Chapter 11

Fail-Silent Hardware for Distributed Systems

The architectural frameworks assumed in this chapter are those of OSA and XPA, and are summarised below:

a) distributed computations are assumed to be structured as software components communicating via messages;

b) XPA: software components execute on fail-controlled nodes with the *fail-silent* property: a node either functions according to the specification or stops functioning;

   OSA: the fail-silent property is not essential for nodes, so software components can execute on ordinary (potentially) fail-uncontrolled nodes; however, all the protocols for message passing are executed on fail-silent hardware (the Network Attachment Controllers, NACs);

c) nodes communicate with each other through redundant communication networks;

d) software components can be replicated on distinct nodes for increased reliability; the degree of replication (if any) for a software component will be determined by the failure characteristic of the underlying nodes: K+1 replicas can tolerate up to K replica failures if the nodes are assumed to be fail-silent, whilst 3K+1 replicas are needed if the nodes are assumed to be fail-uncontrolled.

This chapter concentrates on the issues concerning the design and implementation of fail-silent hardware (nodes and NACs). The basic idea is to replicate processing on two processors that can check each other's performance to form a self-checking processor pair. Since a NAC is really a processor with a network interface, we will begin our discussion by considering how a general purpose, fail-uncontrolled processor can be made into its fail-silent counterpart (a fail-silent node); later on we will discuss some specific details of the fail-silent NAC design. We discuss and evaluate two different approaches to the construction of fail-silent nodes.

11.1. Fail-Silent Node Models

11.1.1. General

Individual hardware components are not inherently fail-silent. For some devices, simple models of their correct behaviour exist, e.g., memory devices should output exactly the data that was originally input to them. In these cases, faults can easily be identified by the addition to the data of redundant information, e.g., parity bits or CRCs, which can be checked when the data is output. However for complex devices, for which there is no simple correlation between their inputs and subsequent outputs, e.g., microprocessors, the easiest method of adding redundancy is to duplicate the device and compare the outputs of the two devices.

Fail-silent nodes have been used widely, for example in commercial transaction processing systems (e.g., [Bernstein 1988]). Such nodes have been designed with the assistance of specialized comparator hardware and clock circuits. A common (reliable) clock source is used for driving a pair of processors that execute in lock-step, with the outputs compared by a (reliable) comparator; no output is produced, once a disagreement is detected by the comparator. We term a node designed this way to be a *hard fail-silent node*. An alternative design being developed within the Delta-4 project requires the processors of a node to execute clock synchronization and order protocols "to keep in step". A node implemented according to this design will be termed a *soft fail-silent node*, since no special clock or comparator circuits are employed. Such nodes can provide an attractive alternative to hard fail-silent nodes since no special hardware assistance is required. Furthermore, as the principles behind the protocols do not change, the protocol software can be easily ported to any pair of processors (including the ones expected to be available in future). In essence, hard fail-silence is implemented in hardware and is transparent to software so that standard software will run. Soft fail-silence is implemented in software and is transparent to hardware so that standard hardware may be used. Note that since only two processors are used within a node to check on each other, the fail-silent characteristics of a node can be guaranteed only if no more than one processor within a node is faulty.

Intuitively, fail-silent behaviour ought to mean that a node never generates an erroneous output, i.e., the node can only either generate correct outputs or remain silent. However, this is impossible to implement since output messages take a finite time to transmit, and a fault may occur leading to an error during the transmission of a message. A definition of fail-silence must include the case where a message receiver rejects such erroneous messages. Thus a two-processor node will be said to exhibit fail-silent behaviour in the following sense: the outputs produced by it (if any) are either valid messages or *detectably invalid* messages; this behaviour is guaranteed so long as no more than one processor in the node fails.

11.1.2. Hard Fail-Silent Nodes

In a hard fail-silent node, duplicated microprocessors are closely coupled and run in micro-synchronization. Each component is initialized to an identical state and then performs identical actions on identical data on a clock by clock basis, so that on every clock cycle the data output by the components is identical. The principles underlying the node architecture can be explained easily by examining figure 1. Since the data streams to be compared are in exact lock-step, a simple hardware comparator can be used to check that the data streams are identical and to prevent any outputs once a discrepancy is detected. Although two replicas of software are actually running, because they are micro-synchronized and compared by hardware, the software takes no part in the replication and comparison process and is "unaware" of it. Thus the replication is transparent and imposes no requirements on the behaviour of software. Indeed, a hard fail-silent environment may be produced such that software is wholly unable to distinguish it from a standard non-duplicated environment. When hard fail-silence techniques are used, the correct and erroneous message sets sent over the network are distinguished by the fact that the only erroneous messages that can be sent are incomplete correct messages, since the occurrence of a fault during the transmission of a message can stop transmission within one clock cycle. Such incomplete messages are easily identified by the receiver since they will contravene the lowest levels of network protocols.

