Dependability Issues in Smart Networks

Bjarne E. Helvik
Department of Telematics, NTNU
O. S. Bragstads Plass 2A
N-7491 Trondheim; Norway
E-mail: Bjarne.E.Helvik@item.ntnu.no

Key words: Dependability, availability, reliability, QoS, network survivability, fault-tolerance, logical faults, error persistence, error propagation, smart networks.

Abstract: The dependency of society and business on telecommunication services is commonly recognized, but is this conception taken into account in our effort towards smarter and more autonomous networks? The objective of the paper is to discuss some dependability issues in the context of these networks and to pinpoint some challenges. As an introduction, a brief review of dependability concepts is given. Next the following issues are discussed: a) strategies for providing a survivable transport network; b) fault-tolerant network nodes vs. fault-tolerant functionality on a distributed platform; c) software faults and their consequences like error propagation and network wide failure modes.

1 INTRODUCTION

Many foresee a future where we will have a “digital” existence in cyberspace that will be an integrated part of our private, social and professional life. Even if this future scenario does not become true, it is beyond doubt that our social and economic welfare is and will become increasingly more dependent on information and communication technology. However, when we pursue new technologies toward smarter networks, do we have in mind that we are going to thrust our welfare to the technology we are developing? Do we take into account that equipment may fail, that information may be deliberately or accidentally corrupted, that the environment may knock-out parts of the network, and that logical flaws in specification, design and implementation may cause instability and breakdown of an entire network?

History has shown that when failures may lead to disasters, the society tends to become conservative with respect to introducing new technology when a proven technology exists. In this situation, the technology deployment trend tends to be more towards marginal improvements than to introduce radically new solutions. The rather circumspect introduction of IN during the past decade may serve as an example. It should also be kept in mind that the additional cost of making a network dependable is similar to the cost of providing its basic functionality. To become a useful technology, smart networks must meet the above outlined conservatism and high cost, and be smart enough to deal with physical and logical faults in an economic and robust manner.

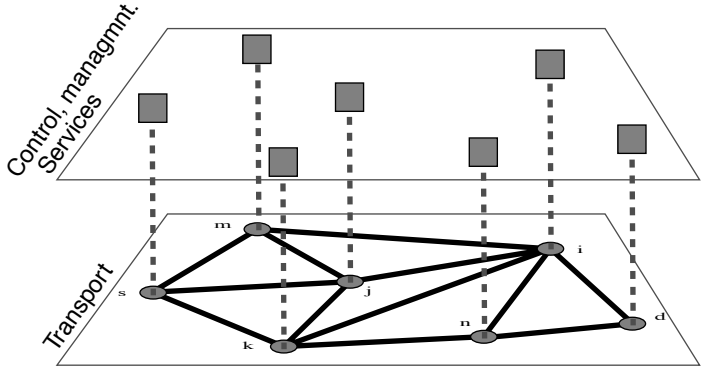


*Figure 1.*Simple two layer network model.

The objective of this paper is to introduce and discuss some issues concerning the dependability related to the coming networks. These challenges has been somewhat overlooked in the network research society. The dependability of the circuit switched communication infrastructure is taken for granted and up to recently, none has trusted their welfare on the Internet. Security issues, arising from intentionally man-made faults/intrusions, is recognized and put on the research agenda. Remember, however, that ‘nature’ causing unintentional physical and logical faults may be more inventive than man. This paper concentrates on these faults of the transport and control parts as the network as illustrated in Figure 1. In Section 3, the various options for making the transport part of the network tolerant to link and node failures are presented and trade-off with respect to speed and efficient use of spare capacity are discussed. How to provide fault tolerant service, control and management functionality are discussed in Section 4. Section 5 presents some observations of the effect of logical faults in network and points out this as a potential cause for major network outages. First, however, the basic dependability concepts are briefly revisited in the next section.

2 DEPENDABILITY

Dependability may be defined as *the trustworthiness of a system such that reliance can justifiably be placed on the service it delivers* [6]. A service delivered by a system is its behaviour, as it is perceived by its users. A user is another system (human or technical) which interacts with the former. For a more comprehensive introduction to dependability concepts, it is referred to [24][1].

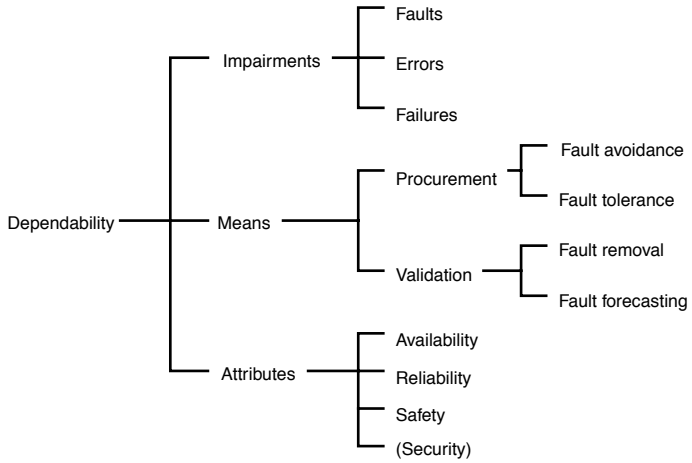


*Figure 2.*The dependability tree, linking various aspects of dependability [24].

2.1 Concepts

From an engineering point of view, we are left with the following questions: “What causes a system to be less than 100% trustworthy?”, “How should this, less than 100%, trustworthiness be quantified?” and “How should a required trustworthiness be achieved and assessed?” To handle these questions three classes of notions are introduced as illustrated in Figure 2:

- *Impairments* to dependability: faults, errors and failures, as well as their causes, consequences and characteristics. Impairments will be discussed in Section 2.2.
- *Means* to achieve reliability by fault avoidance and fault tolerance, see Section 2.3, and the means to validate and reach confidence in the result, like testing evaluation by modelling and mathematical analysis or simulation.

1. Other definitions exist. See for instance [16].

- *Attributes* of a system which describe the properties of a system with respect dependability, and whose quantification determine its dependability. The attributes may be regarded as the following abilities of a system:
  Availability: to provide a set of services at a given instant of time or at any instant within a given time interval.
  Reliability: to provide uninterrupted service.
  Safety: to provide service without the occurrence of catastrophic failures.
  Security: to prevent unauthorized access to and/or handling of information.

It is seen that the a quantification of the dependability attributes are significant QoS parameters of a system´s service together with performance/traffic related parameters as for instance response time and delay.

2.2 Fault, errors and failures

Fault, error and failure are words which in daily language are used interchangeably and mean that something does not work, is incorrect, etc. However, in dependability engineering they denote separate phases from cause to consequence as illustrated in Figure 3. Starting with the consequence:

- *Failure:* delivery of incorrect service or the transition from correct service delivery to incorrect. A failure is a manifestation of an error that we observe on the outside of a system.
- *Error:* a system state that is liable to lead to failure, i.e. the manifestation of a fault within a system.
- *Fault:* the cause of an error. Note that faults may have a multitude of physical and human causes and be of various kinds as illustrated to the right in Figure 4.

