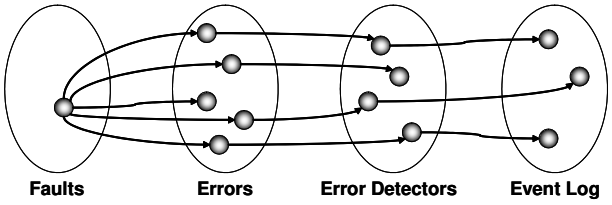


Figure 2.2: Multiple event reporting phenomenon (adapted from [46])

indeed, contain many information which are not related to failure events. In addition, events which are close in time may be representative of one single failure events. They thus need to be coalesced into one failure event.

Filtering is used to reduce the amount of information to be stored, and to concentrate the attention only on a significant set of data, thus simplifying the analysis process. Two basic filtering strategies can be adopted: blacklist and whitelist strategies. The blacklist can be thought as a list of all the terms that surely identify an event which is not of interest for the analysis. The blacklist filtering discards all those events which description message contains at least one of the blacklist terms. On the contrary, the whitelist is the list of all permitted terms, hence only events which contain these terms are not rejected.

Coalescence techniques can be distinguished into temporal, spatial, and content-based. Temporal coalescence, or *tupling* [46], exploits the heuristic of the *tuple*, i.e., a collection of events which are close in time. The heuristic is based on the observation that often more than one failure events are reported together, due to the same underlying fault. Indeed, as the effects of the fault propagate through a system, hardware and software detectors are triggered resulting in multiple events. Moreover, the same fault may persist or repeat often over time. Figure 2.2 shows the multiple reporting of events. When a fault occurs, many

errors can be generated. Some of these errors are detected by error detectors, which in turn may or not report an event into the log file.

To explain how the tupling scheme works, let us represent with X_i the i-th entry in the log, and with $t(X_i)$ the timestamp of the entry X_i. The tupling algorithm, respects the following rule:

$$\texttt{IF } t(X_{i+1}) - t(X_i) < W \texttt{ THEN}$$

$$\texttt{Add } X_{i+1} \texttt{ to the tuple}$$

where W is a configurable time window. The window size is a crucial parameter which need to be carefully tuned in order to minimize collapses (events related to two different faults are grouped into the same tuple) and truncations (events related to the same fault are grouped into more than one tuple).

Spatial coalescence is used to relate events which occur close in time but on different nodes of the system under study. It allows to identify failure propagations among nodes, resulting particularly useful when targeting distributed systems. The techniques adopted for spatial coalescence are usually the same as the ones used for temporal coalescence.

Finally, content-based coalescence groups several events into one event by looking at the specific content of the events into the event log. For example, in [95] this technique is adopted to identify machine reboots: when a the system is restarted, a sequence of initialization events is generated by the system. By looking at the specific contents of these events, it is possible to develop proper algorithms to identify machine reboots sequences and group them into one "reboot" event. Also, content-based coalescence can be used to

group events belonging to the same type [15].

2.2.3 Data Analysis

"Even when the data are available, the challenge is to be able to use it, which is no simple task" [25].

This challenge is addressed in the data analysis step, which consists in performing statistical analysis on the manipulated data to identify trends and to evaluate quantitative measures. Failure classification is a first analysis step, which aims at categorizing all the observed failures on the basis of their nature and/or location. In addition, descriptive statistics can be derived from the data to analyze the location of faults, errors and failures among system components, the severity of failures, the time to failure or time to repair distributions, the impact of the workload on the system behavior, the coverage of error detection and recovery mechanisms, etc. Commonly used statistical measures in the analysis include frequency, percentage, and probability distribution [40]. They are often used to quantify the reliability, the availability, and the maintainability. Their summary characterization can be obtained by the direct measurement of the MTTF and MTTR, as already observed in section 1.2.2.

More detailed analysis try to determine the probability distribution of the "time to failure" variable, and, in some cases, of the "time to repair". This permits to detail the failure model of the system under study. To this aim, the real data are fitted with theoretical, continuous time distributions. The most adopted distributions in this field are the exponential,

the hyper-exponential, the lognormal, and the weibull. The exponential distribution was firstly adopted to model the time to failure and time to repair of electronic components. However, it has been often shown that this distribution does not fit real data, especially when the data involves multiple underlying causes or software failures. This is due to the simplistic memoryless property of the exponential distribution. If a process consists of alternate phases, that is, during any single experiment, the process experiences one and only one of many alternate phases, and these phases have exponential distributions, then the overall distribution is hyper-exponential [106]. This distribution can be used to model failure times of failures which are the manifestation of different, independent and alternate underlying causes.

Recently, the lognormal distribution has been recognized as a proper distribution for software failure rates [78]. Many successful analytical models of software behavior share assumptions that suggest that the distribution of software event rates will asymptotically approach lognormal. The lognormal distribution has its origin in the complexity, that is the depth of conditionals, of software systems and the fact that event rates are determined by an essentially multiplicative process. The central limit theorem links these properties to the lognormal: just as the normal distribution arises when summing many random terms, the lognormal distribution arises when the value of a variable is determined by the multiplication of many random factors.

The weibull distribution has been used to describe fatigue failures and electronic components failures. At present, it is perhaps the most widely used parametric family of failure distributions. The reason is that by a proper choice of its shape parameter, an increasing,

a decreasing, or a constant failure rate distribution can be obtained. Therefore, it can be used for all the phases of the bath-tube mortality curve [106].

In practice, the family of the distribution and its associated parameters have to be estimated from the collected data. Usually, a family of distribution is chosen and the fitting is conducted via a parameter estimation, e.g., maximum likelihood estimation. Goodness-of-fit tests can be conducted to determine whether the data can reasonably be declared to belong to the chosen family.

Other types of analysis are concerned with the correlation between failure distributions. The correlation can uncover possible links between failures in different hardware and software modules or in different nodes constituting the system under study. This analysis can also conduct to the discovery of trends among failure data on event logs. From a theoretical perspective, the trend analysis of event logs is based on the common observation that a module exhibits a period of (potentially) increasing unreliability before final failure. By discovering these unreliability trends, it can be possible to predict the occurrence of certain failures. To this aim, principal component analysis, cluster analysis, and tupling can be adopted [94].

Finally, the analysis activity often conducts to the development of simulation models of the dependability behavior. Models often adopted in the literature are state-machines, fault trees, Markov chains, and Petri nets. The understanding gathered from field data allows to define these models and to populate their parameters with realistic figures, e.g., failure and recovery rates.

2.2.4 Tools for FFDA

Although FFDA has evolved significantly over the last decades, the data analysis work is normally done manually with ad hoc techniques, using programming languages or statistical packages. However, these statistical packages provide only standard procedures, not a complete methodology in the context of dependability analysis for computer systems. It has thus emerged the need to have software packages which integrate a wide range of the state-of-the-art techniques in FFDA (e.g., data collecting, data coalescing, and modeling) and which can generate appropriate dependability models and measures from field data in an automatic fashion. A first example toward this direction was the MEASURE+ tool [103]. Given measured data from real systems in a specified format MEASURE+ can generate appropriate dependability models and measures including Markov and semi-Markov models, k-out-of-n availability models, failure distribution and hazard functions, and correlation parameters.

A more recent and user-friendly tool for critical systems is represented by MEADEP [101]. It consists of 4 software modules: a data preprocessor for converting data in various formats to the MEADEP format, a data analyzer for graphical data-presentation and parameter estimation, a graphical modeling interface for building block diagrams (including the exponential block, Weibull block, and k-out-of-n block) and Markov reward chains, and a model-solution module for availability/reliability calculations with graphical parametric analysis.

Analyze NOW [104] is a set of tools specifically tailored for the FFDA of networks of

workstations. It embodies tools for the automated data collection from all the workstations, and tools for the analysis of such data. Basic analysis tools are: i) the Filter, that filters out all non essential messages, ii) the Analyzer, which performs the tasks of extracting all possible information about each failure, of correlating failures collected from different workstations, and of classifying failures, and iii) tabulator and graph_TBF, which are tools for the presentation of the results, either in a tabular or graphical form.

2.3 Comparison Framework of FFDA Studies

Several FFDA studies have been proposed in the literature over the past three decades, each of them addressing different systems, collecting data from different data sources, and proposing different results. Hence, it is not simple to catch all similarities and differences among different studies, and to draw common conclusions. What the FFDA research has achieved so far, what is still missing, and if in the future we will be able to conduct FFDA studies even more efficiently and effectively are key research questions that need to be answered.

This section proposes several classifying dimensions of FFDA studies. These dimensions represent a common framework useful to compare and to analyze these studies.

In the following subsections, the dimensions are described, then, the related work on FFDA over the last three decades is examined according to them.

2.3.1 Descriptive Dimensions

Six are the proposed descriptive dimensions.

Date: The date when the study was conducted. The time framework is important to contextualize the study.

Purpose: The reasons and goals that are at the basis of the work. In the following, three coarse grained purposes are indicated:

- *FFDA Methodology*: works providing contributions to the FFDA community itself, that is, definition of new techniques and methodologies for FFDA (such as, novel collection/filtering/coalescing techniques), evaluation and comparison of different specific FFDA techniques/methodologies, explicit evidences of FFDA effectiveness.
- *Dependability Study*: traditional works on the failure classification, on the evaluation of dependability measures, such as MTTF, availability, MTTR, coverage, on the identification on statistical/simulation models, such as statistical distributions, markov-chains, finite state machines, and on correlation analysis such as trend analysis.
- *Comparison*: works which compare two or more FFDA studies trying to identify common trends and conclusions.

Actor: Indicates whether the work is performed by academy, industry, or by a collaboration of both the academia and the industry.

Source: Indicates whether the analyzed failure data is provided by third party organizations or if it is internal (e.g., laboratory testbeds, university facilities, etc.).

Target System: The actual computer system under study.

Achieved results: The most interesting results obtained by the study. Examples are the identification of statistical distributions, the percentage of failures classes (e.g., the percentage of software failures), or particular uncoverings, such as the pinpointing of dependability bottlenecks.

2.3.2 Quantitative Dimensions

Quantitative dimensions characterize the physical size of the study, in terms of the time length and the amount of collected failure data items. In particular, the considered dimensions are:

Length (l): the temporal length (measured in months) of the experiment, that is, for how long the system under study has been observed.

Data items (N_d): the number of failure data items used for the study.

When considered together, the length and the data items provide the measure of the density of events which are used for the study. In particular:

$$d = \frac{N_d}{l} \tag{2.1}$$

where d is the density measured as [items/month]. The bigger the density, the more items can be collected in a time unit. The density gives an indication of the length a study should

be performed in order to be statistically significant. For instance, if the density is low, it is necessary to run the experiment for a long period, in order to gather a significant amount of data points. A useful practice would be to observe the density for a short period, and, depending on its value, decide i) whether to increase/decrease the number of system's units to include in the analysis (the more the units, the bigger N_d), and ii) how long the experiment should take.

2.3.3 Methodology-related Dimensions

These dimensions are related to the steps of the FFDA methodology that has been discussed in section 2.2. The following dimensions are defined:

Data source: the source from where field failure data are gathered, e.g. event logs or failure reports.

Levels: the levels at which the data source are considered. They correspond to the abstract machine levels (from hardware to human operators) the gathered failure events correspond to. Examples are: hardware, network, operating system, middleware, application, and human operator.

Workload: whether the workload is idle or automated.

Manipulation: the set of techniques, if any, adopted to filter and/or process the gathered data.

Analysis: the particular statistical analysis performed on the filtered data. The performed analysis can be grouped according to the following classes:

- *Classification*: derivation of the classes of failures/errors along with their causes, if available. In other words, classification consists in the definition of the experimental failure/error model;
- *Dependability measurements*: MTTF, MTTR, FIT, availability, coverage, uptime, downtime;
- *Modeling*: definition of a statistical/simulation model from the real data. Examples are statistical distributions (weibull, lognormal, hyperexponential), markov chains, and finite state machines.
- *Correlation*: identification of either temporal trends among subsequent failures, or spatial correlation among different system's nodes. Examples of the adopted analysis techniques are trend analysis, cluster analysis, factor analysis.
- *Other*: particular analysis which do not fall in any of the above mentioned classes, such as the sensitivity analysis of tupling schemes.

Often, FFDA studies fall into more than one analysis class. For example, it is common that works proposing dependability measurements also perform a classification.

Confidence: It is indicated whether the study provides any measure for the confidence or variability of the data (e.g., standard deviation, coefficient of variation, etc.).

The defined dimensions allow to have a quick survey on the methods and techniques adopted by a particular study to gather and process data, along with the type of conducted analysis.

2.4 Analysis of Related Research

The importance of FFDA studies of computer systems has been recognized since many years. The first seminal contributions date back to the 70s with studies on the Chi/OS for the Univac [72], and CRAY-1 systems [58]. The research has then broadened its scope over the years addressing a wide set of systems and pursuing several objectives. The 80s and the 90s have been characterized by FFDA studies on mainframe and multicomputer systems, such as the IBM 370 with the MVS OS [108] [53] [98] [99] [48] [51] [52], the DEC VAX [16] [15] [102] [110], and Tandem systems [45] [46] [66]. In the second half of 90s the research slightly moved its attention to end-user and interactive operating systems, such as the various flavors of MS Windows [79] [96] [42] [55] and UNIX-based [63] [95] operating systems. At the same time, as the Internet increased in popularity, many studies emerged, trying to assess the dependability of the network of networks [56] [75] [80] [85]. The present decade has witnessed an even broader spectrum of research, adding contributions on virtual machines [32], applications [88] [25], embedded systems [18] [64], large-scale and parallel systems [87] [69] [68] [90] and mobile distributed systems [74] [29]. Over the years, many objectives have been pursued, from the mere statistical classification and modeling of failure events, to the identification of trends and correlations, and the experimental evaluation of

Table 2.1: FFDA Research trends: targets and milestones

Decade	1980s	1990s	2000s
Target Systems	Operating Systems; Applications	Operating Systems, Applications; Internet; Networked Systems	Operating Systems; Applications; Internet; Networked Systems; Large-scale Systems; Virtual Machines; Embedded and Mobile Systems
Research Milestones	FFDA Feasibility; evidences of load-failure relationship	Use of FFDA for correlation analysis and diagnosis; manipulation techniques for event logs; uncovering of the preeminence of software failures; studies on user-perceived dependability	Broader spectrum for the FFDA research; FFDA used to characterize security; manipulation techniques for novel, complex systems.

malicious attacks, such as [37] [35] [24]. Table 2.1 summarizes the trend of the FFDA research, in terms of targeted systems and milestones. These milestones are then detailed in sections 2.4.1 and 2.4.2.

About fifty high level technical papers, either published by IEEE or ACM journals and conference proceedings[1] over the last three decades, have been taken under consideration. Fundamental milestones which provoked a significant shift in the research are analyzed first. Relevant FFDA studies, which provided the community with interesting results, are considered as well. Section 2.5 reports a critical comparison of all considered studies.

[1]As main sources of our study, we considered the following technical journals: IEEE Transactions on Computers, IEEE Transactions on Reliability, IEEE Transactions on Software Engineering, IEEE/ACM Transactions on Networking, ACM Transactions on Computer Systems; and the proceedings of the following international conferences: the International IEEE Conference on Dependable Systems and Networks (DSN), the International IEEE Symposium on Fault Tolerant Computing (FTCS), the IEEE Symposium on Reliable Distributed Systems (SRDS), the Pacific Rim International Symposium on Dependable Computing (PRDC), the IEEE International Symposium on Software Reliability Engineering (ISSRE), the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.

2.4.1 Fundamental Research

Over the last three decades, the FFDA research community devoted its efforts to define methods, strategies, and techniques for failure data gathering, processing, and analysis. Several uncoverings and lessons learned are reported in this section. These experiences traced the path which has then been exploited by other researchers to conduct their studies.

Event Logs Effectiveness, and their Manipulation

Event logs are one of the most adopted failure sources. This is true since the beginning of eighties, when a pioneer FFDA study [108] achieved a relevant echo for the dependability research community. It was one of the first works to demonstrate that FFDA campaigns based on event logs are a viable approach to derive quantitative measures for system fault tolerance and recovery management. The gathered information reveals quite efficient for pinpointing major problem areas where further work could be directed.

An important methodological achievement for event log-based analysis has been the definition of tuple, as an heuristic for temporal-based coalescence [46]. Other than giving the fundamentals of tupling, the study in [46] also proposes an experimental sensitivity analysis, based on event logs from Tandem systems. In addition, authors deeply study the problem of collisions (different errors grouped in the same tuple), proposing a statistical model of collision probability versus clustering time.

A shift in the research is represented by the work in [16], addressing the analysis of event logs from 193 DEC VAX/VMS nodes. For the first time authors propose a critique

digression on the quality of the logs and how it affects the results, besides the mere analysis of data. For example, they show that the incorrect handling of bogus timestamps affects MTTF measures by one order of magnitude. Their conclusions are supported by statistical analysis performed on one of the largest set of failure data, composed by 2.35 million events. The same data set is adopted by a subsequent study from the same authors [15] that, following the wave of its predecessor, does not propose an analysis of failure data, but it rather discusses ways to improve the analysis of event logs. In particular, the comparative analysis of different tupling schemes has been conducted, by adopting these schemes on real data. In particular it demonstrates the usefulness of tupling, and evaluates new heuristic rules by means of sensitivity analysis.

Evidences of the System Load-Failure Relationship

The relationship between failure behavior and system load is clear since the first FFDA works. At the beginning of 80s, during a performance measurement campaign for a large DEC-1OA time-sharing system, it was found that the simplistic assumption of a constant system failure rate did not agree with measured data [20]. Subsequent research by the same authors [21] involves use of a doubly stochastic Poisson process to model failures. The model relates the instantaneous failure rate of a system resource to the usage of the resource considered. Moved by this research, [51] proposed an approach to evaluate the relationship between system load and failure behavior that presumes no model a priori, but rather starts from a substantial body of empirical data. The study was conducted on three

IBM 370 mainframes, and both failure data (maintenance failure reports) and performance counters (via a proprietary IBM system) were gathered. A regression analysis of failure and performance data evidenced the strong correlation between failure manifestation and system load. A similar approach was followed in [52] on two IBM 370 machines. The analysis concentrated on CPU failures. In particular, approximately 17 percent of all failures affecting the CPU were estimated to be permanent. The manifestation of a permanent failure was found to be strongly correlated with the level and the type of the workload. The increase in the probability of a failure was found to be most sensitive to a change in the interactive workload (as measured by the non batch CPU usage, the IO rate and the SVC rate). Although, in strict terms, the results only relate to the manifestation of permanent failures and not to their occurrence, there were strong indications that permanent failures are both caused and discovered by increased activity. This experimental evidence was supported by more recent studies, such as [36], by means of a fault injection campaign.

Using FFDA for Correlation Analysis, Failure Prediction, and Diagnosis

The information gathered from field data has been widely recognized as an enabler for correlation and failure prediction, which in turn leads to the possibility of failure diagnosis. The work in [53] represents a first effort in this direction, evidencing the feasibility of on-line diagnosis approaches based on trend analysis and real data. Specifically, it concentrates on the recognition of intermittent failures and defines a methodology to distinguish between

transient, permanent, and intermittent failures by looking at the correlation between consecutive failure events. About 500 groups of failures are identified over a 14 months time span.

During the same period, [71] proposed a novel failure prediction technique, called the Dispersion Frame Technique (DFT). The technique is defined by starting from the statistical characterization of real data observed on a 13 SUN 2/170 nodes, running the VICE file system, over a 22 months period. By gathering data by both event logs (regarded as errors) and from operator's failure reports (regarded as failures), authors concentrate of the identification of error trends which lead to failures. Interestingly, this is the first FFDA work which takes into account two different data sources at two different levels and which tries to relate them to identify failures root causes and underlying trends. The effectiveness of the DFT is shown via direct experiments on actual data. In particular, it is shown that the DFT uses only one fifth of the error log entry points required by statistical methods for failure prediction. Also, the DFT achieves a 93.7% success rate in failure prediction.

Another important work suggests that failure correlation cannot be underestimated. In other terms this work demonstrates that the common assumption made in dependability analysis: "failures in different components are independent" is not valid in general terms [102]. The study analyzes the dependability of 7 DEC VAX machines in a VAX-cluster. Particular attention is given to the error/failure relationship, and both TBE (time between errors) and TBF empirical distributions are given. As main results, shared resources are identified as a main dependability bottleneck, and both errors and failures are shown to occur in bursts. Although the failure correlation is low, it significantly affects the

system unavailability estimation.

Software Failures as New Protagonists

In the early 90s, field failure studies evidenced the preeminence of software failures as main responsible of system outages. An important study on Tandem systems [45] took a census of six years of failure data, showing a clear improvement in the reliability of hardware and maintenance, while indicating the trend of software as the major source of reported outages (62% in 1990 versus 34% in 1985 whereas hardware-induced outages decreased from 29% in 1985 down to 7% in 1990). The conclusion was clear: hardware faults and hardware maintenance were no longer a major source of outages, whereas software related faults needed to be tolerated. This motivated future research towards software fault tolerance.

Subsequent works emerged in the same period, putting the accent on software failures. Examples are two studies on the IBM MVS OS, [98] [99].

The first study considered software failure data collected from the IBM RETAIN (REmote Technical Assistance Information Network) database from 1985 to 1989. However, since the analysis was performed manually, the total number of reports was sampled down to 241 reports. The analysis concentrate on a detailed classification and related frequencies of errors (programming mistakes) and triggers (errors circumstances). Results pinpoint overlay errors, e.g., those related to memory management, to be more severe in terms of impact than other regular errors. The second work extends the first one by proposing a

comparison between OS errors and IMB DB2 and IMS database management systems' errors. Data are collected the same way as the previous study and the same classification approach is adopted. It's the first time application related data (e.g., DBMS data) are considered in an FFDA study.

Lessons Learned on User-Perceived Dependability

An important contribution of FFDA was the recognition of the difference between the measured and the user-perceived dependability. Specifically, it has been shown how field studies may overestimate some dependability figures, resulting into numbers that may mislead the expectations.

An interesting study in this direction was published in 1999 [55]. The work focuses on a LAN of Windows NT machines and tries to identify reboot causes by looking at what authors call "prominent events", i.e., events that preceded the current reboot by no more than an hour. Authors indicate that while the measured availability of the system was 99 percent, the user-perceived availability was only 92 percent, i.e., the system often can be alive but not able to provide a required service. The work in [96] reports similar figures (99 percent) on system availability of Windows NT and 2K workstation and servers. The need to account for the user's perception of system dependability is also stressed in the analysis of Windows 2000 dependability [79]. The previously mentioned work [55] also proposes a FSM modeling of error behavior and reports several suggestions to improve the Windows NT logging system. For example, authors suggest to add a Windows NT shutdown event

to improve the accuracy in identifying the causes of reboots. This event has actually been introduced into subsequent versions of the Windows logging systems. Other results are the evidence that often more than one reboot are needed to restore proper operation (the 60% of the cases) and that the main responsible for reboots is software (the 90% of the cases).

Using Past Experiences to Improve Next-Generation Products

The essential of field studies is the possibility of building on the results of the analysis to propose precise directions of improvement for the next waves of technologies. An exemplary study was conducted jointly by academy and Microsoft Research on the MS Windows NT operating system [79]. After reviewing a FFDA conducted on Windows NT 4, authors move towards the definition of the new features to be added to the next version of the OS, namely Windows 2000. Just to mention few examples, since the majority of system failures on Windows NT were due to the core NT system and device drivers, Windows 2000 designers and developers decided to place as many new features as possible into the user mode, and to improve the verification of software that resides in the kernel mode, by adopting a new testing and verification process. Also, OS hangs and the well known "blue screens" due to application failures were reduced in Windows 2000 through Kernel memory isolation: the kernel memory space is marked as read-only to user mode applications providing the kernel greater isolation from errant applications.

The work in [96] confirms the availability improvement of Windows 2000 with respect to Windows NT. Interestingly, Windows 2000 decreases the number of failures due to the

core system, and increases application related failures, thus demonstrating the effectiveness of the kernel memory isolation technique.

2.4.2 FFDA Relevant Studies

FFDA studies had pertained a substantial variety of IT systems and concerns, from Operating Systems to embedded systems and security, demonstrating the wide recognition for this kind of research activities. This section summarizes a relevant set of these efforts, with respect to the system or concern they refer to.

Works on Operating Systems

FFDA's first steps mainly concerned the evaluation and modeling of Operating Systems dependability. At the end of eighties, the majority of works were concerned on three popular operating system families: the IBM MVS, Tandem sytems, and the DEC VAX.

A study dated 1988 addressed the MVS operating system, aiming at building a semi-markov model of the system from real data by taking into account both normal and error behavior [48]. A key result of the work was that errors distribution was not simple exponentials. Also, the work was one of the first to use semi-markov chains for the modeling of the error behavior. Building on FFDA fundamentals, data were gathered from the operating system's event logs, and both temporal and content-based coalescence was adopted to manipulate them. Percentage of failures by failure class also shown a significant incidence of software errors: the 36% of errors were due to software, while the rest were hardware

related (CPU, memory, I/O channels, disk). This result was coherent with the trend of increasing relevance of software failures, evidenced by Gray two years afterwards [45].

As an example for the Tandem system, [66] defined an analysis methodology for its event logs through multivariate techniques, such as factor and cluster analysis. The event logs were gathered from three Tandem systems over a 7 months period. A 2-phase hyper-exponential distribution was adopted to model the error temporal behavior, according to the two error behaviors exhibited by the three systems: error bursts and isolated faults.

[110] reports on the validation of an availability model developed for DEC VAX machines in a VAXcluster. Direct availability measurement of system interruptions from the actual systems were used to validate the previously defined model. The interesting contribution is the description of model assumptions that are not supported by the data. In particular, analysis of the data revealed interruption dependencies across devices (of the same and different type) which were not taken under consideration in the model. Moreover, while the model assumes exponentially distributed failures, real failure data results to be non-exponentially distributed.

As a summary of a decade of research, the study in [67] performs a comparison of FFDA analysis on the three operating systems: the Tandem GUARDIAN fault-tolerant system, the VAX/VMS distributed system, and the IBM/MVS system. The relevant results are the following: software errors tend to occur in bursts on both IBM and VAX machines. This is less pronounced in the Tandem system, which can be attributed to its fault-tolerant design. The Tandem-system fault-tolerance reduces the service loss due to software failures by a factor of 10.

Following the wave of their predecessors, more recent works concerned recent Operating Systems, such as Unix and Microsoft Windows. The work in [63] propose an interesting and well conducted FFDA study on a server machine with the Sun SPARC UNIX OS. Starting from event logs, the work performs a classification of failures and identify the potential trends of errors which lead to failures. Summary statistics, such as MTBF and availability, are evaluated.

[42] confirmed the trend of improvement of the Windows OS family, already mentioned in section 2.4.1. The study address Windows XP SP1, and shows how the percentage of OS failures decreases from 12% for Windows 2K to the 5% of Windows XP, thus demonstrating that system crashes are often due to applications and third party software. Differently from previous work, this study exploits the Microsoft's Corporate Error Reporting software to gather failure reports.

Networked Systems Dependability

A natural shift in the FFDA research was represented by the study of networked systems. As compared to operating systems studies, networked systems studies basically add one more dimension for the analysis, i.e., the correlation of failures among the system's nodes.

The work in [95] analyzes a LAN of 298 Unix workstations. The analysis concentrates on the identification of system reboots, via content-based coalescence, on their potential causes, and on their statistical characteristics, such as uptime, downtime and availability

statistics. Also, a correlation analysis among logs belonging to different machines is performed. The analysis however evidences that there is no high correlation between client and server failures, as one could expect. This can be explained in part by the fact that errors affecting the servers persist for a short period of time and thus only clients accessing the servers when errors occur might be affected.

[112] proposes the analysis of a networked Windows NT system, composed of 503 server nodes. The study focuses on machine reboots and presents several interesting statistics. First, the time to failure distribution fits a Weibull distribution. Second, often several reboots are needed to restore normal operation. Third, although the availability of individual server is high (99%), there is a strong indication of correlation between failures in different machines. The correlation was identified through the temporal coalescence of merged event log files.

Internet Dependability

As the Internet increased in popularity, several studies attempted to characterize its dependability.

The study in [56] presents the results of a 40-days reliability study on a set of 97 popular Web sites done from an end user's perspective. The interesting aspect of the work is that it is one of the first studies where an automated workload is used to acquire the data. In particular the workload periodically attempts to fetch an HTML file from each Web site and records the outcome of such attempts. The need for the workload lyes in the spot usage

of the Internet. Some interesting results of the study are that connectivity problems seem to play a major role in determining the accessibility of the hosts and that the majority (70.5%) of the failures are short in time (less than 15 minutes).