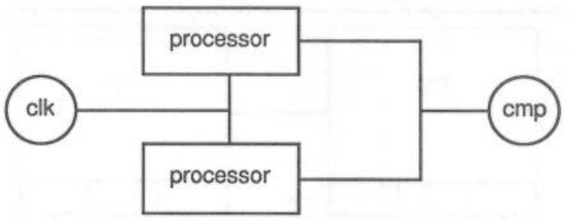

Fig. 1 - A Hard Fail-Silent Node using Duplicated Processors

11.1.3. Soft Fail-Silent Nodes

In the case of hard fail-silent nodes, there is no problem of replica group determinism other than the hardware design issue of ensuring identical results of response to external asynchronous events, so for this purpose replication has no consequences to software design. In contrast, soft fail-silence nodes cannot be expected to run arbitrarily structured programs and preserve replica determinacy. In essence, programs have to be structured as state machines [Schneider 1990]. We assume, as stated before, that distributed computations have been structured as a number of software components (referred to as processes in the rest of this chapter) that interact only by way of messages. A process has an *input port* through which it receives all the messages directed to it and an *output port* through which it can send messages to other processes. This simple model is sufficiently general in that other models, such as clients and servers interacting through remote procedure calls, or objects communicating by messages can be seen as special cases (although, certain enhancements to the model are possible, and will be discussed subsequently). We assume that computations performed by processes are *deterministic* so that if all the correctly functioning replicas of a process have identical initial states then they will continue to produce identical responses to incoming messages provided the messages are processed in an identical order.

The overall node architecture is shown in figure 2. Each of the two processors (P_1, P_2) has network interfaces (n_1, n_2) for inter-node communication over (redundant) networks; in addition, the processors are internally connected by a communication link, l, for intra-node communication needed for clock synchronization and order protocols. Each non-faulty processor in a node is assumed to be able to *sign* a message it sends by affixing the message with its (the processor's) unforgeable signature; it is also assumed to be able to *authenticate* any received message, thereby detect any attempts to corrupt the message. Digital signature based techniques [Rivest et al. 1978] can be relied upon to provide such functionality.

It is necessary that the replicas of computational processes on processors within a node select identical messages for processing, to ensure that they produce identical outputs. Identical message selection can be guaranteed by maintaining identical ordering of messages at input ports and ensuring that processes pick up messages at the head of their respective input ports. An *order protocol* is then required to ensure identical ordering if both processors are non-faulty.

An implementation of this order protocol will require that the clocks of both the processors of a node are synchronized such that the measurable difference between readings of clocks at any instant is bounded by a known constant ε. Algorithms for achieving this abstraction exist (see [Halpern et al. 1984, Lamport and Melliar-Smith 1985]). Communication for clock synchronization and ordering takes place by way of the internal link. Given that the clocks of both the processors are synchronized, the order protocol can be implemented using a version of the signed message algorithm for Byzantine agreement [Lamport et al. 1982]. As there are only two processors, the protocol is particularly simple, since it is expected to work only in the

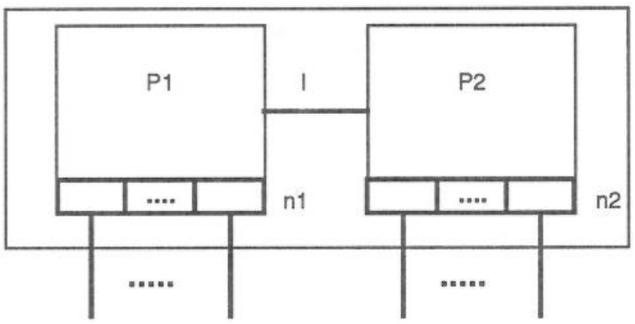

Fig. 2 - Overall Node Architecture

absence of any failures. Essentially, an *order process* of a processor stamps a message to be ordered with its local clock reading. A copy of the time-stamped message is signed and sent over the link to the *order process* of the other processor in the node. If T is the timestamp of the message received from or sent to the order process of the other processor, then the message becomes *stable* at local clock time $T+d+\varepsilon$ where d is the maximum transmission time taken for a time-stamped message to travel from one order process to another order process over the link. A message with timestamp T will be said to become stable, if no message with timestamp $T_1<T$ will be received by an order process. Stable messages are queued at the relevant input ports in the increasing timestamp order (with care being taken not to queue a stable message, if its replica has already been queued).