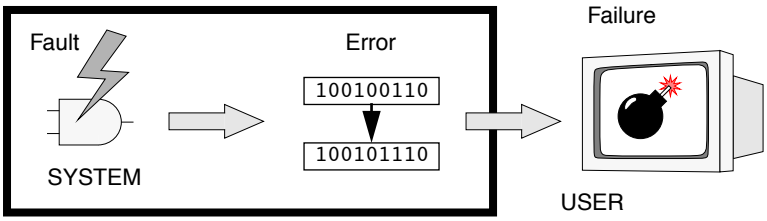


*Figure 3.*Relationship between faults, errors and failures.

This cause to consequence chain may be applied recursively with respect to both the development, deployment and operation, and the system structure. For instance: a misconception in a software developers head (fault) may lead to a wrong design (error) that results in the production of incorrect code (fail-

ure). This incorrect code represent a (dormant) fault in the system which when activated produces an error, which again may cause a failure of the system.

Often, only permanent hardware faults that are considered when the dependability of a system is regarded. This leads to a gross overestimation of the dependability, since these faults are the least frequent cause of failures. Human made logical (e.g. software) and operational faults as well as transient faults are far more frequent. See for instance [2, 10, 21].

2.3 Fault-tolerance

There are two basic approaches to obtain dependable systems, fault avoidance and fault tolerance. The basic principles are illustrated in Figure 4. In the fault avoidance approach, we seek to avoid the faults, e.g. by extensive quality assurance of the software, benign operating conditions for the hardware, etc. In the fault tolerant approach, the system is given a structure and mechanisms, which prevent errors in the system to manifest themselves as failures. A number of mature and well known techniques exist, e.g. FEC (forward error correcting codes) or CRC (cyclic redundancy check) combined with retransmission of errored packets to tolerate transient transmission faults. Other techniques is rerouting of traffic when network links and nodes fail, as introduced in Section 3 and replication of computing functionality as dealt with in Section 4. Introductions to fault tolerant design may for instance be found in [3, 17, 18, 26 Chap. 14, 28, 37].

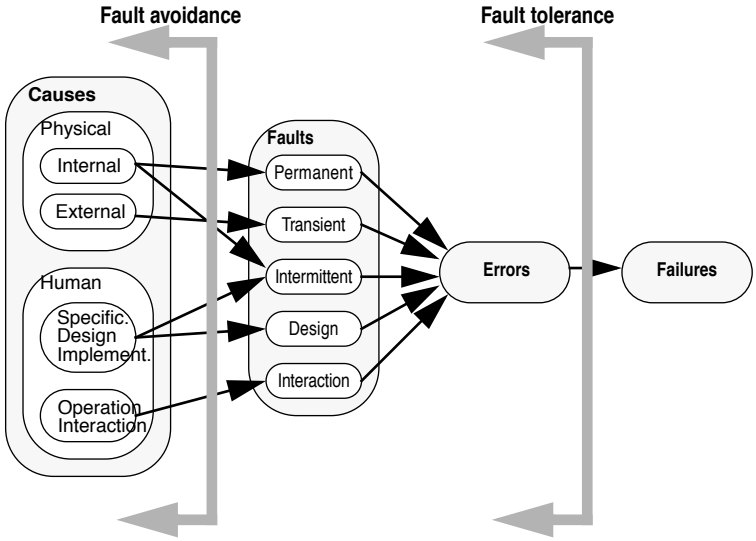


Figure 4. The design barriers of fault avoidance and tolerance to improve dependability.

Below, the options for fault-tolerance in smart network will be discussed. Neither fault avoidance nor fault tolerance comes cheap, and it is not straight forward to obtain a dependable network. Hence, smart networks should be inherently robust, i.e. fault tolerance of network services should be simple to achieve and do not require more additional development effort and deployed equipment than strictly necessary. Preferably smart networks should also minimize the deployment and operation efforts as pursued in the plug-and-play architecture [1].

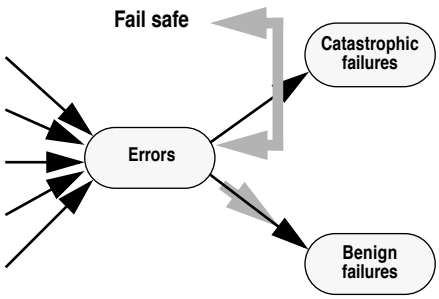


*Figure 5.*The fail-safe principle.

3 SURVIVABLE TRANSPORT

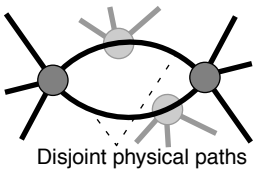
Various fault-tolerance strategies are employed to achieve dependable transport networks. There are three important issues in the choice of strategy:

- The redundant network elements and capacity needed, i.e. the additional equipment cost required to implement the scheme;
- The duration from the failure of a network element and until the transport service is restored;
- Centralized or decentralized control. Control by a dedicated management functionality or as an integrated part of the traffic handling.

In this section, three basic strategies, which differ with respect to the above listed issues are presented. Section 3.4 gives a brief discussion of these in the layered architecture and fault handling in transport networks.

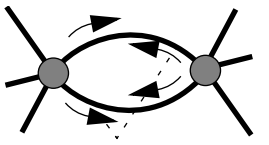
3.1 Protection

Protection is the simplest form of fault tolerance of the transport service between to nodes of a network, where a dedicated spare path is established. There may be intermediate nodes as indicated in the figure, but these do not perform any switching or rerouting of the protected traffic. The principle is, as illustrated in the figure, that the traffic between two nodes follows two (or more) dedicated paths. To have independent failing, the paths must be physically disjoint. In

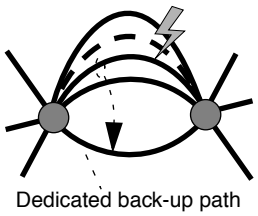


some cases this is sought achieved by minimum physical distance (e.g., 1.5 meters) of the cables combined with other means for physical separation. It is three types of protection switching:

a) *1+1 protection:* The bit stream is sent synchronously on both paths toward the destination. The destination selects the bit stream it considers as the one with the higher integrity, i.e., fewer bit errors. It is ensured that path failures are detected by coding techniques, loss of signal, loss of synchronization, etc. The advantage of 1+1 protection is, as with all masking redundancy techniques, an immediate (in the order of a few ms) handling of path failures. A path failure should be unnoticed by the users of the transport service.



b) *1:1"* and *"1:N protection:* are stand by protection strategies, where a dedicated spare (or back-up) path is available for one and N active paths respectively. Failure detection techniques are as for the masking redundancy, but the receiving node must inform the sending node which redirects the data stream. This incurs a temporary loss of service (in the order of 10 - 100 ms), but is usually tolerated by the end-user (applications).



In conclusion the restoration is simple, fast and proven, but requires more transmission capacity (but less costly control) than the (smarter) alternatives outlined below.

3.2 Reconfiguration

The reconfiguration strategy is based upon a centralized management of the network. A network management system, see Figure 6, supervises and controls all network resources. It seeks to keep a map of all network resources, their utilization and status. Based on this information, it sets up transmission paths between nodes in the network. When a network failure occurs, the network management system reconfigures the routing through the network.

