# Immunet: A Cheap and Robust Fault-Tolerant Packet Routing Mechanism

V.Puente, J.A.Gregorio, F.Vallejo and R.Beivide
*Computer Architecture Research Group*
*University of Cantabria, Spain*
*{vpuente, jagm, fernando, mon }@atc.unican.es*

## Abstract

*A new and efficient mechanism to tolerate failures in interconnection networks for parallel and distributed computers, denoted as Immunet, is presented in this work. In the presence of failures, Immunet automatically reacts with a hardware reconfiguration of the surviving network resources. Immunet has four important advantages over previous fault-tolerant switching mechanisms. Its low hardware costs minimize the overhead that the network must support in absence of faults. As long as the network remains connected, Immunet can tolerate any number of failures regardless of their spatial and temporal combinations. The resulting communication infrastructure provides optimized adaptive minimal routing over the surviving topology. The system behavior under successive failures exhibits graceful performance degradation.*

*Immunet reconfiguration can be totally transparent to the applications running on the parallel system as they will only be affected by the loss of those data packets circulating through the broken components. The rest of the packets will suffer only a tolerable delay induced by the time employed to perform the automatic network reconfiguration. Descriptions of the hardware network architecture and detailed synthetic and execution-driven simulations will demonstrate the benefits of Immunet.*

## 1. Introduction

Parallel computers are being widely used nowadays in mission-critical, scientific, engineering and commercial applications. Consequently, it is becoming increasingly important to provide high system dependability. Hence, fault tolerant systems must be designed in order to support uninterrupted service. Recent scalable systems incorporate a combination of hardware and software techniques to provide some fault tolerance capabilities, such as the direction order routing used in the Cray T3E [24]. Furthermore, failure rates dramatically increase when using highly parallel computers. In certain scenarios, the MTBF (Mean Time Between Failures) can be lower than the execution time of the typical applications running on the system. Such is the case, among others, of the IBM BlueGene/L supercomputer [1].

The interconnection network is a key component of a parallel computer whose design greatly conditions system reliability. The network itself can be a source of failures because of both, its parallelism and its distributed nature. Moreover, after any kind of system fault the network constitutes the media to provide communications among surviving nodes.

In this work, we present and evaluate a complete design of a feasible fault-tolerant interconnection network able to provide unlimited system robustness. Our mechanism guarantees safe communications among all the surviving system nodes that remain connected after any number of faults. Our proposal assumes the use of conventional hardware to dynamically detect failures. After an automatic process of network reconfiguration, Immunet will provide new communication paths able to maintain the surviving part of the system in a degraded but optimal operation. Our reconfiguration mechanism is totally transparent to the applications running in the system and consequently, it is not necessary to stop them. Immunet guarantees that only packets in transit through the devices causing the failure could be lost. Obviously, this packet loss must be recovered and managed by the corresponding hardware and/or software mechanisms. In this paper, we will show the robustness and the graceful degradation exhibited by our mechanism as well as an exhaustive evaluation of its tolerable hardware costs.

We will consider two different kinds of hardware failures: link faults and node faults. Link failures cause a communication loss between pairs of nodes and they can be associated to physical failures in the media used to interconnect neighboring network routers. Only bidirectional link failures will be considered in this work. When a node failure appears, that node interrupts communication with all its neighboring nodes. Any module integrating the node can cause these failures. Nevertheless, from the communications point of view, a node fault will be associated with the death of its corresponding network router. Hence, we will consider that the computing node or nodes attached to this router cannot communicate with any other processor in the system. In essence, a failure in any router introduces the loss of all the communication links attached to it. Our mechanism is able to deal with any number and any kind of temporal and spatial combination of network faults. In short, Immunet guarantees

[1] This work has been supported by the *"Comisión Interministerial de Ciencia y Tecnología"* (CICYT), proyect TIC-2001-591-C02-01.

communication between any pair of computing nodes as long as a physical path exists between them.

A large number of fault-tolerant routing mechanisms have been previously proposed. Some of them, which are related to our proposal, are described in references [3][6][7][10][13][25][26]. None of them is comparable to Immunet. There also exist several works not directly related with fault tolerant networks but with dynamic reconfiguration in high-speed local area networks [2][5][18]. However, the majority of them require discarding application traffic and, in general, they are not able to manage any spatial or temporal fault distribution. Moreover, they use software approaches that are not a desired solution in our context.

We do not know any other integral reconfiguration mechanism like the one presented in this work, exhibiting such a low cost and such a large coverage. In the previously mentioned proposals, the hardware and software cost for implementing the required functionality have important negative impacts on the performance of the system in absence of failures. The implementation of some solutions like the ones presented in [3][7][25] implies an important hardware cost and introduces a non-negligible degradation in the fault-free network performance. This is one of the main reasons for which real systems hardly incorporate efficient fault tolerance mechanisms. Additionally, most of these proposals introduce serious restrictions on the temporal and spatial combinations of components under failure conditions. Even considering a low coverage of faults, they exhibit serious restrictions. For example, in [6][10][13], link and node faults cannot arise anywhere and successive failures cannot appear anytime.

The rest of the paper is organized as follows: In Section 2, we describe the context where our mechanism will be used. Section 3 presents the basis of Immunet. Section 4 is devoted to describe the distributed algorithms that constitute the foundations of our mechanism. Section 5 presents a sketch of a hardware implementation of Immunet, which demonstrates its feasibility. Section 6 is devoted to analyzing the performance exhibited by our proposal under both synthetic and real traffic workloads. Finally, in Section 7, the main conclusions of the work will be summarized.

## 2. Router Architecture and Network Topology

A router in a parallel computer injects packets from one or more computing elements to the network. Conversely, each router ejects packets from the network to one or more computing nodes. Obviously, the router's function is to convey packets towards their destination. The design of this hardware module has to maximize the use of the network resources avoiding communication anomalies such as packet deadlock, livelock and starvation. Although our fault-tolerant routing can be used in arbitrary topologies, we will focus our attention in this paper on analyzing its application to regular networks.

Figure 1 describes our basic router organization, showing the usual hardware modules: crossbar, buffers, arbitration logic, synchronization, etc. Our router has two virtual channels (FIFOs) per input link to support fully Adaptive Bubble Routing (ABR) [19], [20]. This switching mechanism has recently been selected for the design of the IBM BlueGene/L supercomputer [1]. When using ABR, a subset of the total virtual channels is configured as a safe virtual network in which packet deadlock never occurs. The remaining virtual channels are configured as a fully adaptive virtual network. As long as there are available adaptive FIFOs, ABR always routes packets through adaptive paths. Safe paths are only selected when all the profitable adaptive FIFOs are exhausted, which causes the blocking of all the adaptive routing alternatives.

In our ABR, packets move under two different policies. In the adaptive virtual network, the injection and transit of packets are always regulated by Virtual Cut-Through flow control (VCT) [12]. However, in the safe virtual network the packets advance is also regulated by the Bubble Flow Control (BFC), a mechanism for avoiding packet deadlock in topologies based either on a single ring or on a set of rings.