Each non-faulty processor of a node has five processes:

a) **Sender Process**: this process takes the messages produced by the computational processes of that processor, signs them and sends them via the link to the neighbour processor of the node for comparison.

b) **Comparator Process**: this process authenticates all incoming messages from the neighbouring processor; an authenticated message (m_i) is compared with its counterpart produced locally. If the comparison succeeds, the authenticated message m_i is counter signed (by considering the first signature as a part of the message) and this double signed message is handed over to a *transmitter process* for network delivery to destination nodes. A message that cannot be compared because its counterpart does not arrive within a time-out period or a comparison that detects a disagreement indicates a failure. Once a failure is indicated, the comparator process stops, which results in no further double signed message being produced from that node.

c) **Transmitter Process:** this process is responsible for sending the doubly signed messages to destination nodes.

d) **Receiver Process:** this process accepts authentic messages for processing from the network, discarding any duplicates; such valid messages are sent to the local *order process.*

e) **Order Process**: this process executes the order protocol (mentioned earlier) with its counterpart in the other processor and attempts to construct identical queues of valid messages for processing by the computational processes.

A correctly functioning node will generate two identical copies of its output messages. A receiver process at a node will discard any duplicate messages received over the network. When the comparator process of a non-faulty processor in a node detects a failure and therefore stops, no new double signed messages can be emitted by the node; any messages coming from this node that are not double signed will be found to be unauthentic at the receiving nodes. Any authentic but old messages from a faulty node will also be discarded by the receiving nodes as replicas of already received messages.

11.2. A Comparative Evaluation

11.2.1. General

Certain general observations regarding the two architectures can be made at the outset. The main advantages of the soft fail-silent nodes are that: (i) technology upgrades appear to be easy; since the principles behind the protocols do not change, the protocol software can be easily ported to any pair of processors (including the ones expected to be available in future); (ii) the second advantage is that since the replicated computations are loosely synchronized, the architecture is likely to be capable of detecting common mode transient failures. This is because transients are unlikely to affect the computations on the processor pairs in an identical fashion. The advantages of the hard fail-silent nodes are: (i) the ability to execute arbitrary programs that are not necessarily deterministic — this is because the common clock source ensures lock-step synchronized execution; and (ii) better performance than the software-based approach since there is no need for protocols for message ordering.

The disadvantages of the two approaches are very much the converse of the above characteristics. For the software fail-silent nodes, these are: (i) the restriction that application programs need to be deterministic; in particular this requires that potential sources of non-determinism, such as, interrupts and time-outs have to be treated as messages, thus requiring careful treatment; and (ii) concern over the performance of the node due to overheads of the various protocols described above. For the hardware constructed fail-silent nodes, the disadvantages are that every new microprocessor architecture is likely to require substantial design overheads and that tightly synchronized processors may not be resilient to common mode transient failures. Furthermore, lock-step synchronization at very high clock speeds (50 to 100 MHz) may well turn out to be difficult to achieve. We next discuss several specific issues in detail.

11.2.2. Replication

Figure 3 shows a typical fail-uncontrolled node, with constituent components. For this discussion, we will use the term *network interface* (NI) to refer to a simple interface to the network, such that most of the communications protocol is executed by the host processor. The term *network attachment controller* (NAC) will be used to refer to the device with a processor and NI, with the capability of executing virtually all the communication protocols without involving the host processor. A logical way of constructing the fail-silent version of the above node would be to duplicate all the above components. In this subsection we will examine the basic issues concerned with replication: not surprisingly, they are hardware related for the hard fail-silent nodes and software related for the soft fail-silent nodes.