This strategy has the potential to use the transmission capacity in the network very efficiently due to its ability to perform a global optimization. Reconfiguration may, however, be slow, in the order of minutes, due to the time needed for data collection, deciding upon a new configuration, and propagating the new configuration back to the switches. The strategy is vulnera-

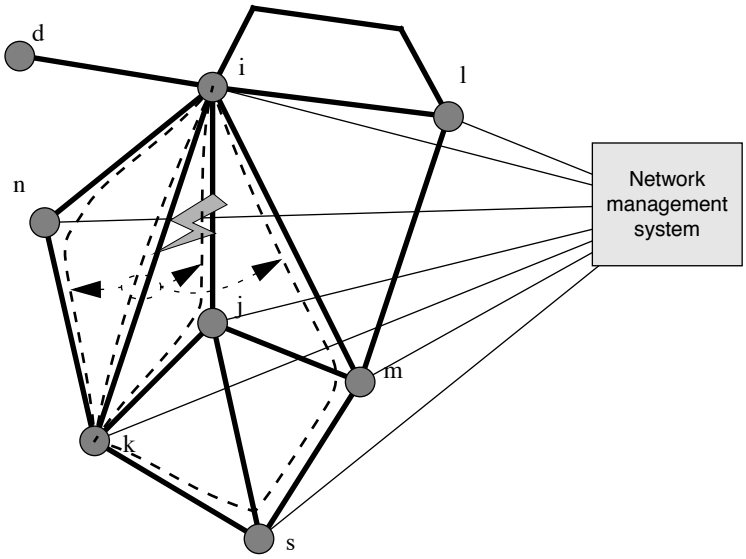


*Figure 6.*Reconfiguration of a network by centralized fault-handling.

ble, since it is both centralized and depends on information transfer in a network with failure(s) not yet handled. To overcome these drawbacks, parts of the management functionality are sought distributed, see for instance [7].

3.3 Self-healing

Self-healing is used as common nominator for a range of strategies which have in common that they have distributed control and requires no dedicated pre-reserved transmission capacity. The next subsections present three basic techniques. The two first are, as protection and reconfiguration, associated with transmission paths in the network, while rerouting in Section 3.3.3 is associated with the individual connections/streams.

3.3.1 Back-up paths

When a path (the active) is established through the network one or more back-up (stand-by) paths are established simultaneously. This/these back-up path(s) may be completely or partially disjoint from the active. A back up path does not carry any traffic unless a node or link along the active fails. Transmission resources along a back-up path may be assigned on-demand when an active path failure occurs. In this case, traffic may be lost if sufficient capacity are unavailable.

Hence, if the network shall meet dependability/QoS requirements, semi-dedicated resources must be reserved. See Figure 7 for an example. The link [i, k] carries the active path between nodes i and k and has reserved stand-by capacity for either the active path {i, n} or the active the path {i, m, s}. Since these paths have no common network element except the originating node i, failures of these two active paths are assumed independent.

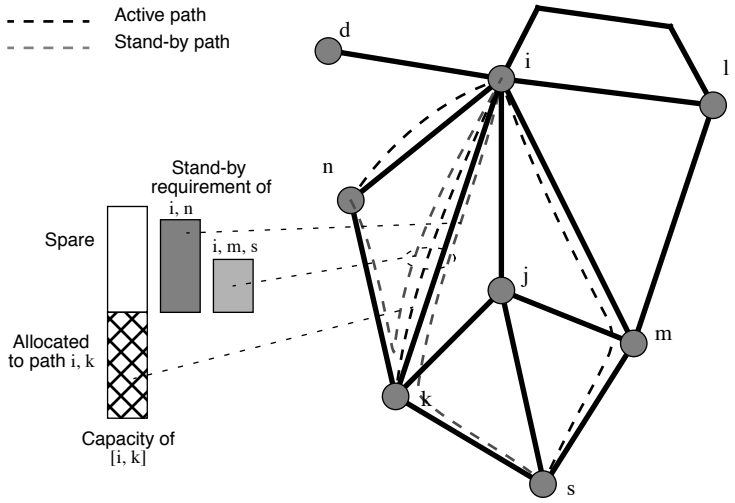


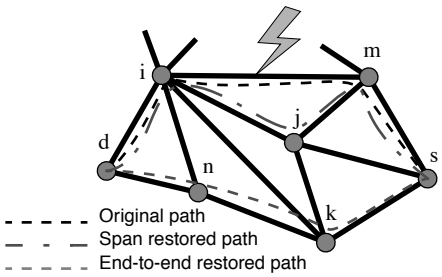
*Figure 7.*Example of allocation of back-up paths with semi-dedicated resources.

A path failure is detected in the end-nodes, for instance by missing OAM cells in an ATM connection. When resources on stand-by paths are available, the service may be restored in the order of 10 - 100 ms. A challenge is to allocate and adapt the back-up paths so the spare capacity in the network is optimally used under a given traffic load.

3.3.2 Flooding

Flooding is a fully distributed network management technique to restore communication after an element failure. The technique has got its name since it floods the network with request in the search for available capacity to replace that of a failed link. No resource reservation is made ahead of the reconfiguration, but the network must be dimensioned so that spare capacity is available. There are four variants of the method, dependent on whether:

- A new path restored by diverting the affected paths around the failed network elements failed, i.e., *span restoration*, or whether a new path is established between the source and destination node, i.e., *end-to-end restoration*. See the illustration to the right for examples.



- The search for a new (unidirectional) path starts form one node and terminates in the other, i.e., *single search*, or starts in both nodes and terminates when the searches "meet in the middle", i.e., *double search.*

Flooding, especially end-to-end restoration, poses a number of algorithmic problems on how to reserve and release capacity to avoid deadlocks and to make an efficient use of the available resources in the network. The service restoration time is at least several seconds, and no dependability/QoS guarantees may be given, since no reservations are made.

3.3.3 Rerouting

This self-healing strategy is executed on the connection level. The traffic through the network finds new routes between source and destination end systems when a network element fails, i.e. by directing the packets (in connectionless communication) along new routes, or re-establishing the end-to-end connections along a new route. See Figure 8 for an illustration. Note that

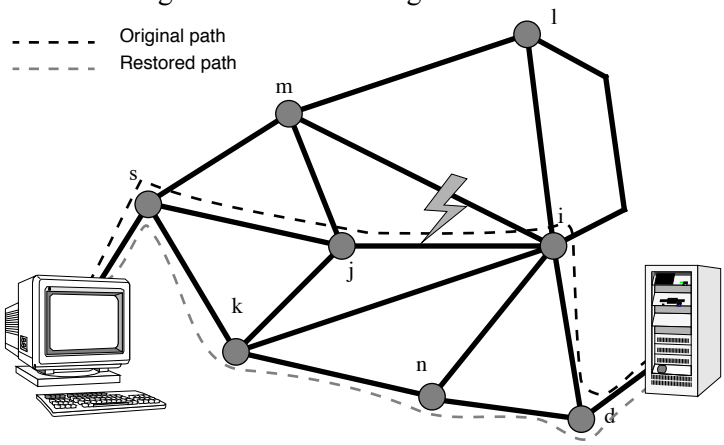


*Figure 8.*Example of self-healing in a network by rerouting connections or flows after a network element failure.

in this strategy, individual packets/connections are rerouted or re-established. This is different from the self healing strategies presented above, as well as protection and reconfiguration, which is aimed at restoring the end-to-end

paths of the network after an element failure. Each path may carry a number of individual end-user connections and/or messages between end-users. This path rerouting/re-establishment may be considered as a bulk restoration of the transport capacity by the network itself without involving the end-systems, and where it is an objective that the fault handling should be as "invisible" to the end-user (applications) as possible. Applying rerouting, this near transparency of network faults to the end-user (systems) will be lost, *no dependability/QoS guarantees can be made* and:

- the end-user application will stop, which may be critical for some applications, e.g., tele-medicine,
- longer time is needed before a service may be re-established after a failure, e.g., a new connection must be made or a period with large packet losses and/or delays will arise in connectionless networks,
- the network may be temporarily overloaded with connection requests or repeated packets after a failure, and
- semipermanent connections established by the management system, e.g., leased lines, require special attention.

