AN APPROACH FOR NETWORK COMMUNICATIONS SYSTEMS RECOVERY

GEORGE G. MITCHELL STEPHEN BROWN

Department of Computer Science,
National University of Ireland, Maynooth.
Maynooth,
Co. Kildare.
Ireland.
{georgem, sbrown} @cs.may.ie

ABSTRACT

In this paper we examine the problem of failures within network communications and telecom systems and outline a localised *Invisible Recovery* solution to such systems. We introduce a new approach that makes use of an envelope surrounding an unchanged software protocol layer, which is running as an independent process. This envelope stores just sufficient information on each connection to enable an invisible recovery (IR) of the protocol layer after a software fault. IR provides the means to permit a reconnection to take the place of the original connections. As a result of knowing the necessary attributes that a recoverable network system should have, including the required state information, our proposed technique will allow for the near continuous operation of a network system using software fault tolerance.

We investigate the problems for the integration of the IR technique into existing network protocols. We also explore the use of recoverable techniques on the grounds that they can be used as a means of providing solutions to failures due to malicious attack or software faults. We believe that it is possible to create a *design pattern*, which is neither operating system nor protocol specific. This technique we believe, has particular applications in the high-volume consumer-technology market.

Keywords
Recovery, Fault Tolerance, Communications Networks.

INTRODUCTION

Failures within Complex Systems have been narrowed down to software inadequacies and software errors [1]. There has been an apparent resurgence in the belief that through the use of good software engineering practice it is possible to avoid the introduction of errors during the design and implementation stages of a large system, this has been primarily touted by manufactures of software engineering tools. Although using modern software design methodologies are an effective counter measure to thoughtless errors, which result in general system failures, experience [2] has shown that the use of these alone are inadequate. When dealing with modern Safety Critical Systems such as those in Avionics Control Systems and those systems that must integrate with an existing legacy system, software fault tolerance is necessary and works well [3].

With the initial introduction of Software Fault tolerance in the late '70's by Randell [4] and others, it has become a widely explored area of research. The fundamental components to a fault tolerant system have remained largely unchanged from those described by Lee and Anderson [5], i.e. checking algorithms, error quantification and then error recovery. The advantages and disadvantages of the implementation of fault tolerance across a number of applications have been examined in detail and the best end use of the techniques has also been looked at [6]

One important thing to note about the implementation of fault tolerant systems is that they may be implemented in two forms, the most predominant form is that of code invasive design time fault tolerance, the second and also the far smaller is non code invasive maintenance fault tolerance. We in this paper will examine an approach that is in fact predominantly non-code invasive and which also makes extensive use of software reuse. It is notable that one should strive for the least code invasive mechanism as this clearly reduce the potential for the number of possible faults that could be injected in the system.

With these concepts in mind let us now focus on the issue of network systems, the provision of fault tolerance to network systems is one which primarily explores the problems of hardware failures and also crashes system. The problems of poor connections resulting in loss of data and or continuous reconnection of the underlying protocol daemons has to date been poorly explored. Development of fault tolerant network systems without a consideration for the similarities between the current protocols results in a diversity of fault tolerant mechanisms being employed.

By focusing our research on the principle of developing a technique for network communications recovery we intend to provide a component, IR Invisible Recovery, which is applicable across a wide variety of network protocols. Initially we examine UDP and TCP over IP, but this we believe is only the start of the design of a system that will have a wide scope of use. Another area in which the use of fault tolerant systems is in the network defence and security field, by providing a staggered reduction in the operability of the network rather than a complete failure users manage the shut down of the system in an orderly manor in the event of a fail which is impossible to recover from.

RELATED RESEARCH

A number of differing approaches have been taken to the design and implementation of Software Fault Tolerant, SFT, computer networks. Each of these techniques has made extensions and improvements to the standard SFT mechanisms. The N-Version programming, NVP, technique consists of N diverse programs executing in parallel and of a counter counting the votes of each of the N outputs. By finding a majority in the outputs the process moves on to its next task. In the event of no majority the system flags an alarm and either relies on another FT technique to cope with the fault or in worst case merely writes to a log file the error and waits for manual intervention. Figure 1 shows a simplified NVP system.