11.2.2.1. Hard Fail-Silent Nodes. One important design consideration is to minimize the need for specially designed interface components. If duplicate busses are employed then this requirement cannot be met, making this form of replication unattractive. On the other hand, many busses (e.g., VME) do not employ any redundancies in their data and address paths, so such busses must be considered potentially fail-uncontrolled. Figures 4 and 5 show two designs based on the assumption that busses are fail-uncontrolled, and not replicated (bold

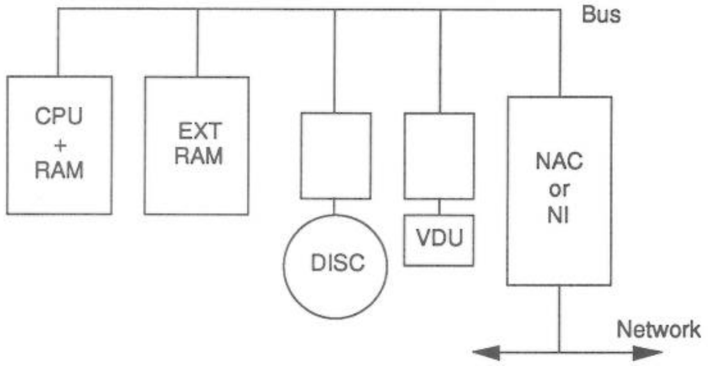

Fig. 3 - Typical Fail-Uncontrolled Node

boxes represent fail-silent components). We note that RAM has been duplicated, rather than being interfaced directly to the bus (a RAM with error detection capability can still be vulnerable on a fail-uncontrolled bus). The common clock that is necessary within a fail-silent component can constitute a single point of potentially uncontrolled failure unless it is itself implemented to be fault-tolerant (see, e.g., [Moreira de Souza and Peixoto Paz 1975, Moreira de Souza et al. 1976]). Also, even though comparators may be built that are self-checking (see, e.g., [Wakerly 1978]), the final comparator output checker can also constitute a single point of fail-uncontrolled behaviour.

Data held on non-replicated discs needs to be protected using checksums or similar forms of redundancy, which must be generated and checked within fail-silent environments (CPUs, figure 4 and in addition NACs, figure 5). Similarly, messages to be transmitted over the network must contain redundant information (checksums, CRCs) generated and checked within fail-silent environments.

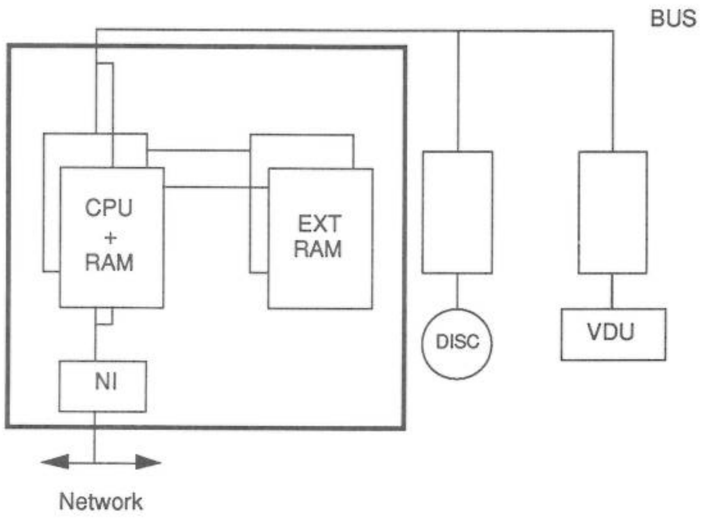

Fig. 4 - Hard Fail-Silent Node *(design 1)*

Methods of handling the interactions between two asynchronous clock domains, each run in lock-step synchronization, are well-understood. Applying such methods requires access to and special treatment of the interface between the domains. The interfaces between asynchronous clock domains are often buried deep within such VLSI chips and so are quite inaccessible. Fortunately, clock signals of processors are accessible, so processors can be replicated. The particular case of concern in this regard is the VLSI Network Interface controller chip, whose clock signals are not accessible, making it impossible to duplicate. An alternative would then be that the chip be discarded and several boards-full of MSI and LSI components be substituted. Such a development is not considered practical, given the over-riding desire of using standard VLSI components. Thus the sensible approach is not to duplicate the NI, but to regard it as a part of the network, which is treated as fail-uncontrolled. For this reason, only a single NI (which is potentially fail uncontrolled) is shown in figure 4. In the similar fashion, the two NACs will share some common interface circuits (a single fail-uncontrolled NI).