The are also a number of advantages by utilizing this strategy:

+ less functionality is needed in the network since ordinary traffic handling functions are used for network element failures,
+ the routing algorithm used for rerouting after a network element failure, may be the same algorithm used to adapt to normal fluctuations of the traffic load, like daily variations and overloads,
+ the restoration is distributed (unless a centralized routing function is introduced) and hence, no centralized function forms a dependability bottleneck,
+ the path restorations are end-to-end and the information flows that shall be restored have a finer granularity, which yields a better resource utilization.

3.4 Multilevel fault handling

Table 1summarizes some of the basic implementation issues related to the redundancy strategies discussed hereto in Section 3. Transport networks has typically a layered design, at least a physical layer, a transmission (link) layer and a switching layer. These layers may contain sublayers, e.g. the ATM virtual path and connection layers. In which of these layers should we introduce the fault tolerance? What fault-tolerance strategy should be used? Since the layers are designed independently of each other, how should we avoid introducing excessive spare capacity and avoiding that several layers compete to handle the same failure?

Table 1. Summary of resource handling issues for the various network redundancy strategies.

	Resources			Control		
	Dedicated	Preplanned Reserved	On demand	Central	Distrib. control	Distrib. mangemnt.
Protection	✔					✔
Reconfiguration		✔	✔	✔		
Self healing		✔	✔		✔	✔

An exaggerated illustration of the problem is given in Figure 9. For instance, the cross-connect switching carried out by the SDH transport system, may alternatively be carried out by the ATM VP layer. If we are interested in IP transport, why do we not get rid of all the underlying layers and use IP-over-light? The answers to these questions are complicated by the fact that:

- in contradiction to what is illustrated in Figure 9, networks are not homogeneously and strictly layered end-to-end. For instance, an IP network may span several underlying networks;
- the lower layers may carry several independent transport services;
- all transport services (as well as end-users) do not have the same dependability requirements and hence the same need for fault-tolerance in the network.

For discussions on design of multilayer networks and on which layer that it is most profitable to introduce redundancy, see for [19, 20, 29 and 40]. Some rules of thumb may be given:

a) The higher up in the network the larger is the flexibility and the number of stand-by resources needed is likely to be less;
b) The higher up in the network, the larger is the delay from the failure event and until the service is restored;
c) Low layer redundancy handling tends to introduce the least complexity since the mechanisms may be simpler and the redundancy handling may be common to a number of higher layer network services.
d) Fault tolerance on a layer can only handle fault at its level and on the levels below. Hence, some fault tolerance is required at the highest level.

A closing question: with the observations of the current internet in mind [33, 22], is the currently so popular IP-over-light scenario able to shortcut today´s multilevel fault handling and provide a fast enough reconfiguration and a stable rerouting in a large network where failures are the rule more than the exception?

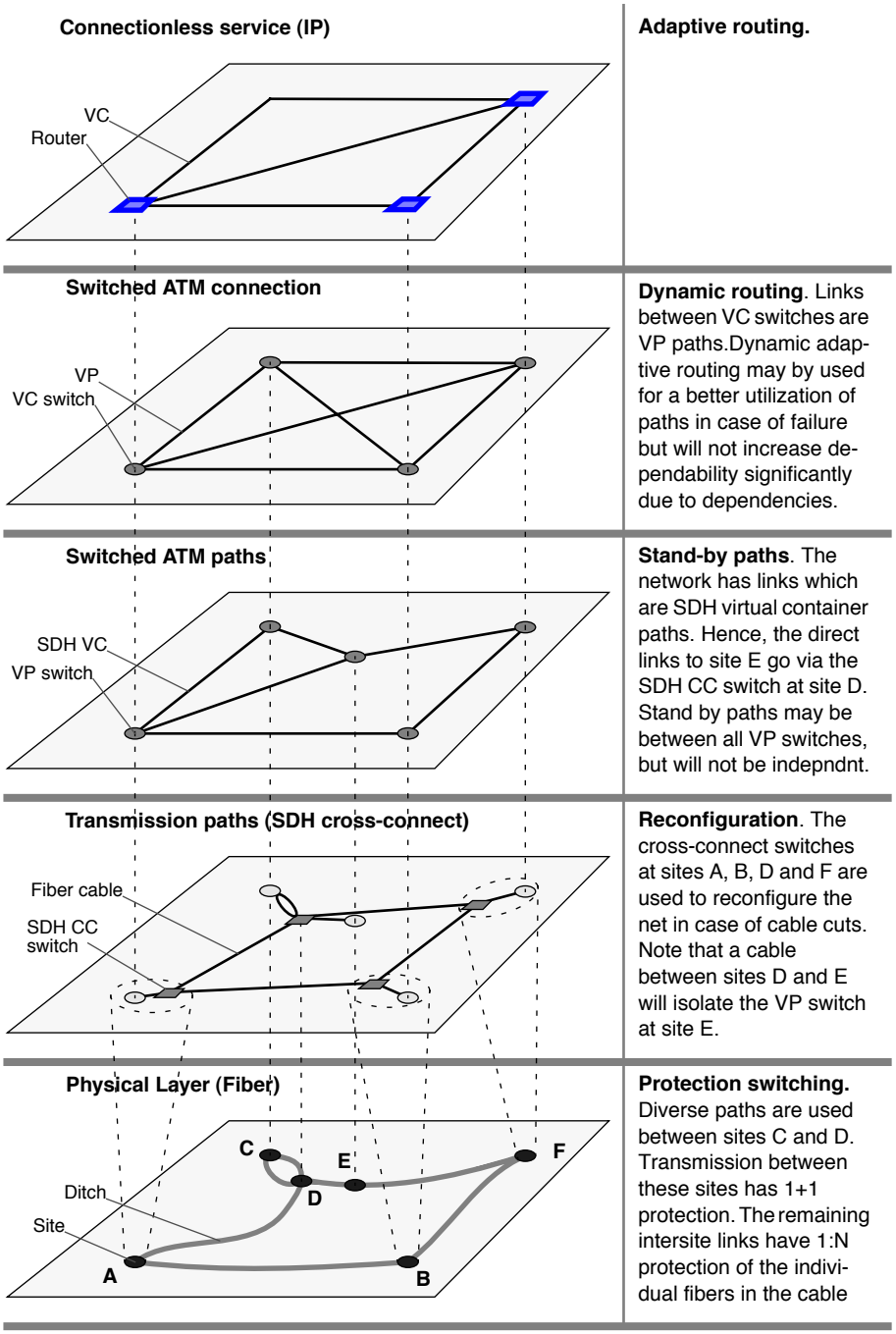


*Figure 9.*Layered network scenario with fault handling in multiple layers.

