

# Measurement-based Analysis of Networked System Availability

Ravishankar K. Iyer, Zbigniew Kalbarczyk, Mahesh Kalyanakrishnan

Center for Reliable and High-Performance Computing  
University of Illinois at Urbana-Champaign  
1308 W. Main St., Urbana, IL 61801-2307 USA  
E-mail: {iyer, kalbar, mahesh}@crhc.uiuc.edu

**Abstract.** The discussion in this section focuses on the issues involved in analyzing the availability of networked systems using the failure data collected by the logging mechanisms built into the system. The process of collecting data and the type of analysis conducted are illustrated by representative studies that include analysis of availability of a LAN of UNIX-based workstations, a LAN of Windows NT-based machines, and the Internet. The analysis of field failure data enables us to study naturally occurring errors and provides feedback to system designers for improving system availability. For example, the study of failures in a network of Windows NT machines reveals that: (1) most of the problems that lead to reboots are software related, (2) rebooting the machine does not appear to solve the problem in many cases. In about 60% of the reboots, the rebooted machine reported problems within a hour or two of the reboot, (3) though the average availability evaluates to over 99%, a typical machine, on average, provides acceptable service only about 92% of the time, and (4) there are indications of propagated or correlated failures.

## 1 Introduction

The dependability of a system can be experimentally evaluated at different phases of its life cycle. In the *design phase*, computer-aided design (CAD) environments are used to evaluate the design via simulation, including simulated fault injection. Such fault injection tests the effectiveness of fault-tolerant mechanisms and evaluates system dependability, providing timely feedback to system designers. Simulation, however, requires accurate input parameters and validation of output results. Although the parameter estimates can be obtained from past measurements, this is often complicated by design and technology changes. In the *prototype phase*, the system runs under controlled workload conditions. In this stage, controlled physical fault injection is used to evaluate the system behavior under faults, including the detection coverage and the recovery capability of various fault tolerance mechanisms. Fault injection on the real system can provide information about the failure process, from fault occurrence to system recovery, including error latency, propagation, detection, and recovery (which may

involve reconfiguration). But this type of fault injection can only study artificial faults; it cannot provide certain important dependability measures, such as mean time between failures (MTBF) and availability. In the *operational phase*, a direct measurement-based approach can be used to measure systems in the field under real workloads. The collected data contain a large amount of information about naturally occurring errors/failures. Analysis of this data can provide understanding of actual error/failure characteristics and insight into analytical models. Although measurement-based analysis is useful for evaluating the real system, it is limited to detected errors. Further, conditions in the field can vary widely, casting doubt on the statistical validity of the results. Thus, all three approaches - simulated fault injection, physical fault injection, and measurement-based analysis - are required for accurate dependability analysis.

In the design phase, simulated fault injection can be conducted at different levels: the electrical level, the logic level, and the function level. The objectives of simulated fault injection are to determine dependability bottlenecks, the coverage of error detection/recovery mechanisms, the effectiveness of reconfiguration schemes, performance loss, and other dependability measures. The feedback from simulation can be extremely useful in cost-effective redesign of the system. For thorough discussion of different techniques for simulated fault injection can be found in [15].

In the prototype phase, while the objectives of physical fault injection are similar to those of simulated fault injection, the methods differ radically because real fault injection and monitoring facilities are involved. Physical faults can be injected at the hardware level (logic or electrical faults) or at the software level (code or data corruption). Heavy-ion radiation techniques can also be used to inject faults and stress the system. The detailed treatment of the instrumentation involved in fault injection experiments using real examples, including several fault injection environments is given in [15].

In the operational phase, measurement-based analysis must address issues such as how to monitor computer errors and failures and how to analyze measured data to quantify system dependability characteristics. Although methods for the design and evaluation of fault-tolerant systems have been extensively researched, little is known about how well these strategies work in the field. A study of production systems is valuable not only for accurate evaluation but also for identifying reliability bottlenecks in system design. In [15] the measurement-based analysis is based on over 200 machine-years of data gathered from IBM, DEC, and Tandem systems (note that these are not networked systems).

In this chapter we discuss the current research in the area of experimental analysis of computer system dependability in the context of methodologies suited for measurement-based dependability analysis of networked systems. We use examples of LAN of UNIX-based workstations, LAN of Windows NT based computers, and Internet to present methods for collecting and analyzing failure data to obtain dependability characterization of the network.

## 2 Measurement-based Studies of Computer System Availability

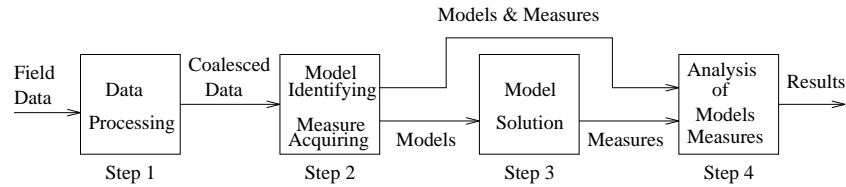
There are many possible sources of errors, including untested manufacturing faults and software defects, transient errors induced by radiation, power surges, or other physical processes, operator errors, and environmental factors. The occurrence of errors is also highly dependent on the workload running on the system. A distribution of operational outages from various error sources for several major commercial systems are reported in [35].

There is no better way to understand dependability characteristics of computer systems (including networked systems) than by direct measurements and analysis. Here, measuring a real system means monitoring and recording naturally occurring errors and failures in the system while it is running under user workloads. Analysis of such measurements can provide valuable information on actual error/failure behavior, identify system bottlenecks, quantify dependability measures, and verify assumptions made in analytical models. Given field error-data collected from a real system, a measurement-based study consists of four steps, as shown in Fig. 1: 1) data processing, 2) model identification and parameter estimation, 3) model solution if necessary, and 4) analysis of models and measures.

Step 1 consists of extracting necessary information from field data (the result can be a form of compressed data or flat data), classifying errors and failures, and coalescing repeated error reports. In a computer system, a single problem commonly results in many repeated error observations occurring in rapid succession. To ensure that the analysis is not biased by these repeated observations of the same problem, all error entries that have the same error type and occur within a short time interval (e.g., 5 minutes) of each other should be coalesced into a single record. The output of this step is a form of coalesced data in which errors and failures are identified. This step is highly dependent on the measured system. Coalescing algorithms have been proposed in [46], [13], [8].

Step 2 includes identifying appropriate models (such as Markov models) and estimating various measures of interest (such as MTBFs and TBF distributions) from the coalesced data. Several models have been proposed and validated using real data. These include workload-dependent cyclostationary models [3], a workload hazard model [10], and error/failure correlation models [41]. Statistical analysis packages such as SAS [33] or measurement-based dependability analysis tools such as MEASURE+ [44] are useful at this stage.

Step 3 solves these models to obtain dependability measures (such as reliability, availability, and transient reward rates). Dependability and performance modeling and evaluation tools such as SHARPE [32] can be used in this step. Step 4, the most creative part of this study, involves a careful interpretation of the models and measures obtained from the data; for example, the identification of reliability bottlenecks and the impact on availability of design enhancement. The analysis methods can vary significantly from one study to another, depending on project goals.



**Fig. 1.** Measurement-based Analysis

Measurement-based dependability analysis of operational systems has evolved significantly over the past 15 years. These studies have addressed one or more of the following issues: basic error characteristics, dependency analysis, modeling and evaluation, software dependability, and fault diagnosis. Table 1 provides a quick overview of the issues addressed in the literature.

Early studies in this field investigated transient errors in DEC computer systems and found that more than 95% of all detected errors are intermittent or transient errors [34], [28]. The studies also showed that the inter arrival time of transient errors follows a Weibull distribution with a decreasing error rate. This distribution was also shown to fit the software failure data collected from an IBM operating system [12]. A recent study of failure data from three different operating systems showed that time to error (TTE) can be represented by a multistage gamma distribution for a single-machine operating system and by a hyperexponential distribution for the measured distributed operating systems [21].

Several studies have investigated the relationship between system activity and failures. In the early 1980s, analysis of measurements from IBM [2] and DEC [3] machines revealed that the average system failure rate was strongly correlated with the average workload on the system. The effect of workload-imposed stress on software was investigated in [4] and [12]. Analyses of DEC [39], [48], and Tandem [19] multicomputer systems showed that correlated failures across processors are not negligible, and their impacts on availability and reliability are significant [6], [40], [41].

In [9], analytical modeling and measurements were combined to develop measurement-based reliability/performance models using data collected from an IBM mainframe. The results showed that a semi-Markov process is better than a Markov process for modeling system behavior. Markov reward modeling techniques were further applied to distributed systems [43] and fault-tolerant systems [20] to quantify performance loss due to errors/failures for both hardware and software.

A census of Tandem system availability indicated that software faults are the major source of system outages in the measured fault-tolerant systems [7]. Analyses of field data from different software systems investigated several dependability issues including the effectiveness of error recovery [47], hardware-related software errors [11], correlated software errors in distributed systems

[42], software fault tolerance [20], [22], and software defect classification [37], [38]. Measurement-based fault diagnosis and failure prediction issues were investigated in [46], [14], [23], [25], [26], [23].

Local Area Networks (LANs) of workstations have become very popular in industry and in academia, however reliability issues and the interdependencies of failures in such networks are still not well understood. Studies presented in [45] and [18] are a step in the direction of understanding of the nature of failures in networked systems. Thakur [45] presented a simple yet effective methodology for collecting and analyzing failures in a network of UNIX-based workstations. The majority of observed failures (68%) encountered were network related. Kalyanakrishnam [18] analyzed machine reboots collected in a network of Windows NT-based servers. This study investigated the major causes of reboots and obtained quantitative measures of machine up-times and down-times. The failure behavior of a typical machine and of the network as a whole was modeled in terms of a finite state machine, providing insights into the typical problems encountered and their dynamics.

**Table 1.** Measurement-based Studies of Computer-System Dependability

Category	Issues	Studies
Data coalescing	Analysis of time-based tuples Clustering based on type and time	[46], [8] [13], [19], [43]
Basic Error Characteristics	Transient faults/errors Error/failure bursts TTE/TTF distributions	[34], [28], [13] [13], [43] [28], [12], [21]
Dependency Analysis	Hardware failure/workload dependency Software failure/workload dependency Correlated failures and impact	[2], [4], [10] [4], [12] [39], [48], [41]
Modeling and Evaluation	Two-way and multiway failure dependency Performability model for single machine Markov reward model for distributed system Two-level models for operating systems	[6], [19], [40] [9] [43] [21]
Software Dependability	Error recovery Hardware-related & correlated software errors Software fault tolerance Software defect classification	[47] [11], [42], [21] [7], [20], [22] [37], [38]
Fault Diagnosis	Heuristic trend analysis Statistical analysis of symptoms Network fault signature	[46], [23] [14] [25], [26] [27]
Network availability	Failure behavior of LANs of machines Internet host behavior	[45], [18] [16], [36], [24], [1], [29], [30], [17]

The issue of Internet host behavior has been the focus of active research. Kalyanakrishnam [17] studied reliability of Internet hosts from the user perspective. Long et al. [24] evaluated mean time to failure (MTTF), mean time to repair (MTTR), availability, and reliability for a sample of hosts on the Internet by repeatedly polling the hosts from different sites at exponentially distributed time intervals. Paxson [30] described in great detail the vagaries of end-to-end Internet routing. The traceroute [36], [16] program (which traces the probable route of data from a remote host to the user site) was extensively used to study, in detail, routing behavior in the Internet. Major routing pathologies and properties were identified.