[75] attempts to discover and discuss Ethernet anomalies from the Carnegie Mellon University network using an ethological approach. Traces of network traffic are collected by means of a hardware monitor deployed on the network. The major result is the ability to classify the network behavior as normal and anomalous.

An interesting study attempts to characterize the pathological behavior of end-to-end Internet routing [85]. The study reports on an analysis of 40,000 end-to-end route measurements conducted by means of automated workload (repeated "traceroutes" between 37 Internet sites). Authors analyze the routing behavior for pathological conditions, uncovering the prevalence of routing loops, erroneous routing, infrastructure failures, and temporary outages. Coherently with the increasing complexity of the Internet, the likelihood of encountering a major routing pathology resulted more than doubled between the end of 1994 and the end of 1995, rising from 1.5% to 3.4%.

From a study of 62 user-visible failures in three largescale Internet services, [80] observes that front-ends are a more significant problem than is commonly believed. In particular, operator error and network problems are shown to be leading contributors to user-visible failures.

Large Scale Systems

Recent studies attempted to study large scale server environments and complex parallel machines [87] [68]. Analysis of event logs from about 400 parallel server machines with the AIX operating system [87] demonstrates that although improvements in system robustness continue to limit the number of actual failures to a very small fraction of the recorded errors, the failure rates are still significant and highly variable. A subsequent work from the same authors deeply studies the logs from a production IBM BlueGene/L system and proposes empirical failure prediction methods which can predict around 80% of the memory and network failures, and 47% of the application I/O failures [68]. Prediction is very useful to on-line system diagnosis: with these prediction schemes deployed online, one is able to effectively predict failures in the future, and possibly take remedial actions to mitigate the adverse impacts that these failures would cause.

A very recent study analyzes failure data recently made publicly available by one of the largest high-performance computing (HPC) sites [90]. The data has been collected over the past 9 years at Los Alamos National Laboratory and includes 23000 failures recorded on more than 20 different systems, mostly large clusters of SMP and NUMA nodes. To date, this is the largest set of failure data studied in the literature, both in terms of the time-period it spans, and the number of systems and processors it covers. This underline the importance of having public failure data repositories available to researches for analysis. We hope this example encourages efforts at other sites to collect and clear data for public release. Specifically, the study classifies failures occurrences and models failure times (with

weibull and gamma distributions) and recovery times (with the lognormal distribution). It is shown that failure and recovery rates vary widely across systems. Moreover, it results that the curve of the failure rate over the lifetime of an HPC system is often very different from the traditional bath-tube curve.

Over the last few years, it has been recognized how FFDA studies can be significant for a wider set of systems and concerns. The three following sections report on FFDA studies on applications, virtual machines, embedded systems, and measurement-based security characterization.

Applications and Virtual Machines

In [25] Chillarege et al concentrate on failure reports from two widely distributed IBM software products. The analysis did not target event logs, rather it started from the service calls made by customers. Once evaluated by an experts team, the calls may result in new APARs (Authorized Failure Analysis Report) which are then analyzed by the software maintenance staff. As an interesting result, author define and evaluate two novel metrics, the fault weight and the failure window, which are demonstrated to be proportional to the qualitative fault severity and that can be used to control the failure reporting process at a costumer base.

A very recent study characterizes the dependability of the Java Virtual Machine starting from Bug Databases, the only public available failure data source for the JVM to date [32].

The study proposes a detailed classification and analysis of bug reports. In particular, key results are: JVM built-in error detection mechanisms are not capable of detect a considerable amount of failures (45.03%); a non-negligible percentage of reported failures indicate the presence of aging-related bugs inside the JVM; and the JVM is not expected to achieve the same level of dependability on different platforms.

Works on Embedded Systems

The work in [18] analyzes the dependability of mobile robots, based on failure reports gathered on 673 hours of actual usage by 13 robots and three manufacturers. It appears that mobile robots, in a given hour, have a 5.5% probability of failure. The reliability is very low, with an average MTBF of about 8 hours and an availability lower than 50%. As expected, field robots have higher failure rates and overall lower reliability than indoor robots, possibly because of the demands of the outdoor terrains and the relative newness of the platforms. The effectors, or platform itself, was the source of most failures for field robots whereas the biggest failure in indoor robots was with the wireless communication link.

A analysis of data on 11 years operating experience of safety critical software for nuclear reactors is presented in [64]. The study is conducted by Technicatome, a French company specialized in the design of nuclear reactors of the submarines and aircraft carriers for the French Navy. Several conclusions can be drawn from the study. In particular authors state

that the analysis of data on operating experience is an efficient mean to improve the development processes, and emergent methods, such as formal methods, would have been of little help to prevent the errors actually encountered in operation, since they concern HW/SW interactions and real-time issues extremely difficult to model. In fact, as it is very difficult to reproduce the real load and asynchronous situations in a laboratory environment, some very particular situations can only be encountered in a life-size trial.

Using Field Data to Characterize Security

As the scope of FFDA campaigns reached the security community, several works appeared which attempted to characterize and to model system vulnerabilities and attacks starting from field data. An outstanding example is represented by the Honeynet project [76]. A honeypot can be regarded as an information system resource whose value lies in unauthorized or illicit use of that resource. By placing honeypots on the Internet and by gathering data on the malicious activity affecting them, one can study the characteristics of attacks and system vulnerabilities. As an example, the study in [37] aimed at using data collected by honeypots to validate fault assumptions required when designing intrusion-tolerant systems. Authors set up three machines equipped with different operating systems (Windows NT and 2K, and Red Hat Linux) and collected network-related data (via *tcpdump*) for four months to analyze the source of attacker and the attacked ports. The work evidenced that, in most cases, attackers know in advance which ports are open on each machine, without performing any port scan. Moreover, there were no substantial differences in the attacks

made on different operating systems.

A similar study analyzes malicious activity collected from a testbed, consisting of two Windows 2K target computers dedicated solely to the purpose of being attacked, over a 109 day time period [35]. The objective was to determine the characteristics of collected attacks data that most efficiently separate attacks, allowing their statistical classification. *Ethereal* was used to collect the data, and filtering techniques were adopted. Clustering is then used to identify the most accurate characteristics for separating attacks. Results show that these characteristics are the number of bytes constituting the attack, the mean of the distribution of bytes as a function of the number of packets, and the mean of the distribution of message lengths as a function of the number of packets. This work hence demonstrate how field data represent an useful mean to recognize attacks.

The work in [24] exploits data from the *Bugtraq* database and proposes a classification of vulnerabilities. In particular, vulnerabilities are dominated by five categories: input validation errors (23%), boundary condition errors (21%), design errors (18%), failure to handle exceptional conditions (11%), and access validation errors (10%). The primary reason for the domination of these categories is that they include the most prevalent vulnerabilities, such as buffer overflow and format string vulnerabilities. Starting from this data and helped by code ispections, authors propose finite state machine models for vulnerabilities, which help to better understand their behavior and/or to uncover new ones.

Recent Contributions on Filtering and Manipulation

Over the last two years, we witnessed some effort in the direction of defining new techniques for processing field failure data. This is mainly due to the new difficulties which arise with emerging systems.

[69] defines a new and effective filtering method for the IBM BlueGene/L event logs. Effective filtering is a very desirable characteristic for these large-scale complex systems, since many log entries are produced and stored. Authors demonstrated how their filtering strategy can remove over the 99.96% of entries from a event log gathered from a IBM BlueGene/L prototype. This represents a valid shift with respect to the previous methods, where a 91.82% filtering effectiveness were reached.

The work proposed in [88] defines a novel failure prediction method, called Similar Events Prediction (SEP), which is based on the recognition of failure-prone patterns utilizing a semi-Markov chain in combination with clustering. Differently from previous approaches, SEP takes into account more information about the current system state. In addition, SEP investigates properties of the error event itself such as the type of error message, the software component that reported the event or the depth of the stack trace. The technique is compared to a straightforward prediction method based on a well-known reliability model and to the Dispersion Frame Technique (DFT) by Lin and Siewiorek. All three models have been applied to data of a complex commercial telecommunication system. Predictive power of the approaches is compared in terms of precision, recall, F-measure and accumulated runtime costs. SEP outperformed the other failure prediction techniques in all measures.

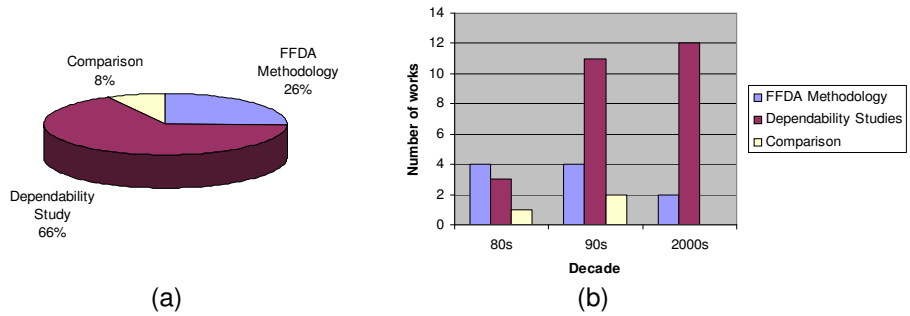


Figure 2.3: FFDA purposes a) break-up, b) distribution over the years

2.5 Comparison

In this section the wide body of research previously presented is compared with respect to the evaluation means for FFDA defined in section 2.3.

2.5.1 Purposes and Target Systems

Figure 2.3a reports a break-up of the purposes followed by the analyzed FFDA studies. As one can expect, the majority of works (66%) performs mere practical studies, addressing a particular system and presenting failure classification, stochastic model, statistical distributions, and dependability measures. A significant amount of works (26%) are devoted to the definition of new methods and techniques for improving FFDA campaigns. It should be noted, however, that, while the number of dependability studies have been increasing over the years, the number of theoretical efforts greatly decreased, if compared with the practical ones (see figure 2.3b). This indicates that the methodological results achieved in the past are still used to conduct today's studies. However, this may not be the case of emerging

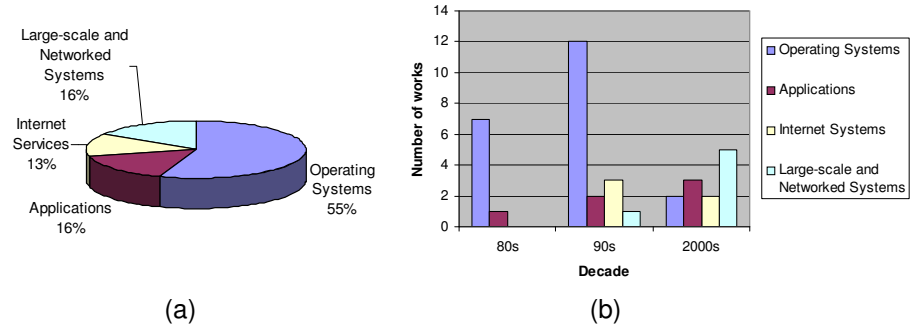


Figure 2.4: FDDA Target Systems: a) break-up, b) distribution over the years

technologies, where the specific constraints and the complexity of the architectures requires new research efforts. For instance, this applies to mobile distributed systems, due to their specificity and resource-constrained characteristics, and to COTS-based software, where the complexity of the interactions along with the absence of any built-in, standard logging scheme, makes it hard to collect the proper data and to generalize the results. Finally, a reduced fraction of works (8%) presents comparisons among various studies, evidencing similarities and differences in the results.

Comparisons are the proof of a certain level of maturity achieved in a particular field. So far, comparisons have been proposed only for studies addressing Operating Systems, which are the most studied target system, as figure 2.4a evidences. Operating Systems account indeed for the 55% of the analyzed systems. The 16% are applications, including virtual machines, and embedded systems, whereas the 13% characterized Internet services, also from a security point of view. Finally, the 16% of works addressed large scale systems and networked systems. As a matter of fact, there are no significant studies so far in the literature addressing MDSs[2]. Figure 2.4b shows how the variety of FFDA targets increased over

[2]Works [27][29][3] are excluded from the comparison, since, although they address MDSs, they are from

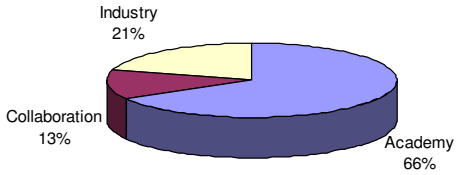


Figure 2.5: Academy and Industry FFDA works break-up

the years: from works mainly addressing Operating Systems in the eighties, to works also concerning applications, Internet servics, large-scale and networked systems in the present decade.

2.5.2 Academic vs Industrial Works

Figure 2.5 reports the break-up of industrial and academic works. Although the academy played an important role for the FFDA research (66% of works), a significant effort has been also profused from industry (21%), as a confirmation that FFDA is a valuable instrument for industries to improve their businesses. Collaborative works from academy and industry are still a little slice of the total number of works (13%). However, more sinergy between the two actors would improve the overall quality of FFDA works, which could benefit from the academy experience and methods from a side, and industry's real data from another side. This is also confirmed by figure 2.6, where it can be noticed how industry works scarcely adopt the manipulation strategies defined in the literature (e.g., filtering, coalescence and so on), differently from what the academia does.On the other hand, collaborative works usually exploit more the existing techniques.

the same author of the present dissertation. Actually, their development also arised from the consideration that there are no FFDA experiences on MDSs in the literature.

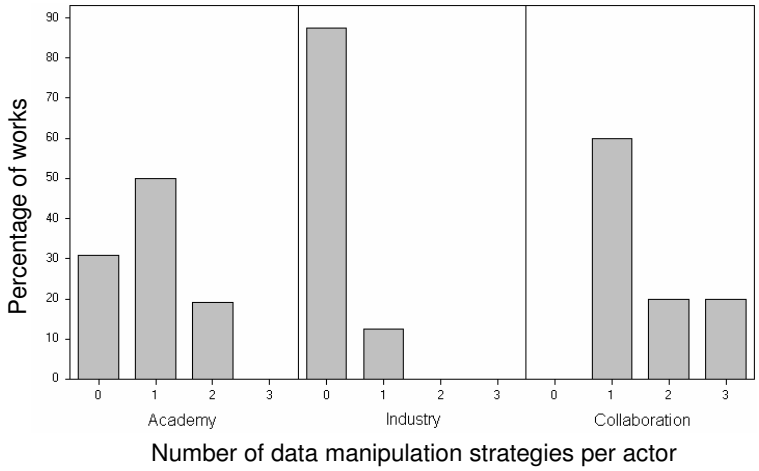


Figure 2.6: Number of adopted FFDA manipulation strategies, per actor; the percentages are normalized for each actor

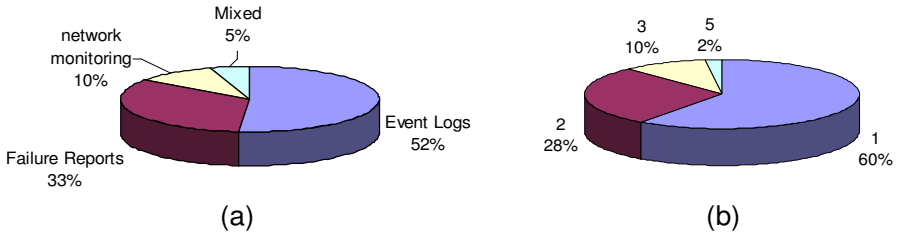


Figure 2.7: FFDA Data sources: a) type break-up, b) number of levels break-up

2.5.3 Methodological Considerations

In this section, a comparison of the related work is conducted according to the FFDA methodology introduced in 2.2. The purpose is to summarize trends and best practices emerged from three decades of FFDA research.

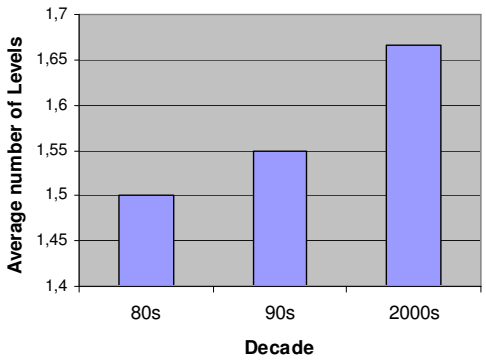


Figure 2.8: Average number of failure data levels considered by FFDA studies, over the years

Data Logging and Collection

The pie chart in figure 2.7a shows a break-up of the data sources adopted by the considered FFDA studies. In particular, the adopted sources are, in a descending order: event logs (52%), failure reports (33%), and network monitoring (10%), i.e., the sniffing of the network traffic. Only a small fraction of the related work (5%) uses data coming from more than one source, hence the common practice is to use a single data source. Another consideration is that the considered failure data often come from internal sources (70% of cases). Only the 30% of studies adopts data provided by third party organizations or companies. This indicates the difficulty of finding publicly available data sources.

As shown in figure 2.7b, the 60% of works uses data captured at a single level, even if a single data source (i.e., one event log) can be sufficient to capture failure data at different machine levels (hardware, operating system, network, middleware, application software, and human interface). The trend of using failure data belonging to more than one level is increasing over the years (see figure 2.8). This is basically due the increased complexity

Table 2.2: Adopted failure data level as a percentage of the total number of analyzed FFDA studies; percentages do not add to 100, since there are studies using data from more than one level

Level	Percentage
Operating System	69%
Application	26%
Hardware	35%
Network	23%
Middleware, human	5%

of systems which in turn requires more data to be captured at different levels to build a thorough understanding of system's faulty behavior. As for the type of level these failure data account for, table 2.2 reports on the frequency of each level, with respect to the total number of analyzed studies. For example, the 69% of studies uses data coming from the Operating System level. This is coherent with the fact that the majority of studies adopts event logs, and targets Operating Systems.

Finally, it is important to note that almost all the considered studies (the 95%) consider failure data captured from systems with idle workloads. Hence, the common FFDA practice suggests to use idle workloads. Not surprisingly, the studies that adopt automated workloads are concerned with the Internet, which spot usage does not always permit to use idle workloads.

Data Filtering and Manipulation

Data manipulation is an important step of FFDA studies. However, as evidenced in table 2.3, a significant fraction of the analyzed studies (38%) do not perform any manipulation on the gathered data. This specially applies to industry works, as previulsly mentioned.

Table 2.3: Percentages of usage of FFDA filtering and manipulation strategies

Manipulation Straregies	Percentage
Not performed	38.46
Filtering	15.38
Temporal coalescence	12.82
Temporal coalescence (merged logs from more nodes)	5.13
Content-based coalescence	10.26
Filtering, Temporal coalescence	7.69
Temporal coalescence, Content-based coalescence	7.69
Filtering, Temporal coalescence, Content-based coalescence	2.56

Data filtering and temporal coalescence are the most common practices, adopted by more of the 30% of works. The table also indicates that there are works performing the temporal coalescence on merged log files from different nodes. This is the case of studies of networked systems, to discover failure propagation phenomena among nodes. Finally, only the 18% of works performs more than one manipulation on the same data, such as filtering and coalescence.

Data Analysis

To have an understanding on the nature of performed studies, table 2.4 reports a summary of the type of conducted analysis per study. The crosses signed on each row evidences the types of analysis, and the percentage of works conducting such analysis. A significant amount of works (33%) performs only the classification of the failures which emerge from the field data. This is especially true for investigation works, i.e., the first works addressing a particular class of system or pursuing a particular objective. Almost one half of the studies produces more than one type of analysis on the data. Among them, about the 10%

Table 2.4: Percentages of types of conducted data analysis

Classification	Modeling	Correlation	Dependability Measurement	Other	Percentage
X					33.33
	X				2.56
		X			5.13
			X		5.13
				X	7.69
X	X				7.69
X		X			7.69
X			X		7.69
X	X	X			10.26
X		X	X		2.56
X	X	X	X		5.13
	X		X		2.56
	X	X			2.56

performs the classification of failures, their modeling, and correlation analysis. These three types of analysis are inherently correlated: the classification is often needed to produce models (note that the majority of modeling works also perform classifications), and the models are then used for correlation analysis, such as trend analysis.

2.5.4 Quality of Conducted Campaigns

The quantitative dimension defined in section 2.3.2, along with the indication of the confidence on collected data, give us a mean to evaluate the quality of conducted FFDA campaigns.

From the considered studies, it results that the average density is 1665 items/month, against an average length of the experiments of 25 months. This means that the experiments are conducted on about 40000 data items, on average. Although these figures seem to be

Table 2.5: Percentiles of the Density of FFDA studies

Percentile	Value
100%	33417.5
99%	33417.5
95%	3304.72
90%	2501
75%	329.02
50%	106.665
25%	27.07
10%	11.75
5%	4.01
1%	2.75
0%	2.75

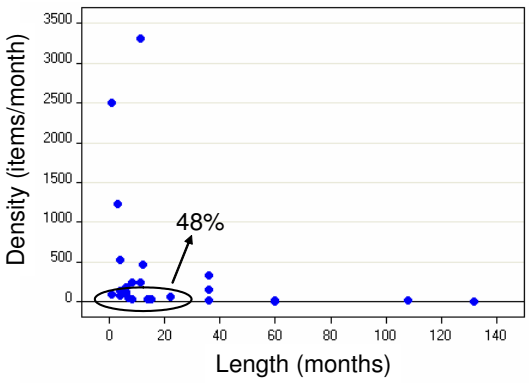


Figure 2.9: FFDA studies density-length relationship

reasonable, a look at the density's percentiles (see table 2.5) evidences that the 50% of works exhibit densities which are lower than 106 items/month. In other terms, several works are concerned with systems which experience a little amount of failures in time. For these systems, longer experiments should be performed. However, this is not always the case, as shown in figure 2.9. Even if the trend is to perform short experiments for high densities and long experiments for low densities, there is still a significant amount of works (48%) exhibiting low densities and short lengths, as indicated in the figure.

For the 31% of works it has not been possible to evaluate the density at all, either because the lenght of the experiment, or the number of data items have not been made available by authors. As for the confidence measures, only the 26% of the considered studies explicitly provides its indication, whereas the 13% furnishes a partial indication, i.e., only on a fraction of the considered data. However, this does not represent an issue for classification studies.

Overall, the quality of conducted experiments can be rated as acceptable. However, the considerations given in this section can be taken into account when conducting future studies, hence improving the quality of future FFDA research.

2.6 FFDA of MDSs: Issues and Challenges

The seek for answers on the dependabilty of MDSs, and the lack of field data on them, as evidenced in section 2.5.1, call for the conduction of FFDA studies on these systems. Recently, some first effort has been conducted towards this direction. The work in [74], proposes a FFDA for a wireless telecommunication system, along with the analysis of failure and recovery rates is discussed. However, the failure data are relative to the fixed core entities (base stations) of a cellular telephone system. [39] reports on a collection of user-perceived failure data from Bluetooth piconets, in order to give a qualitative failures characterization of Bluetooth-enabled devices. The work defines a set of different *test-cases* which have been applied to a variety of Bleutooth devices. Nevertheless, as also authors stated, the results are not purely scientific in that they have no statistical significance. More effort is thus needed. Anyway, new issues and challenges arise when applying the FFDA

methodology to MDSs.

As a MDS is composed of a set of mobile devices linked together via wireless communication means, partial FFDA studies should at least address either the wireless technology or the mobile devices. The following challenging questions are related to the former:

- *Which data source can be adopted?* Failure reports are to be discarder, since there are no publicly available field data on wireless infrastructure, to our knowledge. A possible solution can be event logs, but are we sure that they are able to capture the overall failing behavior of a communication technology? Perhaps, some failure causes lye at the channel level, which cannot be captured by the Operating System's logging facilities. These data can however be gathered via network monitoring. Hence, a mixed approach is needed, able to gather and correlate data from multiple sources. Nevertheless, little experience has been matured on FFDA with mixed sources (the 5% of studied works).

- *Can we use idle workloads?* Wireless channels are generally used in a spot way: the mobility of users and the limited capacity of radio links harden the task of measuring dependability parameters. Dependability continuous-time measures, such as MTBF and MTTR, require the system to be active 24/7 to be properly estimated. Hence, automated workloads should be deployed on actual nodes, to force a 24/7 utilization of the infrastructure. Once again, there poor experience has been achieved about FFDA with automated workloads (the 5% of related work), thus more effort is required in this direction.

The situation gets even worse in the case of mobile devices. In particular, the following open issues arise:

- *How to collect the failure data?* Due to their specificity constraints, mobile devices generally do not provide any built-in logging facility. Even if some instrument is defined, they are used by manufacturers during the development process. As an example, the Symbian OS for smartphones offers a particular server (the flogger) allowing an application to log its information. Yet, to access the logged data of a generic X system/application module it is necessary to create a particular directory, with a well defined name (e.g. Xdir). The problem is that the names of such directories are not made publicly available. It is indeed undesirable for manufacturers to publish their sensible failure data. Hence, ad-hoc logging mechanisms have to be designed, dealing with mobile devices constraints and specificities.

- *Where do we have to start from?* Failure modes of a typical wireless communication mean have already been hypothesized. Examples are connectivity failures, packet losses, and network partitions. Unfortunately, this is not the case for mobile devices, where there is no substantial knowledge on the failure nature. This knowledge cannot be underestimated because, if a logging mechanism is to be built, one should at least know when and how to register a failure event.

The following two chapters try to answer the above mentioned questions, with reference to Bluetooth PANs and Symbian OS smart phones. These chapters are the result of a three years experience, and partially extend previously published results, as [27][29][3].

You always admire what you really don't understand.

Blaise Pascal

Chapter 3

FFDA of Bluetooth Personal Area Networks

The Bluetooth wireless technology is nowadays widespread adopted in a variety of portable devices. In particular, the Bluetooth Personal Area Network (PAN) feature offers IP support over Bluetooth, thus expanding the model of personal communications to provide connectivity and Internet access to and between heterogeneous devices. However, the dependability level of this widely used technology is still unknown.
This chapter presents a failure data analysis campaign on Bluetooth Personal Area Networks (PANs) conducted on two kinds of heterogeneous testbeds (working for more than one year). The obtained results reveal how failures distributions are characterized and suggest how to improve the dependability of Bluetooth PANs.

3.1 Rationale and Characterization of the FDDA Campaign

As demonstrated in recent works, the utilization of Bluetooth as a "last meter" access network, represents an opportunistic and cost-effective way to improve the connection availability of the wide-spread existing 802.11 networks [44, 54]. A significant research quest is to assess whether the Bluetooth technology represents a good candidate for developing mission- or business-critical wireless-based systems. Another interesting quest is to evaluate how Bluetooth meets everyday dependability needs for the consumer-electronic mass-market. To this aim, it is necessary to quantitatively evaluate dependability figures

and to investigate possible fault tolerance means for improving the overall dependability level.

However, the conduction of a FFDA campaign on a wireless communication infrastructure cannot be performed notwithstanding the specificity of the technology and its different scope, as compared to operating systems and applications. Hence, as just mentioned in previous chapter, two challenging questions arise when targeting wireless communication technologies:

- *Which data source can be adopted?*
- *Can we use idle workloads?*

As regarded to first question, it has been previously argued that there are no publicly available failure reports for this kind of systems, while event logs alone cannot assure that all the relevant information is gathered. Most of the failure causes lyes indeed in the volatile nature of the wireless channel, those behavior can hardly be captured by operating system's event logs only. For this reason, a mixed approach is preferred, capable to collect failure-related information from more than one source. Specifically, the analysis on Bluetooth PANs presented in this chapter, is based on failure data collected at three levels: application level (via application logs), operating system level (via OS event logs), and channel level (by means of a Bluetooth channel sniffer). The relationships between data at the different levels allows for a deep understanding of the failure phenomenology, failure causes, and vertical failure propagation, from the channel, up to applications. More in detail, the collected multi-level data permits to obtain the following results:

- The definition of a detailed failure model of Bluetooth PANs, in terms of i) the classification of the failures affecting the applications, ii) the identification of their statistical distributions, which are lognormal for the majority of failure classes, iii) the investigation of possible low level causes of the failures, and iv) the study of failures distribution as Bluetooth channel utilization changes, in order to identify usage patterns that have to be avoided to develop more robust applications.

- The characterization of the self-robustness of Bluetooth wireless channels with respect to faults affecting the radio channels. Low level communication faults are classified, and measures of the coverage of the Bluetooth low level protocols, in terms of the percentage of errors which are detected and corrected, are performed. In particular, a four nine coverage is discovered, demonstrating a good level of self-robustness.