4 FAULT TOLERANT NETWORK FUNCTIONS

In the previous section, the techniques for making the transport services of a network fault tolerant was discussed. This section presents two basic fault-tolerance strategies that enable network functions to be performed, uninterrupted if required, in spite of computing and data storing element failures. Network functions in this context encompass call/connection/stream control, network and service management, and service handling as illustrated in Figure 1. The faults sought tolerated are primarily physical faults, even though some of these techniques also have been experienced to tolerate a subset of logical faults reasonably well [10].

The two strategies regarded for achieving fault tolerance in the computing and data storage functionality are:

- To use fault-tolerant network nodes, i.e. to make the computing and database platforms supporting the functionality of the network nodes fault-tolerant. For instance to make an intelligent network SCP fault tolerant by using a duplicated synchronous computer to execute the functions.
- To introduce fault tolerance at the network level, i.e. to have co-operating replicas of software objects/processes providing network functions in several nodes and thereby enable tolerance of node failures.Network level fault-tolerance is based on dependable distributed computing technologies.

The aim of the section is two view these as competing options for achieving dependability in smart networks. Hence, (too) little attention is paid to the "in-between" systems, e.g. [34, 30], which uses a specially designed transport network to ensure consistencies between replica.

4.1 Fault-tolerant network nodes

This strategy is illustrated in Figure 10, where each node in the network, and the services it delivers, are made fault tolerant by having redundant computing and/or storage capacity in the node. The replicas of the processes executed or stored are located within the same node. The architecture of the nodes may vary as illustrated. Fault-tolerance may for instance be achieved by micro-synchronous duplication, see node *s* of Figure 10, as in for instance the AXE [32]. Alternatively a distributed and mainly load shared architecture, see node *d* of Figure 10, as in Alcatel System 12 [4], may be applied. See for instance [3] for a discussion on the design and analysis of such systems.

As indicated by the type and age of the above references, the fault tolerant network node is the classic approach for public communication systems. This is a proven technology, and the design goal, originally set for the ESS 1 [8], of an expected down time of three minutes per year was achieved between

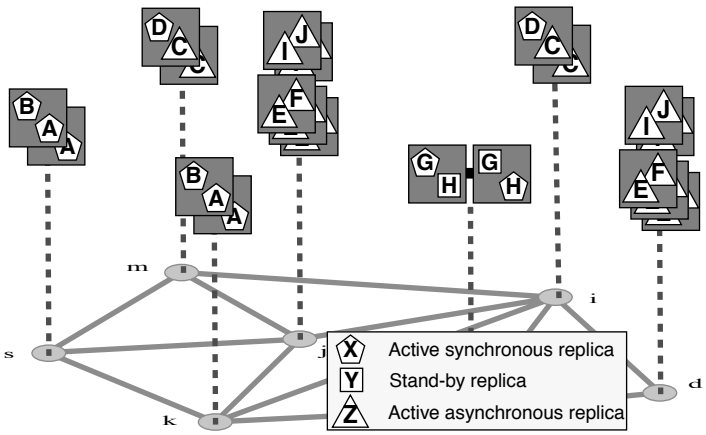


*Figure 10.*Illustration of network with fault tolerant nodes with various architectures.

one and two decades ago. In addition, this approach has several advantages. In most systems, it hides the fault-tolerance mechanisms for the application designer. The synchronization between replica incurs little or no delay, and the strict real time QoS requirements of communication systems is met. However, these systems are dedicated (legacy) systems and are more expensive than off-the-shelf computer and database systems. Located at a single site these systems are vulnerable to environment failures like fire.

Nevertheless, this is a viable approach to providing dependable network functions in a smart network, cf. for instance [15]. With respect to providing network functionality, the problems related to the handling of (physical) faults may be substantially reduced/hidden, but at a certain cost and with reduced applicability to smaller nodes.

4.2 Replication across the network

This strategy is illustrated in Figure 11. The computing and storing hardware of the nodes in the network are *not* fault tolerant. The services delivered by the network, however, may be fault tolerant by having replicas of the various service providing software objects/modules in several network nodes. The replication may be handled by appropriate middelware [27]. As in the strategy of Section 4.1, this requires redundant computing and storage capacity in the network. However, off-the-shelf equipment may be used and the redundancy installed for the various object/modules may be tailored to the dependability requirements of the services they provide and the required consistency between replicas. For instance, in Figure 11, there is three synchronous replicas of object/module A. These replicas have consistent states and by voting, an arbitrary failure of one of the replicas, as well as the node hosting it, may be tolerated. The two synchronous replicas of D and F allow a

crash failure without interrupting the service. The active object/modules B and E have stand-by replicas that may take over if the active replica/node fails. The stand-by replica may have a consistent or partially consistent state with the primary, e.g. by checkpointing, or be in a default state. In the last case illustrated, C has three load-shared replicas without consistent states.

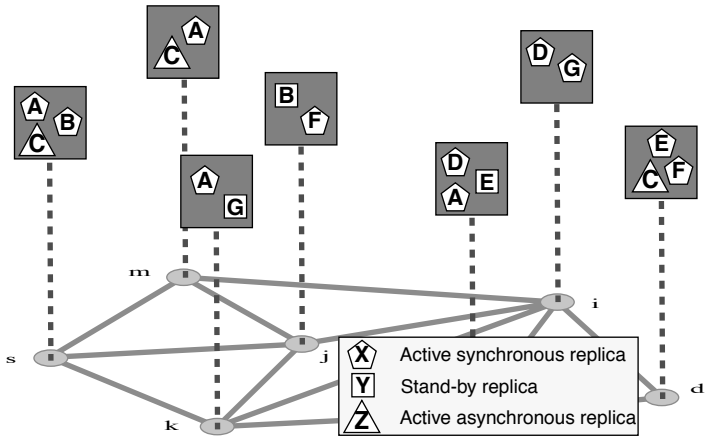


*Figure 11.*Illustration of network where software objects are replicated on disjoint network nodes for fault tolerance.

It is a common objective that replicas shall be location and fault transparent, i.e. users of the services shall not need to know where the replicas are located and whether some of the replicas has failed. Achieving a flexible replication scheme and consistent replicas across an unreliable network is no simple task. To deal with these problems, there has for more than a decade been put considerable research effort into the management of groups of objects/processes and the communication between them. This is referred to as group communication. See [35] for a collection of papers on this issue. Various tool-kits have been developed to facilitate fault-tolerant distributed processing, e.g. Isis and Hourus/Electra [5, 36]. Mechanisms for fault-tolerance are also likely to be included in CORBA [31, 38, 39, 41]. Hence, ensuring the dependability of network control, management and service provisioning should be within reach by fault tolerance techniques developed for distributed systems.