Chimoy [5] examined NSFNET backbone routing. The study was based on the trace information collected over a 12-hour period on all the NSFNET nodes on the T1 backbone. The backbone was evaluated with respect to parameters such as Update Propagation Time (the time taken to propagate update information from one router to another) and Unreachability Cycle Distribution (the time during which a network is unreachable from the backbone. It was shown that only a small number of networks have a highly volatile connectivity to the NSFNET backbone; most networks have a fairly stable connectivity. Paxson [30] obtained metrics that could characterize Internet performance. Arlitt et al. [1] studied Web server workloads and obtained invariants that characterize it. In this study, the access logs of six different Web servers were evaluated in terms of Success Rate, Mean Transfer Size, and Inter-Reference Time. It was found that most of the Web server traffic consisted of HTML and image files.

In the following sections, we discuss issues and representative studies involved in measurements, data processing, error and failure analysis, dependency analysis, and the modeling and evaluation of dependability.

### 3 Measurements

The question of what and how to measure is difficult one. A combination of installed and custom instrumentation is typically used in most studies. From a statistical point of view, sound evaluations require a considerable amount of data. In modern computer systems, failures are infrequent and, in order to obtain meaningful data, measurements must be taken over a long period of time. Also, the measured system must be exposed to a wide range of usage conditions for the results to be representative. In an operational system, only detected errors can be measured.

There are two ways to take measurements: on-line automated logging and human manual logging. Many large computer systems, such as IBM and DEC mainframes, provide error-logging software in the operating system. Similarly, commercial operating systems such as UNIX and Windows NT offer capabilities for error logging. This software records information on errors occurring in the various subsystems, such as the memory, disk, and network subsystems, as well as other system events, such as reboots and shutdowns. The reports usually include information on the location, time, and type of the error, the system state at the

time of the error, and sometimes error recovery (e.g., retry) information. The reports are stored chronologically in a permanent system file. As failures are relatively rare events, it is necessary to meticulously collect and analyze error data for many machine months before the results of the data analysis can be considered statistically valid. Such regular and prolonged data acquisition is possible only through automated event logging. Hence most studies of failures in single and networked computer systems are based on the error logs maintained by the operating system running on those machines. The main advantage of on-line automatic logging is its ability to record a large amount of information about transient errors and to provide details of automatic error recovery processes, which cannot be done manually. Disadvantages are that an on-line log does not usually include information about the cause and propagation of the error or about off-line diagnosis. Also, under some crash scenarios, the system may fail too quickly for any error messages to be recorded.

Before we explore how event logs can be used in analyzing behavior of networked systems, we illustrate how events are generated and logged in the Windows NT operating system. We also show the format and particulars of the event log created by the operating system.

### 3.1 Event Logging Mechanism in Windows NT Operating System

Event logging in the Windows NT operating system is provided through a dedicated *Event Logging Subsystem*. This subsystem consists of multiple execution threads each carrying out part of the event logging functionality. Some threads are dedicated to receiving event log requests through transport layer *ports*, whereas other threads actually write the event to the event log.

The mechanism of event logging is slightly different for different types of events. For events that are logged by applications or other subsystems, the Event Logging Subsystem provides an API (Application programmer interface) to log events. For events that are generated within the *Executive*<sup>1</sup> (e.g., device drivers), the events are directly written to the event log file by the *I/O Manager*, which is a part of the Windows NT Executive.

### 3.2 Event Log Format

The event logs conform to a specific format employed by the Windows NT operating system. Events on Windows NT machines fall into one of three types:

- *Application events* are those that are logged by applications running on NT machines (e.g., error logged by MExchange MTA (Message Transfer Agent)).
- *System events* are those that are reported by components of the Windows NT operating system (e.g., Server, Redirector, NETLOGON service, etc.).

<sup>1</sup> *Kernel* and *Executive* are two basic components of the Windows NT operating system. The two components run in *kernel* (privileged) *mode*.

- *Security events* are those that relate to user authentication or verification of access permissions.

A sample event log is shown below, the format of the log may not be universal, but the fields in it are.

```
1997/09/11 15:02:53 4 0 5715 NETLOGON N/A EXCHOU-CA0201 LSA EXCHOU-CONN02 1
```

The fields in the event log are:

1. *Date* and *time* of the event
2. *Severity* of the event (4) (1 indicates an Error, 2 indicates a Warning, 4 indicates an Information message)
3. Event *id* (5715)
4. The *source* that logs that event (NETLOGON)
5. The *machine* on which the event was logged (EXCHOU-CA0201)
6. *Event-specific information* (EXCHOU-CONN02 1)

Typically, each source that logs an event provides a "message" file that contains a description of the event. So further information about the event can be obtained by looking at the text description in the message file. All Windows NT systems have a built-in Event Viewer, a tool used to examine events logged on a given machine. The Event Viewer uses the message files provided by the system component or application in order to provide a textual description of the event. This particular method of displaying events pose quite a few problems, which are listed below:

- The message files ship with the respective application. In order to interpret the event, the application that logged the message must be running on the machine performing the analysis. Because this is impractical, it is not possible to obtain the textual description of most application events.
- Even for system events, not all message files are accessible. The Windows NT event logging source code contains message files for some, but not all, system components.
- Windows NT stores the event logs in a file on the local machine. Depending on the system configuration, the old event logs get overwritten regularly so that the number of logs collected does not increase without bound. For this reason, it is necessary to retrieve event logs before they are over-written.

## 4 Data Processing

Usually, online logs contain a large amount of redundant and irrelevant information in various formats. Thus, data processing must be performed to classify this information and to put it into a flat format to facilitate subsequent analyses. The first step in data processing is error classification. This process classifies errors in the measured system into types based on the subsystems and components in which they occur. There is no uniform or best error classification, because different systems have different hardware and software architectures. Moreover, the error categories depend on the criteria used to classify errors.

## 4.1 Measurement Overview

As we focus on the network systems, a basic yet useful classification is based on the fault origin. A networked computing environment has two major types of faults based on their source:

- *Machine-related.* A computation node (workstation) encounters failure conditions because of problems that are local to that workstation.
- *Network-related.* A computation node encounters a failure because of a problem in a network component. For example, a failure condition encountered on a client due to a failed server would be classified as a network-related failure.

Table 2 summarizes the classification of errors collected for a LAN of UNIX workstations [45], a LAN of Windows NT machines [18], and Internet [17]. Table 2 should not be used to make a direct comparison between these networks because they are vastly different systems, measured over different periods and operating under different conditions. The data are, however, useful to provide insight into the type of failures encountered in different environments. For example, Table 2 indicates that most of the problems in a LAN of UNIX workstations and on the Internet are network-related. The data for the Windows NT-based network show opposite tendency, i.e., most of the problems are machine-related. The reason is that for the Windows NT based system we analyze only machine reboots, and most of these can be attributed to machine-related problems. The above classification is very generic and does not provide enough insight into the real nature of failures observed in the network system. More detailed analysis is needed to identify weak points in the system.

**Table 2.** Network-related and Machine-related Problems in Three Different Network Systems

	<b>LAN of UNIX Workstations</b>	<b>LAN of Windows NT Machines</b>	<b>Internet</b>
Type of environment	Academic environment	Commercial environment	N/A
Type and number of machines	69 SunOS-based workstations	68 Windows NT 4.0 based mail servers	97 most popular Web sites
Period for data collecting	32 weeks	6 months	40 weeks
Failure context	All types of failures logged in a workstation's system log.	Machine reboots logged in a server system log.	Inability to contact a host and fetch an HTML file from it.
Machine-related failures	32%	69%	47%
Network-related failures	68%	31%	53%

In the following discussion, failure data collected from a LAN of UNIX-based workstations and a LAN of Windows NT-based servers are used to illustrate analysis methods for obtaining detailed failure characterization of the system behavior. In the case of Internet, the process of collecting and analyzing data is significantly different that used for the two LANs. Therefore, the Internet related analysis is presented separately, in Section 10 of this chapter.

## 4.2 Data Extraction and Data Entry

The data processing that follows error classification consists of two steps: *data extraction* and *data coalescing*. Data extraction consists of selecting useful entries, such as error and reboot reports from the log file (throwing away uninteresting entries such as disk volume change reports) and transforming the data set into a flat format. The design of the flat format depends on the necessity of the subsequent analyses. The following is a possible format:

entry number	logging time	error type	device id	error description fields
--------------	--------------	------------	-----------	--------------------------

In on-line error logs, a single fault in the system can result in many repeated error reports in a short period of time. To ensure that the subsequent analyses will not be distorted by these repeated reports, entries that correspond to the same problem should be coalesced into a single event, or *tuple* [46]. A typical data-coalescing algorithm merges all error entries of the same error type that occur within a  $\Delta T$  interval of each other into a tuple. A tuple reflects the occurrence of one or more errors of the same type in rapid succession. It can be represented by a record containing information such as the number of entries in the tuple and the time duration of the tuple.

Different systems may require different time intervals to data coalesce. A recent study [8] defined two kinds of mistakes that can be made in data coalescing: *collision* and *truncation*. A collision occurs when the detection times of two faults are close enough (within  $\Delta T$ ) that they are combined into a tuple. A truncation occurs when the time between two reports caused by a single fault is greater than  $\Delta T$ , so that the two reports are split into different tuples. If  $\Delta T$  is large, collisions are likely to occur. If  $\Delta T$  is small, truncations are likely to occur. The study found that there is a time-interval threshold beyond which collisions rapidly increase. Based on this observation, the study proposed a statistical model that can be used to select an appropriate time interval. In our experience, collision is not a big problem if the error type and device information is used in data coalescing, as shown in the above coalescing algorithm. Truncation is usually not considered to be a problem [8]. Also, there are techniques [14], [23] to deal with truncation which have been used for fault diagnosis and failure prediction.

## 4.3 Error and Failure Analysis

Once coalesced data is obtained, the basic dependability characteristics of the measured system can be identified by a preliminary statistical analysis. Commonly used measures in the analysis include error/failure frequency, TTE or

TTF distribution, and system availability. These measures provide an overall picture of the system and help to identify dependability bottlenecks.