- The improvement of the dependability level of Bluetooth PANs. To this extent, the failure model is used, that is, for each observed failure, we infer its sources. Then, we try to identify the most effective recovery action and, in some cases, we are able to apply error masking strategies by completely eliminating some failures from occurring.

As for the second question, it is desirable to adopt 24/7 automated workloads for several reasons. First, dependability continuous-time measures, such as the Time Between Failures (MTBF) and the Time To Recover (MTTR), require the system to be active 24/7 to be properly estimated. Second, due to the intrinsic characteristics of mobile infrastructures, mobile terminals are generally used in a spot way, differently from server farms or networked systems for which an idle workload is sufficient, as observed in section 2.5. Third, a

continuously operating workload well represents all those critical scenarios where the wireless infrastructure is required to operate continuously, such as remote control of robots, or video-surveillance.

The design of such a 24/7 automated workload plays a crucial role in our analysis, since, as evidenced in section 2.4.1, there is a strong correlation between workloads and failure behavior. To this aim, two classes of workloads have been realized. The first class stimulates Bluetooth channels via completely random workloads in order to study the Bluetooth Channel behavior irrespective of the specific networked application being used. This let us identify good/bad usage patterns that should be adopted/avoided to realize more robust applications. The second class uses more realistic workloads of traditional IP-based application (i.e., Web, streaming, and Peer-to-peer), adopting the traffic models as published in recent works on this research field. This workload allows to characterize the dependability of traditional networked applications using Bluetooth PAN as a last-meter access network.

Table 3.1 summarizes the characteristics of the FFDA campaign, according to the framework introduced in section 2.3. As can be observed from the table, the study is significant from several perspectives: it achieves a variety of interesting results, it benefits from an high density, and it performs a significant amount of analysis.

The rest of the chapter deeply describes the data collection methodology and the results obtained from the FFDA campaign. Such results complete our previous effort in this direction [27][29].

Table 3.1: Characterization of the study conducted on Bluetooth PANs

Analysis of Bluetooth PANs	
Dimension	**Value**
Date	2006
Purpose	Dependability Study
Actor	Academy
Source	Internal
Target System	Bluetooth PANs
Achieved Results	Definition of a failure model, Failure propagation and causes, Bluetooth self-robustness, definition of masking and recovery strategies.
Length	18 months
Data Items	377405
Density	20967 items/month
Data Source	Application logs, System logs, network monitoring
Levels	Channel, OS, Application
Workload	Automated (random and realistic)
Manipulation	Filtering, spatial coalescence, temporal coalescence (from merged log files)
Analysis	Classification, Dependability measurements, Modeling, Correlation
Conficence	Yes

3.2 Bluetooth Background

Bluetooth [12], BT in the following, is a short-range wireless technology operating in the 2.4 GHz ISM band. The BT system provides both point-to-point and point-to-multipoint wireless connections. Two or more units sharing the same channel form a *piconet*. One BT unit acts as the master of the piconet, whereas the other unit(s) acts as slave(s). Up to seven slaves can be active in the piconet.

Applications running on BT-enabled devices use a common set of data-link protocols, the BT core protocols, which are described in the following.

- *Baseband*: this layer enables the physical RF link between BT units forming a piconet. It provides two different physical links, Synchronous Connection-Oriented (SCO) and

Asynchronous Connectionless (ACL). The channel is divided into time slots, each 625 μs in length. A BT ACL data packet may occupy 1, 3, or 5 consecutive time slots. Packets consist of a 72-bit access code for piconet identification and synchronization, a 18 bit header, and a variable length payload. The header contains a header error check (HEC) to check the header integrity. If the HEC does not check, the entire packet is disregarded. The payload consists of three segments: a payload header, a payload and a Cyclic Redundancy Code (CRC) for error detection. The 16 bits CRC-CCITT polynomial $g(x) = x^{16} + x^{12} + x^5 + 1$ is adopted, irrespective of the payload size (i.e., from 1 up to 5 slots), which is able to catch all single and double errors, all errors with an odd number of bits, and all burst errors of length 16 bits or less. It however may fail to detect burst errors which are longer than 16 bits in length, such as 17 bits bursts (with 99.997% coverage) and 18 bits or longer bursts (with 99.998% coverage). In DMx packets (where x is the number of consecutive slots, i.e. 1, 3, and 5), the payload is also coded with shortened Hamming code, in order to perform Forward Error Correction (FEC). DHx packets are instead uncoded. Baseband performs error correction via an ARQ (Automatic Repeat Request) retransmission method. According to this scheme, packets with invalid CRC are retransmitted until an acknowledgment is received or a certain timeout expires.

- *Link Manager Protocol (LMP)*: the LMP is responsible for connection establishment between BT devices, including security aspects, such as authentication and encryption. It also provides BT devices with the inquiry/scan procedure.

- *Logical Link Control and Adaptation Protocol (L2CAP)*: this layer provides connection-oriented and connectionless data services, including multiplexing capabilities, segmentation/reassembly operations, and group abstractions. Error correction and flow control are not performed at this layer since the Baseband channel is assumed, by Bluetooth designers, to be reliable.

- *Service Discovery Protocol (SDP)*: using SDP, device information, services, and characteristics of services can be retrieved.

The BT specification also defines a Host Controller Interface (HCI), which provides developers with an API to access the hardware and to control registers of baseband controller and link manager.

The communication between a BT Host and a Host Controller takes place via either UART or RS232 protocols over serial channels, such as the Universal Serial Bus (USB). Recently, the BlueCore Serial Protocol (BCSP) has been adopted on some devices, such as Personal Digital Assistants (PDAs). It provides a more sophisticated option than its predecessors. BCSP carries a set of parallel information flows between the host and the controller, multiplexing them over a single UART link, and it adds error checking and retransmission.

BT piconets are highly dynamic, with devices appearing and disappearing. For this reason, the BT core specification provides automatic discovery and configuration. Several stages must be completed before BT service can be used: i) Find the device (*Inquiry*); ii) Connect to the device (*Page*); iii) Discover what services the device supports (Service

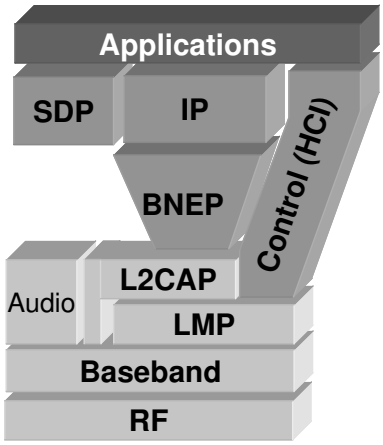


Figure 3.1: The Bluetooth Protocol Stack and the PAN profile

Discovery Protocol- *SDP*); iv) Decide what service to connect to and find out how to connect to it (*SDP*); and v) connect to the service.

The focus of this study is on the use of IP over a BT piconet. The BT Special Interest Group defined the PAN profile, that provides support for common networking protocols such as IPv4 and IPv6. The PAN profile exploits the BT Network Encapsulation Protocol (BNEP) to encapsulate IP packets into L2CAP packets and to provide the Ethernet abstraction. BNEP does not provide any integrity check mechanism for its header. When the PAN profile is used, the master/slave switch role operation assumes a key role. A PAN User (PANU) willing to connect to a Network Access Point (NAP) becomes the master since it initiates the connection. As soon as the connection is established at L2CAP level, a switch is performed, because it is important that the NAP remains the master of the piconet in order to handle up to seven PANUs.

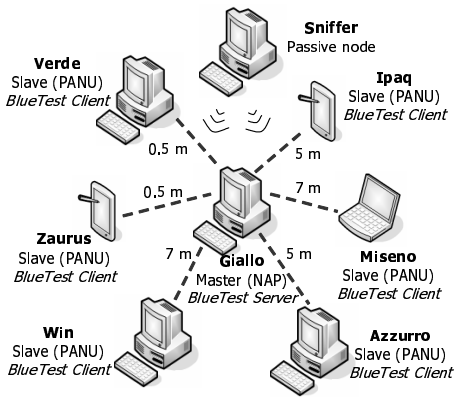
Figure 3.1 gives an overall picture of the described protocols.

3.3 Data Collection Methodology

This section provides the needed insight into the data collection methodology adopted for the Bluetooth FFDA campaign, according to the typical steps of the FFDA methodology, defined in 2.2.

3.3.1 Testbed and Workload Description.

Two testbeds are deployed, composed of actual machines equipped with BT antennas. In order to directly face the intrinsic nature of mobile infrastructure, emulation workloads run on every node of the testbeds. As already introduced, two types of workloads are considered: a totally random workload, so as to stimulate Bluetooth protocols and channels in a uniform way and to identify good/bad usage patterns, and a realistic workload, in order to characterize the dependability behavior of common networked applications when used over BT radio links. Testbeds' topology is shown in figure 3.2. According to the PAN profile, the master node (Giallo) is configured as a NAP to accept incoming connections from slaves, running PANU applications. The workload, called `BlueTest`, runs on all the nodes, in particular `BlueTest` clients on PANUs, and a `BlueTest` server on the NAP. Both the actual testbeds obey to the same hardware and software configurations, i.e. they are composed of 7 devices, 1 master (the NAP) and 6 slaves (the PANUs). To let results be independent on specific hardware platforms or operating systems, the testbed is composed of heterogeneous nodes, ranging from several commodity PCs, with different hardware configurations and OSs, to PDAs. Table 3.2 summarizes technical characteristics of the adopted machines.



Host	O.S.	Distribution	Kernel Version
Giallo	Linux	Mandrake	2.4.21-0.13mdk
Verde	Linux	Mandrake	2.4.21-0.13mdk
Miseno	Linux	Debian	2.6.5-1-386
Azzurro	Linux	Fedora	2.6.9-1-667
Win	MS Windows XP	Service Pack 2	
Ipaq H3870	Linux	Familiar 0.8.1	2.4.19-rmk6-pxa1-hh37
Zaurus SL-5600	Linux	Open Zaurus 3.5.2	2.4.18-rmk7-pxa3-embedix
Sniffer	MS Windows XP	Service Pack 2	

Host	CPU/RAM	BT Stack	BT Hardware
Giallo	P4 1.60GHz/128Mb	BlueZ 2.10	Anycom CC3030
Verde	P3 350Mhz/256Mb	BlueZ 2.10	3COM 3CREB96B
Miseno	Celeron 700Mhz/128Mb	BlueZ 2.10	Belkin F8T003
Azzurro	P3 350MHz/256Mb	BlueZ 2.10	Digicom Palladio
Win	P4 1.80Ghz/512Mb	Broadcomm	Sitecom CN-500
Ipaq H3870	StrongARM 206 MHz/64Mb	BlueZ 2.10	*on board*
Zaurus SL-5600	XScale 400Mhz/32Mb	BlueZ 2.10	*on board*
Sniffer	P4 1.80 Ghz/512Mb	SP2 BT Stack	*2 proprietary BT probes*

Figure 3.2: The topology of both the Bluetooth testbeds, along with the technical details of their machines

Linux machines use the standard BlueZ stack[1], whereas the Windows ones are equipped with Service Pack 2 and use the BroadCom stack[2]. Note that the native Windows BT Stack could not be used because it does not offer any API for the PAN profile (in Windows XP, IP facilities over Bluetooth are provided only via point-to-point RFCOMM[3] connection). All the machines are equipped with Class 2 BT devices, i.e., up to 10 meters communication range. Other than active nodes running the `BlueTest`, also a passive *Sniffer* node is present. The sniffer is equipped with two BT dongles, and with a commercial software for sniffing BT packets over the air. The Sniffer is equipped with Windows XP and it uses the Windows' SP2 Bluetooth stack. In order to reduce hardware aging phenomena, the two testbed have been totally replaced by new ones (having the same configuration), in the middle of the testing period. The PANUs' BT antennas have been placed at several different distances from the NAP's antenna (e.g. 0.5m, 5m, and 7m, see figure 3.2), in order to evaluate the

[1]The Official Linux Bluetooth protocol stack, http://www.bluez.org
[2]Broadcom is a commercial implementation of the BT Stack for Windows, `http://www.broadcom.com`
[3]Serial Cable Emulation Protocol

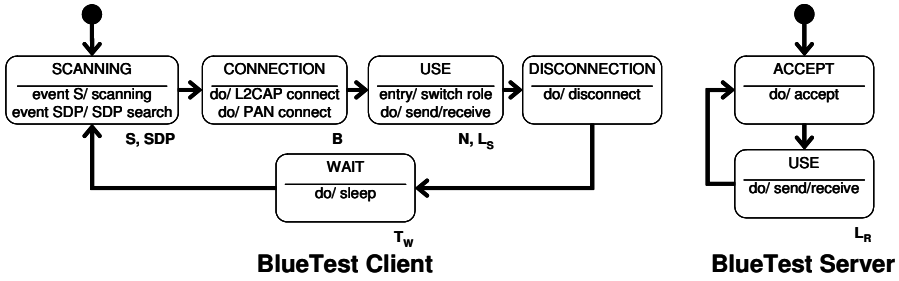


Figure 3.3: State-chart Diagram of the Bluetooth Workload

failure distribution as a function of the distance. Antenna positions are fixed, hence we collect data about fixed PAN topologies. However, this is representative of real cases in which PANs are built among computers on a desk.

The `BlueTest` workload (WL) emulates the operations that can be made by a real BT user utilizing PAN applications. Each `BlueTest` cycle consists of common BT utilization phases, as summarized in the state-chart diagram of figure 3.3. It first executes an inquiry/scan procedure to discover devices in the environment, then it searches the Network Access Point (NAP) service via a SDP Search operation. Once the NAP has been found, the `Bluetest` client connects to it (by creating a BNEP channel on top of a L2CAP connection), switches the role to slave (to let the NAP to be the master of the piconet), uses the wireless link by transfering data to its counterpart (the `Bluetest` server), and finally it disconnects. Before starting a new cycle, the `BlueTest` client waits for a random time, which can be thought as a user passive off time, modeled according to a Pareto distribution, coherently to previous work [34]. To add uncertainty to the piconet evolution, each WL cycle is characterized by several random variables: i) S, scan flag, if true, an inquiry/scan operation is performed; ii) SDP, service discovery flag, if true, a SDP search is performed;

iii) B, the Baseband packet type; iv) N, the number of packets to be sent/received; v) L_S/L_R, the size of sent/received packets; and vi) T_W, the passive off waiting time. S and SDP are introduced since real BT applications do not perform inquiry/scan procedures or SDP searches every time they run. It is possible to exploit caching of the recently discovered devices or services. For each WL cycle, values for S and SDP are chosen according to the uniform distribution, since the lack of publicly available information about the real utilization pattern of inquiry/scan procedures and SDP searches for a typical PANU application. B, N, L_S, and L_R parameters depend on the channel utilization, as described in the following.

Random WL. It generates totally random values for B, N, L_S, and L_R. In particular, B is randomly chosen among the six BT packet types (i.e. DMx or DHx), according to a binomial distribution. This helps to 'stimulate' the channel with every packet types. N, L_S, and L_R are generated following uniform distributions, for the same reasons. To let us analyze the integrity state of packets as they are received by PANUs and the NAP, the random WL uses the User Datagram Protocol (UDP). More details on this WL can be found in our previous work [27].

Realistic WL. It generates values for the parameters according to the random processes which are used to model actual Internet traffic [34, 41]. In particular, the choice for B is left to the BT Stack, whereas N follows power law distributions (e.g. the Pareto distribution) related to the dimension of the resource that have to be transferred. The parameters of the distributions are set with respect to the application being emulated (e.g. Web browsing, file transfer , e-mail, peer to peer, video and audio streaming). Values for L_S and L_R are

set according to the actual Protocol Data Unit, commonly adopted for the various transport protocols over the Internet [41] (e.g. 572 or 1500 bytes for the Transmission Control Protocol - TCP, 825 bytes for the Real-time Transport Protocol - RTP, 40 bytes for acknowledgment packets [41]). Hence the WL adopts TCP, except from streaming traffic, where UDP is used. Finally, since a user can run more applications in sequence over the same connection, the WL runs from 1 up to 20 consecutive cycles over the same connection. Finally, the behavior of the WL may slightly change from an application to another. For example, streaming applications only receive data (that is, $L_S = 0$), whereas the other applications, which use TCP, require a continuous data down stream and acknowledgments upstream and vice versa. Further details on the Realistic WL can be found in our technical report [19].

3.3.2 Failure Data Logging and Collection

Failures might manifest during the normal WL execution. When a failure occurs, the workload is instrumented to register a failure report. Three levels of failure data are produced, as in the following.

User Level Failures: failure reports about the failure as it manifests to a real user, using a PANU device. The report also contains details about the BT node status during the failure (e.g. the WL type, the packet type, the number of sent/received packets);

System Level Failures: failure data registered by system software on the OS system log file, including BT APIs and OS drivers.

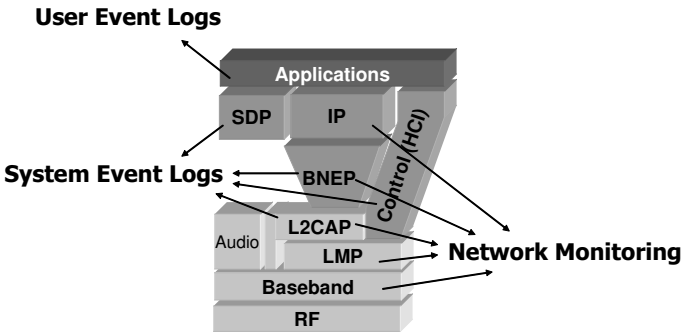


Figure 3.4: Bluetooth multi-level failure data collection

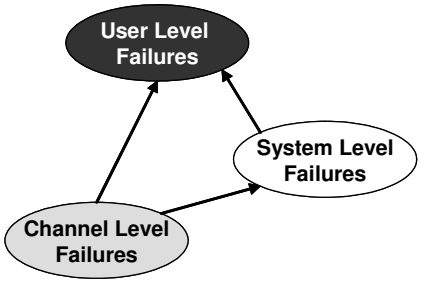


Figure 3.5: Reletionship between User Level, System Level, and Channel Level Failures

Channel Level Failures: failure data gathered by a Bluetooth air sniffer directly over the wireless media.

It is worth to note how this multi-level approach permits to gather failure data at all the Bluetooth layers, as evidenced in figure 3.4. As highlighted in figure 3.5, System Level Failures can be seen as errors for User Level Failures. In other terms, when a User Level Failure manifests, one or more System Level Failures are registered in the same period of time. This helps to understand causes behind the high level manifestation. The same consideration applies for Channel Level Data and both User Level Failures and System Level Failures. In particular, User Level and System Level Failures may also be due to problems at the channel level, such as electromagnetic interference and multipath fading

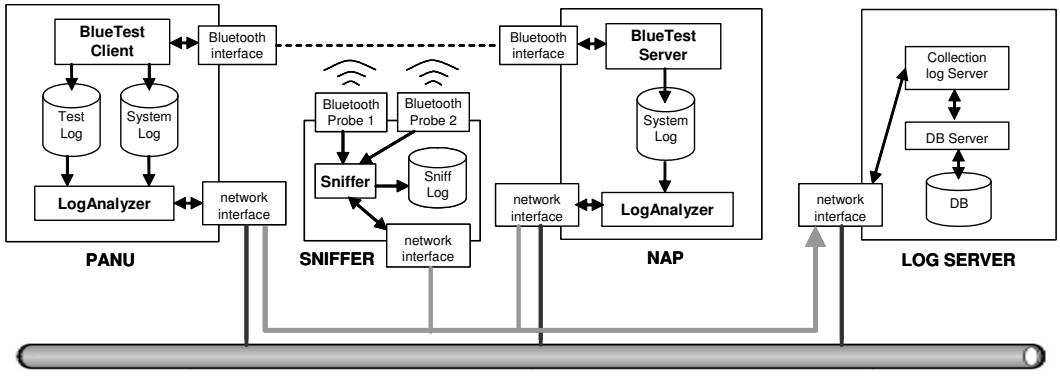


Figure 3.6: Bluetooth Failure Data Collection Architecture

which may cause some corrupted packets to elude the Baseband's error control schemes and to propagate to upper layers. The effects of propagation are discussed in section 3.4.3, whereas the level of self-robustness of the Baseband level with respect to Channel Level Failures is discussed in section 3.4.2.

The collection architecture we adopted is summarized in figure 3.6. User and System Level Failure data on PANUs and the NAP is collected by a `LogAnalyzer` daemon, which sends it to a central repository, the `Log Server`, where data are then analyzed by means of a statistical analysis software. We used the SAS analyzer suite[4].

On each PANU, both User Level and System Level failure data is stored in two files: the `Test Log` file, which contains User Level failures reports, and the `System Log` file, containing all the error information registered by the applications and system daemons running on the BT host machine (on the NAP, only System Level failures are gathered). The `LogAnalyzer` periodically extracts failure data from both the logs, and sends them to the `Log Server`.

[4]SAS is an integrated software platform for business intelligence which includes several tools for statistical analysis; it is produced by SAS Institute Inc., `http://www.sas.com`

As for Channel Level Failures data, we use a commercial Bluetooth air sniffer tool in charge of capturing BT traffic and of organizing data in different structured decoding formats, from frame level to bit level. The sniffer is able to collect all the information we need, such as, failure reports at both the Baseband layer, and the L2CAP, storing data into a `Sniff Log` file. To let sniffing operation not interfere with node activities, the sniffer has been installed on a machine that does not join the piconet (the *Sniffer* node in figures 3.6 and 3.2). Due to their passive nature, all current air sniffers cannot request the retransmission of a missed packet. For this reason, data has been sniffed by installing two redundant Bluetooth antennas (also called probes, and provided with sniffer kit). Probes placement plays a key role to obtain good data. Let us indicate with r the ratio between the number of frames retransmitted by the nodes and the frames the sniffer marks as erroneous. The probes have been placed so as to achieve $r \cong 1$. This simple heuristic metric assures that the sniffed data is not influenced by probes location.

Both testbeds have run for more than one year, from June 2004 to November 2005, with 356551 failure data items being collected. In particular, there were 20854 User Level failure reports from `Test Log`s and 335697 System level failure entries from `System Log`s. The most of the failures (84%) comes from the random workload. Data from the sniffer have been captured during a period of 2 months over a testbed running the random WL, with over 26.5 millions of failure entries from the `Sniff Log`.

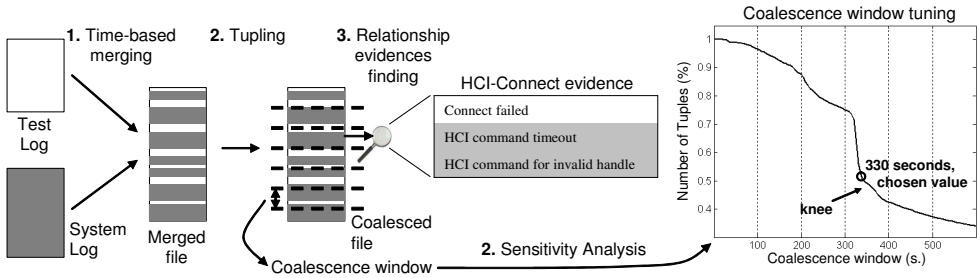


Figure 3.7: The "merge and coalesce" scheme adopted to pinpoint error-failure relationships

3.3.3 Data Filtering and Manipulation

The `System Log` contains data from all applications and daemons running on the host machine; it is thus needed to filter the log in order to capture only BT related information. This helps to reduce the amount of data which have to be stored and simplifies the analysis process.

We also need to manipulate the data in order to i) gain more insights into error-failure relationship on each machine, by relating User Level with System Level and Channel Level failures (via temporal coalescence), and ii) evaluate if and how often the failures manifest together on different machines, by analyzing the User Level Failures correlation among machines (via spatial coalescence).

The huge amount of data does not allow us to perform these manipulation operations manually. Hence, we define a "merge and coalesce" semi-automatic scheme. Let us envision the scheme with reference to the relationship between System Level and User Level failures. Scheme's steps are summarized in figure 3.7. First, for each node a log file is produced by merging its `Test Log` and `System Log` files, on a time-based criteria (entries are ordered according to their timestamps). Second, the merged file is analyzed using the tupling

coalescence scheme [15], i.e., if two or more events are clustered in time, they are grouped into a tuple, according to a coalescence window. The window size has been determined by conducting a sensitivity analysis, as shown in the plot in figure 3.7. The number of obtained tuples (reported as percentage of entries on the vertical axis) is plotted as a function of the window size. A critical "knee" is highlighted in the plot. Choosing a point on the curve before the knee causes the number of tuples to drastically increase, thus generating truncations, i.e., events related to the same error are grouped into more than one tuple. On the other hand, choosing a point after the knee generates collapses because events related to different errors are grouped into the same tuple, due to a decreasing number of tuples. For these reasons, a window size equals to 330 seconds, that is, exactly at the beginning of the knee, is chosen. Third, the error-failure relationship is inferred by analyzing tuples' contents. For instance, if a tuple contains both a "Connect Failed" user level message, and HCI system level messages, an evidence of a HCI-connect relationship is found. Counting all the HCI-connect evidences gives a mean to weight the relationship.

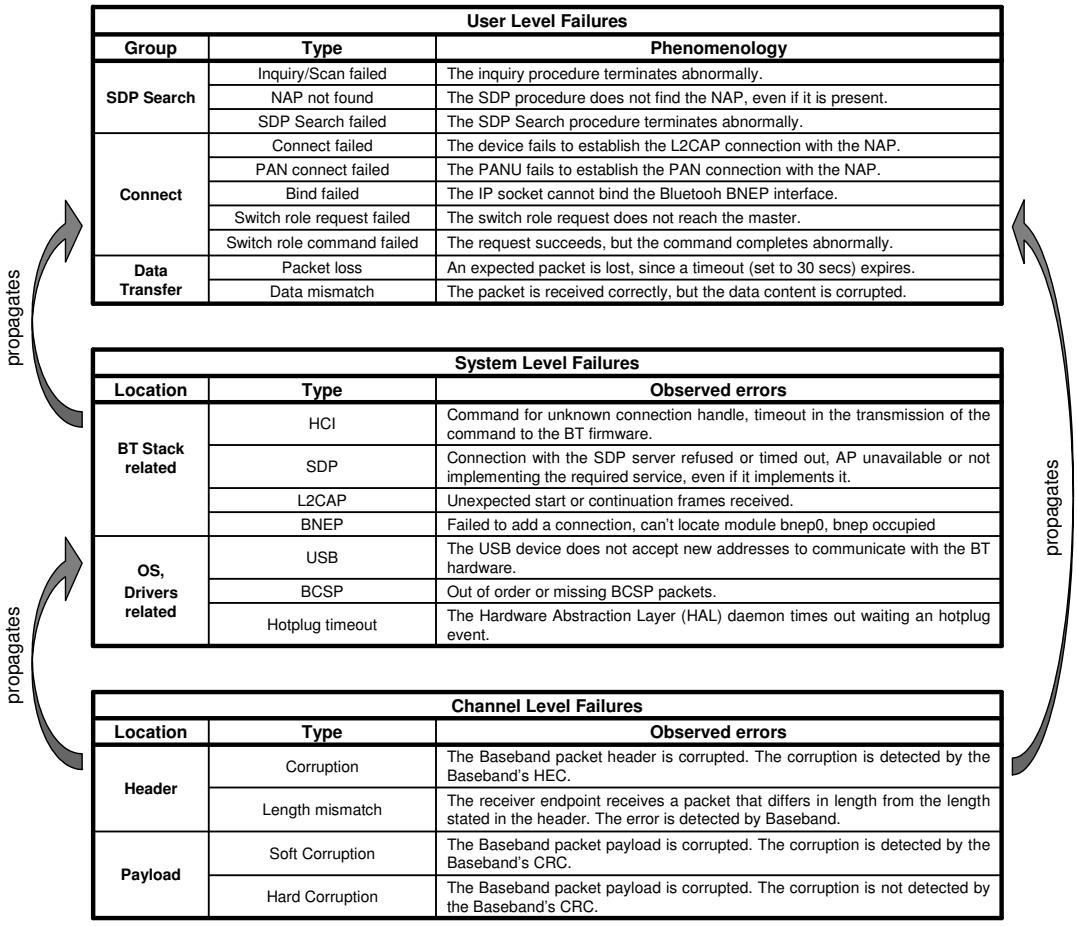
This scheme can easily be adopted in the other cases of our interest. For example, to infer the Channel Level - User Level relationship, the scheme is adopted with reference to the `Test Log` and the `Sniff Log`. In the case of the correlation among User Level failures from different machines, the scheme is adopted by using the `Test Log`s from all the machines.