So where is the challenge in the smart network context? The fault-tolerance and replication strategies applied must be tailored to the requirements and adapted to the particularities of telecommunications. For instance:

- Network operations and end-user QoS requirements put rather strict real time requirements on many tasks. The replica synchronization is based on protocols ensuring causal order [23], which may be time consuming, especially on non-broadcast media like WANs.

- Many tasks carried out in a network, e.g. reservation of a communication resource, have short processing times compared to synchronization times between replicas, hence an approach must be chosen so an immense overhead can be avoided.
- Only a subset of the tasks performed by the network result in state changes that need to be persistent in case of failure. Some of the Internet´s success as a transport network may be attributed to its statelessness. In connection oriented networks, connections in the set-up phase has always been considered expendable when failures occur. For other tasks, persistence of the state in case of failure is of outmost importance, e.g. updates of the home location register in mobile networks.

5 LOGICAL FAULTS[1]

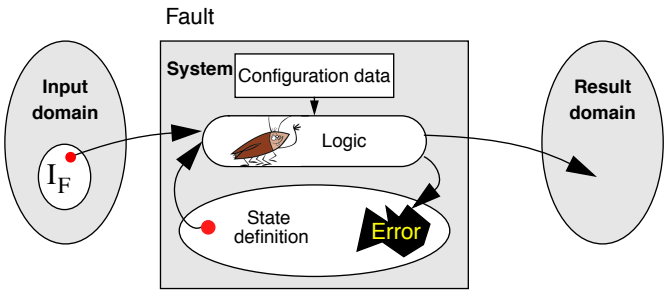
The fault tolerance strategies discussed in the previous chapters are aimed at physical faults. Logical faults are however the cause of a substantial part of the failures. Failures due to logical faults may affect every aspect of the service and network operation, from aborting a single service transaction to a complete network outage. The increasing logical complexity and intervoweness of smart networks makes them more prone and vulnerable to logical faults.

In spite of extensive use of fault avoidance and validation techniques, cf. Figure 2, logical faults, e.g. software faults, are embedded into a system during its specification, design, implementation and configuration. These faults stay dormant in the system until a combination of input/use of the system and internal state activates them and causes an error. For a mature system, the expected dormancy period can be several years. In the subsequent operation of the system the error may cause a failure as illustrated in Figure 12. In dependable systems, the various modules/components are sought made robust to errors by audit routines, exception handling etc.

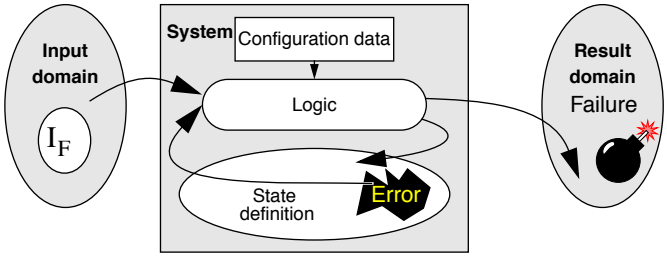
In this presentation, it will not be discussed how logical faults could be avoided and tolerated. The focus is instead put on how logical fault influences the dependability of networks, namely the persistence of errors and error propagation.

---

1. The term logical fault, and not the more common term software fault, is used deliberately since this type of failures encompass much more than software bugs, e.g. design and specification faults in the overall network operation.



a) Activation of a dormant logical fault



b) Failure after a latency period

*Figure 12.*Extended I/O logical failure model based on a generalized Moore/Mealy model.

5.1 Error persistence

A common and rarely disputed assumption in the design and evaluation of dependable systems is that failures occur according to a Poisson process, i.e. failures occur independent of when the previous failures occurred and with a constant occurrence probability per time unit. However, studies of failure logs show that this is not the case, especially for logical caused failures, see for instance [25, 11]. This is demonstrated in Figure 13, which shows the autocorrelation[1], $\rho(j)$, of logically caused failures of a distributed communication control system.

The figure shows the dependency between the number failures in one interval and the number of failures in the *j'th* subsequent (or previous) interval. (The larger the deviation of $\rho(j)$ is from zero, the greater the dependency.) If the failure process was a Poisson process, the autocorrelation would in 95% of the cases be within the two dotted lines of Figure 13. It is seen that

1. Let $n(t_i)$ be the number of failures in the constant length interval i . The autocorrelation is then defined as $\rho(j) = \frac{\mathrm{Covar}(n(t_i), n(t_{i+j}))}{\mathrm{Var}(n(t_i))} = \frac{E(n(t_i) \cdot n(t_{i+j})) - (E(n(t_i)))^2}{E((n(t_i))^2) - (E(n(t_i)))^2}$

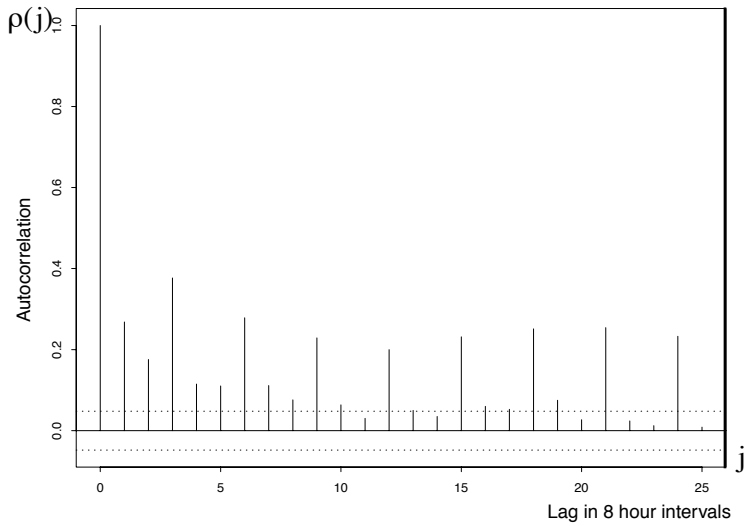


Figure 13. The autocorrelation, $\rho(j)$, between software related failure events in a large distributed communication system.

the failure process has a "memory" of more than a week. Error propagation, as will be discussed in the next section, and changes in operational conditions will also contribute to this correlation. Note also the daily variations seen in Figure 13. The number of failures is more dependent on the number of failures at the same time during the previous day than the number of failures during the previous eight hour interval. Errors may persist in the system for a considerable period and cause a burst of failures, see also Figure 14.

5.2 Error propagation

Figure 13 indicated that errors tend to persist in a system and cause multiple failures. Figure 14 shows a sample from the software related failure pattern of a large distributed communication control system. It is seen that during the first four days we have a normal operational mode with few failures. During the 11'th through the 13'th day severe bursts of failures occur, affecting several processors. These figures show that the failure pattern is not Poissonian, i.e. failures do not occur independently of each other.

- Failures tend to occur in severe bursts, i.e. if one failure is experienced, the next is likely to occur rather soon. Restarts or similar actions after a failure do not always restore the system to an error free internal state, or the internal state may be inconsistent with the rest of the system or the environment.
- Many processors are involved in a failure burst. If one processor fail, there is an increasing probability of failure of the cooperating processors.