**Initial Classification of Data Collected from a LAN of UNIX-based Workstations.** To better understand real failure nature, the machine-related and network-related failures are classified into two groups based on the effect they have on the network components [45]. This kind of analysis helps in focusing resources to the major failure points in the system. Two basic classes are identified:

- *Hard failure.* A failure state, in which a machine becomes unusable. An unresponsive server, a loose cable, or even a keyboard error can cause a hard failure.
- *Soft failure.* A failure in which a system is affected in such a way that it does not become unusable. In most cases, this effect manifests as a performance loss. Failures such as disk errors and delayed responses from the server, are examples of soft failures.

In [45], it was observed that, on a system-wide basis, 72% of the failures are hard failures and 28% are soft failures. Further analysis of these failures shows that: (1) out of the 28% soft failures, 61% are network related, (2) machine-related failures constitute only 25% of the hard failures. With 75% of the hard failures related to the network, it is obvious that network components are the major cause of failures in a network of workstations. Note that by the hard network failure we understand a failure, a workstation encounters due to a failure (hard or soft) in another network component.

*Types of Network and Machine Related Failures.* Having established that network-related failures are the primary cause of problems in a network of workstations, we look at the types of network-related failures.

*Network-related soft failures* include:

- errors in remote procedure calls 0% (RPC errors, not important),
- network file-write errors 3% (NFS errors),
- delayed responses from the server due to an excessive load on the server 27% (performance-related),
- overflowing network buffers 3%.

Thus, soft failures account for approximately 30% of all network-related failures. A vast majority of network-related soft failures are performance related (failures due to inadequate performance response).

*Network-related hard failures* include:

- cable errors 4% (loose cable wire, unplugged transceiver),
- portmapper errors 1% (portmapper stops responding),
- Ethernet jamming due to hardware failure, buffer overflowing 6%, and
- network server unresponsiveness to a client's request 56%.

Thus, approximately 80% (56% of [4+1+6+56]%) of the network hard failures can be attributed to a failure due to an unresponsive server. In our measurement, in 48% of cases of an unresponsive server, a single client failed and identified a healthy server as unresponsive. In 22% of the cases, the client identified a wrong server as the problem source. For example, a client X says that server Y has failed; however, at that time server Z was the one that failed. In the remaining 30% of the cases, the right server was identified, but the cause for the server's hand/halt service could not be ascertained.

*Machine-related soft failures*, which account for a third of the machine-related failures, include:

- memory errors 32% (both system memory errors and disk errors) and
- kernel errors 5% (errors in any kernel structure, such as the signal stack or the process table).

*Machine-related hard failures include:*

- machine reboots 62% (reboot of a machine because of a local fault) including:
  1. halts due to impatient users 13% (user reboots local machine to remove a server problem),
  2. machine service by the system administrator 12%, and
  3. other machine problems that need reboot 75%.
- keyboard errors 1% (unresponsive keyboard makes a machine unusable).

We observed that there are at least twice as many machine-reboots (62%) as disk and memory errors together (32%). This is contrary to the VAXcluster data reported in [15], which indicates dominance of memory and disk errors.

**Initial Classification of Data Collected from a LAN of Windows NT-based Servers.** The initial breakup of the data on a system reboot is primarily based on the events that preceded the current reboot by no more than an hour (and that occurred after the previous reboot). For each instance of a reboot, the most severe and frequently occurring events (hereafter referred to as prominent events) are identified. The corresponding reboot is then categorized based on the source and the id of these prominent events. In some cases, the prominent events are specific enough to identify the problem that caused the reboot. In other cases, only a high-level description of the problem can be obtained based on the knowledge of the prominent events. Table 3 shows the breakup of the reboots by category. (This data pertains to reboots in the domain over a period of six months.)

The categories in Table 3 should be interpreted as follows:

- *Hardware or firmware related problems:* This category includes events that indicate a problem with hardware components (network adapter, disk, etc.), their associated drivers (typically drivers failing to load because of a problem with the device), or some firmware (e.g., some events indicated that the Power On Self Test had failed).

- *Connectivity problems*: This category denotes events that indicated that either a system component (e.g., Redirector, Server) or a critical application (e.g., MExchange System Attendant) could not retrieve information from a remote machine. In these scenarios, it is not possible to pinpoint the actual cause of the connectivity problem. Some of the connectivity failures result from network adapter problems and hence are categorized as hardware related.

**Table 3.** Breakup of Reboots Based on Prominent Events

Category	Frequency	Percentage
Total reboots	1100	100
Hardware or firmware problems	105	9
Connectivity problems	241	22
Crucial application failures	152	14
Problems with a software component	42	4
Normal shutdowns	63	6
Normal reboots/power-off (no indication of any problems)	178	16
Unknown	319	29.00

- *Crucial application failure*: This category encompasses reboots, which are preceded by severe problems with, and possibly shutdown of, critical application software (such as Message Transfer Agent). In such cases, it wasn't clear why the application reported problems. If an application shutdown occurs as a result of connectivity problem, then the corresponding reboot is categorized as connectivity- related.
- *Problems with a software component*: Typically these reboots are characterized by startup problems (such as a critical system component not loading or a driver entry point not being found). Another significant type of problem in this category is the machine running out of virtual memory, possibly due to a memory leak in a software component. In many of these cases, the component causing the problem is not identifiable.
- *Normal shutdowns*: This category covers reboots, which are not preceded by warnings or error messages. Additionally, there are events that indicate shutting down of critical application software and some system components (e.g., the BROWSER). These represent shutdowns for maintenance or for correcting problems not captured in the event logs.
- *Normal reboots/power-off*: This category covers reboots which are typically not preceded by shutdown events, but do not appear to be caused by any problems either. No warnings or error messages appear in the event log before the reboot.

Based on data in Table 3, the following observations can be made about the failures:

1. 28% of the reboots cannot be categorized. Such reboots are indeed preceded by events of severity 2 or lesser, but there is not enough information available to decide (a) whether the events were severe enough to force a reboot of the machine or (b) the nature of the problem that the events reflect.
2. A significant percentage (22%) of the reboots have reported connectivity problems. This is not surprising considering that the workload on the machines is network I/O intensive.
3. Only a small percentage (10%) of the reboots can be traced to a system hardware component. Most of the identifiable problems are software related.
4. Nearly 50% of the reboots are abnormal reboots (i.e., the reboots were due to a problem with the machine rather than due to a normal shutdown).
5. In nearly 15% of the cases, severe problems with a crucial mail server application force a reboot of the machine.
6. Some of the reboots due to connectivity problems might be the result of propagated failures in the domain. Furthermore, it is possible that the machines functioning as the master browser and the Primary Domain Controller (PDC)<sup>2</sup>, respectively are potential reliability bottlenecks of the domain.

## 5 Analysis of Failure Behavior of Individual Machines

After the preliminary investigation of the causes of failures, we probe failures from the perspective of an individual machine as well as the whole network. First we focus on the failure behavior of individual machines in the domain to obtain (1) estimates of machine up-times and down-times, (2) an estimate of the availability of each machine, and (3) a finite state model to describe the failure behavior of a typical machine in the domain. The discussion in sections 5 to 9 is based on the results from the analysis of failure data collected from LAN of Windows NT based servers. Machine up-times and down-times are estimated as follows:

- For every reboot event encountered, the timestamp of the reboot is recorded.
- The timestamp of the event immediately preceding the reboot is also recorded. (This would be the last event logged by the machine before it goes down.)
- A smoothing factor of one hour is applied to the reboots (i.e., for multiple reboots that occurred within an period of one hour, except the last one, are disregarded). (Since the intermediate reboots indicate an incomplete recovery from the failure, the machine would have to be considered as being down until the last of such reboots occurs.)
- Each up-time estimate is generated by calculating the time difference between a reboot timestamp and the timestamp of the event preceding the next reboot.
- Each down-time estimate is obtained by calculating the time difference between a reboot timestamp and the timestamp of the event preceding it.

---

<sup>2</sup> In the analyzed network, the machines belonged to a common Windows NT domain. One of the machines was configured as the Primary Domain Controller (PDC). The rest of the machines functioned as Backup Domain Controllers (BDCs).

*Machine up-times* are presented in Table 4 (the numbers pertain to analysis of 5 months of data.)

**Table 4.** Machine Up-Time Statistics

Item	Value
Number of entries	616
Maximum	85.2 days
Minimum	1 hour
Average	11.82 days
Median	5.54 days
Standard Deviation	15.656 days

As the standard deviation suggests, there is a great degree of variation in the machine up-times. The longest up-time was nearly three months. The average is skewed because of some of the longer up-times. The median is more representative of the typical up-time.

*Machine down-times* are provided in Table 5 (the numbers pertain to analysis of 5 months of data.).

**Table 5.** Machine Down-Time Statistics

Item	Frequency
Number of entries	682
Maximum	15.76 days
Minimum	1 second
Average	1.97 hours
Median	11.43 minutes
Standard Deviation	15.86 hours

As the table clearly shows, 50% of the down-times last about 12 minutes. This is probably too short a period to replace hardware components and reconfigure the machine. The implication is that majority of the problems are software related (memory leaks, misloaded drivers, application errors etc.). The maximum value is unrealistic and might have been due to the machine being temporarily taken off-line and put back in after a fortnight.

*Discussion of up-times and down-times.* Since the machines under consideration are dedicated mail servers, bringing down one or more of them would potentially disrupt storage, forwarding, reception, and delivery of mail. The disruption can be prevented if explicit rerouting is performed to avoid the machines that are down. But it is not clear if such rerouting was done or can be done. In this context the following observations would be causes for concern: (1) average down-time measured was nearly 2 hours or (2) 50% of the measured up-time samples were about 5 days or less.