3.4 Key Findings

3.4.1 Bluetooth PAN Failure Model.

Failures are classified by analyzing their spontaneous manifestation, as recorded in `Test`, `System`, and `Sniff Log`s. The failure model described here is general, in the sense that it considers all failure reports and events gathered from both testbeds and all the machines. Failure manifestations are workload independent, i.e., the same failure types have been observed, regardless of the workload being run. Differences are in the failure rates, as will be detailed in section 3.4.4.

Figure 3.8 gives an overall picture of the Bluetooth PAN failure model. This considers three levels of failures: user, system, and channel level, according to the collection methodology previously described. The reported failure types are the result of an accurate classification of the collected failures' reports. Messages related to the same failure have been classified into one failure type. This gives the model simplicity and understandability. The table at the top in figure 3.8 describes user level failures, the table at the center is dedicated to system level failures, and the table at the bottom reports channel level failures. User level failures can be grouped into three classes, in accordance with the utilization phase where they manifest, i.e., searching for devices and services, connecting, and transferring data. Each group contains one or more failure types. For each failure type, a brief description of its phenomenology is given in the table. Failures during the connection (Connect group) can occur during all the steps that are needed to create a PAN channel, such as the L2CAP and PAN connections setup, the role switch from master to slave, and the binding of the IP



User Level Failures

Group	Type	Phenomenology
SDP Search	Inquiry/Scan failed	The inquiry procedure terminates abnormally.
	NAP not found	The SDP procedure does not find the NAP, even if it is present.
	SDP Search failed	The SDP Search procedure terminates abnormally.
Connect	Connect failed	The device fails to establish the L2CAP connection with the NAP.
	PAN connect failed	The PANU fails to establish the PAN connection with the NAP.
	Bind failed	The IP socket cannot bind the Bluetooh BNEP interface.
	Switch role request failed	The switch role request does not reach the master.
	Switch role command failed	The request succeeds, but the command completes abnormally.
Data Transfer	Packet loss	An expected packet is lost, since a timeout (set to 30 secs) expires.
	Data mismatch	The packet is received correctly, but the data content is corrupted.

System Level Failures

Location	Type	Observed errors
BT Stack related	HCI	Command for unknown connection handle, timeout in the transmission of the command to the BT firmware.
	SDP	Connection with the SDP server refused or timed out, AP unavailable or not implementing the required service, even if it implements it.
	L2CAP	Unexpected start or continuation frames received.
	BNEP	Failed to add a connection, can't locate module bnep0, bnep occupied
OS, Drivers related	USB	The USB device does not accept new addresses to communicate with the BT hardware.
	BCSP	Out of order or missing BCSP packets.
	Hotplug timeout	The Hardware Abstraction Layer (HAL) daemon times out waiting an hotplug event.

Channel Level Failures

Location	Type	Observed errors
Header	Corruption	The Baseband packet header is corrupted. The corruption is detected by the Baseband's HEC.
	Length mismatch	The receiver endpoint receives a packet that differs in length from the length stated in the header. The error is detected by Baseband.
Payload	Soft Corruption	The Baseband packet payload is corrupted. The corruption is detected by the Baseband's CRC.
	Hard Corruption	The Baseband packet payload is corrupted. The corruption is not detected by the Baseband's CRC.

Figure 3.8: The Bluetooth PAN Failure Model

socket over the BNEP interface. Unexpectedly, failures during data transfer (Data Transfer group) encompass "Packet Loss" and "Data Mismatch" failures, despite Baseband's error control mechanisms, such as CRCs, FEC, and HEC schemes. However, as discussed in [84], the weakness of integrity checks is the assumption of having memoryless channels with uncorrelated errors from bit to bit. In our case, correlated errors (e.g. bursts) can occur due to the nature of the wireless media, affected by multi-path fading and electromagnetic interferences. The failure of the integrity checks is further investigated in next subsections.

System level failures are grouped with respect to their location, i.e., BT software stack and OS/Drivers. Failure types have been defined according to the component which signaled the failure. For each failure type, a brief description is given. For instance, the L2CAP module logs the presence of unexpected received frames, that is a start frame when a continuation frame is expected, and vice-versa. This is probably due to interferences, causing either frames or the synchronization with the piconet's clock, to be lost.

Finally, channel level failures are classified into two classes, header related and payload related, according to the portion of the packet that results to be corrupted. The "Hard Payload Corruption" deserves more attention, since it is the only one that propagates to upper layers. The other failures are indeed successfully detected and masked by the Baseband's FEC, HEC and CRC detection and retransmission schemes, as shown in the next subsection.

3.4.2 Baseband Coverage

Failure data from the air sniffer evidenced a very high failure rate at the Baseband layer. The average failure rate resulted to be 6.822 failures per second. If we consider that 596 frames on average are transmitted every second, this means that about one frame out of 100 frames is delivered with errors. This faulty behavior of the wireless media is well balanced by Baseband's error check and retransmission mechanisms. All "Header Corruptions" (which represent the 88.42 % of channel failures), "Length Mismatches" (1.806%), and "Soft Payload Corruptions" (9.77%) are indeed detected and successfully masked. The remaining 0.004% of the failures, i.e., "Hard Payload Corruptions", are not detected by baseband and

hence they propagate to upper layers (the consequences of this propagation are investigated in next subsection). This leads to an overall coverage of the Baseband layer of 0.99996.

The causes of the propagation can be found in the bursty nature of the corruptions. As known, the CRC-CCITT polynomial used by Baseband's CRC cannot always detect error bursts, and, in fact, it detects 18 bits or longer bursts with 0.99998 coverage probability. We analyzed the corrupted packets, discovering that error bursts have an average size of 512 bits.

The time between propagations, that is, the time between "Hard Payload Corruptions", follows an exponential probability distribution, whit a shape parameter $\lambda = 2.73 \cdot 10^{-4}\, s^{-1}$, and mean equals to 3665 seconds. In other terms, about one propagation per hour occurs, and consecutive propagations are uncorrelated with each other. The fitting has been conducted using the SAS analyzer. We tried to fit the data with exponential, lognormal, and weibull distributions, using the Maximum Likelihood Estimators provided by SAS. The goodness of fit is proved by means of three well-known tests: Kolmogorov-Smirnov, Cameron-Von Mises, and Anderson-Darling. The tests confirmed the exponential fitting.

3.4.3 Propagation Phenomena

This section reports the results of a propagation analysis between User Level, System Level, and Channel Level failures, in order to discover the low level causes behind high level failure manifestations. First, we investigate the User Level - System Level failures relationship, then we complete the analysis with the Channel level causes.

Table 3.2 illustrates the results obtained by applying the merge and coalesce approach to the `Test` and `System Log` files, with a coalescence window equals to 330 seconds. The interpretation of the table is simple: the greater is the percentage reported in a cell, the stronger is the relationship between the user level failure (on the row) and the system level failure (on the column). Percentages on each row sum to 100 (except from the "tot" column), so as to have a clear indication of user level failure causes. The "total" row and the "tot" column report the total percentages, e.g., the 49.9% of the user failures are due to HCI system failures. In order to discover error propagation phenomena from the NAP to PANUs, the user level data have also been related with the NAP's system log file (i.e., the server), with the same merge and coalesce approach. Hence, for each system level failure column, the table reports the figures obtained by relating the Test log with both the local `System Log` and the NAP's `System Log`, for each machine. The table contains very useful information about the error-failure relationship, and NAP-PANU propagation phenomena. For example, failures during the L2CAP connection (row "Connect failed" in the table) are mostly due to timeout problems in the HCI module, either from the local machine or from the NAP. This occurs when a connection request (or accept) is issued on a busy device. PAN connection failures are instead frequently related to failures reported by the SDP daemon (the 96.5% of the cases). Interestingly, we observed that exactly the 96.5% of PAN connect failures manifests when the SDP Search is not performed by the workload (in other terms, when the SDP flag is false). This is a clear indication that *avoiding caching and performing the SDP search before a PAN connection is a good practice to reduce PAN connect failures occurrence*. In the remaining cases in which the SDP search is performed (the 3.5% of the

Table 3.2: Error-Failure Relationship: System Failures are regarded as errors for User Failures

	System Level Failures														
User Level Failures	L2CAP		HCI		SDP		BNEP		HOTPLUG		USB		BCSP		
From	local	NAP	local	NAP	local	NAP	local	NAP	local	NAP	local	NAP	local	NAP	**TOT**
Inquiry/scan failed															0.1
Nap not found			18.8		61								20.2		0.2
SDP search failed			20		50.9							9.1	20		0.1
Connect failed		4.9	85.1	2.3							1	1.5	5.2		5.7
PAN connect failed					96.5		3.5								0.5
Bind failed		3.6	55.5	1.8					35.5			3.6			38.6
Sw role command failed	0.9	4.4	10.9	2.4	8.2		18.8				2.7	2	49.7		19.4
Sw role request failed			91.1										8.9		0.8
Packet loss	2.7	3.9	21.8	2.5	0.9		15.4		17.2		0.9	2.6	32.1		33.9
Data mismatch															0.7
Total	1		49.9		7		8.5		11.4		1.1		21.1		

cases), the PAN connection fails due to errors in the BNEP module (see table 3.2). In particular it fails to add the requested connection on the L2CAP connection. One more interesting relationship is between "Switch role request failed" and command transmission timeouts signaled by the HCI module (the 91.1% of switch role request failures). This suggests that *increasing the timeout in the API helps to reduce the switcth role request failure occurrence*. For some failures, such as "Inquiry/Scan failed" and "Data Mismatch", no relationships have been found. Table 3.2 also permits to define masking strategies, as detailed in section 3.5.1.

The same methodology has been applied to discover the relationship between Channel Level and User/System level failures. In this case, a coalescence window equals to 280 seconds had been chosen. Figure 3.9 summarizes the results[5], emphasizing the propagation phenomena from "Hard Payload Corruptions" to User Level and System level failures. Data are gathered

[5]Figure 3.9 only reports the propagation of "Hard Payload Corruptions" to upper layers, and, for a matter of space, it does not evidence all the System level causes of User Level Failures, such as "Packet Loss" failures due to HCI errors, since they are already summarized in table 3.2.

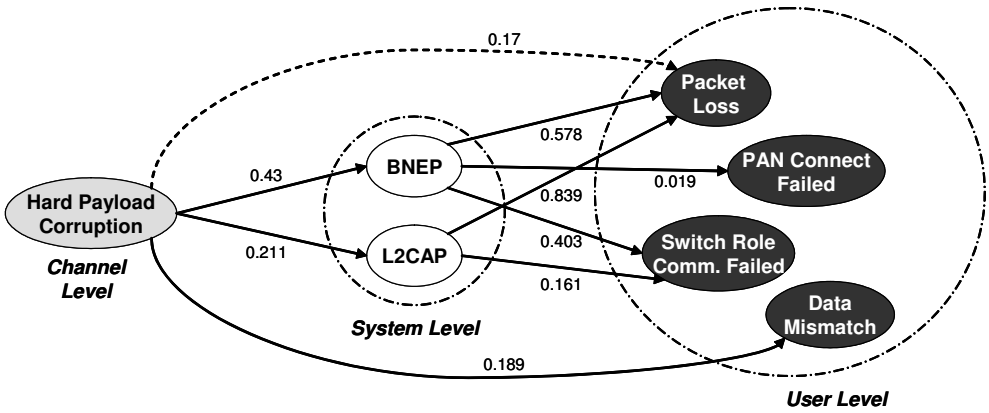


Figure 3.9: Propagation diagram of Hard Payload Corruptions to User and System Level Failures on a single node

from a two months experiment with the random WL on every node. In the figure, the ellipses represent failures (at the various levels) and the arrows represent propagations. The number on each arrow is the average propagation probability; for instance, when a "Hard Payload Corruption" occurs, it generates a "Data Mismatch" failure with a 0.189 probability. The probabilities for the arrows from System to User Level failures are inferred from Table 3.2, whereas the remaining have been obtained by merging and coalescing the `Sniff Log` with the `Test` and `System Log` files, for each node. Note that the diagram only takes into account local propagations, i.e., propagations on the same node.

The diagram shows several interesting points. "Hard Payload Corruptions" directly cause "Data Mismatches" (18.9% of the cases), L2CAP errors (21.1%), BNEP errors (43%), and "Packet Losses" (17%). This is easily justifiable if we look at the structure of the packets that are transmitted among nodes, when the random WL is adopted[6]. As figure 3.10 points out, a burst corruption may affect the L2CAP header, the BNEP header, the IP and UDP

[6]We recall that the sniffer has been used in conjuction with the random WL

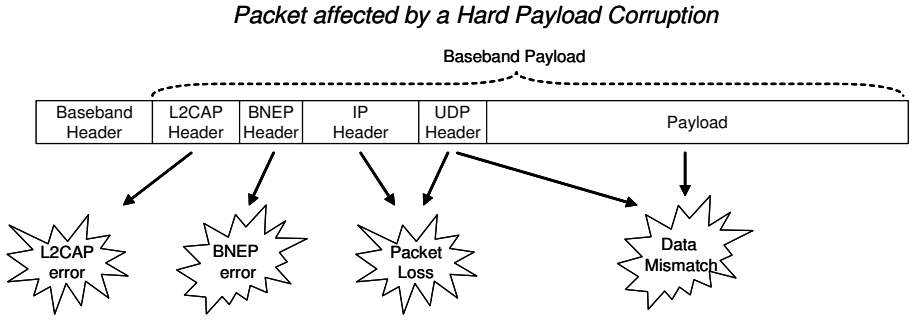


Figure 3.10: Hard Payload Corruptions consequences with respect to the packet's structure; one of the indicated errors/failures manifest, depending on the position of the corruption within the packet

headers, or, finally, the data itself. Hence, one of the above mentioned failures manifests depending on the position of the burst in the packet.

More specifically, if the burst affects L2CAP or BNEP headers, L2CAP or BNEP error manifests. We recall that L2CAP and BNEP are not covered by any error check mechanism, because they assume the Baseband level to be highly reliable. However, we observed that Baseband propagates about one failure per hour, causing L2CAP or BNEP to fail in the 64.1% of the cases. Furthermore, as we will discuss later in section 3.5.2, these failures may also be higly severe, since they can sometimes require a reboot of the machine in order to reestabish proper operation. This cannot be ignored, especially when considering critical wireless systems which have to operate continuosly. The possibility of protecting L2CAP and BNEP headers should thus be considered in the next releases of the Bluetooth specifications.

Once the IP header gets corrupted, the IP module detects the error (the IP header is protected with a CRC) and discards the packet, causing a "Packet Loss" to be manifested at the application. Actually, we had no evidences of these phenomena in the system logs,

since the IP module does not log any information when packets are discarded. However, we found that the 17% of "Hard Payload Corruptions" directly cause "Packet Losses", thus we hypothesize that these losses are due to the corruption of the IP header. This is depicted with a dashed arrow in figure 3.9. Finally, if the corruption affects the UDP header or the payload, a Data Mismatch failure manifests[7]. One could question this last statement by observing that UDP has its own checksum which is computed on the payload also. However, this ckecksum is usually ignored by real UDP implementations.

Surprisingly, Baseband's payload corruptions can also cause the failure of either switch role or PAN Connection operation, other than packet losses or data mismatches. This is because switch role or PAN Connection operations are performed by sending commands over the air between two end-points. If one of this commands is delivered with errors, the whole operation fails. To be clearer, let us envision an exemplary situation. Verde wants to establish a PAN connection with Giallo, hence it sends BNEP commands to Giallo. Unfortunately Giallo's answers are delivered with errors, so the Verde's BNEP module ignores them. When a timeout expires, BNEP registers a "Failed to add a connection" message on the System log, and consecutively the application (the Bluetest) records on the Test log that the PAN connection operation has failed. This explains the "Hard Payload Corruption - BNEP - PAN Connect Failed" chain.

As a last result, we evaluate the spatial correlation of failures among nodes. The analysis has been conducted with the merge and coalesce approach applied to the `Test Log`s of all the slave nodes, i.e., six nodes, for both workloads, with a coalescence window of 120 seconds.

[7]Note that some UDP header corruptions may cause packet losses in the case that the corruption affects the server port

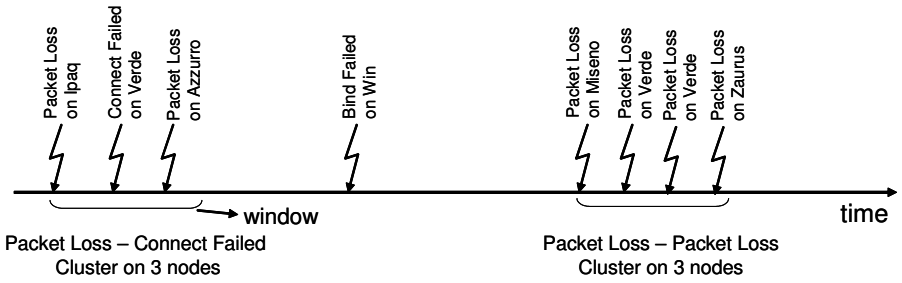


Figure 3.11: Example of clusters of failures.

To simplify the analysis, we only consider failures that cannot be masked. In other terms, we do not consider "Bind Failed", "Switch Role Command Failed", and "NAP not found", for which we defined proper masking strategies that prevent these failures from occurring (see section 3.5.1). As a macroscopic figure, the 10.82% of the total number of failures resulted to manifest as a cluster.

By cluster of failures we mean two or more User Level Failures that manifest *within* the coalescence window onto two or more *distinct* nodes. Figure 3.11 exemplifies the concept of cluster of failures. The figure shows two clusters. The one on the left is a 3-nodes cluster on Ipaq, Azzurro, and Verde. The one on the right is a 3-nodes cluster as well because, even if it contains four instances of failures, two of them come from the same node (i.e., Verde).

A summary of the results is given in table 3.3. Columns represent the clusters of failures we observed, the rows are the number of nodes, and the figures are the percentage of failures with respect to the total number of clustered failures. For example, the 51.01% of the clusters are composed by two "Packet Loss" instances (on two distinct nodes). From the table, it is clear that "Packet Loss" failures are likely to manifest together on more than one node. Moreover, the 38.98% of the total number of "Packet Loss" failures manifests as

Table 3.3: Spatial coalescence of failures among nodes

Clusters of Failures / number of nodes	Packet Loss Packet Loss	Connect Failed Connect Failed	Packet Loss Connect Failed	Packet Loss PAN Connect Failed	PAN Connect Failed Connect Failed	Packet Loss Connect Failed PAN Connect Failed
2	51.01	1.67	10.17	0.94	0.10	
3	21.06		2.09	1.05		0.10
4	7.33		0.42	0.21		0.10
5	1.88		0.10			
6	1.67		0.10			

clusters. This indicates that "Packet Losses" are often due to causes affecting whole parts of the piconet, such as wide-band interference phenomena or temporary unavailabilities of the master node.

3.4.4 Further Results

Figure 3.12 reports the results of the fitting for the time to failure (TTF) of two User Level failures. Failure data comes from two nodes, "Bind Failed" failures on Azzurro, and "Connect Failed" failures on Verde (data are from the random WL). We tried to fit the data with exponential, lognormal, and weibull distribution, using the SAS analyzer. Both the probability plots, and the results from the goodness of fit tests in table 3.4, indicate that these failures follow the lognormal distribution. The probability plots show that real data lies on the line of lognormal percentiles, at least until the 99 and 95 percentile for the "Bind Failed" case and the "Connect Failed" case, respectively. Data points which does not follow the line indicates that real data presents a shorter tail as compared to the lognormal

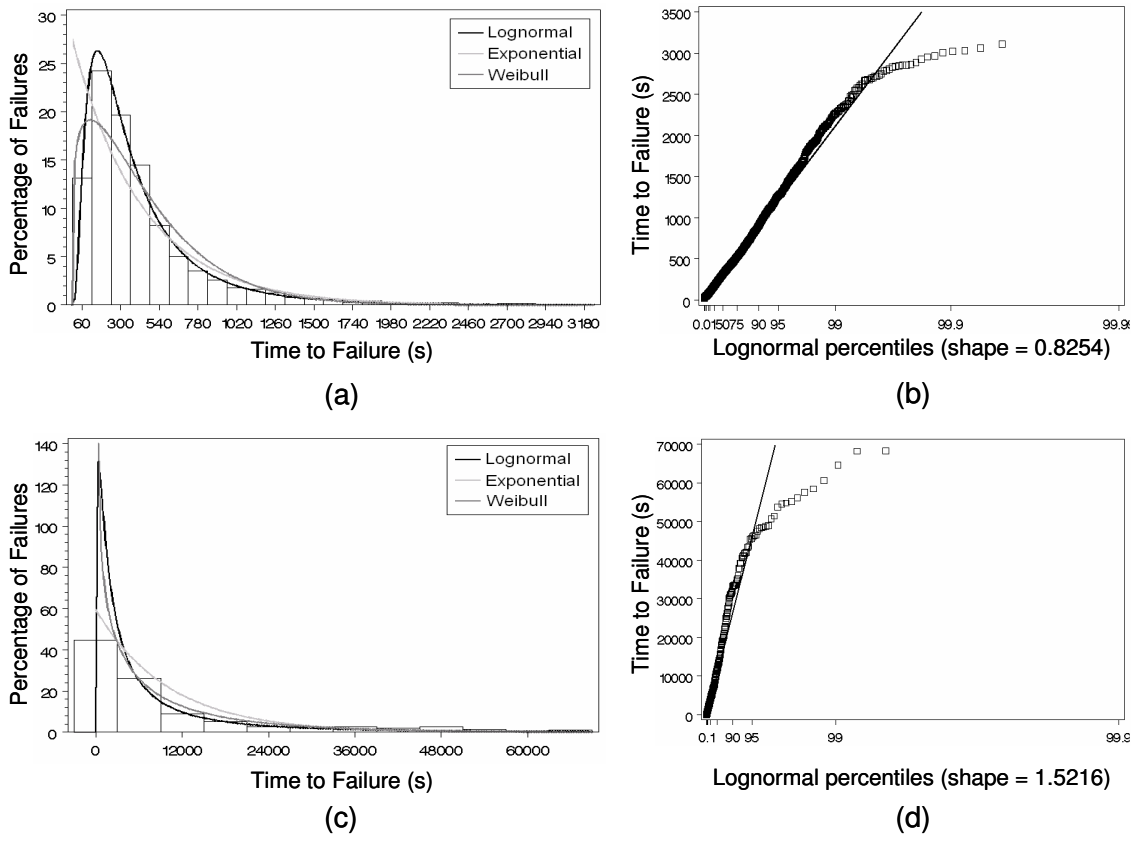


Figure 3.12: Examples of TTF fittings. (a) Histogram and fittings for "Bind Failed" failures on Azzurro, (b) probability plot of Azzurro's "Bind Failed" TTF values for the Lognormal distribution, (c) Histogram and fittings for "Connect Failed" failures on Verde, (d) probability plot of Verde's "Connect Failed" TTF values for the Lognormal distribution.

one. As for the goodness of fit tests (table 3.4), the p-values obtained for the lognormal fitting are the best ones, for both cases.

The same analysis has been conducted for each User Level Failure on each node, and the results are summarized in table 3.5. Failure data are gathered from the Random WL. *Giallo* is not present, since the NAP only records system level data. Each cell in the table contains the fitted distribution and its parameters, the MTTF evaluated in seconds, and the coefficient of variation, evaluated as the standard deviation divided by the MTTF. In

Table 3.4: Goodness of Fit Tests for Bind Failed on Azzurro and Connect Failed on Verde

Results for Bind Failed failures on Azzurro

	Lognormal fitting		Exponential fitting		Weibull fitting	
Test	**Result**	**p-value**	**Result**	**p-value**	**Result**	**p-value**
Kolmogorov-Smirnov	D = 0.014	0.156	D = 0.116	< 0.001	D = 0.075	< 0.001
Cramer-von Mises	$W^2 = 0.191$	> 0.25	$W^2 = 22.76$	< 0.001	$W^2 = 11.23$	< 0.001
Anderson-Darling	$A^2 = 1.699$	0.136	$A^2 = 157.5$	< 0.001	$A^2 = 76.84$	< 0.001

Results for Connect Failed failures on Verde

	Lognormal fitting		Exponential fitting		Weibull fitting	
Test	**Result**	**p-value**	**Result**	**p-value**	**Result**	**p-value**
Kolmogorov-Smirnov	D = 0.032	> 0.25	D = 0.192	< 0.001	D = 0.076	0.007
Cramer-von Mises	$W^2 = 0.077$	> 0.25	$W^2 = 6.941$	< 0.001	$W^2 = 0.682$	0.016
Anderson-Darling	$A^2 = 0.921$	> 0.25	$A^2 = 39.31$	< 0.001	$A^2 = 4.301$	0.007

particular, $Logn(\sigma, \mu)$ indicates a lognormal distribution, with shape parameter σ and scale parameter μ, whereas $Exp(\lambda)$ indicates an exponential distribution with shape parameter λ. A minus symbol in a cell x,y means that no data are available for failure x on node y, whereas the acronym n.e.d. stands for not enough data, meaning that the number of available data items were not enough to conduct a significant fitting. The coefficient of variation resulted to be low for most of the failures, confirming a good consistency of the data.

As can be noticed from the table, almost all the failures on each node are distributed as lognormal. The lognormal distribution is used extensively in reliability applications to model failure times. A random variable can be modeled as log-normal if it can be thought of as the multiplicative product of many small independent factors. In our case, this means that User Level Failures are the product of many small faults at a lower level. These faults can be both software faults, e.g., heisenbugs (i.e., design faults which conditions of activation occur rarely or are not easily reproducible [107]) at the various level of the Bluetooth stack, and channel faults, as the "Hard Payload Corruption" case. As for software faults, this is

Table 3.5: Fitting summary of every User Level Failure on each node. Each cell contains the fitted distribution, the MTTF, and the coefficient of variation. Data comes from the Random WL

	Verde (2.4)	Miseno (2.6)	Azzurro (2.6)	Ipaq (2.4)	Zaurus (2.4)	Win
Inquiry/Scan Failed	n.e.d.	-	-	n.e.d.	-	-
Nap Not Found	Logn(1.11,10.98) 58154 ; 0.70	Logn (1.39,10.73) 54812 ; 0.83	Logn(1.19,10.61) 54348 ; 1.10	Logn(1.19,11.03) 60085 ; 0.92	Logn(1.16,10.62) 54615 ; 0.75	Logn(1.23,10.93) 56148 ; 0.82
SDP Search Failed	n.e.d.	-	n.e.d.	n.e.d.	-	n.e.d.
Connect Failed	Logn (1.52,8.25) 10035 ; 1.42	Logn (1.36,9.36) 21899 ; 1.19	Logn (1.39,10.02) 42408 ; 0.86	Logn (1.86,9.69) 36862 ; 1.18	Logn (1.12,10.05) 36876 ; 0.85	-
Pan Connect Failed	Logn (1.48,10.33) 54851 ; 0.79	Logn (1.39,10.28) 54248 ; 1.05	Logn (1.62,10.30) 54561 ; 0.93	Logn (1.45,10.32) 54812 ; 1.16	Logn (1.51,10.23) 53789 ; 1.21	-
Bind Failed	-	-	Logn (0.83,5.73) 436 ; 0.96	-	-	Logn (0.84,10.06) 44506 ; 1.25
Switch Role Com. Failed	Logn (1.34,10.07) 45218 ; 1.19	Logn (1.37,9.40) 22648 ; 0.88	Logn (1.39,9.50) 25134 ; 0.84	Logn (1.32,7.00) 2695 ; 2.17	Logn (1.72,8.41) 12234 ; 1.27	-
Switch Role Req. Failed	n.e.d.	Logn (1.10,10.02) 41931 ; 1.12	Logn(1.13,10.12) 48201 ; 0.93	Logn(1.13,10.15) 50839 ; 0.90	Logn(1.11,9.89) 29660 ; 0.67	-
Packet loss	Logn (1.80,8.33) 11682 ; 1.25	Logn (1.60,7.40) 5256 ; 2.11	Logn (1.66,6.97) 3386 ; 2.00	Logn (1.89,8.01) 9973 ; 1.47	Logn (1.83,7.74) 7784 ; 1.55	Logn (1.81,9.63) 30976 ; 0.74
Data Mismatch	Exp(46978) 46978 ; 1.03	Exp(50384) 50384 ; 1.12	Exp(47526) 47526 ; 0.98	Exp(48726) 48726 ; 0.96	Exp(46892) 46891 ; 1.01	Exp(51578) 51578 ; 1.09

coherent with recent results arguing that they can be successfully modeled as lognormal [78].