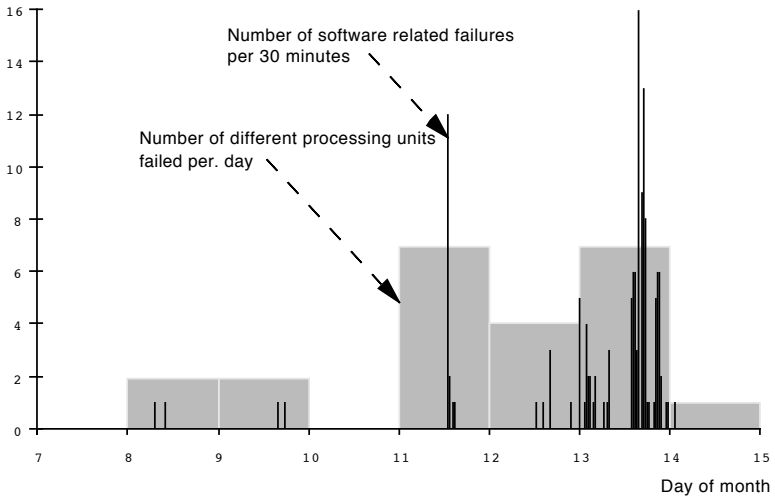


*Figure 14.*Excerpt of the failure log of a distributed communication control system.

An error located in one part of a software subsystem may propagate (spread) to other parts.

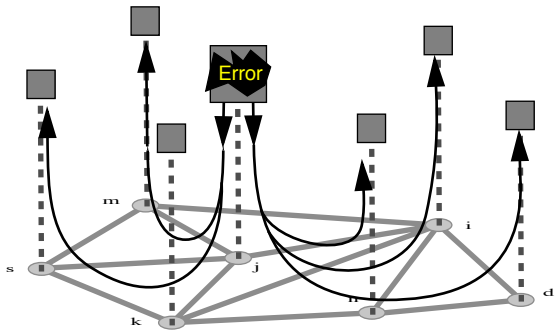


*Figure 15.*Illustration of error propagation in a network.

For an explanation of the latter phenomenon, regard the sketch of Figure 15. Control/management/service providing subsystems span several physically separate nodes of the network. Each of these subsystems is constituted by a number of more or less tightly coupled software modules residing in the various network nodes. These modules cooperate to carry out the services of the system. Incorrect state information may be transferred from one module to another. For instance by transfer of incorrect or inconsistent data, or by a module acting on false information and thereby introducing errors in its own internal state. Hence, subsequent failures of seemingly independent physical units may occur. This phenomenon is denoted error propagation [12, 13]; see the illustration in Figure 16. The error propagation between modules depends on:

- the intensity of the interactions and
- how much information they exchange/have in common.

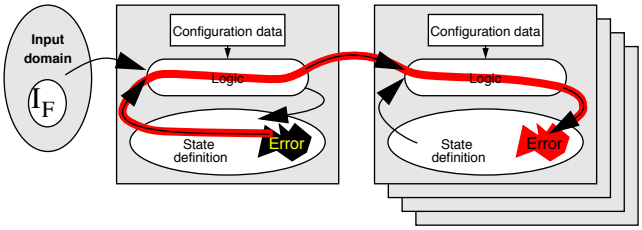


*Figure 16.*Error propagation a generalized Moore/Mealy model.

A variant of error propagation occurs when the same logical fault is present in identical copies of software modules present in various nodes in the network. These faults may be activated by a condition spreading through the system under specific circumstances, cf. the first well known example [9]. The most prominent logical fault of this kind is the Y2K bug. Analysis of operational data from distributed communication control systems by cluster analysis, shows that the systems may have different operational modes as a result of the error propagation. For instance, “good” modes where only few sporadic failures occur and “bad” modes where many units fail repeatedly within a short time interval [14]. A mode-changing behaviour of the system may be identified. The “bad” modes cripple the service offered severely and may cause instability of the entire network. Furthermore, it is illustrated how the mode-changing behaviour of the system may be described by a state model.

Smart networks are, due to their large logical complexity and intervoweness, exposed to persistent and propagating errors. Hence, to be dependable and have a stable operation, these must have an inherent design which

- enables errors to be removed when a recovery action is taken as a result of a failure;
- prevents error propagation between the various parts of the network.

6 CONCLUDING REMARKS

Dependability is an important aspect of networks in the future. The availability and reliability requirements will continue to increase as our social and economic dependence on teleservices as well as the cost of failure continues to grow. Dependability constitutes a significant part of the QoS and is an important factor in the competition among service providers, network operators and equipment manufacturers. High dependability is costly to provide both with respect to development (fault-tolerance, fault management, and

quality assurance), deployed equipment, and operation and maintenance. At least a doubling of "the basic cost" is required if system shall reach a typical communication network component availability in the order of $1 - 10^{-5}$.

New smart network technologies must be proven as dependable as "traditional" network technologies before they become deployed in commercial networks. Hence, the following basic design question arises; shall proven "dumb" low level (in terms of a layered design) fault handling techniques, like protection switching and duplicated synchronous processors be used, or should new smarter technologies, which are more flexible, and which give potentially cheaper solutions be pursued? Anyhow, the "dumb" techniques are only able to cope with physical faults. The major challenge is the logical (software) faults. The consistent design, design methodology and extensive quality assurance, currently used to restrict the number of logical faults and ensure the dependability, seem feasible only within monolithic proprietary systems as we know them to-day. In the future open, heterogeneous networks with multiple hardware and software vendors, service providers and network operators, robust and adaptive network and service components becomes a necessity. Components able to tolerate logical and physical failures of their environment and co-operating components, adapt and contribute to the provision of continued service is a major challenge in the advancement of smart networks.

REFERENCES

[1] Finn Arve Aagesen, Bjarne E. Helvik, Vilas Wuwongse, Hein Meling, Rolv Bræk and Ulrik Johansen, "Towards a Plug and Play Architecture for Telecommunications", This Issue.

[2] Syed R. Ali, "Analysis of Total Outage Data for Stored Program Control Switching Systems", IEEE Journ. on Selected Areas in Communications, Vol. SAC-4, No. 7, pp. 1044 - 1046, Oct. 1986.

[3] Syed R. Ali, "Digital Switching Systems; System Reliability and Analysis" McGraw-Hill, 1997.

[4] Beyltjens, van Houldt, "System 12, Switching System Maintenance", Electrical Communication, Vol. 59, No. 1/2, pp. 80-88, 1985.

[5] Kenneth P. Birman, Robbert van Renesse (eds.), "Reliable Distributed computing with the Isis Toolkit", IEEE Computer Society Press, 1994.

[6] W.C. Carter, "A time for reflection", Proc. 12th International Symposium on Fault-Tolerant Computing" Santa Monica California (FTCS-12), p. 41, June 1982.

[7] Brian A. Coan, Will E. Leland, Mario P. Vecchi, Abel Weinrib, Liang T. Wu, "Using Distributed Topology Update and Preplanned Configurations to Achieve Trunk Network Survivability", IEEE Transactions on Reliability, Vol. 40, No. 4, pp. 404 - 416, October 1991.

