# World Wide Failures.

Werner Vogels

Dept. of Computer Science, Cornell University

4105a Upson Hall,
Ithaca, NY 14853

*vogels@cs.cornell.edu*

## Abstract.

*The one issue that unites almost all approaches to distributed computing is the need to know whether certain components in the system have failed or are otherwise unavailable. When designing and building systems that will need to function at a global scale, failure management will need to be considered a fundamental building block. This paper describes the development of a system-independent failure management service, which allows systems and applications to incorporate accurate detection of failed processes, nodes and networks, without the need for making compromises in their particular design.*

## Introduction.

With the advent of ubiquitous, world-wide distributed systems, it is becoming clear that the systems that are used today in local-area settings, can not simply be employed in their existing form or trivially converted for wide-area, large-scale operation. Whatever form such systems may take in the future, whether they are replicated databases of hyper-links, distributed objects, view or virtual synchronous groups or agents employing lazy consistency schemes, one of the key problems that needs to be addressed, is the detection and handling of faulty components, nodes and networks.

After years of experience with building distributed systems using systems like Isis, Horus, Transis and others [1,2,3,11,12], all of which have built-in failure handling, it is clear that failure management is not just a tool that is unique for these group oriented systems but that it is a fundamental service that should be placed among such established basic services as naming, authentication, security, service brokerage and IPC.

This paper reports on an ongoing research effort to abstract the failure handling strategies from a variety of popular distributed systems and to develop a basic failure management service that can be used by any distributed system regardless of the purpose of that system or the techniques used. The strategies employed by this basic service are specifically targeted towards applications that need to operate on a global scale.

## Research goals.

To build a successful service the following goals were set:

- design a failure management system that is independent of the distributed systems packages in use and provide failure detection of processes, nodes and networks.
- improve the accuracy of detection of process and node failure through systems support.
- design support for failure detectors to work in large scale systems, while maintaining a high level of accuracy.
- provide support for the detection of partitions in networks.
- build a comprehensive software package that can be easily integrated into various distributed systems packages and applications.

The resulting system is implemented and is under test in a wide-area simulator, in a local setting of a mix of high-speed and traditional networks and in the Internet. A first software release is planned for the summer of 1996.

## History.

External failure detector modules originate in asynchronous distributed systems, where they were introduced to decouple the mechanism by which failures are detected from the protocols used to tolerate those failures. In [5] Chandra and Toueg successfully show that it is possible to develop consensus algorithms using failure detectors, even if these failure detectors make frequent mistakes in their observations. In [4,7,9] the failure detector work is extended to systems that also take network failure into account.

The major trade-off in designing practical distributed systems based on the theory developed for asynchronous systems is where and how to introduce the notion of time. Traditionally failure detectors have been implemented using time-out mechanisms in the transport layer that implements inter-process communication. Although time-outs remain an important tool in the failure manager described in this paper, the mechanism is integrated into a more comprehensive approach that treats failure detection

using methods based on an analogy with fault-detection techniques used in daily life.

When trying to contact a person who has allegedly disappeared one would never be satisfied with making repeated phone calls to the same location for half an hour and then declaring the disappearance a fact, no matter whether the phone was not picked up, a busy tone was heard or the phone was disconnected. In practice one would work to gain more confidence in such a decision by talking to the landlord, the neighbors or others that may have a very good idea about the situation of the person in question.

The failure management described in this paper is capable of following a similar strategy; if a process under investigation is not responding it will contact the operating system under which the process is running, or other nodes on the same subnet to help reach a decision in which one can have greater confidence.

## Current practice.

Most distributed systems in use today deal with failure of nodes or networks in some way. In general the problem is detected in the communication subsystem where session or transport protocols are unable to make progress because of the lack of response from remote nodes. Traditionally packets are being retransmitted after a time-out period and after a retry threshold is reached the remote destination is marked as unreachable. Some systems inject additional packets into the data stream to ensure timely detection of failures at moments when the traffic is low or unidirectional.

Popular RPC systems (Sun, DCE) and message-passing libraries (PVM, MPI) expect the application to handle the failure management as the support system does not contain any fault management. Often these systems cannot distinguish between process, node or network failure. The mechanisms used to detect failure do not adapt to changing network conditions, making it almost impossible to use these systems unmodified in wide area systems without resorting to heavy weight solutions like using a TCP connection as the preferred transport method for each RPC call.

Other systems, especially those designed to support high-availability, support fault-management in a more integrated way. Many of these systems are structured as groups of cooperating processes using some form of group membership, which requires fault-detection to be able to reach consensus. Various methods are used, of which fault monitors[3], heartbeats[8,12] and polling[4] are the most popular. However in each of these systems the failure management is an integral part of the particular membership or transport system and not available for general use. Although some research groups[1,4,7,9] are focusing on wide area systems, the majority of the existing failure detectors are not suitable for use in large scale systems, because of their inflexibility or the simplicity of their assumptions.

## Independent failure detectors.

Building a failure detector that is not an integral part of the communication architecture permits the implementation of a collection of failure detection techniques and support for failure detection methods of varying levels of complexity from which the system designer can choose to match the system requirements.

