A Flexible Approach for a Fault-Tolerant Router

Andreas C. Döring, Wolfgang Obelöer, Gunther Lustig, Erik Maehle
Medical University of Lübeck
Institute of Computer Engineering
Ratzeburger Allee 160
D-23538 Lübeck, Germany
E-Mail: doering/obeloeer/lustig/maehle@iti.mu-luebeck.de

Abstract: Cluster systems gain more and more importance as a platform for parallel computing. In this area the power of the system is strongly coupled with the performance of the network, which has to provide high bandwidth and low latency. Besides these performance aspects fault-tolerance within the network is very important. This paper shows how to build a flexible and fault-tolerant router, the main building part of a network. In addition the overhead for the execution of fault-tolerant routing algorithms is examined.

1 Introduction

Today nearly all applications which are computation or data intensive are executed on parallel computers or workstation clusters. Especially cluster systems gain more and more importance. These systems consist of a high number of processors with local memory and exchange messages through a network. In many applications the power of the whole system depends strongly on the performance of the network, i.e. the network has to provide high bandwidth and low latency. Since broadcast-based approaches like Ethernet are not suitable, switched networks are required. These networks consist of routers (router and switch are used synonymously) and links connecting them. Within the switches two parts can be distinguished: a data path and a control unit (Figure 1).

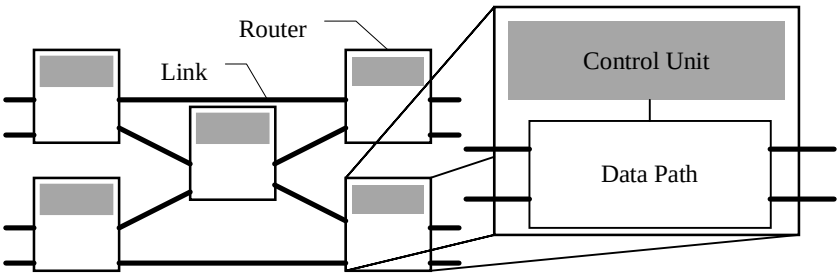

Figure 1. Structure of a switched network

Examples of switched networks with high bandwidth and low latency are Gigabit Ethernet [Coh96], Myrinet [BCF95] or SCI [Gus92]. Once they are installed the behavior of these networks especially the routing scheme is fixed. But applications like multimedia, number crunching or data warehousing require different and flexible behavior in order to achieve an optimized network usage. This leads to the fact that switches with configurable behavior are highly desirable.

Another important aspect in this area is fault tolerance. For example the nodes of clusters are distributed throughout rooms, so faults in the network may not be as rare as for dedicated parallel machines. If the switches - and therewith the network - do not consider fault situations, higher-level software protocols for fault tolerance (e.g. checkpointing, message logging) have to be applied. Additionally in case of a fault a reconfiguration of the system (network and software) is necessary. This produces extra overhead, which can be reduced to a minimum if the network itself is capable to handle faults. This capability and the behavior of the network in general are mainly determined by the applied routing algorithm. These algorithms can be roughly classified into *oblivious* and *adaptive* ones. Using oblivious routing the whole path through the network is fixed. In contrast to that, adaptive algorithms use fault and/or load conditions to determine the path through the network. In order to achieve fault-tolerant behavior, adaptive routing algorithms are recommended. Since these algorithms are difficult to implement, there are only a few fault-tolerant switches realized so far.

Two of them, namely the Spider-Chip [Gal97] and the Hal-Router [MLS96], are quite new and use advanced technology (deep submicron CMOS). While the first one is table-based and mostly used for hypercube topologies, the second one uses source routing. Both methods cannot react to network load states in a sufficient way. The table based approach limits the size of constructable networks. This limitation does not exist for source routing at the expense of additional effort at the source nodes and the restriction of applicable algorithms, e.g. concerning deadlock prevention. The table based approach only allows fault-tolerance by means of reconfiguration. Thus there is the mentioned need for high-level protocols. From the algorithmic point of view the Chaos router [BoY94] and the Planar Adaptive Router [ChK92] are more important. While the principles behind the Chaos router allow many topologies, it is restricted to packet routing. In contrast to that the planar adaptive router works with wormhole routing on k-ary n cubes. Since they are implementations of advanced adaptive routing methods they are good references for the optimizations possible by choosing an appropriate routing algorithm for a given application. Furthermore they are fault-tolerant.