From table 3.5 it can be noticed that the failures follows the same distribution regardless of the particular node's characteristics. For example, Packet Losses are lognormal on every node. This confirms that the underlying nature of the failure is the same. What changes are the parameters of the distributions, since failure dynamics depend on the different software and hardware architectures. For example "Switch Role Command Failed" failures are more frequent on PDAs (Ipaq and Zaurus), due to the complexity introduced by the BCSP. "Bind Failed" failures only appeared on Azzurro and Win. On *Azzurro*, that runs a Fedora Core distribution, the problem remained even after upgrading the hardware to a Pentium 4 1.8 GHz with 512 Gb RAM. The problem is hence probably due to the new

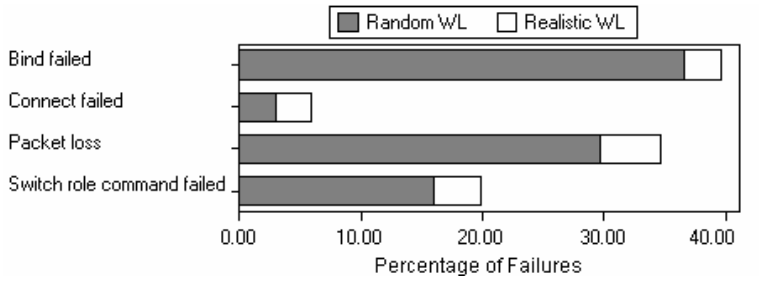


Figure 3.13: Workload influence on failures' manifestation

version of the Hardware Abstraction Layer (HAL), firstly deployed on fedora core distributions, and responsible for the network interfaces hotplug mechanism. As will be discussed in section 3.5.1, "Bind Failed" are indeed mostly due to the hotplug mechanism, which appeared to be extremely slow on Fedora Core and Windows.

Besides these examples, no significant differences have been observed with respect to different versions of the Linux Kernel (2.4 and 2.6). On the other hand, it is evident the difference between Linux machines and the Windows one. In general, the Windows node, running the Broadcomm stack, exhibits better reliability, in that failures are rarer if compared with Linux nodes (Win's MTTF values are in general higher than other nodes MTTF values). Moreover, some failures do not occur at all on Windows nodes, such as "Connect Failed", "PAN Connect Failed", "Switch Role Request Failed", and "Switch Role Command Failed". This can be explained by looking at the implementation of the correspondent API, which are asynchronous and robust with respect to problems during the request.

Interestingly, only "Data Mismatch" failures are distributed as exponential. This is coherent with the fact that the only cause for "Data Mismatches" are "Hard Payload Corruptions", which also resulted to be exponentially distributed.

Let us now investigate how these results change if we consider the failure data obtained with the realistic WL. The histogram in figure 3.13 gives a quick idea of the differences[8], that is, the realistic WL causes in general less failures than the random one. This result is coherent with the major difference between the two WLs, i.e., the random WL is characterized by short data transfer sessions (about 1.5 mins on average); the realistic WL has instead longer data transfer sessions (about 5 mins of average). Hence, the random WL spends most of the time creating and destroying connections, whereas the realistic WL stresses more the established connections. The result is that the random WL causes more failures, since it continuously stimulates the piconet with new connections. Another evidence is that, in the case of realistic WL, most of the failures are "Packet Losses" (which occur during data transfer), whereas the most frequent is the "Bind Failed" in the case of random WL (which occurs during the connection setup).

Even if the realistic WL causes less failures, the nature of them is the same as the ones caused by the random WL. An evidence of this statement is provided by table 3.6, where the results of the fitting are proposed for failure data obtained with the realistic WL. The table shows that the distributions are the same, although we observe a slight change in their parameters. In particular, the "NAP not Found" failure seems to be more frequent in the case of realistic WL. This can be explained considering that when the realistic WL is used, the NAP is often overloaded with data transfers, hence causing the SDP server to fail to answer requests for the NAP service on time. Other failures, such as "Bind Failed" and "Packet Loss", manifest slightly less often than in the case of random WL. This can be

[8]For better readability, we reported only the most frequent failures in the histogram

Table 3.6: Fitting summary for every User Level Failure on each node. Each cell contains the fitted distribution, the MTTF, and the coefficient of variation. Data comes from the Realistic WL

	Verde (2.4)	Miseno (2.6)	Azzurro (2.6)	Ipaq (2.4)	Zaurus (2.4)	Win
Inquiry/Scan Failed	n.e.d.	-	-	n.e.d..	-	-
Nap Not Found	Logn (1.17,10.57) 53915 ; 1.12	Logn(1.21,10.43) 53178 ; 1.81	Logn(1.24,10.39) 52608 ; 0.89	Logn(1.15,10.52) 54846 ; 1.18	Logn(1.19,10.03) 45176 ; 1.59	Logn(1.21,10.12) 47681 ; 0.92
SDP Search Failed	n.e.d.	-	n.e.d.	n.e.d..	-	n.e.d..
Connect Failed	Logn(1.26,9.29) 19165 ; 0.91	Logn(1.42,9.29) 20457 ; 0.84	Logn(1.32,9.92) 41174 ; 0.92	Logn(1.85,9.65) 31450 ; 1.04	Logn(1.14,10.45) 50509 ; 1.53	-
Pan Connect Failed	n.e.d	n.e.d	n.e.d	n.e.d	n.e.d	-
Bind Failed	-	-	Logn(1.69,7.65) 7016 ; 1.85	-	-	Logn(1.65,11.67) 61451 ; 1.52
Switch Role Com. Failed	Logn(1.41,10.16) 48202 ; 1.95	Logn(1.38,9.31) 21154 ; 0.94	Logn(1.51,9.56) 31910 ; 0.89	Logn(1.64,7.29) 4659 ; 2.01	Logn(1.69,9.01) 18174 ; 1.52	-
Switch Role Req. Failed	n.e.d.	n.e.d.	n.e.d.	n.e.d.	n.e.d.	-
Packet loss	Logn(2.23,8.55) 16401 ; 2.15	Logn(2.21,8.23) 15273 ; 1.26	Logn(2.34,7.75) 11883 ; 1.38	Logn(2.18,8.98) 17048 ; 0.98	Logn(2.31,9.01) 17851 ; 1.56	Logn(2.19,10.32) 50146 ; 0.83
Data Mismatch	Exp(48875) 48875; 1.05	Exp(50548) 50547 ; 1.03	Exp(49486) 49486 ; 0.92	Exp(50025) 50025 ; 1.01	Exp(48104) 48104 ; 0.96	Exp(50984) 50984 ; 1.05

due to the less stressful nature of the realistic WL.

Figure 3.14a shows the "Packet loss" distribution as a function of the networked application that was run by the Realistic WL during the failure. Results pinpoint Peer to Peer (P2P) and Streaming applications as the most critical for BT PANs. They are indeed characterized by long sessions with continuous data transfer, which overload the channel and stress its time-based synchronization mechanism. At a first glance, this may surprise, since P2P protocols are TCP-based. However, the most of the packet losses are due to broken BT links, which cause the TCP end-to-end channel to brake as well. Streaming causes less failures than P2P due to its isochronous nature, which better fits the BT time-based nature. Less failures are experienced with Web, Mail, and File Transfer Protocol (FTP) applications, which are characterized by intermittent transfers. This indicates that

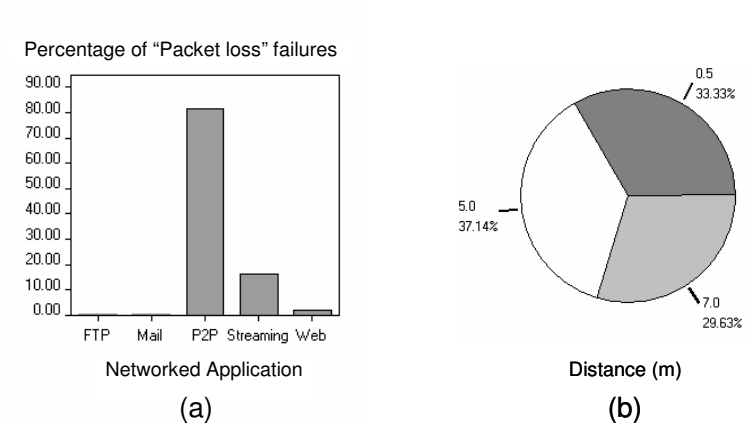


Figure 3.14: a) "Packet loss" failure distribution as a function of the networked application used by the Realistic WL. b) Failures distribution as a function of the distance from PANUs to the NAP

Bluetooth ACL channels are less failure prone when used in an intermittent manner.

Finally, it is interesting to notice that *the failure distribution is not significantly influenced by the distance* between the BT antennas. From data relative to the Realistic WL we measured that the 33.33%, 37.14%, and 29.63% of failures occur with a distance 0.5 m, 5 m, and 7 m, respectively, as summarized in figure 3.14b. Bind failures are not taken into account in the count. They would have biased the measure, since they only manifest on two hosts.

3.5 Masking Strategies and SIRAs

3.5.1 Error Masking Strategies

A deeper investigation on obtained results led us to the definition of three error masking strategies. These strategies have been defined for the following user level failures.

- *Bind failed*: it is mostly related to problems in the host OS hotplug interface or to errors when invoking HCI commands (see table 3.2). Our investigation on source code on BT Kernel modules has led to the following considerations. The creation process of an IP network interface over BT requires: i) a time interval (T_C) for the creation of the L2CAP connection; and ii) a time interval (T_H) needed by the BT stack to build the BNEP virtual network interface over L2CAP, and by the OS hotplug interface to configure the interface. The problem is that the Pan Connect API is not synchronous with T_C and T_H, even in the Broadcomm case, hence, a "Bind Failed" failure occurs whenever the application attempts to bind a socket on the supposed existing BNEP interface before T_C and T_H. In particular, if the bind request is issued before T_C, a HCI command failure (i.e., command for invalid handle) occurs, because the L2CAP connection is not present. If the request is instead issued after T_C but before T_H, a failure occurs, either because the interface is not present or it does not have been configured yet by the hotplug mechanism. To prevent the failure from occurring, it is sufficient to wait for T_C and T_H to elapse. T_C elapses as soon as the L2CAP connection has a valid handle. This check can be easily added in the PAN connection API. As for T_H, the OS hotplug interface can be instrumented so as to notify the application as soon as the BNEP interface is up and configured.

- *Switch role command failed* and *NAP not found*: the switch role failure is often related to out of order packets failures signaled by the BCSP module (49.7% of the cases, see table 3.2). Also, it especially manifests on PDAs (as seen in section 3.4.4),

since they adopt BSCP. The failure can also be related to many other causes, such as unexpected L2CAP frames (0.9% local, 4.4% on the NAP), HCI command for invalid handle (10.9% local, 2.4% on the NAP), and busy BNEP device(18.8% local). This multitude of transient causes does not isolate the symptoms of the failure, and it does not allow to define precise maskings. Therefore we tried to simply repeat the command when it fails. We experienced that repeating the action up to 2 times (with 1 second wait between a retry and the successive) is enough to let the underneath transient cause disappear, and hence to make the command success. The same considerations apply to the case of NAP not found. We experienced that repeating the operation up to 1 time (with 1 second wait) is enough to make the operation success.

3.5.2 SW Implemented Recovery Actions

As soon as a failure is detected[9], several Software Implemented Recovery Actions (SIRAs) are attempted in cascade. This approach allows to pinpoint, for each failure, the most effective recovery action, that is, the one that fixes the problem with a high probability. Note that this is the only viable approach, since we do not have any *a priori* knowledge about the best recovery to perform each time. Upon failure detection, the following recovery actions are triggered subsequently, i.e., when the i-th action does not succeed, the $(i+1)$-th action is performed.

1. *IP socket reset*: the socket is destroyed and then rebuilt; this action is applied only

[9]Failure detection is performed by simply checking the return state of each BT or IP API that is invoked by the WL. Examples are the indication that a PAN connection cannot be created, or a timeout when waiting for an expected packet.

when the connection (PAN and L2CAP) is already up.

2. *BT connection reset*: the L2CAP and PAN connections are closed and established again;

3. *BT stack reset*: the BT stack variables and data are cleaned up, by restoring the initial states;

4. *Application restart*: the `BlueTest` is automatically closed and restarted;

5. *Multiple application restart*: up to 3 application restarts are attempted, consecutively;

6. *System reboot*: the entire system is rebooted;

7. *Multiple system reboot*: up to 5 system reboots are attempted. Actually, we experienced a maximum of four consecutive reboots to restore normal operation.

The given recovery actions are ordered according to their increasing costs, in terms of recovery time. The more attempts have to be done for a failure, the more the failure is severe: if action j was successful, we can say the failure has a severity j. This gives us an indication for failure severity.

Table 3.7 highlights the relationship between user level failures and recovery actions. Data is relative to all the machines and come from both testbeds, and it considers only non-masked failures. Each number in a cell represents the percentage of success of the recovery action (on the column) with respect to the given user level failure (on the row). Therefore, the numbers give an indication of the effectiveness of each SIRA for each failure (which is an estimation of the probability that a certain recovery action goes through).

Table 3.7: User failures-SIRAs relationship

	SIRAs							
User Level Failures	IP socket reset	BT connection reset	BT stack reset	Application restart	System reboot	Multiple app restart	Multiple sys reboot	**TOT**
Inquiry/scan failed			34.5	30		35.5		0.2
SDP search failed		40.1	39.8		20.1			0.2
Connect failed		0.5	14.9	55.8	25.6	0.1	3.1	13.6
PAN connect failed		46.4	35.7	5.4	12.5			1.2
Sw role request failed		28.4	48.2	4.9	17.3	1.2		1.9
Packet loss	5.9	7.2	25.8	33.1	26.7	0.2	1.1	81.2
Data mismatch								1.7
Total	4.9	7.2	24.8	35.2	26.1	0.5	1.3	

The column "Tot" reports the total percentage with respect to unmasked failures, e.g., the 13.6% of unmasked failures are "Connect Failed". Similarly to table 3.2, numbers on each row sum to 100, in order to have a simple indication of which are the effective SIRAs for each user failure. For example, a "Switch Role Request Failed" is most probably recovered by resetting the BT stack (in the 48.2% of the cases). Hence, this should be the first action to be attempted when the failure is detected. The table also allows to calculate failure severity. For instance, the "Connect failure" is one of the most severe, since it is often recovered by expensive SIRAs (the 84.6% of the cases from "Application restart" up to "Multiple system reboot").

Failure-recovery relationship provides further understandings of failure causes and nature. As an example, packet losses recovered by an IP socket reset (the 5.9% of packet losses) are due to "Hard Payload Corruptions" detected by the IP CRC. It is indeed not necessary to reestablish the L2CAP and BNEP connections. The rest of the packet losses are instead likely due to a broken link, since they at least require the connection to be

reestabilished. These broken link failures can be caused by "Hard Payload Corruptions" affecting the L2CAP or BNEP headers, then causing the corruption of the data structures that maintain the link state. Hence, depending on the severity of the corruption, several different recovery actions are needed, from the BT Connection reset to the reboot of the machine. For "Data Mismatch" failures, no recoveries are defined. "Data Mismatch" failures are not realistically recoverable, since a real application only relies on integrity mechanisms furnished by the communication protocols, and cannot know the actual instance of data being transferred.

Multiple Application Restart and BT Stack Reset recoveries are the most frequent. This indicates that the most of the failures are due to corrupted values of the state of the stack or to the corrupted execution state of the application.

It is worth noting that some failures can be recovered only after multiple application restarts or even after multiple reboots. Those failures are probably caused by problems in the NAP. This is confirmed in table 3.2, which highlights such relationship between user failures and NAP's system failures.

In Table 3.8 we summarize the statistical distributions of the time to recover (TTR) for each recovery action on every node. Data comes from both the workloads. This gives an overall idea of the time that is needed to recover from failures, with respect to the different architectures we adopted. Similarly to table 3.5, each cell in table 3.8 contains the fitted distribution and its parameters, the MTTR, and the coefficient of variation. In this case, the distributions are translated in time, by means of the location parameter θ. It is indeed unlikely to find TTR values equals to 0. Specifically, $Logn(\theta, \sigma, \mu)$ indicates a lognormal

Table 3.8: Fitting summary for each Recovery Action on every node. Each cell contains the fitted distribution, the MTTR, and the coefficient of variation

	Verde (2.4)	Miseno (2.6)	Azzurro (2.6)	Ipaq (2.4)	Zaurus (2.4)	Win
IP Socket reset	Exp(1.97;2.30) 4.27 ; 1.05	Exp(1.98,2.42) 4.40 ; 0.83	Exp(1.58,5.00) 6.58 ; 0.78	Exp(1.94,1.38) 3.31 ; 0.71	Exp(1.93,2.86) 4.79 ; 0.77	Exp(9.99,0.25) 10.24 ; 0.07
BT Connection reset	Exp(3.81,10.4) 14.21 ; 0.71	Exp(4.89,23.49) 28.40 ; 0.61	Exp(5.97,14.60) 20.57 ; 0.60	Exp(3.99,17.00) 20.99 ; 0.75	Exp(3.97,12.47) 16.44 ; 0.79	-
BT Stack reset	Logn (11.67,0.57,4.27) 82.12 ; 0.72	Logn (8.75,0.37,1.47) 20.08 ; 0.37	Logn (10.67,0.81,2.48) 27.12 ; 0.55	Logn (7.77,0.77,3.12) 38.71 ; 0.74	Logn (6.47,0.77,2.91) 31.06 ; 0.65	Exp (58.92,4.58) 63.50 ; 0.11
Application restart	Logn (14.62,0.32,4.31) 93.33 ; 0.27	-	Logn (10.47,0.43,3.38) 43.02 ; 0.38	Logn (-6.49,0.34,3.84) 42.65 ; 0.40	Logn (2.82,0.61,3.52) 43.41 ; 0.58	n.e.d.
System reboot	Exp (370.64,33) 403.64 ; 0.85	Exp (269.99,21.38) 291.36 ; 0.70	Exp (231.71,37.07) 268.86 ; 0.71	Exp (98.67,21.89) 120.56 ; 0.82	Exp (102.29,17.3) 119.59 ; 0.73	Exp (612.5,10.5) 623.00 ; 0.59
Multiple application restart	n.e.d.	n.e.d.	n.e.d.	n.e.d.	n.e.d.	n.e.d.
Multiple system reboot	n.e.d.	n.e.d.	n.e.d.	n.e.d.	n.e.d.	-

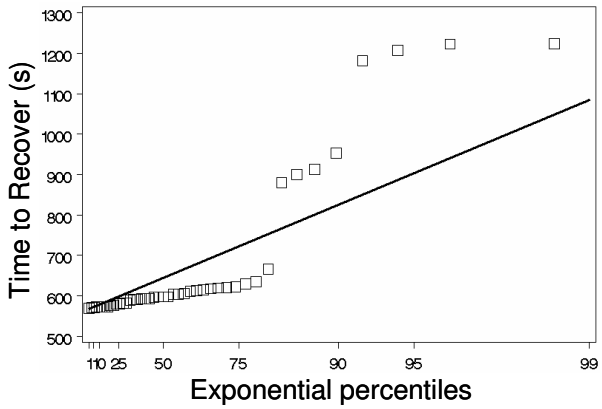


Figure 3.15: Probability plot of Miseno's Multiple System Reboot TTR values for the exponential distribution

distribution, with shape parameter σ, scale parameter μ, and location parameter θ, whereas $Exp(\theta, \lambda)$ indicates an exponential distribution with shape parameter λ and location parameter θ. It is worth noting that "Multiple Application Restart" and "Multiple System Reboot" recovery actions could be probably modeled as hyper-exponential. Indeed, from the probability plot in figure 3.15, which reports the percentiles of "Multiple System Reboot" data from Miseno, it is easy to see that data points are concentrated around 600 s

(roughly equal to the time needed for two consecutive reboots), 900 s (three reboots) and 1200 s (four reboots). This suggests that the TTR for this recovery action could be model as a sum of three exponential random variables each of them centered around 600 s, 900 s and 1200 s, respectively. Unfortunately, we do not have enough data to confirm these hypothesis.

3.6 Dependability Improvement of Bluetooth PANs

From collected data, and from results of the previous sections, it is possible to estimate the dependability improvement which can be obtained by integrating software implemented recovery actions and error masking strategies into the testbeds. To this aim, we consider two typical usage scenarios: i) each time that a failure occurs, the user performs the reboot of the terminal (PC or a PDA); ii) the user performs the following recovery actions, subsequently, ii.1) he/she tries to restart the application, and ii.2)in the case that the application fails again, he/she reboots the terminal. For both scenarios we are able to evaluate Mean TTF and TTR values (MTTF and MTTR), since we can calculate the average recovery time for reboot and for the application restart from the collected data. In order to obtain upper bound measures, we assume that the 'user thinking time' is zero, i.e. we do not encompass it in the TTR value. Finally, we compare the two scenarios with the enhanced facilities that we implemented in the workload, i.e. software implemented recovery actions and error masking. Results are summarized in table 3.9, for both realistic and random WLs and for all the nodes of the testbeds. We evaluate the dependability improvement for both workloads because i) the random one give us indications on the dependability level of BT as used

Table 3.9: Dependability Improvement of Bluetooth PANs

Random Workload

	Only Reboot	App restart and Reboot	With only SIRAs	SIRAs and masking
MTTF (s)	164.39	364.78	422.56	**1097.95**
C.o.V. $_{MTTF}$	7.50	3.66	3.26	2.61
MTTR (s)	331.51	131.12	**75.34**	**141.31**
C.o.V. $_{MTTR}$	0.94	1.60	2.25	1.63
Availability*	< 0.331	< 0.736	**0.852**	**0.886**
% Coverage	0	0	57.76	**73.84****
% Masking	0	0	0	59.98

Realistic Workload

	Only Reboot	App restart and Reboot	With only SIRAs	SIRAs and masking
MTTF (s)	3111.97	3240.97	3351.20	**7090.18**
C.o.V. $_{MTTF}$	2.06	2.00	1.96	1.62
MTTR (s)	328.49	199.49	**89.26**	**124.29**
C.o.V. $_{MTTR}$	1.09	1.43	2.24	1.84
Availability*	< 0.904	< 0.942	**0.974**	**0.983**
% Coverage	0	0	62.15	**75.42*****
% Masking	0	0	0	52.54

* = MTTF/(MTTF+MTTR)

** = 59.98% (masking) + 13.86% (coverage of the remaining failures)

*** = 52.54% (masking) + 22.88% (coverage of the remaining failures)

in high critical scenarios; and ii) the realistic one give us indications of the dependability level achieved form the everyday dependability perspective. For each obtained measure, we also report the coefficient of variation as a measure of the variability. The advantage of using such a coefficient is that it is normalized by the mean, and hence it allows comparison among different measures.

As for the coverage, we refer to failure mode coverage as defined in [1](*failure assumptions coverage*). As we have already observed, the random WL causes more failures hence it exhibits a lower availability and reliability than the realistic WL. In both cases, SIRAs give a good coverage, and specifically the 57.76% for the random WL, and the 62.15% for the realistic WL. In other terms, SIRAs recover more than half of the failures, without rebooting or restarting the application (as a typical user would have done). In the fourth column of both tables we reported results taking also into account the error masking strategies, which gives a coverage of 73.84% and 75.42% for the random and realistic WL, respectively. This, in our opinion, represents a good result for the effectiveness of fault tolerance techniques

we have assessed from the analysis of gathered data. As far as the availability is concerned, results show that the software implemented recovery actions and the error masking strategies actually improved the availability of BT PANs. The effectiveness is evident in the case of the random WL: starting from 0.331 (scenario 1) and 0.736 (scenario 2), which are the upper bound measured values of BT PAN availability, to 0.852 and 0.886, with an improvement of 20.38% (relative to scenario 2), up to 167.67% (relative to scenario 1). The error masking strategies influence the MTTF estimation, which varied from 164 s to 1098 s. This results in an actual reliability improvement of 569%. Similar considerations apply to the realistic WL, even if in this case the margins of improvement are lower. The availability improvement is of 4.35% (relative to scenario 2), up to 8.74% (relative to scenario 1). The MTTF estimation varied from 3112 s to 7090 s, with a reliabilty improvment of 128%. It should be noted that, even using SIRAs and masking, the MTTF values are low, i.e., each 18 minutes on average a node in the piconet fails, in the random WL case. In the realistic WL case, this figure increases up to 118 minutes. Since our measurements are based on a 24/7 experiment, this represents a major reliability issue in all those scenarios in which piconets are permanently deployed and used continuosly, such as, wireless remote control systems for robots, and aircraft maintenance system. In these critical scenarios, SIRAs and masking are not enough, and extensive fault tolerance techniques shoud be adopted, such as, using redundant, overlapped piconets. From an everyday dependability perspective [93], SIRAs and masking strategies allow instead a significant improvement. If we look at the realistic WL, a failure happens about every 54 minutes (scenario 2). This number improves to a failure every 118 minutes when SIRAs and masking are adopted. This means that a

user surfing the web or uploading a file via P2P over Bluetooth will experience the failure of a node in the piconet about every two hours. Finally, as for the maintainability, with reference to the random WL the MTTR decreases from 331.51 s to 75.34, thanks to SIRAs. It results instead slightly longer (141.31 s) in the case with both SIRAs and masking, since the remaining, unmasked failures (the 40.02% of the failures) are more severe, and require costly recovery actions. Same considerations apply for the realistic WL.

If you try the best you can, the best you can is good enough.

Radiohead - Optimistic

Chapter 4

FFDA of Mobile Phones

Modern mobile phones, or smart phones, are becoming more and more complex in order to meet costumer demands. The complexity directly affects the reliability of mobile phones, while the user tolerance for failures becomes to decrease, especially when the phone is used for business- or safety-critical applications. Despite these concerns, there is still little understanding on how and why these devices fail and no techniques have been defined to gather useful information about failures manifestation from the phone.

This chapter addresses these problems by proposing a FFDA campaign on Mobile Phones, trying to answer the key research questions which arise when targeting this type of devices.

4.1 Rationale and Characterization of the FFDA campaign

The measurement-based dependability characterization of smart phones is a complex task, which is exacerbated by the lack of previous studies. At the end of chapter 2, two key research questions have been posed in this respect:

- *Where do we have to start from?* And,
- *How to collect the failure data?*

In order to answer the first question, we believe that it is needed to gain a first understanding of the possible failures affecting mobile phones, in order to clearly identify what has to be collected and how, thus leading to the answer of the second question. Following

this rationale, this chapter presents a study of mobile phones failure data gathered from the only publicly available source: the users. Thus, we found several web forums where mobile phone users post information on their experiences using different devices. This activity allows to answer the first question by gaining useful insight into the nature of the observed failure behavior, and to improve the understanding of the dynamics of typical problems reported by cell phones customers. In particular, freezes and self-shutdown events are reported as two of the more severe failure manifestations.

Building on this experience, a *logger* application is then conceived, in order to collect failure-related information from mobile phones, thus answering the second question. The logger has been installed on several Symbian OS smart phones, and has gathered failure data on freezes and self-shutdowns for more than one year. We chosen the Symbian OS because of i) its open programmability features with C++ and Java programming languages, and ii) its wide spread use at the time of writing.

This rest of chapter gives the needed background on Smart phones and the Symbian OS, then it presents the results from both the experimental campaigns: the preliminary study on web forums, and the results of the data collected by means of the logger. The characteristics of these two campaigns are summarized in table 4.1, according to the framework defined in section 2.3. As one could expect, the first study achieves a lower density than the second one, and it mostly produces qualitative results. On the other hand, the second study benefits from the presence of a bigger number of data items, each one more detailed than web forums' reports, thus allowing for a better quantitative characterization of mobile phones dependability. It also permits to uncover the major failure causes.