The failure management service consists of three functional modules:

1. A library that implements simple failure management functionality and provide the API to the complete service.
2. A service implementing per node failure management, combining fault management with other local nodes to exploit locality of communication and failure patterns.
3. An inquiry service closely coupled with the operating system which, upon request, provides information about the state of local participating processes.

## Basic functionality.

The most fundamental operation offered by a failure detection service is that of the investigation of a suspected process. To make use of this operation it is not necessary for either the local or remote process to run any of the heartbeat or polling patterns; the reasons that the local process began to suspect the remote process are not of any importance to the failure management.

```
fail(address, mode, deadline, &report, &asynch)
```

The process at *address* is investigated and a report is returned within the deadline set by the local process. The local process does not have to wait for the investigation to finish but can make use of the asynch interface to collect the result at a later moment.

The report contains information on whether the remote node was reachable within the deadline and whether the process under investigation was still present at the host. If the *mode* parameter was used to request a more detailed remote reporting, process checkpoint information is returned or the remote process is interrupted to provide status information (see the section on OS integration).

If the node was not reachable and the local process has requested extensive investigation, the failure investigator will try to contact a failure manager at the node's subnet or within its administrative domain which should be able to give a more conclusive answer about the node's failure to respond.

If network failure is the cause of the loss of connectivity, the report will indicate which part of the path is reachable and where the suspected "break" is. If the failure

investigator is configured with alternative outgoing paths, these paths are probed to see if it is possible to circumvent the network failure and in such a way collect information about the remote process. The report contains information about the results of these probes.

## Early triggers

Many systems find it desirable to detect failure of remote processes even if there is no data exchange actually under way. Systems are free to implement whatever scheme they find appropriate and use the failure investigator from the previous section to handle the suspicions, or they can make use of two standardized schemes implemented by the failure manager library.

The first scheme uses a heartbeat mechanism, which sends out *I-am-alive* messages to a group of processes using multiple point-to-point messages or a single IP-multicast message. Each process keeps track of the reception times of messages and if a number of consecutive heartbeats from a destination is missed a suspicion is raised. Inter-heartbeat gaps ( fixed period or an exponential back-off scheme), time-out periods (fixed or estimated by the system), and multiple suspicion levels are configurable by the application. The application can provide application specific data to be piggybacked on the heartbeats.

The second scheme uses a polling method to collect acknowledgments from the peer processes, if no acknowledgments are received after a number of retries a suspicion is raised. Poll periods, time-outs, and retransmission limits are configurable by the application or can be adapted by the failure manager to the network.

## Instrumenting the Operating System.

To achieve greater failure detection accuracy, it is necessary to instrument the operating environment with support for process investigation. It has always been argued that in a distributed system it is impossible to distinguish a crashed process from one that is slow[6], but with the proper system support this is no longer true. If the node is reachable and operating correctly, the operating system can determine whether or not the process has crashed.

The failure management integrated into the OS offers processes a mechanism to register and request a certain level of service. The minimum level (*Process query*) is a simple binary test performed by the OS upon receipt of an inquiry, indicating whether the process is still present in the process table and thus not has crashed or voluntary exited. The two other levels that are currently implemented, provide a remote process with information about the progress the local process is making which is useful in the investigation of processes that are alive, but that appear slow or unresponsive.

*Process checkpoint*: At certain intervals the process logs checkpoint timestamps with the failure service, which simultaneously logs the process' OS communication state. The response to an inquiry request holds the last checkpoint timestamp, the current local time, whether the process has been allocated CPU time since the last checkpoint, and whether the process has consumed any messages since the last checkpoint.

*Process Upcall*. Upon receipt of an inquiry the operating system uses an upcall (Unix Signal) to interrupt the process and requests that the process prepares a special response. This response is returned to the caller.

## Node management.

The previous sections all deal with provisions targeted towards the failure management of processes, exploiting the close coupled nature of a process and the operating system it runs under. To aid accurate detection in the case of node failure the fault management system implements a node management service, which is based on the experience that local failure investigation on a subnet is more accurate than investigation over the Internet.

On a participating subnet one or more Node Failure Monitors (NFM) are present. These are simple services capable of performing local failure investigations upon requests from remote nodes. NFM's use IP-multicast to announce their availability within the organization where their presence is being tracked by the other NFM's. An NFM accepts queries from remote nodes about the availability of a node within its organization. It will forward this request to an NFM on the particular subnet which will investigate the availability of the node by launching a number of fault test requests, if this is support by the host under investigation or by ICMP echo requests if not. The result of the query is then returned to the requesting node. The NFM also functions as proxy for process availability queries in the case where a firewall obstructs the free querying of the nodes by their peers. The NFM's are configured with domain and ACL mechanisms to control access to the information.