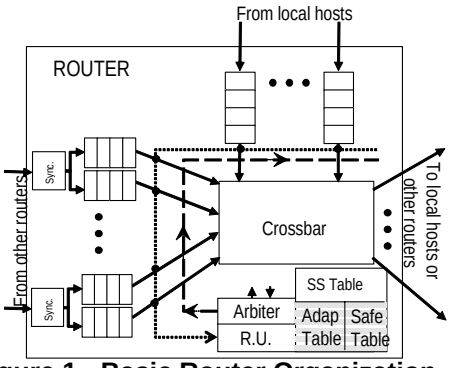



**Figure 1.- Basic Router Organization.**

Although arithmetic routing can be employed in regular topologies, most modern low-to-medium size parallel computers rely on the use of tables. With current hardware technology, the network scalability is not compromised by the size of the tables. The routing table's initializations will be carried out at boot time as in the SGI Spider [9] or the 21364 Alpha [16].

Immunet employs three tables to route packets toward their destination. Two of them are standard routing tables whose sizes depend on the number of system nodes. The *Safe Table* is used to route packets through the safe virtual network and the *Adaptive Table* governs the packet routing inside the adaptive virtual network. The router employs another *Small Safe* (SS) table to record the current shape of

the safe topology inside the node. The SS table size is almost negligible as it only depends on the network degree.

As stated before, we will concentrate our attention on the application of Immunet to regular topologies, namely, *k*-ary *n*-cubes. As is known, these topologies have been frequently implemented in several commercial systems due to both their good cost/performance ratio and scalability [16][24]. Without loss of generality, we will only consider 2D torus to demonstrate our methodology in the rest of the paper.

A 2D torus, or *k*-ary *2*-cube, can be seen as a collection of 4*k* unidirectional rings of length *k*, denoted as *BFC safe rings*. Figure 2(a) shows these 16 rings embedded on a 4x4 torus. Our safe virtual network is composed of these 4*k* rings, which must be visited under Dimension Order Routing (DOR). We employ for its implementation one virtual channel per physical link in every router (a total of $4k^2$ FIFOs). The remaining $4k^2$ FIFOs constitute the fully adaptive virtual network. Packets traveling inside any of the 4*k* BFC safe rings are regulated by VCT but no packet can be injected in any of these rings if it exhausts the buffers of that ring. Packets can be injected inside a BFC safe ring from three sources: from a computing node, from the adaptive virtual network or from another BFC safe ring of a previous dimension. BFC only permits a packet injection in a BFC safe ring if there is space in its local FIFO for, at least, two packets. In this way, there will always be at least one free buffer inside a BFC safe ring (a *Bubble* under our terminology).

Figure 2(b) shows a SS table, which reflects the shape of the safe virtual network topology crossing an arbitrary router. The table has been updated according to the port labeling described in Figure 2(c). Entries set to one indicate an allowed routing action that connects two network ports belonging to the same BFC safe ring. The content of this SS table reflects a fault-free 4x4 torus in which the safe virtual network is configured as the 16 unidirectional rings mentioned before.

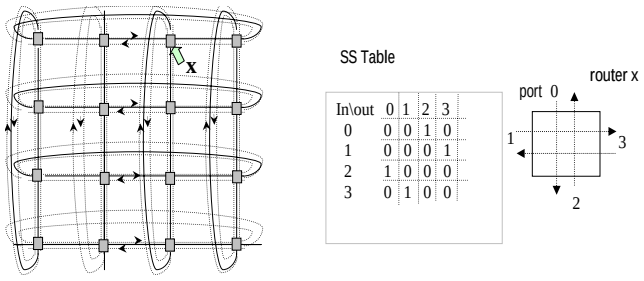



**Figure 2 (a)Safe Paths on a Fault-free 4x4 Torus. (b) Small Safe table. (c) Safe paths inside router "x".**

## 3. The Basis of Immunet

A fault-free ABR 2D torus is able to support a high level of packet throughput $Th_0$, as represented in Figure 3. It must be highlighted that our ABR switching from which we have developed Immunet is, up to our knowledge, the mechanism having the best performance/cost ratio to manage packets in a fault-free interconnection network [20]. A good fault-tolerant network must pursue the maximization of the $Th_0$ value because running programs in a fault-free parallel computer should be the normal mode of operation. Imagine that, at time $t_0$, a fault arises. If the system has the appropriate mechanisms to tolerate this fault, after a certain period $t_r$, the parallel computer should recover and continue in a degraded but normal operation. The performance degradation, $Th_0$-$Th_1$, is caused by both the resources loss and the changes in the resulting topology. Our Immunet fault-tolerant mechanism will provide the parallel system with the necessary functions to reconfigure transparently the network after any number of failures. Due to its low hardware costs and overhead, Immunet maximizes $Th_0$ and simultaneously minimizes $t_r$ and ($Th_0$-$Th_1$), as we will prove in the rest of the paper.

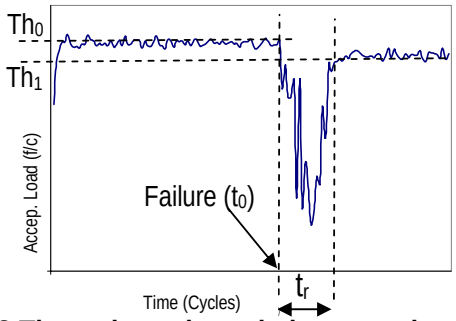



**Figure 3 Throughput degradation: transitory and stationary phases**

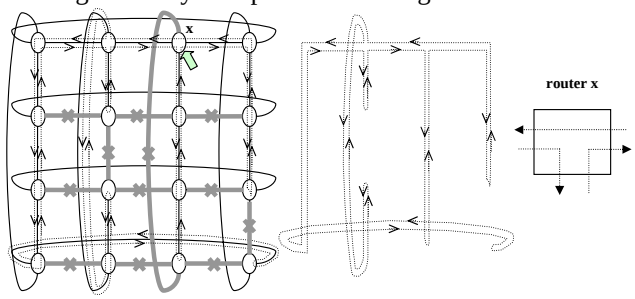
One of the most complex problems inherent in handling any combination of link and/or node faults is that such faults induce topological changes affecting the deadlock avoidance mechanism. For example, a fault in any link of a torus breaks down one ring and, therefore, it is not always possible to use Dimensional Order Routing to route packets through the safe virtual network. Most of the previously proposed solutions add an important number of resources to maintain deadlock-free communications, which notably increases the router complexity.

The first requirement that Immunet must guarantee is that, independently of the number and configuration of network faults, always exists a secure alternative to route packets among the surviving nodes. In [23] we prove that it is always possible to find a unidirectional ring traversing, once or more, all the surviving system nodes after any configuration of faults. Applying BFC over this single ring guarantees the existence of a safe path to interchange packets among all the nodes. A very easy method to obtain

such a single BFC safe ring is based on establishing a "tour" through a spanning tree, always embedded in any arbitrary topology. For example, Figure 4(b) shows a unidirectional ring embedded in a 4x4 torus after the fault of 12 links reflected in Figure 4(a). Applying BFC over this ring assures deadlock-free communications. The rest of the network resources (FIFOs not belonging to the BFC safe ring) can be employed as a minimal adaptive virtual network. Hence, by combining these two virtual networks we can continue using ABR inside the new topology. As packets can be misrouted when entering in the BFC safe ring, the number of movements between virtual networks must be limited in order to avoid packet livelock.