In order to cope with the problems mentioned above, this paper describes how to build a flexible and fault-tolerant switch which is competitive to that ones using only oblivious routing schemes. The remainder of the paper is organized as follows: a short introduction to fault-tolerant routing is given, illustrated by two algorithms. In section three the additional requirements for routing algorithm and hardware are examined, followed by the description of concept and implementation of rule-based routing. Based on this architectures the two examples are used to give concrete numbers on the additional hardware effort for fault tolerance using the rule-based approach.

2 Fault-Tolerant Routing Algorithms

2.1 Overview

The design of a fault-tolerant network includes several topics like an interface to the physical medium with fault diagnostics, node fault diagnostics, handling of transient transmission errors, messages routed in a wrong direction, reconstruction of messages ripped up by a link fault and bypassing faulty links, i.e. fault-tolerant routing. In this paper the last point is focused and hence the following assumptions are made:

i) A link is either faulty and known as such or it transmits messages without destruction. Links are bi-directional and both directions fail together.
ii) In the same way a router node either works correctly or it fails and the adjacent nodes are aware of this.
iii) No messages are sent to disconnected or faulty destinations.
iv) No message is affected during the diagnosis phase after a failure, i.e. until all concerned nodes have changed their fault state information.
v) Multiple faults are allowed.

Though assumption iv is unrealistic, it is typical for the fault model of routing algorithms [Dua97]. In a direct network this problem can be solved by sending an affected message to the nearest home link and reinjecting it into the network. This means very little protocol overhead which is only necessary for a few messages. For indirect networks or for very long messages this may be impractical.

Thus the problem of a fault-tolerant routing algorithm[1] is the following: Parts of the network (nodes, links or both) are out of order and they are known by the routing algorithm. At certain points on time messages with a given destination occur in the nodes of the network. The task of the routing algorithm is to guide these messages through the network as fast as possible. Of course only operating nodes and links may be used.

It is clear that there exists the following simple routing algorithm which solves the problem:

1. Compute a spanning tree for the network graph every time new faults occur.
2. Route messages by only using edges of the tree.

However this algorithm uses only a small fraction of the network links in most cases. This has the effect that the shortest ways (minimal paths) between two nodes are nearly never taken. Hence the performance of the routing algorithm has to be considered too. From this point of view the following three conditions about fault-tolerant routing algorithms should be fulfilled:

Condition 1. If all links of all minimal paths between source and destination are unbroken, then every such path can be selected dependent on the load of the network.

[1] The routing algorithm is designed for a specific topology, i.e. the topology is a property of the routing algorithm and not an input to it.

Condition 2. If there exists at least one path of minimal length between two nodes then the routing algorithm will use one of these paths for the message. Note that it is not demanded that the algorithm chooses among all minimal paths possible in this situation.

Condition 3. If there exists at least one path between two nodes – which may be non-minimal in terms of the original topology – then the message can still be routed.

Condition 1 is the definition of a fully adaptive minimal routing algorithm in the fault-free case [BoC93]. Since the problem of routing in regular graphs is already complex it is not surprising that most of the known fault-tolerant routing algorithms do not fulfill all these points. In opposite they use approximations which reduce the complexity and the memory requirements (i.e. resources) of the routing algorithm. There are a lot of routing algorithms differing in the effort, the degree they fulfill these conditions, the underlying topology, the switching method and other characteristics [ChB95a, GaY95, KiS93, SuS95, SuW97]. To illustrate the methods applied by typical routing algorithms two of them, namely NAFTA [CuA95] and ROUTE_C [ChW96], are discussed in the following subsection.

2.2 Examples of Fault-Tolerant Routing Algorithms

Both algorithms handle link and node faults. They apply wormhole-switching which means that every message in the network is divided into flits (flow control units) transmitted in a pipelined fashion [DaS87]. Every physical link is split into several virtual channels, e.g. by time multiplex. This allows the undisturbed transmission of one message across a link even if another message uses the same physical link.