## 6 Availability

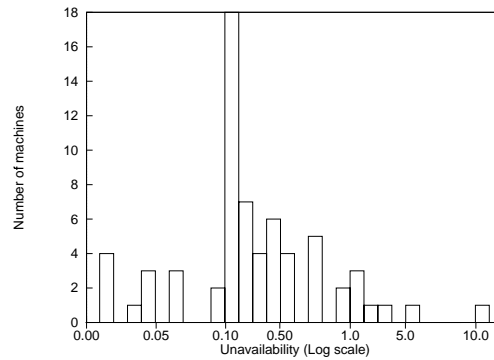
Having estimated machine up-time and down-time, we can estimate the availability of each machine. The availability is evaluated as the ratio:

$$[\langle \text{average up-time} \rangle / (\langle \text{average up-time} \rangle + \langle \text{average down-time} \rangle)] * 100$$

**Table 6.** Machine Availability

Item	Value
Number of machines	66
Maximum	99.99
Minimum	89.39
Median	99.76
Average	99.35
Standard Deviation	1.52

Table 6 summarizes the availability measurements. As it depicts, the majority of the machines have an availability of 99.7% or higher. Also there is not a great deal of variation among the individual values. This is surprising considering the rather large degree of variation in the average up-times. It follows that machines with smaller average up-times also had correspondingly smaller average down-times, so that the ratios are not very different. Hence, broadly speaking, the domain has two types of machines: those that reboot often but recover quickly and those that stay up relatively longer but take longer to recover from a failure.



an availability of 99.9% or higher. However, nearly 90% of the machines had an availability of 99% or higher. It should be noted that these numbers indicate the fraction of time the machine is alive. They do not necessarily indicate the ability of the machine to provide useful service because the machine could be alive but still unable to provide the service expected of it. To elaborate, each of the machines in the domain acts as a mail server for a set of user machines. Hence, if any of these mail servers has problems that prevent it from receiving, storing, forwarding, or delivering mail, then that server would effectively be unavailable to the user machines even though it is up and running. Hence, to obtain a better estimate of machine availability, it is necessary to examine how long the machine is actually able to provide service to user machines.

## 7 Modeling Machine Behavior

**Table 7.** Machine States

State Name	Main Events (id/source/severity)	Explanation
Reboot	6005/EventLog/4	Machine logs reboot and other initialization events
Functional	5715/NETLOGON/4 1016/MSExchangeIS Private/8	Machine logs successful communication with PDC
Connectivity problems	3096/NETLOGON/1 5719/NETLOGON/1	Problems locating the PDC
Startup problems	7000/Service Control Manager/1 7001/Service Control Manager/1	Some system component or application failed to startup
MTA problems	2206/MSExchangeMTA/2 2207/MSExchangeMTA/2	Message Transfer Agent has problems with some internal databases
Adapter problems	4105/CpqNF3/1 4106/CpqNF3/1	The NetFlex Adapter driver reports problems
Temporary MTA problems	9322/MSExchangeMTA/4 9277/MSExchangeMTA/2 3175/MSExchangeMTA/2 1209/MSExchangeMTA/2	Message Transfer Agent reports problems of a temporary (or less severe) nature
Server problems	2006/Srv/1	Server component reports having received badly formatted requests
BROWSER problems	8021/BROWSER/2 8032/BROWSER/1	Browser reports inability to contact the master browser
Disk problems	11/Cpq32fs2/1 5/Cpq32fs2/1 9/Cpqarray/1 11/Cpqarray/1	Disk drivers report problems
Tape problems	15/dlittape/1	Tape driver reports problems
Snmppelea problems	3006/Snmppelea/1	Snmp event log agent reports error while reading an event log record
Shutdown	8033/BROWSER/4 1003/MSExchangeSA/4	Application/machine shutdown in progress

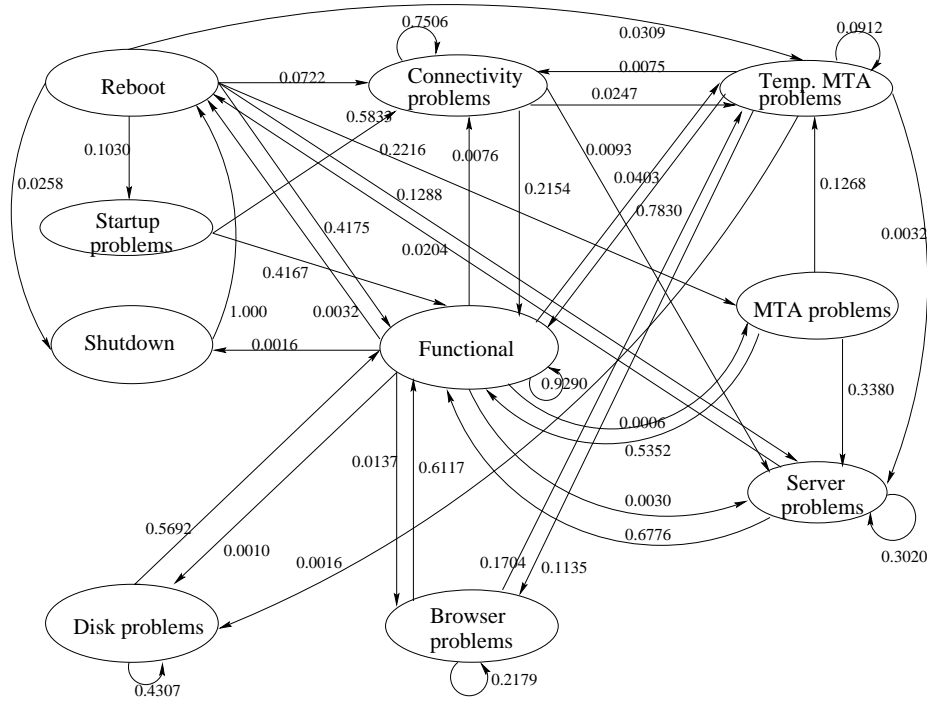
To obtain more accurate estimates of machine availability, we modeled the behavior of a typical machine in terms of a *finite state machine*. The model was based on the events that each machine logs. In the model, each state represents a level of functionality of the machine. A machine is either in a fully functional state, in which it logs events that indicate normal activity, or in a partially functional state, in which it logs events that indicate problems of a specific nature.

Selection and assignment of states to a machine was performed as follows. The logs were split into time-windows of one hour each. For each such window, the machine was assigned a state, which it occupied throughout the duration of the window. The assignment was based on the events that the machine logged in the window. Table 7 describes the states identified for the model.

Each machine (except the Primary Domain Controller (PDC) whose transitions were different from the rest) in the domain was modeled in terms of the states mentioned in the table. A hypothetical machine was created by combining the transitions of all the individual machines and filtering out transitions that occurred less frequently. Fig. 3 describes this hypothetical machine. In the figure, the weight on each outgoing edge represents the fraction of all transitions from the originating state (i.e., tail of the arrow) that end up in a given terminating state (i.e., head of the arrow). For example, if there is an edge from state A to state B with a weight of 0.5, then it would indicate that 50

From Fig. 3 the following observations can be made:

- Only about 40% of the transitions out of the *Reboot* states are to the *Functional* state. This indicates that in the majority of the cases, either the reboot is not able to solve the original problem, or it creates new ones.
- More than half of the transitions out of the *Startup problems* are to the *Connectivity problems* state. Thus, the majority of the startup problems are related to components that participate in network activity.
- Most of the problems that appear when the machine is functional are related to network activity. Problems with the disk and other components are less frequent.
- The self loops on states *Connectivity problems*, *Disk problems*, *Browser problems*, and *Server problems* indicate that these problems might be more persistent than others.
- More than 50% of the transitions out of *Disk problems* state are to the *Functional* state. Also, we do not observe any significant transitions from the *Disk problems* state to other states. This could be due to one or more of the following:
  1. The machines are equipped with redundant disks so that even if one of them is down, the functionality is not disrupted in a major way.
  2. The disk problems, though persistent, are not severe enough to disrupt normal activity (maybe retries to access the disk succeed).
  3. The activities that are considered to be representative of the *Functional* state may not involve much disk activity.



**Fig. 3.** State Transitions of a Typical Machine

- Over 11% of the transitions out of the *Temporary MTA problems* state are to the *Browser problems* state. We suspect that there was a local problem that caused RPCs to timeout or fail and caused problems for the MTA and BROWSER. Another possibility is that, in both cases, it was the same remote machine that could not be contacted. Based on the available data, it was not possible to determine the real cause of the problem.

To view the transitions from a different perspective, we computed the weight of each outgoing edge as a fraction of all the transitions in the finite state machine. Such a computation provided some interesting insights, which are enumerated below:

1. Nearly 10% of all the transitions are between the *Functional* and *Temporary MTA problems* states. These MTA problems are typically problems with some RPC calls (either failing or being canceled).
2. About 0.5% (1 in 200) of all transitions are to the *Reboot* state.
3. The majority of the transitions into the *MTA problems* state are from the *Reboot* state. Thus, MTA problems are primarily problems that occur at startup. In contrast, the majority of the transitions into the *Server problems* state and the *Browser problems* state (excluding the self loops) are from

the *Functional* state. So, these problems (or at least a significant fraction of them) typically appear after the machine is functional.

4. 92.16% of all transitions are into the *Functional* state. This figure is approximately a measure of the average time the hypothetical machine spends in the functional state. Hence it is a measure of the average availability of a typical machine. In this case, availability measures the ability of the machine to provide service, not just to stay alive.

In summary, a detailed examination of the behavior of the individual machines in the domain revealed:

- There is an average up-time of over 11 days and an average down-time of about 2 hours.
- There is a great degree of variation in the average up-time and down-time exhibited by each machine. Some machines stay up for over a month on the average, whereas more than 50% of the machines stay up for less than 2 weeks.
- The behavior of a majority of the machines can be characterized by a finite state model. Examination of the model indicates that problems related to connectivity appear frequently and might be persistent. It also indicates that temporary RPC problems or non-severe problems appear to be dominant.
- About 92% of all transitions are into the *Functional* state. Thus, a typical machine on an average would be able to provide useful service about 92% of the time.
- Average availability (measured as ratio involving up-time and down-time) was estimated to be 99.35%.

## 8 Modeling Domain Behavior

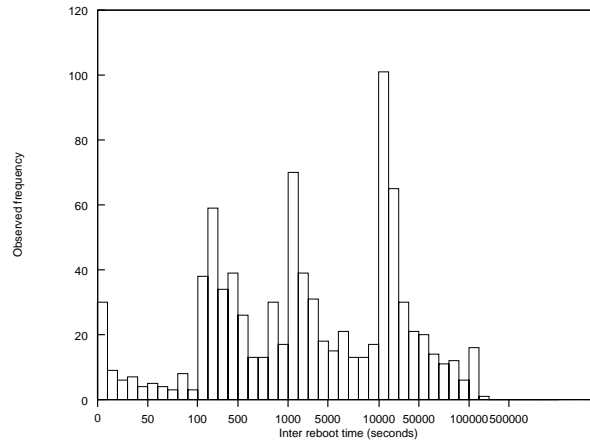
Analyzing system behavior from the perspective of the whole domain (1) provides a macroscopic view of the system rather than a machine-specific view, (2) helps to characterize the nature of interactions in the network, and (3) aids in identifying potential reliability bottlenecks and suggests ways to improve resilience to operational faults.

**Table 8.** Inter-reboot Time Statistics for the Domain

Item	Value
Number of samples	882
Maximum	2.46 days
Minimum	Less than 1 second
Median	2402 seconds
Average	4.09 hours
Standard Deviation	7.52 hours

*Inter-reboot Times.* An important characteristic of the domain is how often reboots occur within it. To examine this, the whole domain is treated as a black box, and every reboot of every machine in the domain is considered to be a reboot of the black box. Table 8 shows the statistics of such inter-reboot times measured across the whole domain.