One of the main advantages of our mechanism relies on its low hardware cost. We will employ the same router architecture core as the one used in a fault-free network. Immunet will reconfigure all the router SS tables, as shown in Figure 2, for dynamically recording the configuration of the current safe virtual network inside each node. Obviously, when the network achieves a new topological configuration Immunet must also modify the safe and the adaptive routing tables.

In the following Section, we are going to prove that it is always possible to build automatically the BFC safe ring with a limited cost. Later on, we will also present a low-cost methodology to automatically reconfigure and optimize the safe and adaptive routing tables in order to achieve graceful system performance degradation.




(a) (b) (c)

**Figure 4 (a) A 4x4 torus with 12 faulty links. (b) Its tree-based BFC Safe Ring. (b) BFC Safe Ring topology inside router "x".**

## 4. Immunet Distributed Algorithms

We divide this section into three parts. The first one will be devoted to presenting a distributed algorithm to dynamically generate trees over which we will implement our BFC safe rings. The second and third subsections are dedicated to describe the distributed processes of updating the routing tables used for the safe and the adaptive virtual networks.

### 4.1. Establishing an emergency path

We begin our analysis starting from a healthy network, such as the one described in Section 2. When one or more failures take place, the router or routers detecting them will enter in an *emergency state* that will be propagated to the rest of the surviving network nodes. After stopping new packet injections, the first goal of Immunet is to establish an *emergency path* to guarantee that all the surviving connected nodes can communicate among them. During the emergency state, the emergency path will be the only secure communication medium for all the pending in-transit packets. In addition, as we will see below, the emergency path will be the communication support used to reconfigure the routing tables. Once ended the emergency state, the emergency path will act as a safe virtual network. Remember that our safe virtual network will be a BFC unidirectional ring obtained from a "tour" through a spanning tree always embedded in the surviving topology.

The presence of failures induces changes on the network topology. The 4*k* BFC safe rings, which visited under DOR guaranteed deadlock-free communications, no longer exists. Consequently, some packets cannot achieve their destination. Hence, to rebuild the new unique BFC safe ring we can only rely on local communications among the surviving neighbors.

For the sake of clarity, we will follow an automatic generation of one of these trees by means of an example. Consider a 3x3 torus such as the one represented in Figure 5. Now, assume that there is a fault in the link connecting nodes #4 and #1. Suppose, initially, that only node #4 detects the failure. To begin with, node #4 will be the root of the tree. Node #4 will communicate to its neighboring nodes #3, #5 and #7 this emergency state by means of a special signal together with its *emergency priority level* (EPL), which is the ID of the root node (#4). After that, nodes #3, #5 and #7 will retransmit the emergency state to their corresponding neighbors until reaching the most distant nodes.

When a node detects a failure or receives an EPL, it enters in a special emergency state. First, the node stops new packet injections. Second, it compares the incoming EPL with its local EPL (in a fault-free network all the local EPLs are initially set to zero). Then, the node analyzes, using a first-come-first-serve policy, the remaining input links searching for other emergency states with higher EPLs. After a bounded time, the node records the highest EPL and sends back a special ACK to the winner, which will be its parent in the tree. Now, in order to determine its children, the router will retransmit the emergency state to all its neighbors except to its parent. Again, after a bounded time, the router will receive the corresponding special ACKs coming from the nodes that accept this router as its parent.

Subsequent communications of emergency states will compare their EPLs with the one recorded in each surviving router. In the same way, the local priority level will be changed if the received EPL is higher than the one recorded in the router, modifying again the shape of the

tree. In this way, every node will know every time who its parent is and who its children are. Hence, we have the necessary information distributed among nodes to rebuild a new safe virtual network in the form of a BFC ring over the basis of the current emergency tree. In fact, to route packets over the emergency path it is enough to update, in each router, the SS table, which records the local topology of the BFC safe ring.

An SS table updated according to a certain port labeling can be seen in Figure 5(b) for router #3. The table reflects the "tour" through the emergency tree described in Figure 5(c). This SS table, directly induced from the emergency tree, records the valid routing actions to traverse this node through the emergency ring. So, for node #3, any packet entering from link #3 will be sent to port #2, any packet entering from link #2 will be sent to port #0 and any packet coming from link #0 will be sent to port #3.

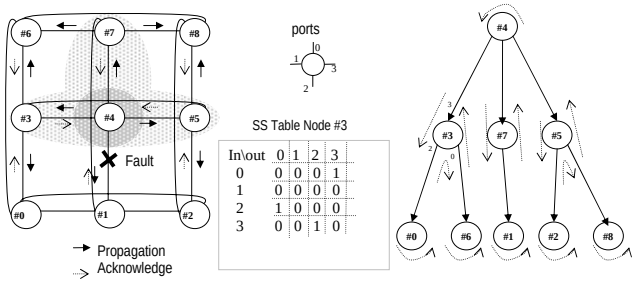



**Figure 5 Example of BFC safe ring rebuilding after a link failure.**

This methodology to generate trees avoids ambiguity problems related with multiple detections of emergency states. If there is only one node detecting a single fault, a router can receive from different links an emergency state with the same EPL. In this case, the selected parent will be the one corresponding to the first detected emergency state. Nevertheless, a more usual scenario is that two or more routers simultaneously detect a failure because all routers behave identically. For example, in Figure 5, nodes #4 and #1 will simultaneously detect the failure of the link that communicates them. Hence, both nodes try to be the roots of two different spanning trees. It is clear that only the root with higher ID will succeed and just one single tree is going to be generated.

Our mechanism can also tolerate nested failures. It is possible that after the detection of a fault, another fault arises in the middle of the generation of one emergency path. In this case, if the node detecting the second fault had lower ID than the one detecting the first failure it would not be possible to obtain the correct tree. This undesirable behavior can be overcome using an EPL that depends not only on the ID of the node that detected the fault but also on the times that the node managed an emergency state. Therefore, in an *N* node network, a router *x* detecting a new fault will generate an EPL equal to $tN+x$, $t$ being the number of emergency states experienced by this node. In this way, the EPL generated for a detecting node after every fault is always different. Therefore, we can assure that only one spanning tree will succeed in implementing the BFC safe ring regardless of the number and the configuration of faults.

Figure 6 shows several tree propagations that illustrate the cases of multiple emergency detections and nested failures. Initially, nodes #1 and #4 simultaneously detect the fault of the link that connects them. Each of these nodes tries to become a root and begin the corresponding tree generations, propagating their emergency states. Node #7 will receive simultaneously two emergency notifications but it will just consider the one with higher EPL, i.e. #4. Transmission of emergency states is represented by continuous arrows on the left tori of Figure 6 and the special ACKs sent back only to the winner parents are represented by doted lines.