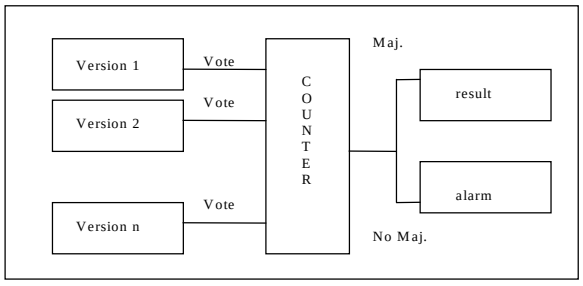

Figure 1. N-Version programming model.

A second widely used technique is checkpoint this can be viewed as the placing of a flag along a path taken, in the event of taking the incorrect turn, it is possible to return to you last good position which has been flagged. Then from this good point it is possible to continue by using new information as if the system had not failed. In fault tolerant systems we implement this technique through the use of checkpoints in critical data or in invasive FT programming checkpoints in the code. It is then possible to recover back to a particular point through the use of a rollback function

Another important technique for the checking of the actual working of a computer system is the acceptance test. This is a check on a condition expected to be met by successful program execution. It is not intended to guarantee complete result correctness and thus includes a wide scope of possible comprehensiveness levels, ranging from a cursory check for anomalous states in the program to completely exhaustive output verification. The effectiveness of an acceptance test, obviously, will be highly dependent on the quantifiable parameters selected. Aiming at too high completeness may lead to large and complex test programs and therefore excessively high costs and design fault proneness [5].

Kim and Subbaraman [7] introduced Acceptance Test Checks at ever point in which the reasonableness of a computation result was in question. Kim's main interest was in over coming 'AT failure' results due to transient hardware faults or design faults in the method of computation and even the possibility of timing failures indicating a violation of the method execution deadline. To do this, the primary and shadow technique was taken to a new level, he used the principle of design diversity (similar to NVP) to create two nodes which were both identical from, a design specification point but were entirely different by means of their implementation. Kim also made use of distributed recovery blocks. The primary and shadow nodes were constructed using the active replication principle, this is based on the underlying principles of the Real-time Fault Tolerance DRB scheme [7]. This technique has been in constant re-development over the past 15 years, with each object replicated to form a pair of partner objects running on differing nodes the two partner objects control the updating of there own ODS, object data store. This together with NVP design and implementation provides for a more reliable result for the AT's in question.

At this point we must note that all the previous research was extensively code invasive and what follows is the best of the non-code techniques. This brings us to the research conducted by Tso in SoHaR Corporation [3] over the past ten years. Much of the work they have published has been focused on the use of fault tolerance in the area of critical computer systems such as health care and aircraft control. The extent of there research in the areas of networks is also interesting, although they have developed all of there systems with the use of one single environment, Ada. By the use of recovery block programming they provide a recoverable mechanism to network systems primarily Distributed Systems. Again all of their techniques are code invasive except for reusable watchdog processes that are incorporated ion to all of their systems.

González [8] developed a method for a graceful degradation of a system which was intended to provide a mechanism to allow the correct control of an avionics system but his technique raised the idea of allowing our system to continue to work for a length of time so as to permit the controlled shut down of the system if it so happened that it was impossible to continue to work with our recovered connection.

Huang and Kintalas [6, 9] fault tolerant network systems, made use of application level software components for the detection and recovery from faults not handled by the operating system or the hardware fault mechanism built

into the actual networking computer system. What was significant about this work was the novel idea of a self recovering component which was a watch dog daemon that monitored the running of the system and was merely used to flag the occurrence of a detected error (fault) this ran in parallel with the rest of the system and was implemented in the for m of a C library for inclusion in the implementation / design of the system. Although the self-recovering daemon was not used directly in the actual recovery of the system it did raise the possibilities which we intend to explore in our work. Whilst Huang and Kintalas work was developed to cope with static errors we are more interested in the exploration of software fault tolerance in applications directly affected by transient software failures / errors.