Fig. 4 shows the distribution of inter-reboot times across the domain. Over a third of the reboots in the domain occur within 1000 seconds of each other. Probing these reboots further, about 25% of them (about 8% of the total) are due to the same machine being rebooted multiple times. Also, only about 43% of them occur during the working hours. So, it appears that the majority of such reboots are primarily due to planned shutdowns for maintenance. However, it is also possible that some of the reboots are correlated and that they are due to common cause. Such reboots are indications of propagated failures across the domain.



**Fig. 4.** Inter-reboot Time Distribution for the Domain

### 8.1 Finite State Model of the Domain

The proper functioning of the domain relies on the proper functioning of the PDC and its interactions with the Backup Domain Controllers (BDCs). Thus it would seem useful to represent the domain in terms of how many BDCs are alive at any given moment and also in terms of the PDC being functional or not. Accordingly, a finite state model was constructed as follows:

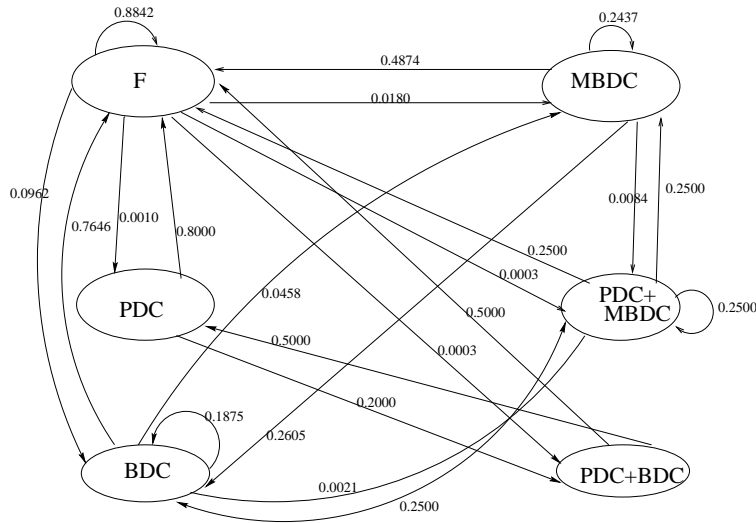
1. The data collection period was broken up into time windows of a fixed length,
2. For each such time window, the state of the domain was computed, and
3. A transition diagram was constructed based on the state information.

The state of the domain during a given time window was computed by evaluating the number of machines that rebooted during that time window. More specifically, the states were identified as shown in Table 9.

**Table 9.** Domain States and Their Interpretation

State Name	Meaning
PDC	Primary Domain Controller (PDC) rebooted
BDC	1 Backup Domain Controller (BDC) rebooted
MBDC	Many BDCs rebooted
PDC+BDC	PDC and One BDC rebooted
PDC+MBDC	PDC and Many BDCs rebooted
F	Functional (no reboots observed)

Fig. 5 shows the transitions in the domain. The results are based on the analysis of event logs spanning a 6-month period. Each time window was one hour long.



**Fig. 5.** Domain State Transitions

Fig. 5 reveals some interesting insights.

1. Nearly 77% of all transitions from the F state, excluding the self-loops, are to the BDC state. If these transitions do indeed result in disruption in service, then it is possible to improve the overall availability significantly just by tolerating single machine failures.

2. A non-negligible number of transitions are between the F state and the MBDC state. This would potentially indicate correlated failures and recovery among BDCs. (These transitions are explored in detail later.)
3. Again, a non-negligible fraction of transitions are between states BDC and MBDC. This would indicate potentially correlated failures and recovery (explored in detail later).
4. Majority of transitions from state PDC are to state F. This could be explained by one of the following:
  - Most of the problems with the PDC are not propagated to the BDCs,
  - The PDC typically recovers before any such propagation takes effect on the BDCs, or
  - The problems on the PDC are not severe enough to bring it down, but they might worsen as they propagate to the BDCs and force a reboot.

However, 20% of the transitions from the PDC state are to the PDC+BDC state. So there is a possibility of the propagation of failures. We obtain further useful insights into the dynamics of the domain by considering what percentage of the total transitions of the domain go into each state. Table 10 shows the details.

**Table 10.** Domain Transitions by State

Target state	Percentage of total transitions
F	85.68
PDC	0.11
BDC	11.04
MBDC	2.74
PDC+BDC	0.04
PDC+MBDC	0.08

From Table 10 we observe the following:

- Less than 1% of all transitions involve the reboot of the PDC. So the PDC exhibits a higher availability than the BDCs.
- Over 11% of all transitions are into the BDC state. If the domain is configured to tolerate the failure of a single BDC, then disruption of mail service could be reduced significantly.
- A non-negligible percentage (2.74%) of transitions end in MBDC state. Since each BDC is responsible for handling part of the mail service, the MBDC state represents a potentially serious disruption of service.

In summary, we conclude the following:

- Of the reboots in the domain 50% occur within 40 minutes of each other. Moreover, about 30% of them occur within 15 minutes of each other. Thus, there exists the possibility of correlated/propagated failures.

- The domain is in the MBDC state for about 3% of the time, on average. Since each machine contributes to the mail delivery, this represents a potentially serious disruption of service.
- About 77% of the transitions from the F state (excluding the self-loop) are due to the rebooting of a single machine. Thus, providing single-machine failure tolerance could significantly improve the overall availability of the domain.
- A non-negligible fraction of the domain transitions are between states BDC and MBDC. These transitions suggest the possibility of correlated failures and recovery.

## 9 Error Propagation

Examination of the domain-wide behavior suggests the possibility of correlated/propagated failures. Below, we examine indications of propagation of problems within the domain. It should be noted that the networked-based systems are usually loosely coupled; consequently the propagation of failures may not be as strong or easily identifiable as in tightly coupled systems.

*Classification of prominent events.* An approach to studying propagation of failures in the domain is to classify each occurrence of the prominent events (i.e., the events upon which reboots in our target system were categorized) as caused by one of the following:

1. a problem specific to the local machine (i.e., the machine on which the event was observed),
2. a problem related to a remote machine (this could be true in case of connectivity problems), or
3. a general, network-related problem.

Each occurrence of a prominent event that was caused by a remote-machine problem would indicate propagation of failure.

*Approach to classifying prominent events.* Classification of the occurrences of prominent events was based on tests designed using event-specific information. These tests were designed as follows. Essentially, for each event occurrence, the entire set of events occurring in the domain around that event (e.g., within an hour of it) was obtained. Based on the events that were found in this *event window*, the current occurrence of the event was classified into one of the above-mentioned categories.

*The anatomy of a test.* The tests were generated based on manual analysis of events. However, they were applied to events in an automated fashion. Thus, the test had to be specified in a format conducive to interpretation by a script. A sample test is shown below.

```
Event 2006 Srv
MP_test_begin
  4320 NetBT 7200 7 local-machine yes 1 1
  3013 Rdr 3600 8 local-machine 7 !local-machine yes 3 1
```

```

5711 NETLOGON +3600 7 local-machine yes 1 2
5715 NETLOGON +3600 7 local-machine yes 1 2
6005 EventLog +3600 7 local-machine yes 1 1
2006 Srv 3600 7 local-machine yes 3 3
MP_test_end

```

The first line indicates the *event id* and the *source* of the event for which the test is applicable. The keywords *MP\_test\_begin* and *MP\_test\_end* denote that the set of entries in between comprise a test for a local machine problem (MP). Each line between these two keywords corresponds to a *test event*. A typical test event description contains the following fields:

- event id (4320),
- source logging the event (NetBT),
- time window within which the event should occur relative to the event being tested (7200) (a plus sign before the number indicates that the test event should occur after the event being tested), and
- other constraints on the test event observed.

The constraints on the test event are better understood with an example. Let us consider the first test event in the sample shown above.

1. The entries *7 local-machine*, indicate that the test will be passed only if an occurrence of the test event is found whose 7th field matches the local machine on which the event being tested was logged. If the same entries read *7 !local-machine*, then the test would be passed only if the 7th field did *not* match the machine name.
2. The field *yes* indicates that if a test event satisfying all the constraints is found in the event window, then that test is considered passed. A *No* for that field would indicate that the test would pass if no such test event was found.
3. The last two fields in the test event indicate, respectively, the *number of times* the event must occur (for the test to be passed) and the *group* to which the test event belongs.

*Application of the test.* A specific instance of a prominent event is considered to be due to a local machine problem if it passes the test designed for that purpose. The tests are applied in *groups*. If any one test in a group is passed by the given event instance, then the whole group is considered to be passed. If the majority of the groups are passed, then the test itself is considered to have passed. The tests for a remote machine problem and a network problem are performed in a similar manner.

*Designing a test.* The test events are framed based on manual analysis of the events. It is instructive to explore in detail the sample test event depicted above. The event to be tested has an *event id* of 2006 and is logged by the *Srv* (server). The text description for this event states that the *server received an incorrectly formatted request from an unknown source*. To evaluate if a given occurrence of this event represents a problem specific to the local machine, we perform the following tests:

1. Do we observe a problem with the transport service in the local machine? (This corresponds to event 4320.)
2. Do we observe at least three instances of the Redirector service on a remote machine trying to contact the Server on the local machine and reporting a failure? (This corresponds to event 3013.)
3. Does the local machine show evidence of normal network activity within an hour of the 2006 event? (This is indicated by the 5711/5715 event.)
4. Do we observe a reboot of the local machine within one hour of the 2006 event? (This corresponds to event 6005.) The reasoning behind this is that the system administrator might have suspected a problem with the local machine and attempted to solve it by rebooting the machine (possibly after some maintenance).
5. Do we observe at least three instances of 2006 from the same machine in a 1-hour period? (This corresponds to event 2006.)

As evident from the explanation, designing test events involves a heuristic approach. However, in the absence of more specific event descriptions, this seems to be reasonable. In an ideal situation, the accuracy of these tests could be improved by validating the predictions against operator logs that report the actual cause of the problem. However, in the data that was analyzed, no operator log was available.

*Results of the tests.* Table 11 summarizes the results obtained on selected events.

**Table 11.** Breakup of Selected Events

Event	Source	Severity	Number	Local Machine Problem	Remote Machine Problem	Unknown
3012	Redirector	1	218	218	0	0
8021	Browser	1	790	261	0	509
2006	Server	1	19586	9907	0	9679

It is observed that most of the identifiable problems seem to be local machine related. This would suggest that propagation of failures is not observed on a regular basis. However, in quite a few cases, the tests were not able to classify the event one way or another. It is possible that some of these unknowns represent propagated failures. Besides, designing such tests requires a fair amount of operating-system-specific knowledge, which is not easy to obtain. However, the idea of designing event-specific tests and using them does appear to be a reasonable approach to implementing automated propagation detection.