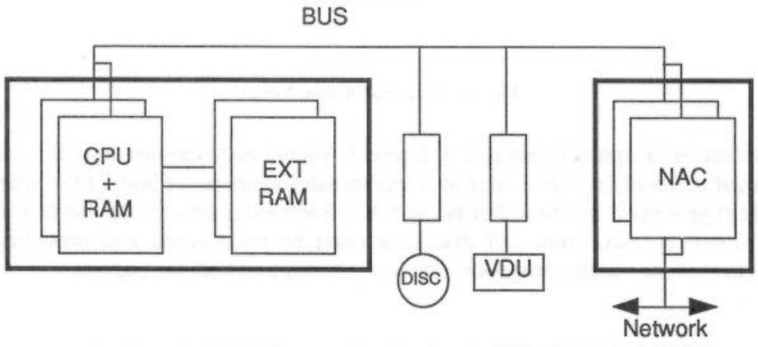

Fig. 5 - Hard Fail-Silent Node *(design 2)*

11.2.2.2. Soft Fail-Silent Nodes. In this architecture, two fail uncontrolled nodes (figure 3) can be duplicated as illustrated in figure 6; the node can be built entirely from standard components. It must be stressed however that the guarantee of the fail-silent behaviour rests on the coverage achieved by the signature and authentication mechanisms used. Of course, as observed before, such nodes, unlike the hard fail-silent nodes, cannot be expected to execute arbitrary "black box" software; rather software modules must be structured as state machines (section 11.1). Most practical systems require some additional functionality in their concurrent processing model than that assumed in section 11.1. Typically, this will include the capabilities of selecting a message within a given time interval (waiting for a message with a time-out), selecting the highest priority message from the set of available messages, and responding to interrupts coming from "alarm messages". Naturally, protocols to enforce identical replica behaviour will be necessary to deal with the treatment of such interrupts, priority messages and time-outs; some work in this direction is discussed at length in [Tully and Shrivastava 1990]. We mention it here to indicate that the soft fail-silent architecture presented here can be adapted to incorporate these functionalities. However, it must be stated that performance implications are not fully well understood and that no operational experience is yet available (a prototype node is currently being implemented [Shrivastava et al. 1991]).

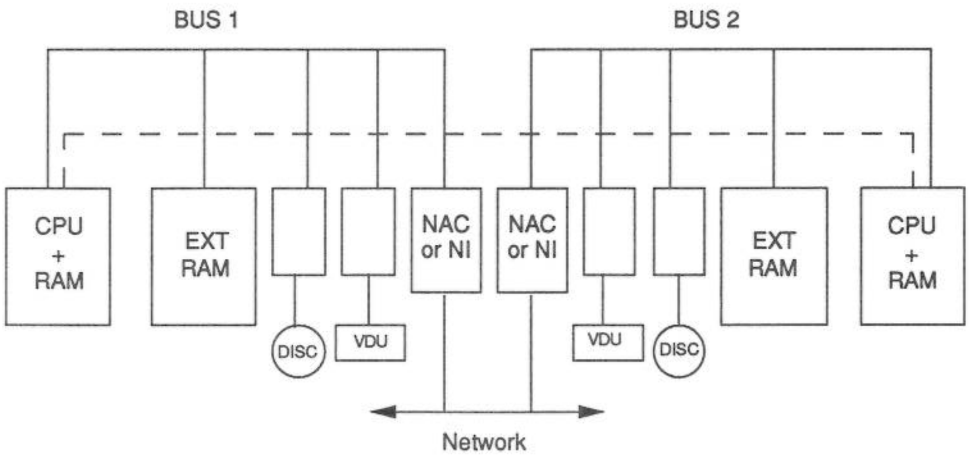

Fig. 6 - Soft Fail-Silent Node

An alternative design is illustrated in figure 7. Such a node employing a fail-silent NAC will be termed a *hybrid fail-silent node* as it incorporates elements of both of the architectures: soft fail-silent processor and hard fail-silent NAC. Such a design permits use of a single bus and disc. However, two copies of disc data must be maintained and protected against corruption (to prevent a faulty processor identically corrupting both the copies).