Table 4.1: Characterization of conducted studies on mobile phones

Analysis of failure reports from web forums	
Dimension	**Value**
Date	2006
Purpose	Dependability Study
Actor	Academy
Source	Third Party
Target System	Mobile Phones
Achieved Results	Definition of an high-level failure model, understanding of potential causes
Length	30 months
Data Items	533
Density	18 items/month
Data Source	Failure reports
Levels	Human operator
Workload	Idle
Manipulation	Filtering
Analysis	Classification
Confidence	Not given

Analysis of failure data gathered via the logger	
Dimension	**Value**
Date	2006
Purpose	Dependability Study
Actor	Academy
Source	Internal
Target System	Symbian OS Smart Phones
Achieved Results	Quantitative measures, uncovering of actual underlying causes
Length	14 months
Data Items	1246
Density	89 items/month
Data Source	Event logs (Panics and high level events)
Levels	OS, Application
Workload	Idle
Manipulation	Filtering, temporal coalescence
Analysis	Classification, Dependability Measurement, Other (root cause analysis)
Confidence	Not given

It is interesting to notice how the framework defined in section 2.3 allows for a quick comparison of FFDA studies. For example, from table 4.1, we can quickly conclude that the logger study is of higher quality if compared to the web forums study. Similarly, both works are not deep as the one performed for Bluetooth (see table 3.1)

4.2 Background on Smart Phones and the Symbian OS

4.2.1 The Evolution of Mobile Phones

Mobile/cellular phone evolution can be described according to three waves, each one characterized by a specific class of mobile terminal [47]:

- *Voice-centric mobile phone* (first wave): a hand-held mobile radiotelephone for use

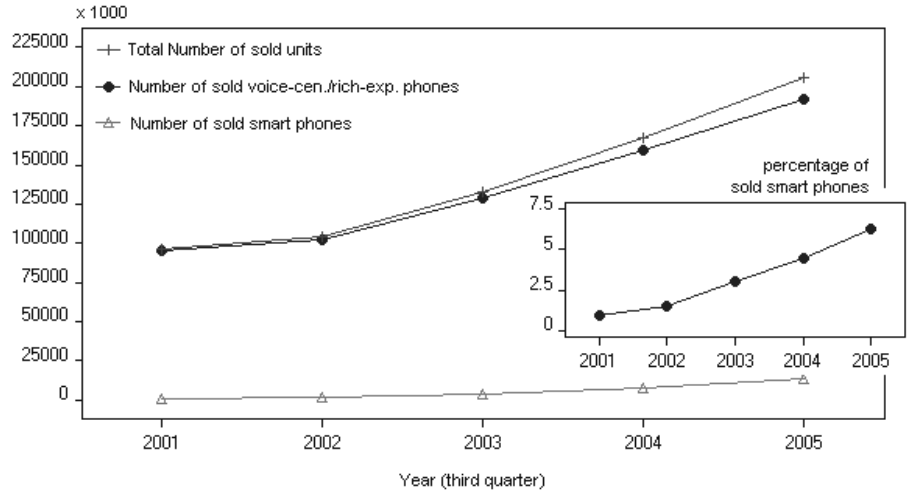


Figure 4.1: Mobile phones' market growth

in an area divided into small sections (cells), each with its own short-range transmitter/receiver. These devices support a special text notification service called SMS (Short Message Service).

- *Rich-experience mobile phone* (second wave): a mobile phone with numerous advanced features, typically including the ability to handle data (web-browsing, e-mail, personal information management, images handling, music playing) through high-resolution color screens.

- *Smart phone* (third wave): a general-purpose, programmable mobile phone with enhanced processing and storing capabilities. It can be viewed as a combination of a mobile phone and a Personal Digital Assistant (PDA), and it may have a PDA-like screen and input devices.

Newer mobile phone models on the market feature more computing and storing capabilities, new operating systems, new embedded devices (e.g., cameras, radio) and communication technologies/protocols (Bluetooth, IrDA, WAP, GPRS, UMTS), as well as new shapes and designs. As shown in Figure 4.1[1], the number of units sold during the third quarter of 2005 (205 millions) doubled with respect to the third quarter of 2001 (97 millions units). In the same period, the percentage of smart phones sold has sextupled. While innovative features are attractive and meet customer demands, the race toward innovation increases the risk of delivering less reliable devices, since new mobile phones are often put on the market without comprehensive testing.

According to industry sources the time from conception to the market deployment of a new phone model is between 4 to 6 months. Clearly the pressure to deliver a product on-time frequently results in compromising its reliability. The hope is that any potential reliability problems can be fixed quickly by developing new releases of phone firmware, which can be installed on the phone by service phone centers.

4.2.2 Mobile Phone Architectural Model

This section introduces a software layered architecture model for mobile phones. The model, which is depicted in Figure 4.2, is based on the architecture of the current generation of smart phones.

In the case of simpler devices, e.g., voice-centric or rich-experience mobile phones, some layers may not be present or may be integrated in another layer. As Figure 4.2 shows, the

[1]sources: http://www.itfacts.biz, http://www.theregister.co.uk

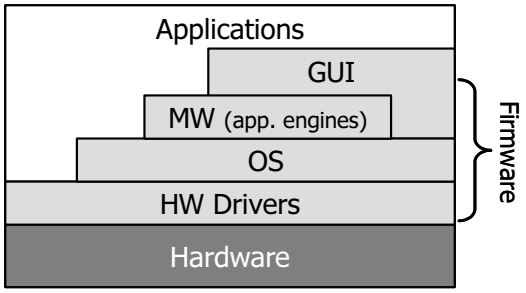


Figure 4.2: Mobile phones' architectural model

term *firmware* encompasses everything between *Hardware* and *Applications*. The software layers are written onto read-only memory (ROM) and can be updated by "flashing" the ROM. The firmware encompasses: i) device drivers supporting storage or communication hardware, ii) the operating system kernel (OS), iii) application engines or middleware (MW) providing a set of Application Programming Interfaces (APIs) for mobile phone programming (e.g., SMS or phonebook management), and iv) the Graphical User Interface (GUI) API. It should be noted that, on some mobile phones, the GUI layer is not implemented as firmware, and can be upgraded without flashing the ROM.

4.2.3 Symbian OS fundamentals

Symbian [47] is a light-weight operating system specifically tailored for smart phones and carried out by several leader mobile phone's manufacturer companies. It is based on a hard real-time, multithreaded kernel that is designed according to the micro-kernel approach. Specifically, the kernel only offers basic abstractions, i.e., address spaces, thread scheduling and message passing interprocess communication; all other system services are provided by *server* applications. Clients access servers using message passing kernel's mechanisms.

Examples of servers are the File Server, for files' management, the Window Server, for user interface drawing, and the Message Server for the Short Message Service (SMS) management.

Since mobile phones resources are highly constrained, special care has been taken for memory management issues, throughout the design of the Symbian OS. Specific programming rules are defined so as to free unused memory and to avoid memory leaks, even in the case of failures. In particular, the following mechanisms have been designed: the clean-up stack, the trap-leave technique, and the two-phase construction paradigm. All these mechanisms are tightly linked together. The clean-up stack is an OS resource storing the references of all the objects allocated on the heap memory. As for the trap-leave technique, it is similar to the try-catch paradigm defined for C++ and Java languages: if problems arise during the execution of a trap block, the current method "leaves", and the control steps back to the caller which handles the problem. In the meanwhile, the OS is responsible to free all the objects which have been stored in the clean-up stack during the execution of the trap block, thus avoiding potential leaks. Finally, the two-phase construction paradigm is defined to properly construct objects with dynamic extensions. The mechanism assures that, when errors occur during the construction of an object, the dynamic extension is properly freed by means of the clean-up stack.

As far as multitasking issues are concerned, the Symbian OS defines two levels of multitasking: threads and Active Objects (AOs). Threads are scheduled by the OS thread scheduler, which is a time-sharing, preemptive, priority based scheduler. At the upper

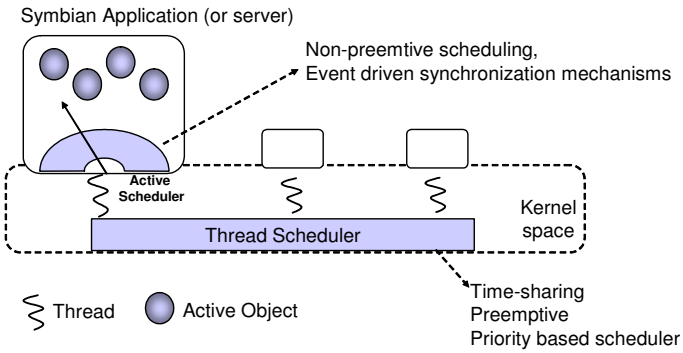


Figure 4.3: Symbian OS multitasking model

level, multiple AOs run within a thread (see figure 4.3). They are scheduled by a non-preemptive, event-driven scheduler, called *active scheduler*. In other terms, AOs multitask cooperatively using an event-driven model: when an AO requests a service, it leaves the execution to another AO. When the requested service completes, it generates an event that is detected by the active scheduler, which in turn inserts the requesting AO in the queue of the AOs to be activated. Non-preemption was chosen to meet light-weight constraints, avoiding synchronization primitives such as mutexes or semaphores. Moreover, AOs belonging to the same thread all runs in the same address space, so that a switch between AOs incurs a lower overhead than a thread context switch.

As shown in figure 4.3, a typical Symbian OS application (or server) consists of a thread running an active scheduler which coordinates one or more AOs. Each application is in turn scheduled by the OS thread scheduler. However, AOs non-preemption characteristics make them not suitable for real-time tasks. On Symbian OS, real-time tasks should be rather implemented using threads directly. The whole design constitute a good compromise between real-time and light-weight design requirements.

A crucial aspect of interest for our activity is represented by panics. In the Symbian OS world, a panic represents a non-recoverable error condition launched by either user or system applications. Applications which are not able to treat an occurred error have to notify a PANIC EVENT to the Kernel. The panic information associated with the event is a record composed of its category and type. Once this event has been notified, the application is killed by the kernel. As for panics notified by system servers, the kernel can decide to reboot the phone to recover them, based on the severity of the panic.

Finally, as for failure logging, the Symbian OS offers a particular server (the *flogger*) allowing an application to log its information. Yet, to access the logged data of a generic X system/application module it is necessary to create a particular directory, with a well defined name (e.g. Xdir) under a particular path. The problem is that the names of such directories are not made publicly available to developers, and are used by manufacturers during the development. Recently, a tool called D_EXC[2] has been proposed to register all panic events generated on a phone. However, the tool does not relate panic events with failure manifestations, and running applications and phone's activity at the time of the failure. This solicited us to build an ad-hoc failure logger for Symbian Smart Phones.

[2]D_EXC is a Symbian project, available at http://www.symbian.com/developer/downloads/tools.html

```
The phone freezes whenever I
try to write a text message,
and stays frozen until I take
the battery out.
```

a)

```
random wallpaper disappearing
and power cycling, due to UI
memory leaks.
```

b)

Figure 4.4: Mobile phones' failure reports examples: a) a user reported problem; b) a known issue

4.3 Starting Point: Web Forums-based analysis

In this section a high level characterization of mobile phones' failures is proposed as the starting point of the subsequent FFDA campaign conducted via the logger. Such a characterization is based on data gathered from several web forums[3] where mobile phone users post information on their experiences using different devices.

The identified failure reports have been divided in two groups:

- *User-reported problems*: These reports contain qualitative information and allow identifying a failure type, user activity when the failure occurred, and actions taken by the user to recover. Figure 4.4a shows an example of a user-reported problem.

- *Known issues*: These reports are more accurate, since they correspond to known design or implementation flaws confirmed by manufacturers. From these reports, it is possible to understand an underlying failure cause and pinpoint the software component(s) responsible for the problem. Figure 4.4b shows an example of a known issue.

The posted information has been carefully filtered in order to consider only those posts

[3]We considered *www.howardforums.com* *cellphoneforums.net*, *www.phonescoop.com*, and *www.mobiledia.com*.

that signal a failure of the device. The reports have then been classified along several dimension and then analyzed.

4.3.1 Classifying Dimensions

Failure Types

A failure is defined as an event that occurs when the behavior exhibited by the phone deviates from the specified one. These failure categories we selected so that they represent the bulk of the reported data. (Clearly it is possible that other categories not present in the analyzed logs exist.) Based on the available data, five failure categories were defined:

- *Freeze* (lock-up or a halting failure [1]): The device's output becomes constant, and the device does not respond to the user's input.
- *Self-shutdown* (silent failure [1]): The device shuts down itself, and no service is delivered at the user interface.
- *Unstable behavior* (erratic failure [1]): The device exhibits erratic behavior without any input inserted by the user, e.g. backlight flashing, continuous self-shutdowns, and self-activation of applications or modes of operation.
- *Output failure* (value failure [13]): The device, in response to an input sequence, delivers an output sequence that deviates from the expected one. Examples include inaccuracy in charge indicator, ring or music volume different from the set one, event reminders going off at wrong times, and unexpected text displayed when browsing the Internet.

- *Input failure* (omission value failure [13]): User inputs have no effect on device behavior, e.g. soft keys and/or key combinations do not work.

The failure classes defined above indicate that mobile phones exhibit the kinds of failures observed in traditional computer systems, e.g., timing failures (freeze and self-shutdown), value failures (output and input failures) and erratic failures (unstable behavior).

User-Initiated Recovery

User-initiated actions to recover from a device failure can be classified according to the following categories:

- *Repeat the action*: Repeating the action is sometime sufficient to get the phone working properly, i.e., the problem was transient.
- *Wait an amount of time*: Often it is enough to wait for a certain amount of time (the exact amount is not reported by users) to let the device stabilize and deliver the expected service (i.e., the device simply fixed itself).
- *Reboot* (power cycle or reset): The user turns off the device and then turns it on to restore the correct operation (a temporary bad state is cleaned up by the reboot).
- *Remove battery*: Battery removal is mainly performed when the phone freezes. In this case, the phone often does not respond to the power on/off button. Battery removal can clean up a permanent bad state (e.g., due to a user's customized settings) and enables the correct operation.

- *Service the phone*: The user has to bring the phone to a service center for assistance in fixing the problem. Often, when the failure is firmware-related, the recovery consists of either a master reset (all the settings are reset to the factory settings and the user's content is removed from the memory) or a firmware update, i.e., uploading a new version of the firmware. Hardware problems are fixed by substituting malfunctioning components (e.g., the screen or the keypad) or replacing the entire device with a new one. Reports indicate that experienced users can apply master reset or update the firmware by themselves.

It is reasonable to assume that, in the case of a failure, a user would first repeat the action (which initially caused the failure) wait for an amount of time, and if the device still does not work, try to reboot it. If the reboot cannot be performed, the user would remove the battery, and as a last resort, he/she would bring the phone in for service. Note that if a failure report does not contain any information about the recovery action taken by the user, we classify the recovery action as *unreported*.

Failure Severity

In introducing failure severity, this study takes the user perspective and defines severity levels corresponding to the difficulty of the recovery action(s) required to restore the correct operation of the device.

- *High*: A failure is considered to be highly severe when recovery requires the assistance of service personnel.

- *Medium*: A failure is considered to be of medium severity when the recovery requires *reboot* or *battery removal*. A reboot may abruptly interrupt the action performed by the user (call, message composing/reading, or browsing), whereas a battery removal often causes a loss of customized phone settings, which have to be manually re-set by the user.

- *Low*: A failure is considered to be of low severity if the device operation can be reestablished by *repeating the action* or *waiting for an amount of time*.

Failure Causes

Ideally, using the insights provided by the failure data, one could attempt to determine root causes (in terms of software and/or hardware components) of failure. Unfortunately, user-reported problems do not usually contain enough detail to trace back to a failure cause or to pinpoint the component that creates problems. Therefore, in many cases, we simply indicate *firmware* as a cause of failure, meaning one of the layers between the hardware and the applications. On the other hand, reports of known issues allow more accurate identification of the component(s) responsible for the failure.

Device activity at the time of a failure

This information allows us to understand what application the device was running at the time the failure. This way, critical applications can be pinpointed.

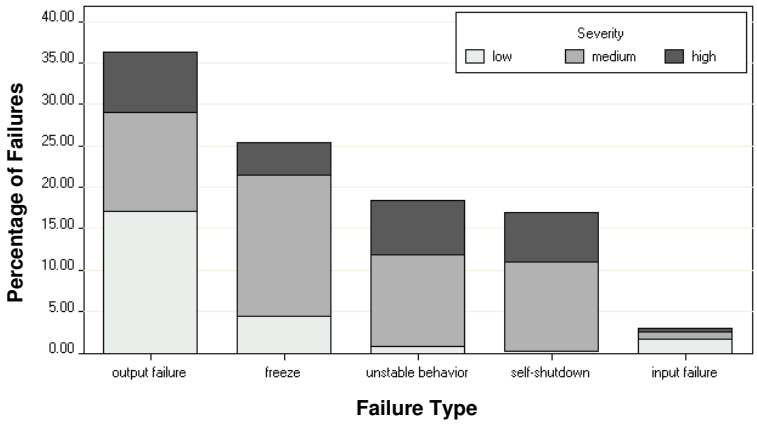


Figure 4.5: Mobile phones Failure types and severity

4.3.2 Results from the Reports' analysis

The results discussed in this section are obtained from the analysis of failure reports posted between January 2003 and June 2005. A total of 533 problems (466 user-reported and 67 known issues) were used in this study. Phone models from all the major vendors are present: Motorola, Nokia, Samsung, Sony-Ericcson, LG, besides Kyocera, Audiovox, HP, Blackbarry, Handspring, and Danger. It is worth noting that 22.3% of failure reports are from smart phones, although smart phones represent only 6.3% of the market share in 2005 (see Figure 4.1). We attribute this to the fact that smart phones (i) have more complex architecture than voice-centric or rich-experience mobile phones and (ii) are open for users to download and install third party applications and/or develop their own applications.

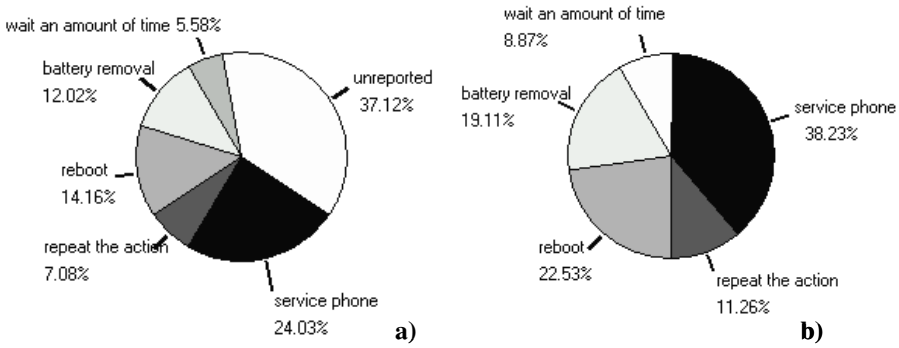


Figure 4.6: User initiated recovery: a) break-up including unreported recovery actions and b) break-up without unreported recovery actions; the numbers are percentages of the total number of failures

Analysis of Data on User-Reported Problems

The results discussed in this section are based on user-reported problems. Figure 4.5 depicts a bar chart of failure types and failure severity. The most frequent failure type is output failure (36.3%), followed by freeze (25.3%), self-shutdown (16.9%), unstable behavior (18.5%), and input failure (3.0%). Despite their high occurrence, output failures are usually of low-severity since repeating the action is often sufficient to restore a correct device operation. On the other hand, self-shutdown and unstable behavior can be considered as high-severity failures. The two categories contribute to 52.2% of the high-severity failures, although they represent only 35.4% of the total number of failures. Phone freezes are usually of medium severity, since reboot (2.4% of the total number of failures; see Table 4.2) or the battery removal (9.0%; see Table 4.2) usually do the job and reestablish the proper operation. Only in about 3.7% (see Table 4.2) of cases must the user seek assistance.

Figure 4.6 gives the break-up of user-initiated recovery actions. While the advance smart phones have a watch-dog, which can perform an auto-reboot of the phone, it does

Table 4.2: Failure frequency distribution with respect to failure types and recovery actions; the numbers are percentages of the total number of failures

	Recovery action					
Failure Type	**service phone**	**reboot**	**battery removal**	**wait an amount of time**	**repeat the action**	**unreported**
freeze	3.65	2.36	9.01	4.29	0	6.01
input failure	0.64	0.64	0.21	0	0.64	0.86
output failure	6.87	8.80	0.43	0.64	5.79	13.73
self-shutdown	6.65	0	2.15	0.43	0	7.73
unstable behavior	6.87	1.72	0.21	0.21	0.64	8.80

not seem to be effective. About 37% (see Figure 4.6a) of user reports do not indicate a recovery action. In order to better understand the distribution of recovery actions, Figure 4.6b depicts the break-up considering only entries with a reported recovery actions. This, given a recovery is attempted, shows that in 38.23% of the cases users bring the phone to be fixed by the service personnel. This could imply that often users are not able to reestablish correct operation of the device. The most common recovery action is a reboot of the device (22.53%), followed by battery removal (19.11%).

To gain an understanding of the relationship between failure types and recovery actions, Table 4.2 reports failure distribution with respect to failure types and corresponding recovery actions. From the recovery action perspective, one can see that reboots are an effective way to recover from output failures (8.8% of the total number of failures). This indicates that output failures are often due to a temporary bad state in the software, which is cleaned up by the reboot. Battery removal is required to fix phone freezes. From this one can infer that freezes are mainly due to a permanent bad state (e.g., invalid user settings) that can be cleaned up by removing the battery. Data in Table 4.2 show also that a significant number of freezes (4.29% of the total number of failures) are recovered by simply waiting an amount

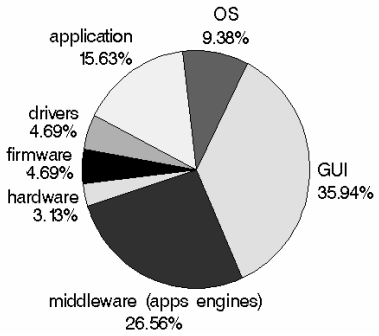


Figure 4.7: Break-up of failures per component; the numbers are percentages of the total number of failures

of time for the phone to respond. This may indicate that a certain fraction of battery removals and reboots in response to freezes are due to impatient users. Furthermore, it can be observed that self-shutdown and unstable-behavior failures usually require the phone to be serviced by a specialist. This once again confirms the high severity of this type of failure.

Analysis of Data on Known Issues

In contrast to the user-reported problems, data on known issues provides better explanations for failure causes and enable identifying components responsible for the failure. The break-up of failures per component is depicted in Figure 4.7. One can see that GUI and application engines (middleware) are serious dependability bottlenecks, together accounting for 62.5% of failures. Failures originating in applications themselves are responsible for about 15.63% of failures. Thus it can be desirable to provide application developers with programming tools/frameworks for fault/error management.

The data also provide insight into failure causes, which we divided into three categories:

- *Software errors/bugs*: The most common software bugs found in the reports include memory leaks, use of incorrect resources (such as wrongly written strings, or images converted with an erroneous bit depth), wrong data type casting operations, bad handling of indexes/pointers to objects such as phonebook entries, incorrect management of buffer sizes, and writing incomplete data to memory. These bugs are usually discovered by manufacturers (by keeping track of user complaints) and fixed by providing new firmware to be uploaded on devices. Insufficient testing is one of the primary reasons that these problems escape.

- *Resource exhaustion or interferences*: Two runtime conditions are often indicated as leading to failures: i) full internal secondary memory - the erasable programmable ROM used to store program binaries and ii) process interferences, e.g., race conditions when accessing shared resources such as operating system. Figure 4.7 indicates that OS-related problems account for 9.38% of failures. Consequently it might be beneficial to harden OS tasks responsible for memory and resource management.

- *Hardware and drivers of communication protocols*: A significant percentage of failures are caused by: (i) hardware glitches/faults (3.13%), often due to a physical damage, such as accidental drops or knocks. [70] indicates, for example, that a mechanical stress can cause failures of the interconnections within the phone; (ii) communication protocol errors (4.7%) due to flaws in communication drivers, e.g., Bluetooth or CDMA/GSM protocols. Often, as discussed in the next section, these failures manifest during Bluetooth utilization or during a CDMA/GSM search for the tower signal.

Table 4.3: Failure frequency distribution with respect to: a) running application, b) failure type and number of running applications; the numbers in the table are percentage of total number of failures

Running application(s) during the failure	Percentage of Failures
No active applications	46.14
Call	12.65
Using Bluetooth	3.65
Searching for signal	2.79
SMS Composition	2.79
SMS Receiving/reading	2.58
Playing with images	2.36
Browsing	1.93
mp3 listening + call	1.72
Gaming	1.50
Wallpaper set + camera	1.29
Using the phonebook	1.29
Other	20.47

a)

	Number of running apps		
Failure Type	0	1	2
freeze	6.22	17.17	1.93
input failure	1.29	1.72	0
output failure	16.52	18.03	1.72
self-shutdown	9.23	6.22	1.50
unstable behavior	12.88	4.72	0.86
all	46.14	47.85	6.01

b)

Failure Frequency versus Mobile Device Activity

Table 4.3a reports the percentage of failures with respect to the mobile device activity at the time of a failure. A significant fraction of failures (46.14%) manifests without any application running on the phone aside from OS tasks. Since most phones are based on well tested commodity hardware, we speculate that the problem is an OS and/or firmware (most likely the drivers).

Most of failures, while running applications, occur during a call (12.65%). This should be expected, since voice connection still remains a primary function of mobile phones. Relatively few failures manifest during Bluetooth utilization (3.65%). While this percentage is small (Bluetooth is not deployed on all phones), it may indicate a problem of potentially

greater significance in future phones, if we consider the results of the previous chapter. However, we do not have data on application usage, which would enable relating application failure rates with the application's usage.

Table 4.3b shows that the majority of failures occur while no applications or a single application executes on the device. As for failures that manifest while two (or more) applications are running on the device (the third column in table 4.3b), most often one of the running applications is either a search for the tower signal or a call (which account, respectively, for 33% and 52% of cases in which multiple applications are running during the failure). The second application is usually a data-driven application, e.g., mp3 player, Internet browser, or the ring tone setting. This indicates that interactions/interferences between the communication firmware and the runtime environment (for data driven applications) is one of the most common failure causes.

4.4 Data Collection Methodology

The results presented in previous section guided the definition of a data collection methodology for smart phones, based on the development of an ad-hoc logger application, to be deployed on actual phones. We concentrate on freeze and self-shutdown failures, since they are easy to detect, yet severe failure manifestations. Some unstable behavior failures, such as repeated self-reboots, can be captured as well. As for input and output failures, we do not pay attention to them for their less important severity and for the fact that the automatic detection of value failures would require the implementation of a perfect observer

which has a complete knowledge of the system specification [13].

The main objective of the logger is to detect and record the occurrences of freezes and self-shutdowns. Other than this, it is important to catch the status of the phone during the failure.

In the following subsection, the methodology is described according to the FFDA steps defined in section 2.2.

4.4.1 Smart phones Under Test

The system under test is composed of a set of 25 smart phones with different versions of the Symbian OS, from the 6.1 to the 8.0 and 9.0. The majority of them carries out the version 8.0, which was the most released one at the time the analysis started. The targeted phones belong to students, researchers and professors from both Italy and the USA. The phones had the logger installed on them and were normally used from users during the collection period (hence, an idle workload is supposed to be adopted).

4.4.2 Failure Data Logging and Collection

In this section, the architecture of the logger, and the related collection infrastructure, is presented. The content is extracted from a previously published work, devoted to the design of the logger [3].

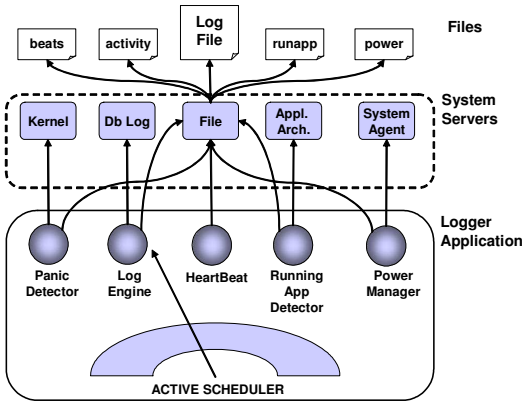


Figure 4.8: Overall architecture of the Logger for Symbian OS smart phones

Logger High Level Architecture

We designed a logger application as a set of AOs, each one responsible of a particular task. The logger architecture is shown in figure 4.8. Each AO interacts with a particular OS server to perform its task, and all AOs use File Server facilities to store their data. The logger is conceived as a daemon application that starts at the phone start-up and that executes in background. The AOs building the logger are detailed in the following.

- *Heartbeat*: it is in charge of detecting both freezes and self-shutdowns. More details about the heartbeat technique can be found in the next subsection.

- *Running Applications Detector*: In order to be conscious of the phone status during the failure, this AO periodically stores on the *runapps* file the list of IDs of all the applications running on the phone. The list is obtained by requesting it to the Application Architecture Server. This way, the applications running during a failure can be pinpointed.