INVISIBLE RECOVERY

Our invisible recovery technique takes on three separate and distinct parts:

- Timing
- Restart
- Protocol Mangling

Time:
Crucial to the correct operation of many network systems is the timing both of packets and the overall session. The OSI protocol stack introduced session layers that provided a compensation operation for intermittent connections by token management and check-pointing data transfer. This permits multiple network sessions to be integrated together to form one continues data transfer. This type of method is not available in all network and telecom protocols for example, TCP / IP, the timing and also the non-interrupted communication between networked machines is crucial to the continues operation of the protocol. The length of time that a packet remains unacknowledged is directly linked with the retransmission of the data packet in question. Our system must contend with this and it is anticipated to take no more than a 2-3 seconds for the complete resumption of normal service, through the use of our restart / protocol mangling.

Restart:
The initial communication required for a connection between networked computers normally requires the issuing a number of protocol data units (PDU). Each of these are used to enable the client to register itself with the server, typically 3 to 5 PDU's are required, in the TCP protocol, 3 PDU's are used in the well know 3-way hand shake. Our IR system operates in the form of an envelope that surrounds an unchanged protocol entity, with this design it is possible to operate between upper and lower network layers providing a "restartable" component for most telecommunications and network protocols. Our envelope makes a data copy of all data that passes through the layer. On detection of a fault our restart component reinitialises the connection by reissuing PDU's and then creating a new connection. Our envelope, running as a process, uses a database containing the state information of the application process and also the protocol process. Typically this consists of; port number, IP address, unacked_data, last_seq_number, last_acked_request, etc. This provides sufficient information to enable the IR process to restart and to then hide the failure of the underlying network entity, we accomplish this with the aid of *protocol mangling.*

Protocol mangling:
This is essentially a mapping of all the last known good state information to the current new state information. It is necessary so as to cope with the newly regenerated protocol entity that will have been "reincarnated" with differing sequence numbering, possibly different port information and possibly incorrectly matched data. We use the stored data to correct the data problems of acknowledged verses unacknowledged data. Then by mangling the state information such as old_port address to new_port addresses, old_ip to new_ip and so forth we make it is possible to disguise the fact of a protocol entity failure, and continue to operate correctly.

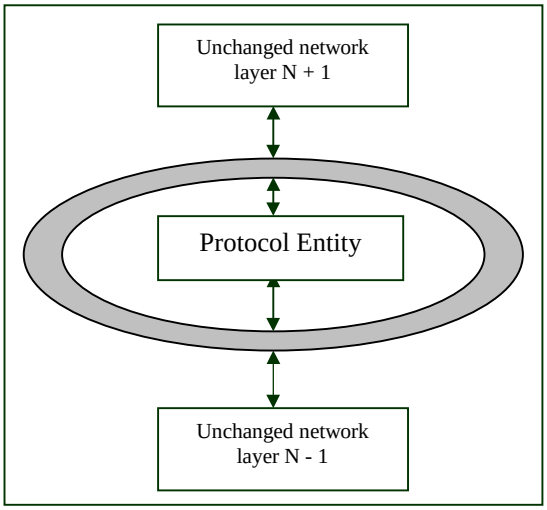

Figure 2. IR envelope communication

CURRENT WORK

Currently we are in the process of validating the operation of our technique. Examining Figure 2 we see the unchanged network layers and the envelope surrounding the protocol entity that may have undergone a localised failure. What is important to note here is the requirement for the extra two jumps incurred in our system, although this is a slight overhead we believe that it is small when compared against a complete system failure which may have occurred when our technique is not in use. The relationship our system has with other systems is depicted in Figure 3 here our system, with its IR process running between the network application and the network process,

can communicate effectively with a standard system which remains unchanged.