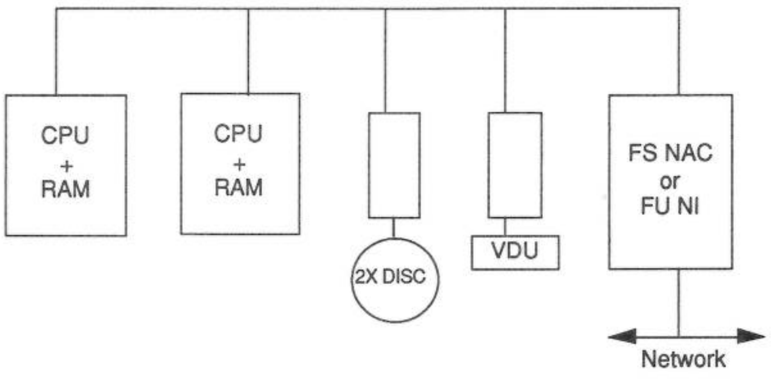

Fig. 7 - Hybrid Fail-Silent Node

11.2.3. Discs

The disc drives are one of the most expensive components of a node, and require particular attention. Various techniques are described in the literature to achieve very reliable but expensive fault-tolerant disc subsystems; typically this involves mirrored discs dual-ported to dual disc channel cards, with special attention paid to live-unplugging of busses for maintenance purposes. Since we are only looking for fail-silent properties rather than fault-masking or recovery, the problem is simplified. With much less cost and performance penalties, this can be met by added redundancy without full duplication. If standard discs and channels are to be used, the redundancy must, however, be added and checked within a (hard or soft) fail-silent environment elsewhere in the node. In the case of hard fail-silent nodes, this means that disc drivers resident in the fail-silent host must add and check the redundancy information of all data held on disc that could affect the correctness of computation leading to output messages. Only one copy of each data item is held. Soft fail-silent nodes really need duplicated discs, unless the hybrid arrangement is used which will require some form of data protection technique.

11.2.4. NIs and NACs

In Delta-4 XPA, the fail-silent host assumption should lead to two simplifications: (i) communication protocols (e.g., for reliable multicast) may reside in either the fail-silent host or in the fail-silent NAC; and (ii) there is no need for voting protocols for managing replicas. As a consequence of (i), the use of simple NIs, with protocols implemented within the device drivers running on fail-silent hosts becomes an attractive design option. Work on the design of protocols for soft fail-silent nodes with NIs has been reported in [Ezhilchelvan and Shrivastava 1991].

We now discuss some specific design issues for hard fail-silent NACs that have been constructed for use within OSA. The NAC and the host (which could be either fail-uncontrolled or fail-silent) share a bus (the VME bus, in the particular design under discussion).The main system components of a NAC are: (i) a NAC processor plus RAM pair, acting as a hard fail-silent node (where all the communication system protocols are run); and (ii) a NI with communications RAM (for reasons discussed earlier, the NI's RAM remains non-replicated). A fail-silent NAC must be able to cope with potentially uncontrolled failures in the host, the bus connecting the host and NAC, parts of the NAC associated with the interface or accessible across the bus, the NI and the network. In the case of a fail-uncontrolled host, errors cannot be detected by the host itself as any checksums or similar methods used to validate data might themselves be incorrect. Therefore any checking must be performed by comparing messages generated by replicated software components running on distinct hosts. This checking is carried out in the communication system by voting on active replicas' messages. This can only detect faults that are transmitted on the network and it is possible to have long periods of latency before faults are detected.

For fail-silent hosts, the majority of hardware faults will be detected by the host itself. However, the data that is transmitted from the host can be corrupted. Since this can happen as soon as the data leaves the fail-silent environment, it is necessary to generate some form of checksum within the fail-silent environment if voting on active replicas is not performed. This checksum should not be a simple adding of bits since it will be transmitted over a large area of fail-uncontrolled environment before its validity is checked and therefore a combination of errors could lead to the checksum appearing valid. Any data arriving at the host should be checked for validity. The data should be copied into the fail-silent environment and the checksum should then be used to verify the data. The checksum can then be discarded if required.

The bus must always be considered as fail-uncontrolled if it is not replicated and no checksum or parity is automatically generated (this is the case in the design being considered). Even if the bus itself could be considered as reliable, the interface at either end certainly cannot. Therefore, as mentioned above, a checksum of some sort must be generated from within the fail-silent environment on the host or NAC if the data is being transmitted from the fail-silent part of the NAC. This checksum must be checked wherever the data is used.