## 10 Reliability of Internet Hosts

The previous sections used examples of failure data collected from networked systems configured as Local Area Networks. With the ever-increasing use and rapid expansion of the Internet, connection to a host from a user site, and the reliability of this connection are becoming increasingly important. The key question is whether the approaches to collecting and analyzing failure data discussed here are applicable in assessing the availability of Wide Area Networks such as Internet. From the perspective of an *average Internet user* (i.e., the majority of users that access various Web sites to obtain information from them) how can we address issues such as:

1. What is the (stationary) probability that a user request to access an Internet host succeeds?
2. On an average, what percentage of hosts might remain accessible to the user at a given moment?
3. What are the major causes of access failures as seen by the user?
4. Typically, how long could a host be unavailable to the user?

To answer these questions, we will use the results of a 40-day availability study of Internet hosts for the 97 most popular Web sites [17]. The term *failure*, in this context, refers to the inability to contact a host and fetch an HTML file from it. The failure might be due to problems with the host (e.g., the host is too busy handling other requests) or problems with the underlying network (e.g., no proper route to the host exists). Thus, the failure is an integration of the behavior of the remote host, the intermediate routers, and the local network at the user site. We believe that this corresponds well to the reliability of the "black box" between a user's local machine and a remote host. Although network instability and absolute failures of remote hosts are individually important, it is the combined impact of these failures that is critical for the end user.

### 10.1 Data Collection

We selected 97 of the top 100 Web sites, as rated by the PC Magazine [31], for the study. The data collected during our study consists of success/failure logs corresponding to attempts to connect to and fetch an HTML file from a Web site. On a failure, the cause of the failure, if recognizable, was also recorded. A sample of the logs is shown below:

```
Server www.cnn.com time: 5-3-97 : 7:48:23;
Downloaded 22009 bytes successfully
```

```
Server www.columbiahouse.com time : 5-3-97 : 7:48:24;
server returned error 302, Trying with new URL:
http://www.columbiahouse.com/?86264561812817419329132317
Downloaded 11260 bytes successfully
```

```
Server www.americangreetings.com time : 5-3-97 : 7:48:35;
```

```
Connection failed with error code Connection timed out
```

```
Server www.egghead.com time : 5-6-97 : 6:17:25;
server returned error 500
```

The logs include the following information: (1) server (Web site) name, (2) date and time of the attempt, and (3) result of the attempt. The sample above depicts the outcome of four separate access (the term *access* refers to the process of connecting to a Web site and fetching a HTML file from it) attempts, each to a different Web site. The first attempt (www.cnn.com) was a success. The second attempt (www.columbiahouse.com), though successful, required a re-attempt as the HTTP server first returned an error response. The third attempt (www.americangreetings.com) was a failure due to a problem with connection establishment. The last entry in the sample depicts another failure (www.egghead.com), which was due to an error response from the HTTP server at the Web site.

The data collection was performed by a tool (a Perl script) that ran continuously at the test site. The tool maintains a list of the Web sites under study. It periodically iterates through the list. During an iteration, the tool attempts to access each of the Web sites, one after the other. Web sites that are found to be inaccessible during a regular iteration are handled by the tool as follows. Initially, the tool attempts to access each of these sites once every five minutes. If such an attempt succeeds, the frequency of access attempts to that Web site is reduced to the default rate (once every two hours). However, if failures continue, the frequency of attempts to access that particular Web site is gradually reduced until it reaches the original rate of once every two hours, and then the whole process repeats. In addition, the first few re-attempts to access a Web site (i.e., after a failure has been encountered) are supplemented by simultaneous runs of the traceroute program on the same site. This aids in detecting routing problems that might be the cause of the problem. The default period of a single iterations is two hours. This value was determined by considering the following: (a) requirements on the sample frequency should be high enough to provide insights into the daily behavior of the servers, but (b) it should not be so high that it would artificially increase the workload on the Web sites significantly and cause congestion at the test site. Such a scenario would bias the results and possibly lead to an underestimation of the accessibility and availability of the Web sites. The motivation behind this variation of the rate of access attempts is to get a tight upper bound for the duration of the failure and to progressively "back off" if the failure seems to be of long duration.

## 10.2 Identification of Internet Hosts Accessibility Parameters

We commence the discussion with a brief examination of the parameters that, we believe, reflect the accessibility and reliability of Internet hosts as seen from the user's perspective. Table 12 lists these parameters and their corresponding measured (mean).

**Table 12.** Parameters to Describe Accessibility/Availability of Internet Hosts

Parameter	Value
Average success frequency	94.4
Failure duration	43.022 min (mean)
Interfailure time	4.016 days (mean)
Hourly failure rate	2.16 (mean)
Modes of failure	Connection Timeouts (42.8%) Connection Refusals (27.0%)
Mean availability	0.993

The *average success frequency* (ASF) for a Web site is defined as the percentage of HTML file-fetch attempts to that Web site that are successful. It functions as an indicator of host accessibility, since the greater the ASF, the more accessible the host. We obtained a high mean ASF indicating that, on an average, host accessibility is high.

The *failure duration* is the amount of time for which a particular remote host is inaccessible to the user. It represents the collective failure of the host and the network and is a measure of the ability of the host/network to recover. Though we observed the mean failure duration to be around 43 minutes, most (70%) of the failures lasted less than 15 minutes.

The *interfailure time* is the time period between two consecutive instances of failure of the same remote host. It provides an estimate of the degree of nonstop operation of the host. Though the mean value was around 4 days, individual sites differed greatly in their mean interfailure time.

The *hourly failure rate* is the percentage of Web sites that fail at least once during any given hour interval. It is a measure of the fraction of the network unavailable to the user at any time (the granularity being an hour). We observed that, on average, only 2% of the hosts are unavailable at any time.

The *modes of failure* provide insights into the actual causes of the failures. Our study found that failures due to connection timeouts and connection refusals are the two most frequently observed failure modes. Also, the majority of connection timeout failures appear to be network related.

The *mean availability* is defined as the ratio of interfailure time (mean) to the sum of the failure duration (mean) and the interfailure time (mean). It denotes, on average, what fraction of the time a remote host is available. The data showed that the mean availability was very high, with a small variation.

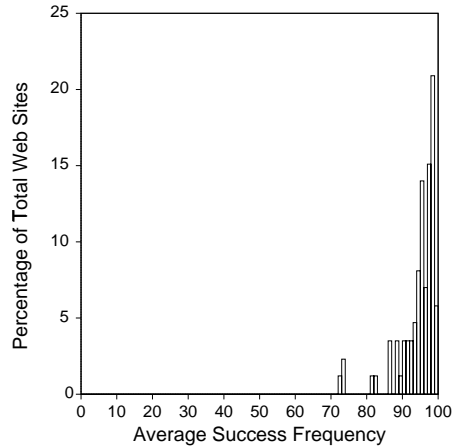
### 10.3 Average Success Frequency (ASF)

The ASF is a measure of the stationary probability of the success of a given file-fetch request to a host. The average success frequency for 86 servers<sup>3</sup> is given in Table 13.

**Table 13.** Measured ASF

Parameter	Value
mean	$94.4 \pm 6.3$ (90%)
median	95.8
standard deviation	5.6

The stationary probability of an HTML file-fetch attempt being successful is 0.944. Fig. 6 shows the distribution of ASF for this reduced set. The distribution appears highly skewed, with most of the Web sites (95.3%) showing an ASF of 80 or more. A high percentage (83.7%) of the sites showed an ASF of 90 or more. A small fraction (about 4.7%) exhibited a lower ASF (less than 80) and were consequently less accessible than the rest. We also examined the distribution of the average failure frequency, defined as  $[(1-ASF)/100]$ . The distribution fitted an exponential distribution at a significance level of 0.05.



**Fig. 6.** Distribution of ASF

<sup>3</sup> Eleven of the Web sites almost always responded with error codes 302/404 (which are HTTP-specific errors) or with a connection refused error message; hence they were excluded from further analysis.

## 10.4 Failure Duration

The failure-duration parameter is an upper bound on the duration of an instance of failure. As discussed earlier, this parameter indicates the ability of the network/remote host to recover from a failure.

In this context, each instance of failure represents the string of consecutive unsuccessful attempts to access the same Web site (i.e., successive failed attempts were coalesced into a single instance of failure). Thus the failure duration truly represents the time period for which the host is inaccessible. Table 14 summarizes the results for the measured failure duration. The observed mean failure duration (MFD) is distorted somewhat by a small fraction of failures that lasted much longer (a day or more) than the rest. The median however, is only slightly greater than 5 minutes. On closer examination, we found that most of the failures had a much shorter duration than the MFD. Fig. 7 shows the distribution of the failure duration. More than a third (37.2%) of the failures lasted less than 5 minutes. Nearly a third (33.3%) of the failures lasted between 5 and 15 minutes. Approximately 18% of the failures lasted longer than 30 minutes. There were only 3 instances of failure that lasted longer than a day. The longest failure recorded lasted 3.375 days. The peak at the 5-6 minute range was predominantly due to two kinds of failures: failures due to a connection refusal by the server, and failures due to a timeout<sup>4</sup> while reading the HTTP header that precedes the HTML file.

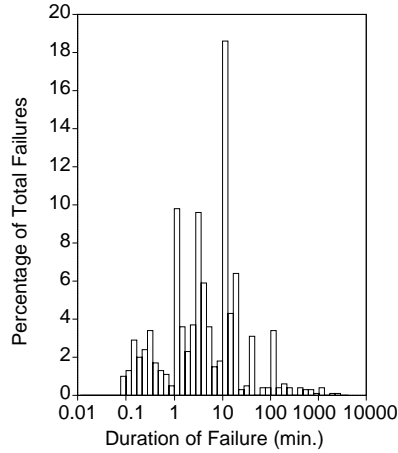
**Table 14.** Measured Failure Duration

Failure Instances	Mean(min)/ Confidence	Standard Deviation	Median (min)
783	43.022 ± 41.990 (75%)	233.504 min	6.667

In order to understand the failures better, we further categorized them based on the number of unsuccessful access attempts made during the failure period. Specifically a *one-time failure* is defined as an instance of failure that involves only one unsuccessful access attempt. On the other hand, a *multiple failure* is defined as an instance of failure that involves two or more unsuccessful access attempts.

We observed that 44.8% of the failures were multiple failures. This would suggest that, once a failure is encountered, the probability of an immediate re-attempt being a success is not high. We examined the results of all re-attempts (on encountering a failure) made within five minutes of encountering a failure. Out of the 1268 such re-attempts, only 38.2% succeeded, indicating that the probability that an immediate re-attempt will succeed is very low. The value

<sup>4</sup> Once the tool successfully establishes connection with the HTTP server process on the remote host, it waits for a predefined timeout interval to receive the HTTP header from the server.