- *Log Engine*: it is responsible to collect the smart phone activity (e.g. calls, messages, and browsing). The information is gathered from the Database Log Server, which logs phone's activity, and it is stored into the *activity* file. This allows to gather more information on the phone status.

- *Power Manager*: it provides information about the battery status, in order to distinguish self-shutdowns due to failures from those due to low battery. The battery status is gathered from the System Agent Server, and it is stored into the *power* file.

- *Panic Detector*: collecting panics as soon as they are launched is one of the main objectives of the logger. It allows to identify the underlying causes of a self-shutdown or freeze. In order to gather panics and the related information (e.g. panic category and type), the Panic Detector exploits the services offered by the RDebug object offered by the Symbian OS Kernel Server. In particular, the Panic Detector registers to the RDebug's `getException` API so as to be notified whit the panic information whenever a panic occurs on the phone.

Other than collecting panics, the Panic Detector is also responsible of putting all the information produced by the other components together into one *Log File*. This operation is performed either when a panic is detected or when the logger application starts (i.e., when the phone starts). A drawback of the logger is that it cannot store data about File Server's failures. Nevertheless, this cannot be avoided in that there is no way to permanently store any information when the File Server fails.

Detecting Freezes and self-shutdowns

The Heartbeat AO periodically writes a heartbeat item on the *beats* file. The item is composed of a time-stamp and a status info, i.e. ALIVE, REBOOT, MAOFF, and LOWBT. During normal execution, the Heartbeat writes an ALIVE item. When a shutdown is performed either by the user or automatically undertaken by the kernel, the Heartbeat writes a REBOOT item, since it is capable to capture the phone shutdown event. It is worth to mention that when the phone is rebooted the OS leaves a certain time to applications to complete their tasks. This time is sufficient for the Heartbeat to write the REBOOT item. A user initiated turn off of the whole logger application causes instead a MAOFF (Manual OFF) item to be written. Finally, if a shutdown is due to low battery (the information about the battery status is requested to the Power Manager), a LOWBT (LOW BaTtery) item is written.

When the phone is turned on and the logger starts, the Panic Detector checks the last written item by the Heartbeat. When an ALIVE is found, the phone has been shut down by pulling out the battery. In all other cases (i.e., a shutdown due to low battery, user, or kernel) the Heartbeat would have written REBOOT or LOWBT. This means that the phone was frozen, coherently with the fact that pulling out the battery is the only reasonable user-initiated recovery action for a freeze. Therefore, a freeze is registered by the Panic Detector, along with the information gathered by the Log Engine and the Running Applications Detector.

On the other hand, a REBOOT can be found for three reasons. First, the phone rebooted

itself. Second, it was rebooted by the user to recover a failure (e.g., output failure). Third, it was regularly shut down. Hence, the problem of distinguishing this three cases arises. Unfortunately, we are not able to systematically distinguish phone induced reboots from manual ones, because the generated event i.e., the one captured by the heartbeat, is the same in both the cases. However, they can be distinguished by looking at the off time of the phone, or reboot duration. It is reasonable to state that:

$$T_{SS} < T_{MS} \tag{4.1}$$

where T_{SS} is the duration of a self reboot and T_{MS} is the duration of a manual shutdown. In other terms, the duration of a shutdown (e.g., when the phone is shut down over the night) is greater than the duration of a self-shutdown. A manual reboot requires at least the user to press the on button, which generally requires more time than a self-shutdown. The Panic Detector registers a self-shutdown event and its duration. This way, the reboot duration can be analyzed a posteriori.

Choice of the Heartbeat Frequency

The hertbeat frequency f_h is a logger's crucial parameter since it determines the time granularity at which the above mentioned durations, i.e., T_{SS}, and T_{MS}, can be measured. It would thus be desirable to choose an arbitrary big f_h to increase the precision of the measurements. However, this is not possible for two practical reasons: i) the battery consumption induced by the logger increases as f_h increases, and ii) the heartbeat precision decreases as f_h increases, as will be shown later by our experiment. The heartbeat precision

can be defined as:

$$precision = \frac{T_h}{T_h + \Delta_w} \ , \ T_h = \frac{1}{f_h} \tag{4.2}$$

where Δ_w is the write delay induced by the File Server. In other terms, once fixed a heartbeat frequency f_h, and thus a heartbeat period T_h, the heartbeat items will not be written exactly each T_h, but they will be written each T_h plus the time Δ_w needed to invoke the File Server, transfer to it the information to write, access the file, and actually write it. The bigger is f_h, the smaller is the precision, because as T_h decreases, it becomes comparable with Δ_w. Moreover, as T_h decreases, Δ_w increases because the File Server starts to be overloaded with requests. This is confirmed by our experiment.

We evaluated the average Δ_w achieved as a function of f_h, with different workloads running on the phone: SMS, phone call, video call, listening of an audio clip, and Bluetooth file transfer. We also performed measurements with a "stand-by workload", i.e., when the phone is in stand-by mode. The measurements were performed on two different Symbian smart phones: Nokia 6680 and Motorola A1000. For each fixed f_h, we run the heartbeat AOs concurrently with one of the mentioned workloads. As an effect of the writing delay, the timestamps on the *beats* file are written with a period $T_h + \Delta_w$. Hence, from the timestamps we can evaluate the average Δ_w.

Figures 4.9 and 4.10 shows the results of the experiment. As we expect, the average Δ_w is an increasing function of f_h, independently from the workload. The case of idle workload (figure 4.9) shows that a $f_h = 2Hz$ ($T_h = 0.5s.$) is a physical upper bound after which Δ_w starts to increase almost exponentially. For this reason, the experiments with the other

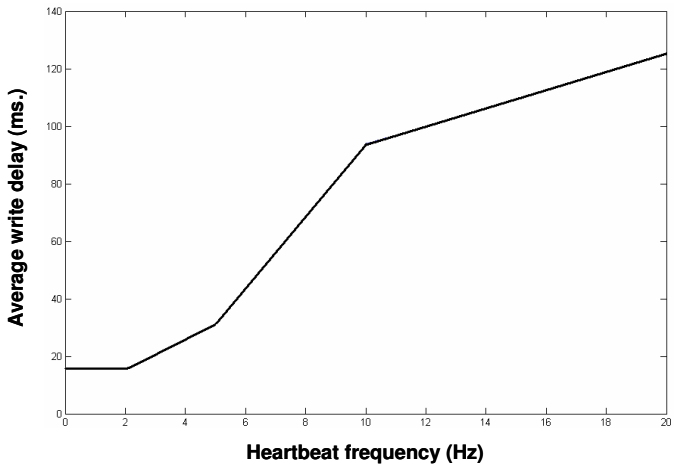


Figure 4.9: Write delay Δ_w as a function of f_h, stand-by workload

workloads have been run with f_h ranging from 0.1 Hz to 2 Hz. Figure 4.10 shows that the most critical application is the video call. This could be expected as the video call use a wide range of phone's resources. From figure 4.10 one could conclude that the best choice to meet precision requirements would be $f_h = 0.1Hz$ or even lower. On the other hand, very low frequencies affect the time granularity. As a compromise we chose $f_h = 0.33Hz$ ($T_h = 3s.$), that is, the point after which the precision becomes unacceptable if a video call is performed.

Distributed Data Collection Architecture

In order to collect the Log Files of all the monitored phones, we developed a data collection architecture for Log Files gathering. The architecture is 3-tier and it is depicted in figure 4.11.

The first tier is the phone. In particular, we developed a Java midlet for the phone using the Java 2 Micro Edition technology. The logger requests the user to send the Log File when it

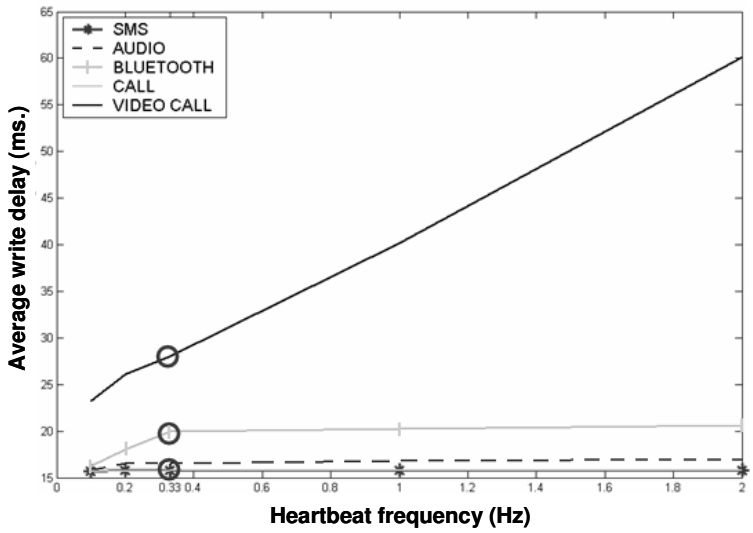


Figure 4.10: Write delay Δ_w as a function of f_h, other workloads

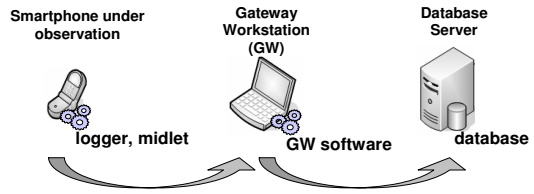


Figure 4.11: Distributed Data Collection Architecture

reaches a certain size. When the user is ready, the midlet can be used to send the Log File to the tier 2, the Gateway Workstation (GW), via a Bluetooth connection. However, if the user's phone or GW does not provide Bluetooth connection facilities, he or she can avoid to use the midlet and can transfer the file via the serial cable usually used to synchronize the phone with a computer.

The GW (tier 2) is a user's computer connected to the Internet. It runs our software to receive the Log File via Bluetooth, and to send it to our Database node (tier 3) using the Internet. To do so, the user must authenticate himself/herself to the Database node. Again,

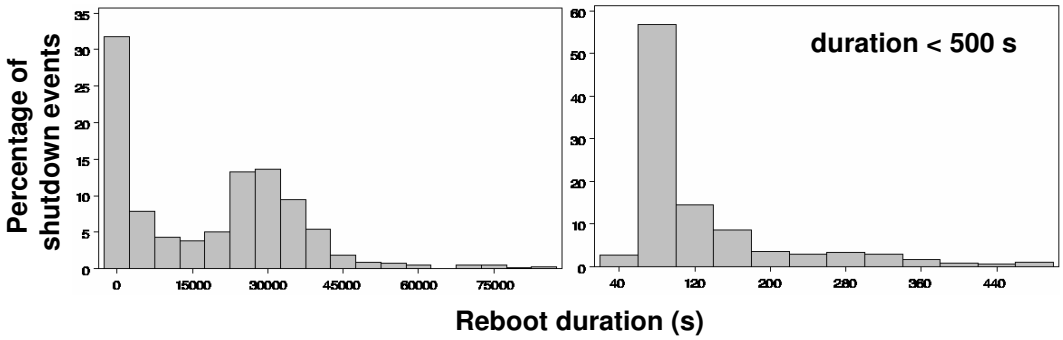


Figure 4.12: Distribution of reboot durations. The right-side histogram zooms the left-side one for durations less than 500 seconds.

if Bluetooth connections are not available, the GW allows the user to select the Log File to send from his/her computer's file system.

Finally, the tier 3 stores the received files on a centralized database, after checking the Log File format. The data collected on the database can then be used to perform the analysis.

4.4.3 Data Filtering and Manipulation

As a first step to be performed, it is necessary to filter out regular shutdown events that are normally triggered by users, thus isolating self-shutdown events, according to equation 4.1. Unfortunately, it is not possible to automatically distinguish between the two, since the generated event i.e., the one captured by the Heartbeat AO, is the same in both cases. However, they can be discriminated by looking at the off time of the phone, or reboot duration, which is registered by the Panic Detector.

Figure 4.12 shows the distribution of reboot durations. The histogram on the left-side considers all the registered shutdown events (1778 events). From the histogram, two peak bars can be noticed: a first one close to the origin, which should contain all self-shutdown

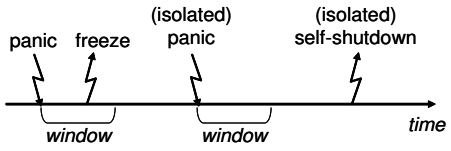


Figure 4.13: Panics and HL events coalescence scheme

durations, and a second one around 30000 seconds (about eight hours and 20 minutes), corresponding to the off time over the nights (users usually turn off their phones at night time). The histogram on the right-side is a zoom on the first peak, for durations less than 500 seconds. It clearly evidences a peak around 80 seconds, which corresponds to the median self-shutdown duration. Also, the number of events slowly approaches to 0 for durations bigger than 360 seconds. Based on these data, we filtered-out all shutdown events with a duration bigger than 360 seconds. The remaining events are assumed to be self-shutdown events. In particular, 471 such events (the 24.2% of the overall sample) are identified.

From the collected data, it is possible to infer the relationship between panics and high level (HL) events (i.e., freeze and self-shutdown). To do so, it is necessary to coalesce panic events with freeze and self-shutdown events, similarly to the merge and coalesce scheme already defined for Bluetooth. The adopted scheme is depicted in figure 4.13. When a panic is found in the `Log File`, freeze and self-shutdown events registered on the same phone after the panic are searched, within a temporal window. The figure evidences that there can be panics which do not relate to HL events, as well as isolated HL events. Careful attention must be payed to the tuning of the temporal window. An analysis of the data in our possess evidences that the number of coalesced events increase significantly for window's sizes up to five minutes. After this size, we must wait for relative big sizes (around one hour) to

appreciate a further, slight increase of this number of coalesced events, meaning that the window starts to collapse events which are most probably uncorrelated. Hence, we choose a window size equals to five minutes.

4.5 Key Findings

This section reports the results of a 14 months collection campaign conducted on the previously mentioned 25 Symbian smart phones. A total number of 360 freezes and 471 (filtered) self-shutdown HL events have been collected. As for panics, the logger captured 415 panic events, belonging to several different categories and types.

4.5.1 Freeze and Self-shutdown Measurements

The Mean Time Between Freezes (MTBFr) and the Mean Time Between Self-shutdowns (MTBS) have been evaluated, averaged for each phone. It results: MTBFr = 313 hours, and MTBS = 250 hours. Hence, a user experiences a Freeze about every 13 days, and a self-shutdown about every 10 days, on average. These figures give an overall idea of today's mobile phones user-perceived dependability. While these values are acceptable for everyday dependability requirements [93], they evidence the limits for the application of smart phones in critical applications.

Table 4.4: Collected panics typologies, descriptions, and frequencies

Panic	Type	%	Meaning
KERN-EXEC	0	6.31	This panic is raised when the Kernel Executive cannot find an object in the object index for the current process or current thread using the specified object index number (the raw handle number).
	3	56.31	This panic is raised when an unhandled exception occurs. Exceptions have many causes, but the most common are access violations caused, for example, by dreferencing NULL. Among other possible causes are: general protection faults, executing an invalid instruction, alignment checks, etc.
	15	0.51	This panic is raised when a timer event is requested from an asynchronous timer service, an RTimer, and a timer event is already outstanding. It is caused by calling either the At(), After() or Lock() member functions after a previous call to any of these functions but before the timer event requested by those functions has completed.
E32USER-CBase	33	5.56	Raised by the destructor of a CObject. It is caused, if an attempt is made to delete the CObject when the reference count is not zero.
	46	0.76	This panic is raised by an active scheduler, a CActiveScheduler. It is caused by a stray signal.
	47	0.25	This panic is raised by the Error() virtual member function of an active scheduler, a CActiveScheduler. This function is called when an active object's RunL() function leaves. Applications always replace the Error() function in a class derived from CActiveScheduler; the default behaviour provided by CActiveScheduler raises this panic.
	69	10.10	This panic is raised if no trap handler has been installed. In practice, this occurs if CTrapCleanup::New() has not been called before using the cleanup stack.
	91	0.51	Not documented
	92	0.76	Not documented
USER	10	1.52	This panic is raised when the position value passed to a 16-bit variant descriptor member function is out of bounds. It may be raised by the Left(), Right(), Mid(), Insert(), Delete() and Replace() member functions of TDes16.
	11	5.81	This panic is raised when any operation that moves or copies data to a 16-bit variant descriptor, causes the length of that descriptor to exceed its maximum length. It may be caused by any of the copying, appending or formatting member functions and, specifically, by the Insert(), Replace(), Fill(), Fillz() and ZeroTerminate() descriptor member functions. It can also be caused by the SetLength() function.
	70	0.76	This panic is raised when attempting to complete a client/server request and the RMessagePtr is null.
KERN-SVR	0	0.25	This panic is raised by the Kernel Server when it attempts to close a Kernel object in response to an RHandleBase::Close() request. The panic occurs when the object represented by the handle cannot be found. The panic is also raised by the Kernel Server when it cannot find an object in the object index for the current process or current thread using the specified object index number (the raw handle number). The most likely cause is a corrupt handle.
ViewSrv	11	2.53	occurs when one active object's event handler monopolizes the thread's active scheduler loop and the application's ViewSrv active object cannot respond in time (the View Server monitors applications for activity/inactivity, if it thinks the application is in some kind of infinite loop state it will close it. Clever use of Active Objects should help overcome this).
EIKON-LISTBOX	3	0.25	occurs when using a listbox object from the eikon framework and no view is defined to display the object.
	5	0.76	occurs when using a listbox object from the eikon framework and an invalid Current Item Index is specified.
Phone.app	2	0.25	Not documented
EIKCOCTL	70	0.25	Corrupt edwin state for inlining editing
MSGS Client	3	6.31	Failed to write data into asynchronous call descriptor to be passed back to client
MMFAudioClient	4	0.25	it appears when the TInt value passed to SetVolume(TInt) gets 10 or more

4.5.2 Captured Panics

Table 4.4 reports the list of all the encountered panics during the observation period. The table contains panic categories, panic types, the percentage of their occurrence with respect to the total number of panics, and the description of the panic, mostly extracted by the Symbian OS documentation. The meaning of all encountered panics gives an overall picture of the software defects conducting to failures. Among them, the most frequent ones are

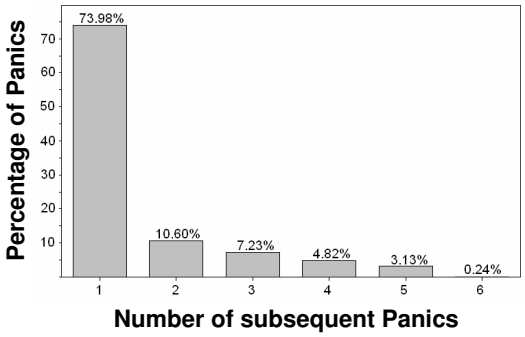


Figure 4.14: Distribution of panics registered as a cascade

access violations caused by deferencing null, which cause the Symbian kernel executive to kill the responsible application and to signal a KERN-EXEC type 3 panic. However this panic type includes other causes such as general protection faults, executing an invalid instruction, and alignment checks. Other frequent causes are invalid object indexes (KERN-EXEC type 0 panic), runtime errors related to the heap management (causing E32User-CBase panics), and copy operations causing a descriptor to exceed its maximum length (USER type 11 panic). It is interesting to notice that these results are coherent with what was found on web forums.

In some cases, more than one panic can be registered as a cascade, as evidenced in figure 4.14. Since panicking is the last operation an application or system module performs (just after, the application is killed by the kernel), this proofs the existence of error propagation phenomena in the OS, despite its micro-kernel architecture.

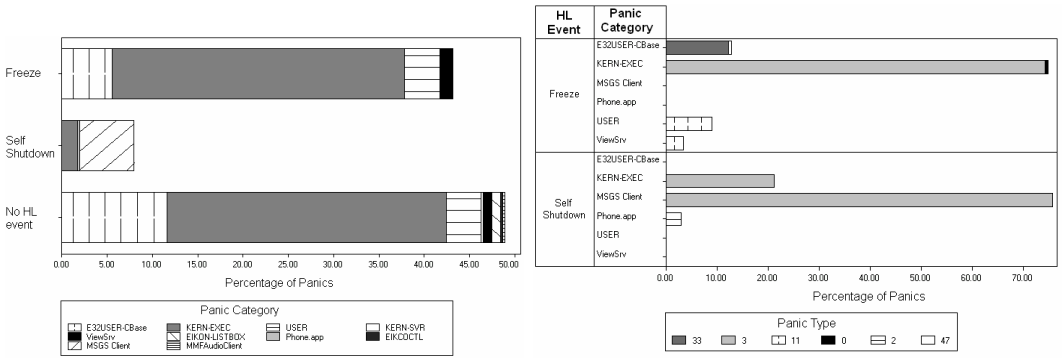


Figure 4.15: Panics and HL events: a) overall summary, b) details with respect to freeze and self-shutdown events

4.5.3 Panics and High Level Events

Figure 4.15 shows the results of the previously defined coalescence activity. In particular, figure 4.15a gives an overall summary, reporting also the distribution of isolated panics, i.e., those panics which are not related to any HL event. Perhaps these panics cause output failures. However, our logger is not able to intercept this kind of failures.

A first important evidence is that more than a half of the registered panics (51%) relate to HL events. If we consider the fewness of HL events (about one every 11 days), these relationships cannot be just a coincidence. As a further confirmation, if we include all the shutdown events in the analysis (hence a 300% increase of the number of events, from 471 to 1778 shutdown events), the percentage of panics that relate to HL events gets only a 4% increase, to 55%. This also reinforce the results given in section 4.4.3: the shutdown events which are filtered out are regular user-triggered shutdowns.

From figure 4.15a, we can observe that there is a set of panic categories which is never related to HL events, such as EIKON-LISTBOX, EIKCOCTL, MMFAudioClient, and KERN-SVR.

The first three panics are typical application panics, related to the view or the audio streaming. This indicates a good resilience of the OS with respect to application panics. More frequent system panics, such as KERN-EXEC, E32USER-Cbase, USER and ViewSrv, can either cause an HL event or not. In our opinion, this depends on the component that caused the panic: if it is an indispensable system server, its death will then cause a whole phone crash. Otherwise, if it is an application, once it is killed by the kernel, the phone keeps working properly. As a further observation, there have been panics, such as the Phone.app and MSGS Client, which always cause a self-shutdown. We recall that the kernel might decide to reboot the phone in response to some panics and to the component that provoked them. This means that these two panics were always caused by important system modules or applications. For example, Phone.app is an always running system application, representing the phone itself.

Figure 4.15b details the relationship between panics, freezes and self-shutdowns, by reporting also the panic type. As already observed, there are panics which only relate to self-shutdown events, such as Phone.app and MSGS Client. In the same way, we can isolate panics that are the potential symptoms of freezes, such as heap management (E32USER-Cbase), USER, and ViewSrv, and KERN-EXEC type 0 panics. On the other hand, access violation-related panics (KERN-EXEC type 3) can cause both the anomalous HL behaviors.

Table 4.5: Relationship between panics and the phone activity

categ. / act. \ type	E32USER-CBase 33	E32USER-CBase 47	KERN-EXEC 0	KERN-EXEC 3	MSGS Client 3	Phone.app 2	USER 11	View Srv 11	All
message	1.10	.	.	4.41	.	1.10	.	.	6.62
Voice call	6.62	1.10	.	17.3	.	.	9.56	4.04	38.6
unspecified	4.78	.	0.37	40.4	9.19	.	.	.	54.8

4.5.4 Phone Activity at Panic Time

As a further result, we investigate on the phone's activity, in terms of user activity and running applications, at the time a panic is registered. Using the same coalescence methodology already mentioned in previous section, table 4.5 reports the activity that resulted to be performed by the user at the time of the panic, in terms of voice calls and text messages (the only ones registered on the Database Log Server). Only the panics which are related to an HL event are considered in this analysis. Interestingly, about the 45% of panics are registered while the user were performing real-time activities, such as a voice call, or sending/receiving a short message. This confirms what we observed on web forums, and evidences the presence of interferences between normal applications/system modules, and real-time threads concurrently running on the phone. In other terms, this is a symptom of the lack of isolation between real-time modules and time-sharing, interactive modules. Thus, more effort should be directed to the enforcement of the isolation between the two system modules. Also, there are panics, such as USER and ViewSrv, that show-up only while a voice call is performed. Equally, there are panics, such as Phone.app, appearing only while a short message is sent/received. Hence, these particular panics are more likely than the other ones to be due to the presence of real-time operations.

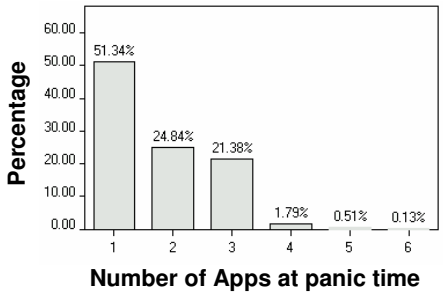


Figure 4.16: Distribution of the number of running applications at panic time

Table 4.6: Relationship between panics ans running applications

HL event	Panic category	Application: Messages	Messages Log	Camera Log Telephone	Log	Clock	Log Telephone	BT_Browser Log Teleph.	Log Contacts	Contacts	battery	Messages Contacts	Telephone	FExplorer	Clock Log	TomTom
Freeze	KERN-EXEC	0.51	.	.	3.20	3.20	.	.	1.02	1.28	.	.	.	0.28	0.90	1.02
Self-Shutdown	KERN-EXEC	.	.	.	.	.	.	.	.	.	.	.	.	0.18	.	.
	MSGS Client	.	.	6.39	.	.	3.20	.	.	.	.	.	.	.	.	.
No HL event	E32USER-CBase	0.38	6.39	.	.	.	.	.	0.26	.	.	.	0.26	.	.	.
	EIKCOCTL	.	.	.	.	.	.	.	0.13	.	.	.	.	.	.	.
	EIKON-LISTBOX	.	.	0.26	.	.	.	.	.	.	.	.	.	.	.	.
	KERN-EXEC	6.78	0.26	.	1.66	1.28	.	.	1.02	1.15	2.56	1.53	1.28	0.89	0.38	0.26
	USER	.	.	.	.	.	.	3.07	.	0.38	.	.	.		.	.
	ViewSrv	.	.	0.13	.	.	0.13	.	.	.	.	.	.		.	.
Total		8.18	6.91	6.78	5.50	4.48	3.32	3.07	3.07	2.94	2.56	1.53	1.53	1.35	1.28	1.28

As far as applications are concerned, the Running Application Detector allowed as to collect the set of running application at the time of the panic. It is interesting to notice that often only one application is found to be running at panic time, other than always running system applications, as can be observed by figure 4.16. Hence, as opposite from intuition, running more applications at the same time does not cause more panics.

Table 4.6 gives a summary of the more frequent sets of applications that were found in correspondence to panics. Only the sets with significant percentage were taken into account, covering 53% of the total number of panics. The rows reports panic categories organized with respect to the HL event they cause, in order to have an overall picture of

what manifestation we can expect from a certain set of applications. On the column, the sets of applications that were coalesced with the panic events are reported. The numbers reported into every cell of the table are percentages of the total number of panics, e.g., the Clock application has been found in correspondence of the 3.2% of panics, and, in particular, of KERN-EXEC panics which cause the phone to freeze.

Coherently with what was found on web forums, the Message application is one of the main responsible of panics. Also, the Bluetoth browsing tool is present, as pointed out by users on the forums. Other potential dependability bottlenecks are the camera, and the log of incoming/outcoming calls. The table also give some insights on the running applications which, even panicking, do not cause HL events. Perhaps, in these cases, only output failures are manifested to the user.

Conclusions

This dissertation addressed dependability issues of mobile distributed systems. Due to the spread use of these systems into our everyday life, the field failure data analysis approach has been adopted. This approach represents indeed a viable methodology to evaluate qualitative and quantitative dependability aspects of operational systems.

Two particular technologies had been chosen as representative of a wider class of systems. The Bluetooth wireless technology and smart phone devices equipped with the Symbian OS.

The research activity dealt with fundamental issues which arise when conducting FFDA studies on these kinds of systems, by proposing novel solutions or by enriching existing techniques. In particular, a multi-source approach has been used for the Bluetooth campaign to monitor the thorough behavior of the protocol stack. Such an approach enables to vertically investigate failure causes, to the extent of providing masking strategies, thus improving the overall dependability level. Moreover, automated workloads have been developed and deployed on real systems to address the inability of studying this kind of systems under idle workloads, i.e., the normal load at which they operate. As for the mobile phones campaign, we started from the failure information which is available on the web. Although unstructured, this information allowed to have a first understanding on the failures phenomenology,

which in turn lead to the definition of a specific failure logger application. The logger has been demonstrated to be an efficient mean to capture more structured failure data, which permitted the measurement of failure times along with the investigation of related causes.