[8] R.W. Downing, J..S. Novak, L.S. Tuomenoksa, "No. 1 ESS Maintenance Plan", The Bell System Technical Journal, Vol. 43, No. 5, Part 1, pp. 1961 - 2019, Sept. 1964.

[9] Karen Fitzgerald, “Vulnerability exposed in AT&T´s 9-hour glitch”, The Institute, Vol. 14, No. 3, Pages 1 and 6, March 1990.

[10] Jim Gray, “A Census of Tandem System Availability between 1985 and 1990”, IEEE Trans. on Reliability, Vol. 39, No. 4, pp. 409 - 418, Oct. 1990.

[11] Bjarne E. Helvik, Anders Rygh Swensen, “Modelling of Clustering Effects In Point Processes. An Application to Failures in SPC-Systems”, Scandinavian Journal of Statistics, Vol. 14, pp. 57 - 66, 1987.

[12] Bjarne E. Helvik: “Modelling the Influence of Unreliable Software in Distributed Computer Systems”, Proc. 18th International Symposium on Fault-Tolerant Computing (FTCS-18), pp. 136 - 141, June 1988.

[13] Bjarne E. Helvik, “The Error Propagation Phenomenon; An introduction”, Telektronikk, Vol. 93, No. 1, pp.109 - 117, 1997

[14] Bjarne E. Helvik, Sven Arne Gylterud, “Identification of Operational Modes of Distributed Systems by Clustering Analysis”, Telektronikk, Vol. 93, No. 1, pp.118 - 127, 1997.

[15] Svein-Olaf Hvasshovd, Øystein Torbjørnsen, Svein Erik Bratsberg, Per Holager, “The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response”, Proceedings of 21th International Conference on Very Large Data Bases (VLDB'95), pp. 469 - 477, September 11-15, 1995, Zurich, Switzerland.

[16] ITU-T, “Terms and definitions related to quality of service and network performance including dependability”, Rec. E.800, August 1994.

[17] Pankaj Jalote, “Fault Tolerance in Distributed Systems”, Prentice Hall, 1994.

[18] Barry W. Johnson, “Design and Analysis of Fault Tolerant Digital Systems”, Addison-Wesley, 1989.

[19] K.R. Krishnan, R.D. Doverspike, C. D. Pack, “Unified Models of Survivability for Multi-Technology Networks”, Proc. ITC-14 (eds: Labetoulle and Roberts), pp. 655 - 666, Antibes Juan-les-Pins, France, Elsevier, June 1994.

[20] K.R. Krishnan, R.D. Doverspike, C. D. Pack, “Improved Survivability with Multi-layer Dynamic Routing”, IEEE Communication Magazine, Vol. 33, No. 7, pp. 62 - 680, July 1995.

[21] D. Richard Kuhn, “Sources of Failure in the Public Switched Telephone Network”, Computer, Vol. 30, No. 4, pp. 31 - 36, April 1997.

[22] Craig Labovitz, Abha Ahuja, Farnam Jahanian, “Experimental Study of Internet Stability and Backbone Failures”, Proc. 29th International Symposium on Fault-Tolerant Computing (FTCS-29), pp. 278 - 285, June 1999.

[23] L. Lamport, “Time, Clocks and the Ordering of Events in Distributed Systems”, Communication of the ACM, Vol. 21, No. 7, pp. 558-565, July 1978.

[24] Jean-Claude Laprie (Ed), “Dependability: Basic Concepts and Associated Terminology”, Dependable Computing and Fault Tolerant Systems. Vol-5; Springer, 1992.

[25] P.A.W. Lewis, “A Branching Poisson Process Model for the Analysis of Computer Failure Patterns”, Journ. of The Royal Statis. Soc. Ser. B, Vol. 26, pp. 493 - 503, 1964.

[26] Michael R. Lyu (ed.), “Handbook of Software Reliability Engineering”, McGraw-Hill/IEEE Comp. Soc. Press, 1996.

[27] Silvano Maffeis, Douglas C. Smidt, “Construction of Reliable Distributed Communication Systems with CORBA”, IEEE Communication Magazine, Vol. 35, No. 2, pp. 56 - - 60, Feb. 1997.

[28] John J. Metzner, “Reliable Data Communications”, Academic Press, 1997.

[29] L. Nederlof &. al. “End-to-end Survivable Broadband Networks” IEEE Communication Magazine, Vol. 33, No. 9, pp. 63 - 70, Sept. 1995.

[30] Louise E. Moser, P. M. Melliar-Smith, Deborah A. Agarwal, Ravi K. Budhia, Colleen A. Lingley-Papadopoulos, “Totem: A fault-Tolerant Multicast Group Communication System“, Communications of the ACM, Vol. 39, No. 4, pp. 54 - 63, April 1996.

[31] Louise E. Moser, Roger J. Martin, “Fault Tolerance for CORBA” (Joint initial fault tolerance RFP submission by Eternal Systems and Sun Microsystems), ftp://ftp.omg.org/pub/docs/orbos/98-10-08, October 19, 1998.

[32] Bengt Ossfelt, Ingmar Jonsson, "Recovery and Diagnostics in the Central Control of the AXE Switching System", IEEE Trans. on Comp., pp. 482-491, June 1980.

[33] Vern Paxson, “End-to-End Routing Behaviour in the Internet, IEEE/ACM Transactions on Networking, Vol. 5, No. 5, pp. 601 - 615, Oct. 1997.

[34] David Powell, “Distributed Fault Tolerance: Lessons from Delta-4", IEEE Micro, February 1994, pp. 36 - 47.

[35] David Powell (Guest Editor), “Group communication”, Communications of the ACM, Vol. 39, No. 4, pp. 50 - 97, April 1996.

[36] Robbert van Renesse, Kenneth P. Birman, Silvano Maffeis, “Horus: A flexible Group Communication System”, Communications of the ACM, Vol. 39, No. 4, pp. 76 - 83, April 1996.

[37] Daniel P. Siewiorek, Robert S. Swarz, “Reliable Computer Systems; Design and Evaluation”, Digital Press, 2nd edition, 1992.

[38] Chris Smith, "Fault Tolerant CORBA" (Fault tolerance joint initial submission by Ericsson, IONA, and Nortel supported by Alcatel), ftp://ftp.omg.org/pub/docs/orbos/98-10-10, October 20, 1998.

[39] Jeffrey R. Spirn, "Fault Tolerance RFP: Initial Submission" (Oracle), ftp://ftp.omg.org/pub/docs/orbos/98-10-13, October 20, 1998.

[40] Paul Veitch, Dave Johnson, “ATM network Resilience”, IEEE Network, Vol. 11, No. 5, pp. 26 - 33, Sept./Oct. 1997.

[41] Shalini Yajnik (ed.), “Fault Tolerant CORBA using Entity Redundancy” (Joint initial fault tolerance RFP submission: Highlander Communications, L.C., Inprise Corporation, Lockheed Martin Corporation, Lucent Technologies, TIBCO Inc., and supported by Academia Sinica, Taiwan), ftp://ftp.omg.org/pub/docs/orbos/98-10-09, October 20, 1998.