**Fig. 7.** Duration of Failure (min.)

for MFD obtained in our study is significantly different from the values for the mean time to repair (MTTR) obtained in [24]. The discrepancy, we feel, is due to the difference in the types of hosts evaluated. In [24] the sample included hosts with a widely varying degree of reliability. Many of these hosts used to be shutdown nightly or weekly for maintenance purposes (which would significantly increase MTTR). In our study, however, the hosts were dedicated Web sites that receive millions of hits every day and are designed to stay up most of the time (i.e., shutdowns are very minimal. This is reinforced by the fact that most of the failures lasted only 15 minutes or less, which is too short for a regular shutdown. Hence we observed a lower MFD.

### 10.5 Classification of the Causes of Failure

We classify failures based on the contents of the error messages associated with them. An error message can be returned either by the connect system call or the HTTP server on the remote host. Table 15 summarizes the distribution of failures based on the error message associated with the failure. In this case, each failed access attempt was considered as a unique instance of failure (i.e., no coalescing of failures was performed.).

The following error categories are related to the host: (1) connection refused (the network was able to reach the host but the host did not respond favorably to the request) and (2) server returned error 500/401. Errors 500 (internal error) and 401 (unauthorized request) indicate a problem with the HTTP server on the host.

The following error categories are related to the network: (1) no route to host and (2) network is unreachable. Connection timeout failures and failures due to timeout while reading the header could be due to either the host or the network.

**Table 15.** Cause-wise Distribution of Failures

Cause of Failure	Number of Failures	Percentage
Connection timed out	1073	42.8
Connection refused	677	27.0
Could not read HTTP header within the timeout interval	531	21.2
No route to host	92	3.7
Server returned error 500	76	3.0
Server returned error 401	51	2.0
Network is unreachable	1	0.0

### 10.6 Connection Timeouts and Timeouts While Reading Header

Connection timeouts can arise due to (among other factors) a route establishment problem, the server having too many pending connect requests at that time, or congestion along the route to the server. Timeouts while reading the header can occur due to sudden failure of the host/network or (more plausibly) temporary congestion along the route. To categorize these failures as network-related problems or host-related problems, we use the results of the traceroute runs. Specifically, we examine all traceroute runs to each Web site that were executed within five minutes before/after facing a connection timeout or a timeout while reading header. Table 16 shows the results of such traceroutes.

**Table 16.** Summary of Traceroute Runs Made While Experiencing Timeouts

Parameter	Connection Timeout	Header-Read Timeout
Total instances of failure	1073	531
Total traceroutes executed within 5 minutes of failure	659	436
Number of such traceroutes that failed to reach the host	434 (66%)	100 (23%)

Evidently, the majority of the traceroute runs executed on facing connection timeouts failed. Moreover, most such failures occurred at one of the intermediate routers, (i.e., not near the host) clearly indicating a network problem. Thus the majority of these traceroutes failed due to network-related problems. Since each such traceroute was executed within minutes of a connection timeout failure, we can conclude that the corresponding connection timeout also failed due to a network problem. If we extrapolate this result to cover all the connection timeouts, about 66% of the connection timeouts are network related.

For the timeouts while reading the header, however, most traceroute runs reached the host. Since a successful traceroute run implies proper functioning of both the host and the network, the actual cause of failure is not evident. However, since the host did respond favorably to a connection request a few seconds before

the failure (the tool waits for the header only if it is able to successfully connect to the HTTP server), and since such hosts (dedicated to maintaining the Web site) typically have some built-in fault-tolerance, a sudden host failure seems unlikely. A more plausible explanation is a temporary network problem, such as congestion along the route to the host, that might have delayed the reception of the header. Thus, with a small margin of error, we can conclude that almost all of these failures were due to network problems (though the problem may not be as severe as in the case of connection timeouts, which lasted much longer). Thus, approximately 53% ( $42.8 * 0.66 + 21.2 + 3.7$ ) of the failures are network related.

### 10.7 Comparing of the Major Classes of Failure

A comparative study of the three major classes of failure (i.e. connection timeouts, connection refusals and timeouts while reading header) yields useful insights. We focus on the failure duration and the inter-failure time for each class of failure. For this study, all consecutive access attempts that failed with the same error message are coalesced into a single instance of failure. Table 17 and Table 18 summarize the failure duration and the inter-failure time, respectively.

**Table 17.** Comparison of Failure Duration for Different Classes of Failures

Cause of Failure	Mean (min)	Median (min)	Standard Deviation (min)
Connection timeout	46.558	12.117	134.106
Connection refused	30.259	6.742	191.815
Timeout while reading header	17.486	2.725	76.714

**Table 18.** Comparison of Interfailure Time for Different Classes of Failures

Cause of failure	Mean (days)	Median (days)	Standard Deviation (days)
Connection timeout	5.652	4.002	6.077
Connection refused	7.274	4.076	7.980
Timeout while reading header	2.695	1.080	3.882

These tables reveal the following:

1. Connection timeouts last longer than most other failures. Since majority of the connection timeouts are network related, it appears that the network is relatively slow in recovering from failures as compared to the remote host.

2. Timeouts while reading the header lasted for a much shorter period than the other failures. (In fact nearly 37% of these failures lasted less than a minute.)
3. Connection refusals occur less frequently than the other two classes.

In summary, we observe the following: (1) connection timeouts (majority of which are network related) and connection refusals (which are host related) account for nearly 70% of the failures, (2) network related failures marginally outnumber host- related failures, and (3) network related failures last longer than host related ones.

### 10.8 Mean Availability

To quantitatively describe the behavior of the hosts from the user’s perspective, we evaluated the *mean availability* of the hosts. The mean availability is defined as this ratio:

$$(\text{Mean Inter-Failure Time})/(\text{Mean Failure Duration} + \text{Mean Inter-Failure Time})$$

The mean availability evaluates to 0.993. To obtain a range for this parameter, we calculate the availability for each Web site using the corresponding mean values of failure duration and inter-failure time for that Web site. The results of this computation are shown in Table 19<sup>5</sup>.

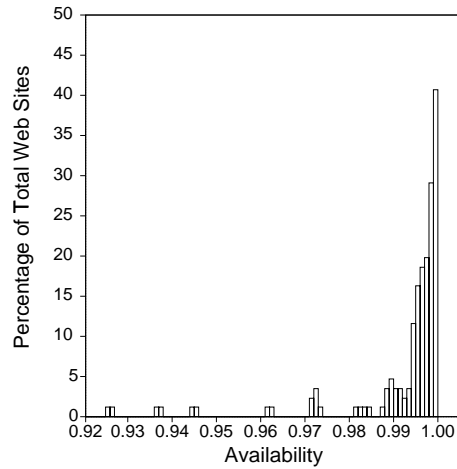
**Table 19.** Measured Availability

Parameter	Value/Confidence Interval
Mean	0.993 ± 0.067 (99%)
Standard Deviation	0.013
Median	0.997

Fig. 8 shows a histogram of the availability distribution over all 86 participant sites. The distribution is skewed, with all the sites having an availability of at least 0.92. A significant percentage (40%) of the sites exhibited an availability of 0.99 or more. However, there was no site that had an availability of 1.00. Comparing the results obtained here with the results obtained in [24], the hosts in our study exhibit a higher availability than the hosts in [24] (where the mean availability was 0.926). In summary, this study of the Internet hosts reveals:

1. Most of the user requests to the Web servers are successful, hence there is a high stationary probability (0.94) that a given request goes through successfully.
2. On an average, only about 2% of the servers fail within a given one hour interval.

<sup>5</sup> The mean of the availability of individual sites has the same value as the overall mean availability (computed using MFD and the mean inter-failure time).



**Fig. 8.** Availability

3. Connectivity problems play a major role in determining the accessibility of the hosts. Network-related failures tend to outnumber host-related failures. Also, the network appears to be slower in recovering from failures than the hosts.
4. The majority (70.5%) of the failures are short (less than 15 minutes). However, a few failures spanned several days.
5. On average, Web sites stay up for over four days without any failures, though a good fraction of them fail at least once a day.
6. Connectivity to the hosts seems to be good on average. However, we did observe a few major network-related failures that rendered nearly 70% of the hosts inaccessible for a significant period of time.
7. The mean availability of the hosts is very high (0.993).

## 11 Conclusions

In this chapter, we discussed methodologies for measurement-based analysis of computer system dependability. The discussion focused on the issues involved in analyzing the availability of networked systems using the failure data collected by the logging mechanisms built into the system. The process of collecting data and the type of analysis conducted were illustrated by representative studies that include analysis of availability of LANs (Local Area Networks) of UNIX-based workstations, LANs of Windows NT-based machines, and the Internet (from the end-user perspective).

This type of analysis enables us to study naturally occurring errors and all measurable dependability measures, such as failure and recovery rates, reliability, availability. However, the analysis is limited to detected errors. Our experience in designing and evaluating dependable systems shows that to achieve accurate

and comprehensive system dependability evaluation the analysis must span the three phases of system life: design phase, prototype phase, and operational phase (which is covered in this chapter). The thorough discussion of techniques suited for conducting dependability evaluation during the system design phase (simulated fault injection) and the system prototype phase (physical fault injection) the reader is referred to [15].

Significant progress has been made in all the three fields over the past 15 years, Increasing attention is being paid to 1) combining analytical modeling and experimental analysis and 2) combining system design and evaluation. In the first area, state-of-the-art analytical modeling techniques are being applied to real systems to evaluate various dependability and performance characteristics. Results from experimental analysis are being used to validate analytical models and to reveal practical issues that analytical modeling must address to develop more representative models. In the second area, dependability analysis tools are being combined with each other and with other CAD tools to provide an automatic design environment that incorporates multiple levels of joint evaluation of functionality, performance, dependability, and cost. Software failure data from the testing and operational phases are also providing feedback to software designers for improving software reliability.

For example, the presented study of the various kinds of failures in a network of Windows NT machines reveals useful and interesting insights into network system failure behavior.

1. Most of the problems that lead to reboots are software related. Only 10% are attributable to specific hardware components.
2. Connectivity problems contribute most to reboots.
3. Rebooting the machine does not appear to solve the problem in many cases. In about 60% of the reboots, the rebooted machine reported problems within a hour or two of the reboot.
4. Though the average availability evaluates to over 99%, a typical machine in the domain, on average, provides acceptable service only about 92% of the time.
5. About 1% of the reboots indicate memory leaks in the software.
6. There are indications of propagated or correlated failures. Typically, in such cases, multiple machines exhibit identical or similar problems at almost the same time.