In both cases, it is clear how FFDA can be successfully applied to mobile distributed systems, and how it keeps representing an effective way to gather a substantial understanding on existing systems' dependability characteristics. This understanding can thus be used to improve future instances of mobile distributed systems, towards the goal of dependable ubiquitous computing.

Other than providing specific considerations on both the presented case studies, this chapter summarizes the general lessons which have been learned from this three years experience and that can be reasonably taken into account when performing future FFDA studies.

Lessons Learned

On the Use of a Multi-source Approach

Existing literature on the FFDA field mainly considered field data originated from one source, i.e., system event logs or maintenance staff failure reports. Conversely, both the conducted studies adopted a multi-source approach for the data collection. Such a multi-source approach provides deep insights into the failure phenomenology. In addition, it allows for a vertical investigation of the failure occurrences, thus enabling the identification of the "chain of threats" from faults to errors and failures. For instance, the "bind failed" failure on Bluetooth PANs was discovered to be an heisenbug with particular underlying

activation conditions. We then completely masked the failure by preventing such conditions from occurring.

As a final remark, using more data sources enables for an higher number of failure data items to be collected in the time unit, thus improving the density of the campaign.

On the Use of Multiple Automated Workloads

As observed in section 2.5, the majority of FFDA works adopt idle workload, since the aim of FFDA itself is to characterize the dependability of a system under its normal conditions. The only works adopting automated workloads are concerned with the Internet, since its spot usage does not permit to evaluate continuous time dependability measures. The same consideration applies to the Bluetooth campaign presented in chapter 3. However, a a major issue to be addressed by "field data researchers" is the choose of what can be the appropriate workload to deploy. Indeed, as observed in section 2.4.1, there is a strong correlation between the system's load and the failing behavior of a system.

With respect to the Bluetooth campaign, we have seen how the use of two different workloads can provide a more thorough view of dependability aspects, thus fostering more considerations and avoiding the risk of obtaining results which are biased from the particular adopted workload. The use of a more stressful workload, such as the random one, permits to activate failure modes more quickly, giving statistical significance to collected data in a shorter period. It also permits to identify usage patterns which have to be avoided to develop more robust applications. On the other hand, the use of a realistic workload

cannot be neglected, since it allows to characterize the dependability behavior of the system under study when used for everyday applications.

Bluetooth Campaign Specific Considerations

Presented results have shown how failure data provide helpful insights to design fault tolerance means for operational systems. Respectively, up to 168% and 569% availability and reliabilty improvements have been demonstrated with respect to the random workload, and up to 9% and 128% improvements in the case of the realistic workload. Several lessons have also been learned about preferable usage patterns, from a dependability perspective. Examples are to avoid caching by performing the SDP search before the PAN connection, and to increase the timeout in the switch role API.

L2CAP and BNEP protocols do not perform error checks and flow controls since the Baseband channel is assumed, by Bluetooth designers, to be reliable. However, the analysis evidenced that often hard payload corruptions propagate to L2CAP and BNEP errors, due to corruptions of the L2CAP or BNEP headers. A valid improvement would derive from the use of even the simplest error check strategy, such as parity bits or low-overhaed CRC schemes.

Based on this experience, an enhanced version of the Linux BlueZ BT protocol stack has been successfully submitted, which includes all the findings we gathered from the analysis, and that developers can use for building more robust BT applications.

Mobile Phones Campaign Specific Considerations

The conducted campaign produced several interesting results. For instance, the results show that the majority of Kernel exceptions are due to memory access violation errors (56%), and heap management problems (18%), despite the Symbian OS design goals, i.e., the adoption of a micro-kernel model and the provision of advanced memory management facilities.

There is also an evidence of failure propagation between multiple applications, due to the uncovering of cascaded panic events. It is interesting to notice how the results are often coherent with the information gathered from publicly available web forums. The forums indeed pinpointed memory leaks as one of the main causes of failures. In the same way, both the forums and the logger-based analysis evidence that the majority of failures manifest while the user is performing real-time tasks, such as a voice call or the sending/receiving of a text message. This suggests to strength the isolation between interactive and real-time tasks.

From the user-perceived dependability point of view, the analysis shows that users experience a failure every 11 days, on average, which manifest in the form of freeze or self shutdown.

Although the effort made to enforce proper heap management via trap-leave and clean-up stack facilities, there is a non negligible amount of panics (E32User-Cbase panics, 18%) related to the misuse of heap management mechanisms, such as the absence of a trap handler (type 69) and the attempt to delete an object with a non-zero reference counter (type 33). The severity of such panics cannot be understimated, since often they lead to the freeze

of the phone. Since this especially applies to E32User-Cbase type 33 panics, more effort should be dedicated to strength the runtime support so as to mask the error and avoid the panic.

KERN-EXEC type 3 panic are the more frequent ones, and are the unique panics which may lead to both freeze and self-shutdown. This is probably due to the fact that such a panic is pretty generic and embrace a wide set of problems, such as access violations, general protection failures, executing an invalid instruction, alignment checks, etc. A deeper comprehension could be derived if this panic was split into multiple types by designers.

Finally, it is worth observing that there are freezes and self-shutdowns which remain isolated. This means that not all the anomalous phone activity is properly managed by panicking, hence applications and system modules may be wrongly programmed in a way that conduct them to crash without signaling any panic.

FFDA: Towards a Unified View

During this three years experience matured on FFDA issues and practice it has been recognized the need for the assessment of a more comprehensive methodology for FFDA. As it has been observed in section 2.5, the wider scope of the FFDA research, dealing with more and more different types of systems, is not accompanied by novel methodological achievements. In fact, all the related works recently published are based on results and approaches which have been proposed between 80s and 90s. In addition, while there are several different studies targeting the same system, the results are difficult to generalize because both the

environmental conditions and the data collection methodology may differ.

The definition of a unified FFDA methodology would instead enable a consistent comparison of results and conclusions from studies conducted from different actors (both academic or industrial) or performed on different instances of the same target system. This would in turn facilitate the communication within the FFDA research community, thus achieving more credible results.

A valid starting point for the definition of such methodology is represented by the framework defined in section 2.3. It provides a mean for comparing different studies and it suggests to researches which is the relevant methodology-related and quantitative information that they should evidence into their work. In addition, such a methodology should be capable of drawing guidelines and best practices about i) which kind of sources of failure data should be adopted for a given class of systems; ii) which kind of workloads should be deployed; and iii) which type of collection infrastructure should be implemented. Finally, a particular effort should be devoted towards the recognition of a common structure of the collected field failure data, i.e., one of the formats adopted by already proposed tools for FFDA (see section 2.2). This would let data be always stored in the same format, thus letting more research teams to conduct different analysis on the same data set, and then compare the results.

The definition of data sets conforming to a common format, along with the creation of publicly available failure data repositories would also enable more exchange between the academy and the industry.

This need is being more and more recognized within the dependability research community. The work in [90], published in the current year, is a first example of a study conducted on a large and publicly available failure data source. Authors conclude with the hope that this data might serve as a first step towards a public data repository, thus encouraging efforts at other sites to collect and clear data for public release. Even in this dissertation, we recognized the importance of analyzing public failure data, i.e., the data from web forums, although unstructured. The same approach has also been followed by other studies [32]. All this ferment has been recently discussed in the context of an interesting panel, named "In Search of Real Data on Faults, Errors and Failures", and held at the sixth European Dependable Computing Conference (EDCC-6, Coimbra - Portugal, October 2006). Panelists evidenced that, even if industries demonstrate great interest in the FFDA research field, they often hide collected failure data due to strategic reasons that can compromise the market view [31]. The new approach would instead encourage industries to publish their data into the novel, anonymous format. Then, the adherence to the common format and the existence of public repositories, would assure the data to be analyzed by third party actors.

Bibliography

[1] A.Avizienis, J.C. Laprie, B.Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, January-March 2004.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communication Magazine*, pages 102–114, August 2002.

[3] P. Ascione, M. Cinque, and D. Cotroneo. Automated Logging of Mobile Phones Failure Data. *Proc. of the 9th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC 2006)*, April 2006.

[4] V. Astarita and M. Florian. The use of Mobile Phones in Traffic Management and Control. *Proc. of the 2001 IEEE Intelligent Transportation Systems Conference*, August 2001.

[5] A. Avizienis and J.P.J. Kelly. Fault Tolerance by Design Diversity: Concepts and Experiments. *IEEE Computer*, 17(8):67–80, August 1984.

[6] D. Avresky, J. Arlat, J.C. Laprie, and Y. Crouzet. Fault Injection for Formal Testing of Fault Tolerance. *IEEE Transactions on Reliability*, 45(3):443–455, September 1996.

[7] A. A. Aziz and R. Besar. Application of Mobile Phone in Medical Image Transmission. *Proc. of the 4th National Conference on Telecommunication Technology*, January 2003.

[8] S. Baatz, M. Frank, R. Gopffarth, D. Kassatkine, P. Martini, M. Scheteilg, and A. Vilavaara. Handoff support for mobility with IP over Bluetooth. *Proc. of the 25th Annual IEEE Conf. on Local Computer Networks (LCN 2000)*, November 2000.

[9] J. E. Bardram. Applications of context-aware computing in hospital work-examples and design principles. *Proc. of the 19th ACM Symposium on Applied Computing (SAC 2004)*, March 2004.

[10] C. Bettstetter and C. Hartmann. Connectivity of Wireless Multihop Networks in a Shadow Fading Environment. *The Sixth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, September 2003.

[11] Bluetooth SIG. *Personal Area Networking Profile*, 2001.

[12] Bluetooth SIG. *Specification of the Bluetooth System - core and profiles v. 1.1*, 2001.

[13] A. Bondavalli and L. Simoncini. Failures Classification with Respect to Detection. *Proc. of the 2nd IEEE Workshop on Future Trends in Distributed Computing Systems*, 1990.

[14] P. Bracchi and V. Cortellessa. A framework to Model and Analyze the Performability of Mobile Software Systems. *The fourth ACM SIGSOFT International Workshop on Software and Performance*, January 2004.

[15] M. F. Buckley and D. P. Siewiorek. A Comparative Analysis of Event Tupling Schemes. *proc. of The 26th IEEE International Symposium on Fault-Tolerant Computer Systems (FTCS '96)*, June 1996.

[16] M.F. Buckley and D.P. Siewiorek. VAX/VMS Event Monitoring and Analysis. *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing (FTCS-25)*, June 1995.

[17] S. Cabuk, N.Mahlotra, L. Lin, S. Bagchi, and N. Shroff. Analysis and Evaluation of Topological and Application Characteristics of Unreliable Mobile Wireless Ad-hoc Network. *Proc. of 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, December 2004.

[18] J. Carlson and R.R. Murphy. Failure Classification and Analysis of the Java Virtual Machine. *Proc. of the 2003 IEEE International Conference on Robotics and Automation (ICRA'03)*, September 2003.

[19] G. Carrozza, M. Cinque, F. Cornevilli, D. Cotroneo, C. Pirro, and S. Russo. Architecting a Realistic Workload for Bluetooth PANs Stressing. TR-WEBMINDS-58, University of Naples Federico II, web-minds.consorzio-cini.it, November 2005.

[20] X. Castillo and D. P. Siewiorek. A Performance-Reliability Model for Computing Systems. *Proceedings of the 10th IEEE Symposium on Fault Tolerant Computing (FTCS-10)*, October 1980.

[21] X. Castillo and D. P. Siewiorek. Workload, Performance, and Reliability of Digital Computing Systems. *Proceedings of the 11th IEEE Symposium on Fault Tolerant Computing (FTCS-11)*, June 1981.

[22] D. Chen, S. Garg, C. Kintala, and K. S. Trivedi. Dependability Enhancement for IEEE 802.11 with Redundancy Techniques. *proc. of IEEE 2003 International Conference on Dependable Systems and Networks (DSN '03)*, June 2003.

[23] D. Chen, S. Garg, and K.S. Trivedi. Network Survivability Performance Evaluation: A Quantitative Approach with Applications in Wireless Ad hoc Networks. *The Fifth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, September 2002.

[24] S. Chen, Z. Kalbarczyk, J. Xu, and R.K. Iyer. A Data-Driven Finite State Machine Model for Analyzing Security Vulnerabilities. *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN 2003)*, June 2003.

[25] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of Failure Rate in Widely Distributed Software. *Proceedings of the 25st IEEE Symposium on Fault Tolerant Computing (FTCS-25)*, June 1995.

[26] R. Chillarege, R. K. Iyer, J. C. Laprie, and J. D. Musa. Field Failures and Reliability in Operation. *Proc. of the 4th IEEE International Symposium on Software Reliability Engineering*, November 1993.

[27] M. Cinque, F. Cornevilli, D. Cotroneo, and S. Russo. An Automated Distributed Infrastructure for Collecting Bluetooth Field Failure Data. *Proc. of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, May 2005.

[28] M. Cinque, D. Cotroneo, and S. Russo. Achieving All the Time, Everywhere Access in Next-Generation Mobile Networks. *Mobile Networks and Applications, Special Issues on Mobility of Systems, Users, Data and Computing*, 9(2):29–39, April 2005.

[29] M. Cinque, D. Cotroneo, and S. Russo. Collecting and Analyzing Failure Data of Bluetooth Personal Area Networks. *Proc. of the 2006 International Conference on Dependable Systems and Networks (DSN '06)*, June 2006.

[30] T. Clouqueur, K.K. Saluja, and P. Ramanathan. Fault Tolerance in Collaborative Sensor Networks for Target Detection. *IEEE Transactions on Computers*, 36(7):74–84, March 2004.

[31] D. Cotroneo. The Hide and Seek Field Data Game. *Proc. of the sixth IEEE European Dependable Computing Conference*, October 2006.

[32] D. Cotroneo, S. Orlando, and S. Russo. Failure Classification and Analysis of the Java Virtual Machine. *Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, July 2006.

[33] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Sistems - Concepts and Design*. Addison-Wesley, 2000.

[34] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *Proc. of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 1996.

[35] M. Cukier, R. Berthier, S. Panjwani, and S. Tan. A Statistical Analysis of Attack Data to Separate Attacks. *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN 2003)*, June 2003.

[36] E.W. Czeck and D.P. Siewiorek. Observations on the Effects of Faults Manifestation as a Function of Workload. *IEEE Transactions on Computers*, 41(5):559–566, May 1992.

[37] M. Dacier, F. Pouget, and H. Debar. Honeypots: A practical Mean to Validate Malicious Fault Assumptions. *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, December 2004.

[38] A. Das, A. Ghose, V. Gupta, A. Razdan, H. Saran, and R. Shorey. Adaptive Link-level Error Recovery Mechanisms in Bluetooth. *The IEEE International Conference on Personal Wireless Communications*, December 2000.

[39] M. Elangovan D. D. Deavours and J. E. Dawkins. User-Perceived Interoperability of Bluetooth Devices. Technical report, The University of Kansas 2335 Irving Hill Road,Lawrence, KS 66045-7612, June 2004.

[40] J. Arlat et. al. DBench: State of Art, Deliverable CF1. Technical report, LAAS - CNRS, 2001.

[41] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S.C. Diot. Packet-level Traffic Measurements from the Sprint IP Backbone. *IEEE Network*, 17(6):6–16, December 2003.

[42] A. Ganapathi and D. Patterson. Crash Data Collection: A Windows Case Study. *Proc. of IEEE International Conference on Dependable Systems and Networks (DSN 2005)*, June 2005.

[43] R. Gandhi. Tolerance to Access-Point Failures in Dependable Wireless LAN. *Proc. of the 9th Int. Workshop on Object-Oriented Real-Time dependable Systems (WORDS'03)*, June 2003.

[44] M. Gerla, P. Johanssona, R. Kapoor, and F. Vatalaro. Bluetooth: "Last Meter" Technology for Nomadic Wireless Internetting. *Proc. of 12 th Tyrhennian Int. Workshop on Digital Communications*, 2000.

[45] J. Gray. A Census of Tandem System Availability Between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, October 1990.

[46] J.P Hansen and D. P. Siewiorek. Models for Time Coalescence in Event Logs. *Proceedings of the 22nd IEEE Symposium on Fault Tolerant Computing (FTCS-22)*, June 1992.

[47] R. Harrison. *Symbian OS C++ for Mobile Phones Volume 2*. Symbian Press, 2004.

[48] M.C. Hsueh, R.K. Iyer, and K.S. Trivedi. Performability Modeling Based on Real Data: a Case Study. *IEEE Transactions on Computers*, 37(4):478–484, April 1988.

[49] IEEE. *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications:*, 1999.

[50] R. K. Iyer, Z. Kalbarczyk, and M. Kalyanakrishnam. Measurement-Based Analysis of Networked System Availability. *Performance Evaluation Origins and Directions, Ed. G. Haring, Ch. Lindemann, M. Reiser, Lecture Notes in Computer Science, Springer Verlag*, 1999.

[51] R.K. Iyer, S.E. Butner, and E.J. McCluskey. A Statistical Failure/Load Relationship: Results of a Multicomputer Study. *IEEE Transactions on Computers*, C-31(7):697–706, July 1982.

[52] R.K. Iyer, D.J. Rossetti, and M.C. Hsueh. Measurement and Modeling of Computer Reliability as Affected by System Activity. *ACM Transactions on Computer Systems*, 4(3):214–237, August 1986.

[53] R.K. Iyer, L.T. Young, and P.V.K. Iyer. Automatic Recognition of Intermittent Failures : An Experimental Study of Field Data. *IEEE Transactions on Computers*, 39(4):525–537, April 1990.

[54] P. Johansson, R. Kapoor, M. Kazantzidis, and M. Gerla. Personal Area Networks: Bluetooth or IEEE 802.11? *International Journal of Wireless Information Networks Special Issue on Mobile Ad Hoc Networks*, April 2002.

[55] M. Kalyanakrishnam, Z. Kalbarczyk, and R.K. Iyer. Failure Data Analysis of a LAN of Windows NT Based Computers. *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, October 1999.

[56] M. Kalyanakrishnan, R. K. Iyer, and J. Patel. Reliability of Internet Hosts - A Case Study from the End User's Perspective. *Proc. of the 6th International Conference on Computer Communications and Networks*, September 1997.

[57] D. Katic and M. Vukobratovic. Survey of Intelligent Control Techniques for Humanoid Robots. *Journal of Intelligent and Robotic Systems - Kluwer Academic Publishers*, 37:117–141, 2003.

[58] T.W. Keller. CRAY-1 Evaluation Final Report. LA-6456.MS, Los Alamos Scientific Laboratory, Los Alamos, CA, December 1976.

[59] L. Kleinrock. Nomadicity: Anytime, Anywhere in a disconnected world. *Mobile Networks and Applications*, 1(1):351 – 357, December 1996.

[60] P. Koopman and J. Ray. Efficient High Hamming Distance CRCs for Embedded Networks. *Proc. of the 2006 International Conference on Dependable Systems and Networks (DSN '06)*, June 2006.

[61] P. Krishna, N.H. Vaidya, and D.K. Pradhan. Recovery in Distributed Mobile Environments. *Proc. of the 1993 IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993.

[62] T. Kubik and M. Sugisaka. Use of a Cellular Phone in mobile robot voice control. *Proc. of the 40th SICE Annual Conference*, July 2001.

[63] R. Lal and G. Choi. Error and Failure Analysis of a UNIX Server. *Proc. of the third IEEE Symposium on High-Assurance Systems Engineering*, 1998.

[64] J.-C. Laplace and M. Brun. Critical Software For Nuclear Reactors: 11 Years of Field Experience Analysis. *Proc. of the 9th IEEE International Symposium on Software Reliability Engineering*, November 1998.

[65] J.C. Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology. *Proc. of the 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15)*, June 1985.

[66] I. Lee, R.K. Iyer, and D. Tang. Error/Failure Analysis Using Event Logs from Fault Tolerant Systems. *Proceedings of the 21st IEEE Symposium on Fault Tolerant Computing (FTCS-21)*, June 1991.

[67] I. Lee, R.K. Iyer, D. Tang, and M.C. Hsueh. Measurement-Based Evaluation of Operating System Fault Tolerance. *IEEE Transactions on Reliability*, 42(2):238–249, June 1993.

[68] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, and Morris Jette. BlueGene/L Failure Analysis and Prediction Models. *proc. of the 2006 International Conference on Dependable Systems and Networks (DSN'06)*, June 2006.

[69] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, Jose Moreira, and Manish Gupta. Filtering Failure Logs for a BlueGene/L Prototype. *proc. of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 476 – 485, June 2005.

[70] C.T. Lim. Drop Impact Study of Handheld Electronic Products. *Proc. of the 5th International Symposium on Impact Engineering*, July 2004.

[71] T.T.Y. Lin and D.P. Siewiorek. Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis. *IEEE Transactions on Reliability*, 39(4):419–432, October 1990.

[72] W.C. lynch, W. Wagner, and M.S. Schwartz. Reliability Experience with Chi/OS. *IEEE Transactions on Software Engineering*, 1(2):253–257, June 1975.

[73] L. Marchesotti, R. Singh, and C. Regazzoni. Extraction of Aligned Video and Radio Information for Identity and Location Estimation in Surveillance Systems. *Proc. of the 7th International Conference on Information Fusion*, pages 316–321, June 2004.

[74] S. M. Matz, L. G. Votta, and M. Malkawi. Analysis of Failure Recovery Rates in a Wireless Telecommunication System. *proc. of the 2002 International Conference on Dependable Systems and Networks (DSN'02)*, 2002.

[75] R.A. Maxion and F.E. Feather. A Case Study of Ethernet Anomalies in a Distributed Computing Environment. *IEEE Transactions on Reliability*, 39(4):433–443, October 1990.

[76] The Honeynet Research Project Members. *The Honeynet Project*. http://www.honeynet.org, 2004.

[77] P. J. Mogowan, D. W. Suvak, and C. D. Knutson. *IrDA Infrared Communications: an Overview*. www.irda.org.

[78] R. Mullen. The Lognormal Distribution of Software Failure Rates: Origin and Evidence. *Proc. of the 9th IEEE International Symposium on Software Reliability Engineering*, November 1998.

[79] B. Muprphy and B. Levidow. Windows 2000 Dependability. MSR-TR-2000-56, Microsoft Research, Microsoft Corporation, Redmond, WA, June 2000.

[80] D. Oppenheimer and D.A. Patterson. Studying and Using Failure Data from Large-Scale Internet Services. *Proc. of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, July 2002.

[81] C.H. Ou, K.F. Ssu, and H.C. Jiau. Connecting Network Partitions with Location-Assisted Forwarding Nodes in Mobile Ad Hoc Environments. *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, December 2004.

[82] B. Parhami. From Defects to Failures: a View of Dependable Computing. *ACM SIGARCH Computier Architecture News*, 16(4):157–168, September 1988.

[83] T. Park, N. Woo, and H.Y. Yeom. An Efficient Optimistic Message Logging Scheme for Recoverable Mobile Computing Systems. *IEEE Transactions on Mobile Computing*, 1(4):265–277, October-December 2002.

[84] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico, and P. Koopman. Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems. *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN 2005)*, June 2005.

[85] V. Paxon. End-to-End Routing Behavior in the Internet. *ACM Conference proceedings on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'96)*, August 1996.

[86] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, M. Barbera, and I. Mura. Service-Level Availability Estimation of GPRS. *IEEE Transactions on Mobile Computing*, 2(3), July-September 2003.

[87] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure Data Analysis of a Large-Scale Heterogeneous Server Environment. *proc. of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, June 2004.

[88] F. Salfner and M. Malek. Predicting Failures of Computer Systems: A Case Study for a Telecommunication System. *Proc. of 11th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS'06)*, April 2006.

[89] P. Santi and D.M. Bough. An Evaluation of Connectivity in Mobile Wireless Ad Hoc Networks. *Proc. of the 2002 IEEE International Conference on Dependable Systems and Networks (DSN '02)*, June 2002.

[90] B. Schroeder and G.A. Gibson. A Large-Scale Study of Failures in High-Performance Computing Systems. *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN 2006)*, June 2006.

[91] A. Sekman, A. B. Koku, and S. Z. Sabatto. Human Robot Interaction via Cellular Phones. *Proc. of the 2003 IEEE Int. Conf. on Systems, Man and Cybernetics*, October 2003.

[92] S. Sesay, Z. Yang, and J. He. A Survey on Mobile Ad Hoc Wireless Network. *Information Technology Journal*, 3(2):168–175, 2004.

[93] M. Shaw. Everyday Dependability for Everyday Needs. *Proc. of the 13th IEEE International Symposium on Software Reliability Engineering*, November 2002.

[94] D.P. Siewiorek, R. Chillarege, and Z.T. Kalbarczyk. Reflections on Industry Trends and Experimental Research in Dependability. *IEEE Transactions on Dependable and Secure Computing*, 1(2):109–127, April-June 2004.

[95] C. Simache and M. Kaaniche. Measurement-based Availability Analysis of Unix Systems in a Distributed Environment. *Proc. of the 12th IEEE International Symposium on Software Reliability Engineering*, November 2001.

[96] C. Simache, M. Kaaniche, and A. Saidane. Event Log based Dependability Analysis of Windows NT and 2K Systems. *Proceedings of the 8th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, December 2002.

[97] A.P. Snow, U. Varshney, and A.D. Malloy. Reliability and Survivability of Wireless and Mobile Networks. *IEEE Computer*, pages 49–55, July 2000.

[98] M. Sullivan and R. Chillarege. Software Defects and Their impact on System Availabilty - A Study of Field Failures in Operating Systems. *Proceedings of the 21st IEEE International Symposium on Fault-Tolerant Computing (FTCS-21)*, June 1991.

[99] M. Sullivan and R. Chillarege. A Comparison of Software Defects in Database Management Systems and Operating Systems. *Proceedings of the 22nd IEEE International Symposium on Fault-Tolerant Computing (FTCS-22)*, June 1992.

[100] R.S. Swarz and D.P. Siewiorek. *Reliable Computr Systems (3rd ed.): Design and Evaluation*. A.K. Peters, 1998.

[101] D. Tang, M. Hecht, J. Miller, and J. Handal. MEADEP: A Dependability Evaluation Tool for Engineers. *IEEE Transactions on Reliability*, 47(4):443–450, December 1998.

[102] D. Tang and R.K. Iyer. Dependability Measurment and Modeling of a Multicomputer System. *IEEE Transactions on Computers*, 42(1):62–75, January 1993.

[103] D. Tang and R.K. Iyer. MEASURE+ - A Measurement-Based Dependability Analysis Package. *Proc. of the ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 1993.

[104] A Thakur and R. K. Iyer. Analyze-NOW - An Environment for Collection and Analysis of Failures in a Networked of Workstations. *IEEE Transactions on Reliability*, 45(4):560–570, 1996.

[105] J. Tourrilhes and C. Carter. P-handoff: A protocol for fine grained peer-to-peer vertical handoff. *Proc. on the 13th IEEE Int. Symposium on Personal, Indoor and Mobile Radio Communcations (PIMRC '02)*, 2002.

[106] K. S. Trivedi. *Probability and Statistic with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons, 2002.

[107] K. Vaidyanathan and K. S. Trivedi. A Comprehensive Model of Software Rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2(2):124–137, April-June 2005.

[108] P. Velardi and R.K. Iyer. A Study of Software Failures and Recovery in the MVS Operating System. *IEEE Transactions on Computers*, C-33(6):564–568, June 1984.

[109] Y.M. Wang, Y. Huang, K.-P. Vo, P.Y. Chung, and C. Kintala. Checkpointing and its Applications. *Proc. of the 25th IEEE Fault-Tolerant Computing Symposium (FTCS-25)*, June 1995.

[110] A.S. Wein and A. Sathaye. Validating Complex Computer System Availability Models. *IEEE Transactions on Reliability*, 39(4):468–479, October 1990.

[111] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):74–84, June 1993.

[112] J. Xu, Z. Kalbarczyc, and R. K. Iyer. Networked Windows NT System Field Data Analysis. *Proc. of the 5th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'99)*, December 1999.

[113] B. Yao and W.K. Fuchs. Message Logging Optimization for Wireless Networks. *Proc. of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS'01)*, October 2001.

[114] V. C. Zandy and B. P. Miller. Reliable Network Connections. *Proc. of the 8th International Conference on Mobile Computing and Networking (MOBICOM '02)*, September 2002.