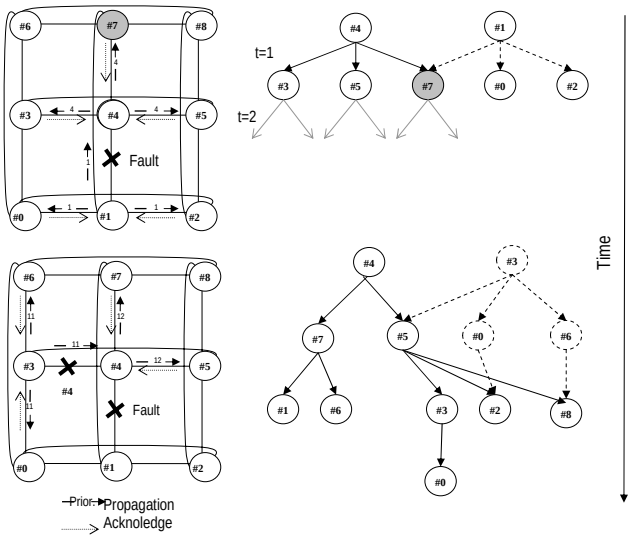



**Figure 6 Examples of multiple fault detections and nested faults.**

Now, consider that before ending the establishment of the tree rooted in node #4, a new fault arises in the link connecting nodes #3 and #4. Two new emergency states will be detected and propagated by nodes #3 and #4 sending respectively priorities $(1\cdot9+3)=11$ and $(1\cdot9+4)=12$. In this case, the partial tree caused by the previous fault will be removed when the new higher emergency priority level is detected in each node. There will be nodes that receive simultaneously both notifications (nodes #5, #2 and #8), selecting only the higher EPL. Nevertheless, some nodes (#0 and #6) will first receive the emergency state with lower priority. Immediately afterwards, these nodes will receive the emergency notification of priority #12, corresponding to the second tree rooted on node #4,

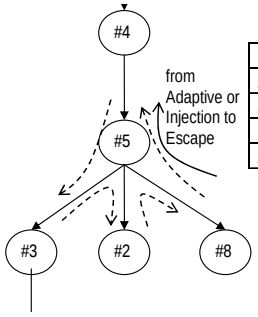
canceling the previous tree. At the end, just one tree will progress. The definitive tree used to establish the final BFC safe ring is the one shown with continuous arrows in Figure 6. In consequence, we can build a unique BFC ring through which, after a finite time, any packet will eventually reach its destination. Our idea to rebuild the tree has some points similar to the technique proposed in [2] but, unlike it, we can successfully solve multiple link/node faults (simultaneous or not).

### 4.2. Safe routing table rebuilding

Once the BFC safe ring and the SS table have been obtained, every surviving node knows how to route packets in transit inside that ring. Nevertheless, the routing tables of the nodes are no longer valid. Consequently, no packet can be injected into the BFC safe ring coming from adaptive channels or from network interfaces. Hence, routing table rebuilding is compulsory before leaving the emergency state.

Two steps are employed to rebuild the safe routing table. In the first one, we use a secure but provisional routing table and in the second phase, we rebuild it in order to improve packet routing performance through the BFC safe ring.

The provisional safe table fulfils the following routing rule: *Every packet, regardless of its destination, trying to enter into the BFC safe ring at a certain node will be routed towards the output port that communicates this node with its parent*. This routing rule causes that certain packets travel towards their destinations through paths longer than necessary. Although the strategy is not optimal, it guarantees deadlock-free communications. In addition, it is quite easy to build this provisional routing table using only local information (the port linking each router with his parent). Figure 7 shows the provisional routing table for node #5, considering again the example represented in Figure 6. The routing table simply indicates that every packet trying to be injected inside the BFC safe ring on node #5 must be routed towards node #4 (output port #1).




| | Port 0 | Port 1 | Port 2 | Port 3 |
|---|---|---|---|---|
| Node 0 | 0 | 1 | 0 | 0 |
| Node 1 | 0 | 1 | 0 | 0 |
| ... | 0 | 1 | 0 | 0 |
| Node 8 | 0 | 1 | 0 | 0 |

**Figure 7 Provisional routing table for the BFC safe ring.**

Although improbable, it is possible that all the FIFOs belonging to the resulting BFC safe ring are full. In this case, the BFC safe ring would be deadlocked. To avoid such a situation, several mechanisms can be employed. For example, is possible to employ the network interfaces to temporarily "consume" the first packets of the queues and reincorporate them into the ring when fulfilling the rules of our Bubble Flow Control.

Once the BFC safe ring and the provisional routing table for this ring are obtained, the router can abandon the emergency state because, although non-optimal, there is a secure path between any pair of surviving nodes. From this moment, new packet injections will be allowed and the BFC safe ring will be shared between packets belonging to the applications running on the machine and some control packets employed in a second phase devoted to the optimization of the safe routing table. At the present state and considering again the example of Figure 7, a packet injected in node #5 and addressed to node #2 would follow the longest possible path instead of using the shortest one.

We will use a distributed algorithm for optimizing the safe routing tables. At the end of the process, every node will know who all its descendants are, for all the branches of the tree. To do that, every node sends to its parent a tagged control packet composed of its own ID and its EPL. The parent will compare the received EPL with his own EPL. If they do not match, the packet will be discarded and the table updating will not take place. In this case, the network is managing a new emergency state. If they match, the parent updates its safe routing table, setting to one in the corresponding row the column corresponding to the port from which the control packet was received and setting to zero the remaining columns. Next, this node will retransmit the same packet to its corresponding parent performing the same actions until reaching the root, when the optimization process will finish. Now, every node can send packets through the BFC safe ring employing the shortest paths. The low overhead imposed by our method must be highlighted consisting of just one control packet emitted per node.

### 4.3. Adaptive routing table rebuilding

It is clear that the adaptive routing tables after an emergency state caused by a failure do not record the correct information to route all the packets through the adaptive virtual network. Packets using the wrong adaptive paths during the reconfiguration process can be misrouted but they will find the proper routes either through the BFC safe ring or through the adaptive one when this process finishes.

For updating the adaptive routing table we employ a similar distributed procedure to the one employed with the safe routing table. Nevertheless, in this case there can be several adaptive minimal paths to communicate two arbitrary nodes. Our update algorithm employs the hop count or distance among nodes to rebuild correctly the adaptive table, in a similar way that the distance-vector

routing [15] employed for example in RIP [11]. Each entry of the adaptive routing table will have an associated field recording the distance from the node to the corresponding destination. At the beginning, this field can be set to a maximum value, i.e. the number of nodes.

In parallel with the safe table updating, every node begins with the process of rebuilding its adaptive routing table. Each node will send to all its neighbors another tagged control packet composed of its own ID, its EPL and its distance. Obviously, when a node sends this control message to its local neighbors the distance field must be set to zero. When a node receives this control packet, it must compare the local and transmitted EPLs. If the transmitted EPL is lower, the packet is discarded and no action will take place. This means that a new emergency state is in progress and consequently, the previous tree is no longer valid. If the EPLs match, the receiver checks if the distance field received in the packet is higher than the one stored in the corresponding entry of the routing table. If so, the packet is discarded as well and no action will take place. If the transmitted distance is lower, all the columns belonging to this entry in the routing table will be set to zero except the column corresponding to the output port from which the control packet was received that will be set to one. Finally, if the transmitted and stored distances coincide, only the column corresponding to the output port from which the control packet was received will be set to one. Anyway, if the packet is not discarded, the node will increment the distance field and it will retransmit the packet through all the output ports except to the one from which it received the control packet.