Data that is transmitted to the NAC across the bus can be placed in either the communications RAM (non-replicated) or the processor RAM (replicated). Both the host and the network interface may perform direct memory accesses into the RAM of the fail-silent NAC. This allows messages to be passed without extra copying processes between protected and unprotected memory areas. This means that a fault in either the host or the network interface could cause corruption of data or program within the NAC. To prevent this occurring, while allowing messages to be placed in the RAM and then protected whilst being processed, a dynamically controlled protection scheme is provided which allows the duplicated processors to control the protection on their respective RAMs. Each environment has two masks that may only be altered by the NAC processor in the associated environment. One mask in each environment prevents the host from writing to the RAM, the other prevents the network interface from writing to the RAM. Each bit in each mask protects 128 bytes in the RAM, so that the NAC software can allocate buffers to each interface in granules of 128 bytes. An attempt to write to a protected granule causes an interrupt to both NAC processors so that the NAC software is aware that the host or the network interface may be faulty. No write occurs, but the access terminates normally. This protection mechanism can be tested at system start up, and therefore it would require failure of the independent protection mechanisms on both RAMs before a fault in the host or network interface could cause a RAM corruption that might lead to a malicious fault.

Since these masks are the only means of protection from an arbitrarily faulty host, they are replicated. This should prevent a failed host from transmitting apparently correct messages over the network. It has already been stated that any data transmitted over the bus needs to be checked. If the data is to be altered by the NAC, it is essential that software on the NAC checks the validity of the data before altering it. If data is altered the NAC must also generate a new checksum in order for the data to be recognised as valid at its destination. This checksum must be generated from within the fail-silent environment since if it is not, the data that is being read cannot be relied upon and therefore a valid checksum could be generated for incorrect data. Wherever possible, to reduce processing and transfer overheads, data received from the host is not altered on the NAC, but additional data is added around the original data, i.e., an envelope is created which encapsulates the original data. The additional data should have an additional checksum or checksums added and should be produced within the fail-silent environment and should not rely on any of the original data that has not been validated. This will ensure that the initial checksum generated in the fail silent part of the host is preserved and therefore the encapsulated data can still be verified against the original checksum.

Any data that is received by the NAC over the network must carry out a check on the envelope around the initial data and make sure that no corruption of data has occurred in transmission. The envelope should then be stripped from the original data and this data transmitted to the host. Any data that is to be transmitted over the network should already have had checksums added within the fail-silent part of the NAC and this can therefore be checked at the destination NAC. The network interface is effectively treated as part of the network and therefore liable to data corruption.

The design just discussed has illustrated the practical considerations that go into the construction of fail-silent hardware, particularly when some of the components cannot be duplicated.

11.3. Concluding Remarks

In addition to meeting technical criteria, to be marketable the system implementation must also have other attributes.

a) **Visibility:** The fault tolerance of the system must be made visible and comprehensible to the customer.

b) **Credibility:** The dependability mechanisms must be credible to customers. The mechanisms must also be developed and tested to high quality control standards; it is expected that it will be necessary to offer such evidence to potential customers. The fault injection testing of NACs (see section 15.3) has been performed with this requirement in mind.

c) **Cost-effectiveness:** The perceived gains in dependability must outweigh the increased cost and performance penalties of a fault tolerant system, and do so at least as well as competitive methods (see also section 15.4, where dependability evaluation is discussed).

In this respect, both hard and soft fail-silent approaches provide visible and credible solutions; redundancy within a node is visible, and its exploitation can be explained to non-expert customers. The ability to run third party software, e.g., concurrent database systems with non-deterministic behaviour, with minimum performance penalty makes hard fail-silent nodes cost-effective. Factors working against such nodes are that there could be market resistance from customers to the use of non-standard processor and NAC boards and that new processors require considerable re-design effort. In contrast, the performance of a soft fail-silent node can never be made as good as its hard fail-silent counterpart; furthermore, not every third party software can be made to run on such nodes without any changes. Factors working in the favour of a soft fail-silent node, as observed earlier, are that they can be built entirely using standard hardware components and no major re-design effort is required for new processors. Given these observations, we can only conclude at this stage that both types of nodes have a role to play in dependable distributed systems.