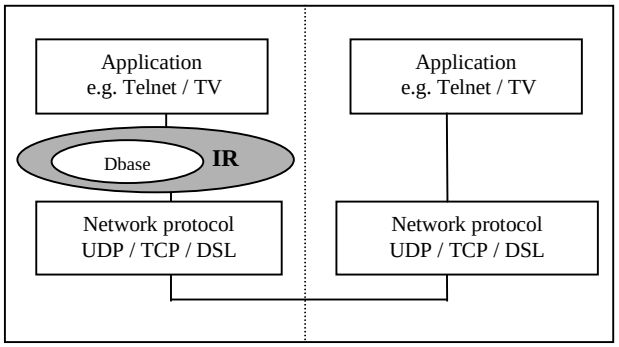

Figure 3. IR system communications with standard system

VALIDATION & TESTING

Validation of the correctness and performance of the invisible recovery model is currently in progress. We are using LINUX as the initial operating system environment, as it provides a number of highly desirable attributes that we required in the construction of our test model. By effecting the inetd process which is in control of the socket requests for the system it is possible to cause a protocol failure and examine the quality of our technique. We have of late decided upon the use of SFIT, Software Fault Insertion Testing. The testing of transient errors is completely unpredictable but it is possible to mimic them by inserting errors to cause a process crash or even by the removal of the hardware network connection. SFIT the Federal Communications Commission (FCC) Network Reliability Council (NRC) conducted an extensive study that involved all major key telecommunications / computer networking system suppliers and research institutions and issued a report entitled *Network Reliability: A Report to the Nation* [10]. In this report Software Fault Insertion Testing, *SFIT* was recommended to be performed as a standard part of a telecommunications system supplier's development process for improving fault tolerance.
We intend to measure the restart time required for the system and also the overhead impact of introducing two further data jumps in the protocol this together with the small amount of data coping that is required will have a performance issue but one which will be small.

CONCLUSION & FUTURE WORK

Invisible Recovery is important for the provision of software fault tolerance to existing network protocol systems. It enables code reuse and provides a high availability of the protocol entities that are surrounded buy the protective envelope. With knowledge of the necessary attributes that a recoverable network system must have, we can derive a design pattern for use on a wide range of protocols.
Once we have validated our design pattern on UDP and then TCP, we intend to complete further work on validating the model for significantly different protocols such as ADSL or xDSL. With the increased interest in providing digital television to the domestic market our technique should enable the recover of such systems in a fast and low cost manner.

REFERENCES

[1] J.Gray, *Why computers stop and what can be done about it?* Proceedings of the 5th Symposium on the Reliability in Distributed Software and Database Systems, 1986, 3-12.

[2] A. Spector & D. Gifford, *The Space Shuttle Primary Computer System*, Communications of the ACM, 27(9), 1984.

[3] K. Tso, E. Shokri, A. Tai , R. Dziegiel, Jr. *A Reuse Framework for Software Fault Tolerance*, AIAA Computing in Aerospace 10 Conference, 1995, 490-500.

[4] B.Randell, *System structure for software fault tolerance*, IEEE Transactions on Software Engineering, SE 1, June 1975, 220-232.

[5] P.A. Lee, and T. Anderson, *Fault Tolerant Principles and Practice 2 ed.*, (Wien - New York: Springer - Verlag, 1990).

[6] M.R. Lyu, *Software Fault Tolerance*, (New York: John Wiley and Sons Inc, 1995).

[7] K.H. Kim and C. Subbaraman, *Fault Tolerant Real-time Objects*, Communications of the ACM, 40(1), 1997, 75 - 82

[8] H. Gonzàlez, *Adaptive Fault Tolerance and Graceful Degradation Under Dynamic Hard Real-Time Scheduling*, Proceedings of the 18rth IEEE, RTSS, 1997.

[9] C. Kintala, *Software Implemented Fault Tolerance: Technologies and Experience*, Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing, 1993, 2-9.

[10] NRC, *Network Reliability: A Report to the Nation*, Network Reliability Council, USA,1993.