At the end of this process, all the adaptive routing tables will record a map of minimal distances reflecting the new topology resulting from the last emergency state. Those nodes, whose distances in the adaptive tables maintain their initial value *N*, correspond to lost nodes. Nodes can be lost for two reasons: the node has died or the network has been disconnected into two or more regions. In both cases, the final values of the adaptive routing tables will be valid for each surviving region. The corresponding layers of the system software must manage this situation.

The simplicity of the above algorithm guarantees an efficient distributed implementation. Other solutions based on centralized algorithms would add higher overheads. Moreover, our method has a number of advantages. All the resulting routes are minimal. The number of generated control packets is similar to the number of minimal paths between every pair of nodes. The algorithm is invulnerable to nested failures. It is impossible to reach a deadlocked situation caused by flooding in the consumption queues because the protocol is not reactive. In short, the complete methodology can be directly implemented in the router hardware, broadening its applicability to indirect networks in which some switching nodes do not have an associated network interface.

## 5. A Router Implementation for Immunet

In order to show the feasibility of our fault-tolerant routing, we describe here a sketch of a possible hardware implementation that incorporates all the above-mentioned algorithms and features.

In the presence of faults, only surviving local communications are reliable. Our Immunet implementation employs only local communications to generate the required embedded spanning tree to route packets through the BFC safe ring. When a node detects a failure, it enters in an emergency mode. The router stops all incoming traffic except the packets on flight and begins the propagation of the emergency state. Such a propagation can be implemented asserting simultaneously the two “request” protocol lines existing for each physical link. It should be noticed that this combination of the “request” lines is impossible in normal operation. The EPL can be transmitted using the data lines of the communication channels. Once all the active emergency notifications are tested the router selects the one with the highest EPL and if it is higher than the local EPL, the router will send back a special ACK to the parent node. We implement this special ACK asserting again the two “request” protocol lines and a special EPL (all ones) to distinguish it from a conventional emergency propagation. Then, the router will propagate the winning emergency state to all its neighbors except to its parent. Before finishing the emergency state, which can be easily bounded, the router will receive several special ACKs to determine its children. Then the node will generate the provisional safe routing table and ends its emergency state. From this moment, the router will continue in normal operation but sharing the safe network resources for a period among applications traffic and control traffic for updating the routing tables. At the end of the process, the communication subsystem will work in a degraded but optimized manner.

Figure 8 shows the basic hardware building blocks of our fault-tolerant router. We add four new modules to the router core of Figure 1. The *Input Block* will accept both emergency requests and special ACKs from its children. The *Reconfiguration Module* will process them updating the SS table. The *Output Block* will propagate emergency states and will send special ACKs to its selected parent. Obviously, the *Table Updated Module* will reconfigure the routing tables, as described before. The added buffers associated to the *Table Updated Module* are employed both for delivering control packets and for performing a temporary drain of the first packet in the queues of BFC safe ring to avoid deadlocked initial states.

Although the *Input* and *Output Blocks* are in the router’s critical path, their effect on the elapsed time is extremely small because its only purpose is to detect and propagate emergency states. As can be seen in Figure 9, both the *Input* and *Output Blocks* need only a single level of logic gates per physical link to detect the simultaneous assertion

of the "request" protocol lines. When the emergency state is detected, all the conventional transmissions are stopped by activating the STOP lines in all the input channels. Packets in transit will be stored in the local buffers and the router will stop their retransmission to the output ports.

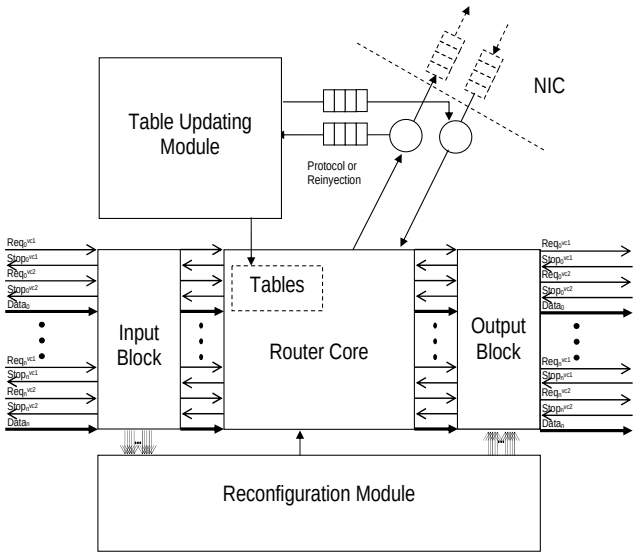



**Figure 8 Router Building Blocks.**

The *Reconfiguration Module* (RM) must select just one of the inputs, which have activated their two "request" protocol lines during an emergency state. Following a first-come first-serve policy, the RM compares each incoming EPL with the local EPL. Such EPLs travel through the input data lines and they are properly multiplexed to be compared with the local EPL, stored in the EPL latch. If the incoming EPL is higher than the local one, the RM will update the EPL latch and the input port from which the router received that higher EPL will be recorded as the link to communicate this node with its parent.

Having determined over which link a reconfiguration is required, the RM sends back a special ACK asserting both "request" lines and all data lines. Then, the RM will propagate the emergency state with its corresponding EPL through the remaining output ports. The neighbors sending back the corresponding ACK before a time-out, will be written in an additional small table that records the children of this node. The time-out required to validate special ACKs can be easily determined taking into account the RM pipeline length and maximum wire delay between two adjacent routers. From this moment, the router knows which port links it with its parent and the ports connecting it with its children. Such information is enough to route packets in a secure way through our BFC safe ring and to build the provisional safe routing table.

The root node, i.e. the node detecting a fault, constitutes a singular case in the propagation of emergency states through the network. Its "*Local fault Detector*" will stop the incoming traffic after detecting the failure. Its *parent latch* will be set to zero and its EPL latch will store its own ID. After that, it will propagate the emergency state until reaching the leaf nodes that having no children will stop the emergency state propagation.

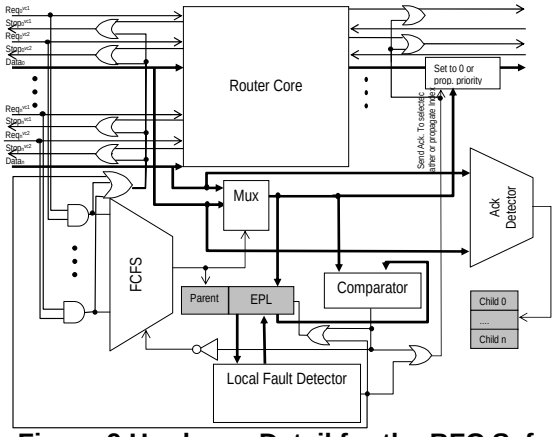



**Figure 9 Hardware Detail for the BFC Safe Ring Reconfiguration.**

It should be noted that the complexity of this mechanism only has impact in terms of the required silicon area. The RM and the *Table Updated Module* are not in the router critical path and they are used only to manage emergency states as well as to rebuild all the routing tables: the SS table recording the shape of the BFC safe ring in this node and the safe and adaptive routing tables. In a normal mode of operation, the overhead added by the *Input* and *Output Blocks* is just one level of logical gates.