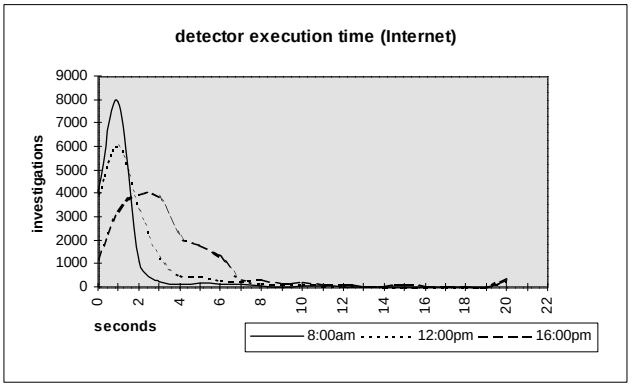
An extension which is under investigation is to have nodes multicast heartbeats with local node information periodically. This information can be collected by the local NFM's and shared in compressed form among the other NFM's in the organization. Local system management tools can connect to an NFM to retrieve the information and set trap conditions.

## Network failure.

In distributed systems build on top of a web of interconnected networks, like the Internet, we have to take network failure into account. Failures at network level are in general related to crash failures of routers and gateways, or to severe degradation of the service level due to network congestion, causing minimum performance requirements to be violated.

The failure investigator will, when not able to reach the node under investigation or a relevant NFM, perform a path search to find the trouble spot in the network. It uses the *traceroute* technique of emitting small messages with limited TTL's, triggering ICMP responses from routers among the path. If an obstruction is found it is reported to the caller. The failure management library offers functionality to keep the obstruction under investigation and to notify the application once the obstruction seems to be removed. This way the process does not need to keep the partitioned processes under investigation but can wait until the connectivity is restored by simply monitoring the trouble spot.

In case the network topology permits it, the investigator can be configured to use alternate paths (using encapsulated-IP) to reach one of the destination NFM's (from Cornell for example it is possible to construct 2 alternative routes to anywhere in California). The request contains sufficient information for the NFM to construct a symmetric return path. Protocols that can exploit this type of information are under development[10,12].




## Result Summary.

Failure investigation of a process at the same subnet has always been viewed as a reasonably accurate. Reasons for false suspicions were overload in the receiver OS (e.g. UDP message buffering), which could cause high message loss, or unresponsiveness due to application overload (although that could be seen as a design error). Although confident about the result, one was never **guaranteed** that the process had truly crashed. Using the OS failure management extensions, this assurance is now available.

The time needed by the failure detector to come to a result has been greatly reduced in the optimistic, common case that the node on which the process was running is reachable. Regardless if the process has failed or not, the node is able to indicate whether or not the process has crashed. In general a single round-trip time is sufficient at the local network to get a result. In the wide-area case this time is a function of the level of congestion in the network path. The graph above this section shows the measured execution time of the failure detector in the Internet under different congestion levels (executed 12,000 times).

The OS extensions also improve the confidence in the failure investigation process in the wide-area case. Using the old strategy of simply polling a process until a time-out occurs gives much less confidence in the result of the failure investigation. If no response was received after the maximum number of retransmission is reached, it was not certain whether this was because of network failure, host failure or process failure. With the new scheme it is possible to distinguish among these different failures.

The full paper contains the detailed results of the trace study of the accuracy and performance of the failure detector in the Internet, the effectiveness of its partition detection mechanism, process/host failure measurements and measurements of failure detection for server fail-over in high-speed networks.

## References


[1] Amir, Y., Dolev D. , Kramer, S. and Maki, D., Transis a Communication Subsystem for High Availability, Digest of Papers of 22nd IEEE FTCS, 1992

[2] Amir, Y., et al. Fast Message Ordering and Membership Using a Logical Token-Passing Ring. Proc. of ICDCS-13, May 1993.

[3] Birman, K. and Joseph, T., Reliable communication in the presence of failure, *ACM TOCS*, Feb. 1987

[4] Babaoglu, O., Davoli, D. And Montresor, Failure Detectors, Group membership and View-Synchronous Communication in Partitionable Asynchronous Systems, Technical Report UBLCS-95-18, University of Bologna, November 1995.

[5] Chandra, T. and Toueg, S., Unreliable Failure Detectors for Reliable Distributed Systems, to appear in *Journal of the ACM*.

[6] Fisher, J., Lynch., N. and Paterson M., Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, April 1985.

[7] Guo, K., Renesse, R van and Vogels, W., Structured Virtual Synchrony: Support for Highly Reliable, Scaleable group Communication, Cornell Univ. CS Technical Report in preparation, Feb 1996

[8] Holbrook, H., Singhal, S. And Cheriton, D., Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation, Proc. of ACM SIGCOMM'95, Sept. 1995

[9] Malloth, C. and A. Schiper, A., View Synchronous Communication in Large Scale, Proc. 2nd Open BROADCAST Workshop, July, 1995

[10] Malloth, C., Increasing Reliability of Communication in Large Scale Distributed, Proc. of, PDCS-7, October, 1995

[11] Powell, D., editor, Delta-4 - A Generic Architecture for Dependable Distributed Computing, ESPRIT Research Reports, Springer Verlag, Nov. 1991.

[12] Renesse, R. van, et al., Horus: A Flexible Group Communications System University Technical Report, TR95-1500, March, 1995.