## Acknowledgments

This manuscript is based on a research supported in part by the US National Aeronautic and Space Administration (NASA) under grant NAG-1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), by the Defense Advanced Research Project Agency (DARPA) under contract DABT63-94-C-0045 and by Tandem (now Compaq) Computers. The findings, opinions, and recommendations expressed herein are those of the authors and do not necessarily reflect the position or policy of the United State Government and no official endorsement

should be inferred. We are pleased to thank Fran Baker for her assistance in editing this manuscript.

## References

1. Arlitt, M.F., Williamson, C. L.: Web Server Workload Characterization: The Search for Invariants. SIGMETRICS '96. (1996) 126–137
2. Butner, S.E., Iyer R.K.: A Statistical Study of Reliability and System Load at SLAC. Proc. 10th Int. Symp. Fault-Tolerant Computing. (1980) 207–209
3. Castillo, X., Siewiorek, D.P.: Workload, Performance, and Reliability of Digital Computer Systems. Proc. 11th Int. Symp. Fault-Tolerant Computing. (1981) 84–89
4. Castillo, X., Siewiorek, D.P.: Workload Dependent Software Reliability Prediction Model. Proc. 12th Int. Symp. Fault-Tolerant Computing. (1982) 279–286
5. Chimoy, B.: Dynamics of Internet Routing Information. Proc. SIGCOMM '93. (1993) 45–52
6. Dugan, J.B.: Correlated Hardware Failures in Redundant Systems. Proc. 2nd IFIP Working Conf. Dependable Computing for Critical Applications. (1991)
7. Gray, J.: A Census of Tandem System Availability Between 1985 and 1990. IEEE Trans. Reliability. **39** (1990) 409–418
8. Hansen, J.P., Siewiorek, D.P.: Models for Time Coalescence in Event Logs. Proc. 22nd Int. Symp. Fault-Tolerant Computing. (1992) 221–227
9. Hsueh, M.C., Iyer, R.K., Trivedi, K.S.: Performability Modeling Based on Real Data: A Case Study. IEEE Trans. Computers. **37** (1988) 478–484
10. Iyer, R.K., Rossetti, D.J.: A Statistical Load Dependency Model for CPU Errors at SLAC. Proc. 12th Int. Symp. Fault-Tolerant Computing. (1982) 363–372
11. Iyer, R.K., Velardi, P.: Hardware-related Software Errors: Measurement and Analysis. IEEE Trans. Software Engineering. **SE-11** (1985) 223–231
12. Iyer, R.K., Rossetti, D.J.: Effect of System Workload on Operating System Reliability: A Study on IBM 3081. IEEE Trans. Software Engineering. **SE-11** (1985) 1438–1448
13. Iyer, R.K., Rossetti, D.J., Hsueh, M.C.: Measurement and Modeling of Computer Reliability as Affected by System Activity. ACM Trans. Computer Systems. **4** (1986) 214–237
14. Iyer, R.K., Young, L.T., Iyer, P.V.K.: Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data. IEEE Trans. Computers. **39** (1990) 525–527
15. Iyer, R.K., Tang D.: Experimental Analysis of Computer System Dependability. Chapter 5 in Fault Tolerant Computer Design, D.K. Pradhan, Prentice Hall. (1996) 282–392
16. Jacobson, V.: traceroute, <ftp://ftp.ee.lbl.gov/traceroute.tar.Z>. (1989)
17. Kalyanakrishnam, M., Iyer, R.K., Patel, J.: Reliability of Internet Hosts: A Case Study from End User's Perspective. Proc. 6th Int. Conf. on Computer Communications and Networks. (1996) 418–423
18. Kalyanakrishnam, M.: Analysis of Failures in Windows NT Systems. Master Thesis, Technical Report CRHC 98-08, University of Illinois at Urbana-Champaign. (1998)
19. Lee, I., Iyer, R.K., Tang, D.: Error/Failure Analysis Using Event Logs from Fault Tolerant Systems. Proc. 21st Int. Symp. Fault-Tolerant Computing. (1991) 10–17

20. Lee, I., Iyer, R.K.: Analysis of Software Halts in Tandem System. Proc. 3rd Int. Symp. Software Reliability Engineering. (1991) 227–236
21. Lee, I., Tang, D., Iyer, R.K., Hsueh, M.C.: Measurement-Based Evaluation of Operating System Fault Tolerance. IEEE Trans. Reliability. **42** (1993) 238–249
22. Lee, I., Iyer, R.K.: Faults, Symptoms, and Software Fault tolerance in the Tandem GUARDIAN90 Operating System. Proc. 23rd Int. Symp. Fault-Tolerant Computing. (1993) 20–29
23. Lin, T.T., Siewiorek, D.P.: Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis. IEEE Trans. Reliability. **39** (1990) 238–249
24. Long, D., Muir, A., Golding, R.: A Longitudinal Survey of Internet Host Reliability. Proc. Symposium on Reliable Distributed Systems. (1995) 2–9
25. Maxon, A.: Anomaly Detection for Diagnosis. Proc. 20th Int. Symp. Fault-Tolerant Computing. (1990) 20–27
26. Maxon, R.A., Feather, F.E.: A Case Study of Ethernet Anomalies in a Distributed Computing Environment. IEEE Trans. Reliability. **39** (1990) 433–443
27. Maxon, R.A., Olszewski, R.T.: Detection and Discrimination of Injected Network Faults. Proc. 23rd Int. Symp. Fault-Tolerant Computing. (1993) 198–207
28. McConnel, S.R., Siewiorek, D.P., Tsao, M.M.: The Measurement and Analysis of Transient Errors in Digital Computer Systems. Proc. 9th Int. Symp. Fault-Tolerant Computing. (1979) 67–70
29. Paxson, V.: Towards a Framework for Defining Internet Performance Metrics. Proc. INET '96.
30. Paxson, V.: End-to-End Routing Behavior in the Internet. SIGCOMM '96. (1996)
31. PC Magazine. **16** (1997) 101–124
32. Sahner, R.A., Trivedi, K.S.: Reliability Modeling Using SHARPE. IEEE Trans. Reliability. **R-36** (1987) 186–193
33. SAS User's Guide: Basics. SAS Institute. (1985)
34. Siewiorek, D.P., Kini, V., Mashburn, H., McConnel, S.R., Tsao, M.: A Case Study of C.mmp, Cm, and C.vmp: Part I - Experience with Fault Tolerance in Multiprocessor Systems. Proc. of IEEE. **66** (1978) 1178–1199
35. Siewiorek, D.P., Swarz, R.W.: Reliable Computer Systems: Design and Evaluation. Digital Press. (1992)
36. Stevens, W.R.: TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley. (1994)
37. Sullivan, M.S., Chillarege, R.: Software Defects and Their Impact on System Availability-A Study of Field Failures in Operating Systems. Proc. 21st Int. Symp. Fault-Tolerant Computing. (1991) 2–9
38. Sullivan, M.S., Chillarege, R.: A Comparison of Software Defects in Database Management Systems and Operating Systems. Proc. 22nd Int. Symp. Fault-Tolerant Computing. (1992) 475–484
39. Tang, D., Iyer, R.K., Subramani, S.: Failure Analysis and Modeling of a VAXcluster System. Proc. 20th Int. Symp. Fault-Tolerant Computing. (1990) 244–251
40. Tang, D., Iyer, R.K.: Impact of Correlated Failures on Dependability in a VAX-cluster System. Proc. 2nd IFIP Working Conf. Dependable Computing for Critical Applications. (1991)
41. Tang, D., Iyer, R.K.: Analysis and Modeling of Correlated Failures in Multicomputer Systems. IEEE Trans. Computers. **41** (1992) 567–577
42. Tang, D., Iyer, R.K.: Analysis of the VAX/VMS Error Logs in Multicomputer Environments-A Case Study of Software Dependability. Proc. 3rd Int. Symp. Software Reliability Engineering. (1992) 216–226

43. Tang, D., Iyer, R.K.: Dependability Measurement and Modeling of a Multicomputer System. *IEEE Trans. Computers.* **42** (1993) 62–75
44. Tang, D., Iyer, R.K.: MEASURE+-A Measurement-Based Dependability Analysis Package. *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems.* (1993) 110–121
45. Thakur, A., Iyer, R.K.: Analyze-NOW-An Environment for Collection and Analysis of Failures in a Network of Workstations. *IEEE Trans. Reliability.* **R-46** (1996) 561–570
46. Tsao, M.M., Siewiorek, D.P.: Trend Analysis on System Error Files. *Proc. 13th Int. Symp. Fault-Tolerant Computing.* (1983) 116–119
47. Velardi, P., Iyer, R.K.: A Study of Software Failures and Recovery in the MVS Operating System. *IEEE Trans. Computers.* **C-33** (1984) 564–568.
48. Wein, A.S., Sathaye, A.: Validating Complex Computer System Availability Models. *IEEE Trans. Reliability.* **39** (1990) 468–479

## Glossary of Terms

*Availability ( $A(t)$ )* A measure of the service delivery with respect to the alternation of the delivery and interruptions.  $A(t)$  is the probability that the system delivers a proper (conforming to specification) service at a given time  $t$ . Expected value:  $EA = MTTF / (MTTF + MTTR)$ .

*Average success frequency (ASF)* For a Web site, defined as a percentage of HTML file-fetch attempts to that Web site that are successful.

*BDC (Backup Domain Controller)* The domain typically has multiple Backup Domain Controllers that receive updates from the primary domain controller (PDC). Each BDC can potentially take over as the PDC for that domain if the original PDC fails.

*Hard failure* A failure state, in which a machine becomes unusable. An unresponsive server, a loose cable, or even a keyboard error can cause a hard failure.

*Hourly failure rate* The percentage of Web sites that fail at least once during any given hour interval. It is a measure of the fraction of the network unavailable to the user at any time (the granularity being an hour).

*Interfailure time* The time between two consecutive instances of failure of the same remote host. It provides an estimate of the degree of nonstop operation of the host.

*Machine-related failure* A computation node (workstation) encounters failure conditions because of problems that are local to that workstation.

*Maintainability ( $M(t)$ )* A measure of the service interaction.  $M(t)$  is the probability that the system will be repaired within the time less than  $t$ ; expected value: MTTR (Mean Time To Repair).

*Network-related failure* A computation node encounters a failure because of a problem in a network component. For example, a failure condition encountered on a client due to a failed server would be classified as a network-related failure.

*PDC (Primary Domain Controller)* Every Windows NT domain has a Primary Domain Controller (server), which maintains data related to membership of the domain and other relevant information.

*Reliability ( $R(t)$ )* A measure of the continuous delivery of service.  $R(t)$  is the probability that the system survives (does not fail) throughout interval  $[0, t]$ . Expected value: MTTF (Mean Time to Failure).

*Soft failure* A failure in which a system is affected in such a way that it does not become unusable. In most cases, this effect manifests as a performance loss. Failures such as disk errors and delayed responses from the server, are examples of soft failures