Resetting of the *parent* latch is not considered, which means that there is a limited number of nodes that can be supported by our fault-tolerant routing. This number is bounded by the data line width of the communication channels used to transmit EPL values. Assuming 32 bits for data lines as in [16], the maximum number of tolerable consecutive faults in a network having $N$ nodes is approximately $2^{32}/N$. For example, Immunet can manage three-dimensional torus networks having around 100000 nodes without problems. As the torus has $3N$ links, we can cope with up to 150000 links in failure, a situation in which the network would probably be disconnected. This scenario indicates that there is no need to reset the *parent* latch avoiding the subsequent problems derived from this action.

As we have demonstrated in this Section, our complete fault-tolerant Immunet system can be directly implemented in the router hardware although software implementations that use the network interface are also possible. In this paper, we have selected a hardware implementation approach but depending on the failure probabilities and on

cost issues, Immunet can admit different cost/performance implementations.

## 6. Immunet Evaluation

In this section, we will analyze the performance loss experienced by Immunet under faulty conditions and the effectiveness of the table rebuilding processes in terms of both packet overhead and required time to carry out the reconfiguration procedure. In addition, we will measure the behavior of Immunet under synthetic and real loads.

The simulation environment employed in this study is based on the SICOSYS (Simulator of Communication Systems) interconnection network simulator [22]. This simulator allows us to take into consideration most of the VLSI implementation details with high precision but with much lower computational cost than hardware-level simulators. SYCOSIS has been integrated into the RSIM simulator [17], replacing RSIM's original network simulator. The combination of both simulators provides a powerful tool to emulate a complete CC-NUMA machine when running real parallel applications.

To carry out the evaluation process we firstly need to establish some initial parameters. We will assume that a node employs 100 clock cycles to locally manage and retransmit an emergency state. In addition, we will assume that a node employs 1000 clock cycles to manage and retransmit a control packet employed to rebuild the routing tables Although both times have been arbitrarily established, there is no problem for adequately setting them when dealing with a real implementation. Anyway, it seems realistic for the hardware approach previously described.

### 6.1. Synthetic Workloads

First of all, we will analyze the time required to finish a reconfiguration process assuming only a single fault in the network for different network sizes under random traffic. We will inject a fault in an arbitrary link for two network sizes: 8x8 and 16x16 tori. Our analysis assumes that the networks are beyond the saturation point (worst case) managing a uniform traffic pattern. Figure 10 shows the network throughput degradation during the reconfiguration process and the performance exhibited once this process is finished. In order to clearly assess the Immunet performance, the curves in this figure and in the following ones only record the application traffic. The critical data associated to both reconfiguration processes are shown in Table 1. It can be seen that the number of control messages for updating the safe routing tables increases linearly with the number of nodes. Nevertheless, the number of control messages for updating the adaptive tables grows as n-squared.

During a reconfiguration process, the existence of reconfiguration control messages affects the application traffic reducing the effective network throughput. Typically, parallel applications consume a considerable number of billions of cycles. Taking this into account, the effect of the reconfiguration process will be almost negligible. Then, even in the case of a 256-node network, the reconfiguration gap will be, in most cases, transparent to the parallel application.

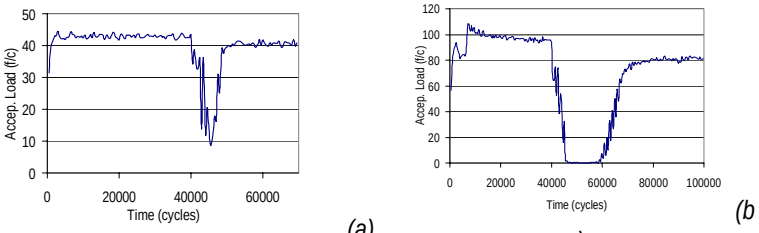



**Figure 10 Time required for doing online reconfiguration on (a) 8x8 torus, and (b) 16x16 torus.**

| | 8x8 Torus | 16x16 Torus |
|---|---|---|
| Number of control messages required for updating safe routing tables | 64 | 256 |
| Number of control messages required for updating adaptive routing tables | 12240 | 244908 |
| Time required to finish the network reconfiguration (cycles) | 9945 | 36125 |

**Table 1. Main values for network reconfiguration**

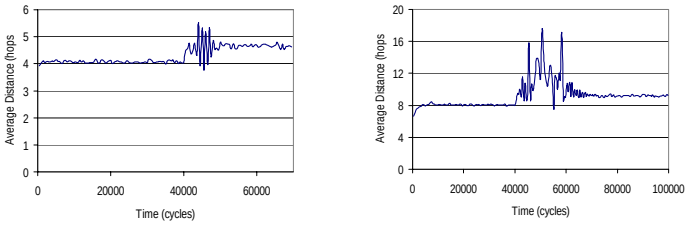



**Figure 11. Average distance impact: (a) 8x8 torus, and (b) 16x16 torus.**

Some performance degradation can be observed in the network when a reconfiguration process ends. This degradation is caused for two reasons. The obvious one is the resource loss associated to the missing link. The second reason originates from the topological change experienced by the network. Before the failure, we were using a routing algorithm optimized for the selected topology. After the failure, an enlargement in the safe virtual network appears, increasing the packet average distance as can be seen in Figure 11. Although, at the end of the reconfiguration process the increment of the packet average distance is reasonably low, its effect on network throughput can be more noticeable. When the number of network nodes increases, this negative effect is higher and consequently the performance degradation. In this experiment, the throughput degradation is nearly 5% for the 64-node network and 15% for the 256-node network.

To assess the viability of Immunet in a more demanding scenario we will proceed to inject incrementally a higher number of faults under different conditions. Figure 12 and Figure 13 show the effect of injecting a growing number of faults in both networks. The failures considered are either link faults or node faults. The failures are uniformly distributed and the only required restriction is that the surviving network nodes must remain connected. The robustness of Immunet can be clearly observed, always converges to a steady state. The behavior exhibited by Immunet is more than acceptable even in the hardest scenario in which almost all faults are simultaneously injected. When the faults are sufficiently distant the reconfiguration process ends before the arrival of the next failure. When the network performance dramatically falls as a consequence of a high resource loss, nested fault detections can appear and the time required for reconfiguring the tables increases. In any case, the good behavior of Immunet is clear. Even in extremely adverse scenarios under a very high number and frequency of failures, the network recovers itself and converges to a steady state.

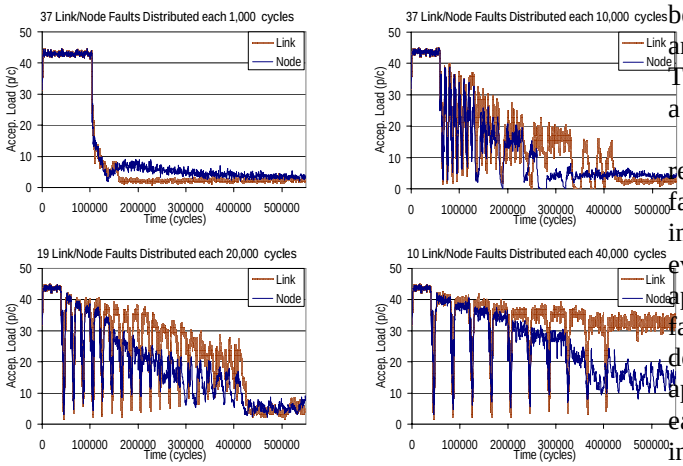



**Figure 12. Throughput degradation for different numbers of link/node faults in an 8x8-torus network under uniform traffic.**

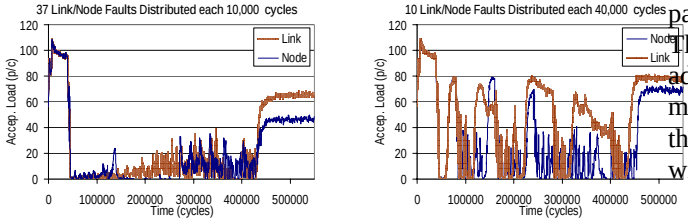



**Figure 13. Throughput degradation for different numbers of link/node faults in a 16x16-torus network under uniform traffic.**

## 6.2. Real Workloads

To assess the network behavior under realistic workload conditions, the impact of an increasing number of faults on the execution time of different parallel applications has been analyzed. To assure the ending of the programs we have considered only link faults that do not isolate any computing node. We will emulate a multiprocessor system with 64 nodes assuming that each network router has a single-processor computing node attached.

The parameters of the nodes employed in the CC-NUMA multiprocessor emulated in this paper (cache coherence protocol, processor architecture, memory hierarchy, etc.) are similar to those of an Alpha 21364 [16]. Cache line size is 64 bytes and command packet size is 12 bytes long. The processor speed has been established at 1.2 GHz. As the physical channel width or phit size is 4 bytes, a data packet containing 76 bytes will be 19 phits long. Consequently, command packets (request or invalidation), will be three phits long. The router clock has been set to 0.8GHz.

To carry out this realistic evaluation, we fed our simulation platform with three applications selected from the SPLASH-2 suite: Radix, FFT and LU, which had already been ported into RSIM by researchers at Rice University [17]. These three applications were selected because they have significant communication demands, and each one represents a different case of network load. The problem sizes are 64K double complexes for FFT, half a million keys for Radix, and a 256x256 matrix for LU.

To assess the effectiveness of Immunet under this realistic scenario, we will inject a different number of faults when running each application and analyze their impact on the execution times. We will also show the evolution of the network behavior in some of the analyzed applications. As stated before, we only consider link failures because the higher levels of the simulated system do not allow the dynamic loss of processors along the application execution. The number of faults considered for each case is 3, 9 and 17. In addition, the network faults are injected using a non-periodic pattern, once a pre-established number of packets are consumed. This reflects, in a way, a worst-case scenario because the faults tend to be injected during high network load phases.

The degradation observed on the execution times of the parallel phases of each application is shown in Table 2. These data are normalized in Figure 14. Even in the most adverse situation, the impact of our reconfiguration mechanism is close to 30%, which includes the effect of the application running on an arbitrary irregular topology with a single long BFC safe ring.

Figure 15 shows the evolution of the main network parameters when executing the central phase of Radix. We can see how the maximum accepted load drops from 40 phits per network cycle to 35 phits when 17 faults are considered. The Immunet behavior is quite similar to the one observed under synthetic workload conditions. During small lapses of time with respect to the total execution time, the effective network throughput drops to values near

to zero and packet latencies rise, in some cases, more than one order of magnitude. Nevertheless, when the reconfiguration processes ends, latencies return to values close to those corresponding to the fault-free network. Therefore, the main cause of the observed performance degradation is the reduction of the maximum achievable throughput. When the application enters in an intensive communication phase, the amount of traffic tends to be high and constant. Hence, a reduction in throughput will imply a proportional increase in execution time. It must be remembered that throughput reductions are not only due to the routing algorithm employed in the presence of failures but also due to the resources loss.

| | *Fault Free* | *3 Links* | *9 Links* | *17 Links* |
|---|---|---|---|---|
| *FFT* | 3.13E5 | 3.33E5 | 3.75E5 | 4.21E5 |
| *RADIX* | 1.33E6 | 1.36E6 | 1.43E6 | 1.50E6 |
| *LU* | 1.37E6 | 1.40E6 | 1.42E6 | 1.46E6 |

**Table 2. Absolute execution times (processor cycles).**

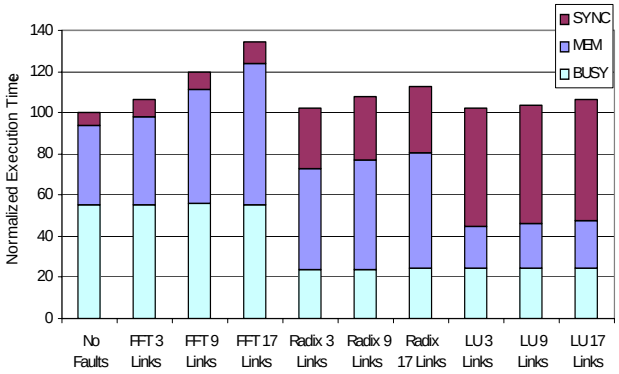



**Figure 14. Normalized execution times.**

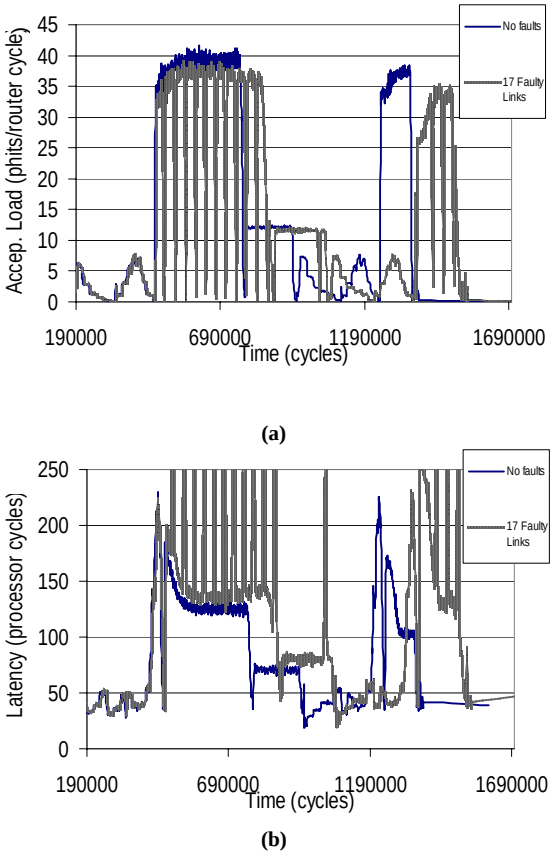



**Figure 15. Radix execution trace: (a) Throughput, (b) Latency (zoom).**

Among the three applications, FFT is the one exhibiting higher performance degradation for two underlying reasons. First, FFT employs an "all-to-all" communication pattern and hence, the resources in failure cause a higher impact. The second, and more important reason is that the execution time of this application is much shorter than Radix or LU as can be seen in Table 2 and consequently, the impact is comparatively more important. Note that in realistic scenarios, it is not usual to have such short applications. Due to the extreme complexity of our simulated system, we have reduced the size of the input set in order to achieve results in manageable amounts of time.

When executing LU, the performance degradation is even lower than with Radix. Its execution time is slightly longer than Radix and the traffic pressure on the network is much lower than in the other applications analyzed here. Consequently, neither the reconfiguration processes nor the maximum throughput degradation have a significant impact on its execution. In the worst case, the observed performance degradation is close to 7%.

## 7. Conclusions

A new methodology able to tolerate any combination of faults inside a typical interconnection network has been presented in this work. The unique limitation of Immunet is the network connectivity. Our proposal exhibits a set of characteristics that makes it extremely effective. Firstly, Immunet hardly affects the interconnection network performance when running under fault-free conditions. Secondly, Immunet tolerates any spatial and/or temporal combination of faults. Only packets on flight through faulty links will be affected and in consequence, they can be recovered with little effort. In addition, Immunet exhibits graceful performance degradation when it is integrated in a parallel computer. In such a scenario, our network is able to carry out an automatic reconfiguration without intervening in the upper levels of system software. Reconfiguration times are short enough to be supported by almost any real application.

All these features have been verified by means of an exhaustive simulation process of *k*-ary *n*-cube networks for

which their main performance figures were obtained. Notwithstanding, Immunet can be used in any regular or irregular topologies. In fact, using a routing algorithm for irregular networks like that presented in [21], the application of our proposal is straightforward.

Measurements of reconfiguration times and performance degradation are both excellent. For example, in a 16x16 torus network managing random traffic beyond its saturation point, a link fault is solved in less than 30,000 processor cycles only degrading the performance around 17%. On the other hand, 35 randomly distributed link failures cause a performance degradation of about 90%. Nevertheless, after 400,000 cycles, although heavily degraded, the network will recover its normal operation mode. Furthermore, detailed application-driven simulations of FFT, Radix and LU parallel codes not only corroborate the results obtained using synthetic traffic but also the robustness of the methodology.

## 8. References


[1] NR Adiga, GS Almasi, Y Aridor, M Bae, Rajkishore Barik, et al., “An Overview of the BlueGene/L Supercomputer”, Supercomputing 2002.

[2] D. Avresky, N. Natchev, V. Shurbanov, “Dynamic Reconfiguration in High-Speed Computer Clusters”, 3rd IEEE International Conference on Cluster Computing (CLUSTER'01), October 2001

[3] R.V. Boppana and S. Chalasani. “Fault-tolerant wormhole routing algorithms for mesh networks”. IEEE Trans. on Computers, vol. 44, no.7, pp. 848-864, July 1995.

[4] J. Bruck, R. Cypher, and C. Ho. “Fault-tolerant meshes with small degree”. SIAM Journal of Computing, vol. 26, no. 6, pp. 1764-1784, December 1997.

[5] R. Casado, A. Bermudez, F. J. Quiles, J. L. Sanches, and J. Duato, “Performance evaluation of dynamic reconfiguration in high-speed local area networks”. International Symposium on High-Performance Computer Architecture (HPCA), January 2000.

[6] S. Chalasani and R V. Boppana, “Communication in Multicomputers with Nonconvex Faults”. IEEE Trans. on Computers, vol. 46, no.5, pp. 616-622, 1997.

[7] M.S. Chen and K.G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers", IEEE Trans. on Computers", vol. 39, no.12, pp. 1406-1416, December 1990.

[8] D. Avresky, “Embedding and Reconfiguration of Spanning Trees in Faulty Hypercubes”, IEEE Transactions on Parallel and Distributed Systems vol. 10 no.3, pp. 211-222, March 1999.

[9] M. Galles, "Spider: a high-speed network interconnect," IEEE Micro, vol. 17, no.1, pp. 34-39, Jan-Feb. 1997.

[10] P.T. Gaughan and S. Yalamanchili, "A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks", IEEE Trans. on Parallel and Distributed Systems, vol. 6, no.5, pp. 482-497, May 1995.

[11] C. Hedrick, “Routing Information Protocol”, June 1988. Internet RFC 1058.

[12] P. Kermani and L. Kleinrock, “Virtual Cut-Through: A New Computer Communication Switching Technique”. Computer Networks, Vol. 3, pp. 267-286, 1979.

[13] D. Linder and J. Harden, “An Adaptive and Fault-Tolerant Wormhole Routing Strategy for k-ary n-cubes", IEEE Trans. Computers, vol. 40, no1, pp. 2-12, Jan. 1991.

[14] O. Lysne and J. Duato. “Fast Dynamic Reconfiguration in Irregular Networks”, International Conference of Parallel Processing (ICPP), 2000.

[15] G. S. Malkin and M. E. Steenstrup. “Distance-vector routing”, In M. E. Steenstrup, ed., Routing in Communications Networks, pp. 83–98. Prentice Hall, 1995.

[16] S. Mukherjee, P. Bannon, S. Lang, A. Spink, D. Webb, “The Alpha 21364 Network Architecture“, IEEE Micro, vol. 22, no. 1, pp 26-35, Jan-Feb 2002.

[17] V.S.Pai, P. Ranganathan, and S.V.Adve, "RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors". IEEE TCCA Newsletter, October 1997.

[18] R.Pang, T.Pinkston, “The Double Scheme: Deadlock-free reconfiguration of Cut-through Networks”, International Conference of Parallel Processing (ICPP) August 2000.

[19] V. Puente, C. Izu, R. Beivide, J.A. Gregorio, F. Vallejo and J.M. Prellezo, “The Adaptive Bubble Router”, Journal of Parallel and Distributed Computing. vol 61, no. 9, September 2001.

[20] V. Puente, J.A. Gregorio, J. M. Prellezo, R.Beivide, J. Duato, and C. Izu, “Adaptive Bubble Router: a Design to Improve Performance in Torus Networks”, International Conference of Parallel Processing (ICPP) 1999.

[21] V.Puente, J.A. Gregorio, R. Beivide, F.Vallejo, A.Ibañez, "A New Routing Mechanism for Networks with Irregular Topology", Supercomputing, 2001.

[22] V.Puente, J.A. Gregorio, R.Beivide, “SICOSYS: An Integrated Framework for studying Interconnection Network in Multiprocessor Systems”, Euromicro Workshop on Parallel and Distributed Processing, 2002.

[23] V. Puente, J.A. Gregorio, R. Beivide and F. Vallejo, “A Low Cost Fault Tolerant Packet Routing for Parallel Computers”, International Parallel and Distributed Processing Symposium (IPDPS), 2003.

[24] S. L. Scott, “Synchronization and Communication in the T3E Multiprocessor, ASPLOS VII, 1996.

[25] J.Shih, “Wormhole routing for torus networks with faults”. Parallel Computing, 27 (2001), 1817-1829.

[26] L.Ziang, “Fault Tolerant Networks with Small Degree”, Symposium on Parallel Algorithms and Architectures (SPAA), 2000.