The algorithms explained, NAFTA and ROUTE_C, are similarly structured. The fault information is distributed in the network by exchanging information between adjacent nodes. Dependent on the message this information restricts the usage of the outgoing links (set 1). From the destination and source address of an arriving message a set of outgoing links offering a deadlock-free path to the destination is calculated (set 2). From intersection of these two sets one output is selected according to an adaptivity criterion. The following description is based on this structure.

The first example of a fault-tolerant routing algorithm is called *NAFTA* (New Adaptive Fault-Tolerant routing Algorithm [CuA95]). It is illustrative because of its relation to the non fault-tolerant NARA introduced in the same publication. Thus, for this special case the fault-tolerant aspects can be clearly separated from the remaining routing algorithm (see Section 3). The underlying topology is a two dimensional mesh, which is very popular and widely used.

In order to build *set 1*, fault information is propagated in a wave like way beginning with the node where a fault is known first. The information is encoded using a state assignment for each node. Adjacent nodes exchange it every time there is a change. The states are oriented at geometric patterns like columns and rows. For instance, one state is called "dead-end-east" meaning that all columns to the east have at least one fault. Therefore a message destined to north-east may not use a node in this state, because it is possible that there is no way to forward the message to northern

directions anymore. Especially concave fault patterns are completed to a convex shape excluding the use of some non-faulty nodes, violating condition 3. However the restriction to a few states makes NAFTA attractive for fast implementations.
For NAFTA a *deadlock prevention* method derived from the turn model [GlN92] is applied. Two virtual channels are used per link forming two virtual networks, called south-last and north-last. By prohibiting a direction change for messages that once have been transmitted southern (resp. northern), cycles of dependencies are avoided. In this way all paths of minimal length in the original topology can be used for adaptivity or fault tolerance, meeting condition 1.
For wormhole-routing it is known how long the remainder of a message is, which still has to be transmitted even if this remaining part has not been received yet. This is exploited by using the amount of data that still has to pass a node as *adaptivity* criterion for the neighboring nodes. With intersection of the two sets of possible output links and the adaptivity criterion a link is selected.
In contrast to the previous algorithm *ROUTE_C* introduced in [ChW96] focuses on a hypercube topology. It uses five virtual channels and a constant number of states for each node.
The meaning of the states is derived from the properties of the hypercube which offers a lot of freedom for message paths. E.g. for every message that has to be transmitted two hops two alternative paths are available. Using this property nodes are avoided that have two faulty neighbors or are ends of two faulty links (state "unsafe"). The algorithm has the interesting property that it is known for a node, whether condition 2 (about fault-tolerant routing) can be met or not. The state propagation scheme is very similar to NAFTA. The way in which error states are combined forms a partial order. Therefore the propagation scheme settles fast. Condition 3 on the network is provided as long as the network is not in a state called "totally unsafe" which can be easily detected. This will only occur if more than n-1 nodes are faulty.
Deadlock avoidance is done similar to [Kon90]: First all links with increasing coordinates in any dimension are useable, afterwards all links with decreasing coordinates towards the destination. An extension of four additional virtual channels is used in the hops-so-far scheme after this in case of faults. Besides, by applying the method from [BoC96] three additional virtual channels suffice. In which way the freedom offered is used for improving performance, i.e. *adaptivity* is not specified.
Altogether the separation of a knowledge propagation scheme and the application of this information in the routing decision is a common structure not only found in the two examples above.

3 Requirements for Fault Tolerance

The published fault-tolerant routing algorithms differ mainly in the number of faults allowed, the fault model and whether all non-faulty nodes and links can be used. In most cases their performance has been demonstrated by simulation.
Based on a systematic investigation in this section the cost of hardware (e.g. transistor count) and algorithms (time and space) for providing fault tolerance is examined. In particular it is shown how the additional effort of fault-tolerance is

generated. To do so beginning with non-fault-tolerant algorithms the necessary changes to the algorithm and its implementation are elaborated. In order to systematize the variety of routing algorithms in high-speed networks non-fault-tolerant routing algorithms can be divided into subgoals according to following topics:

- Deadlock avoidance
- Lifelock avoidance
- Directing to destination (purposiveness)
- Adaptivity
- Scheduling and Fairness

By adding fault tolerance these points are influenced differently. Thus the additional cost of fault tolerance is caused by the application of modified methods for maintaining employability of these subgoals.

Deadlock Avoidance

First of all a routing algorithm has to be deadlock-free. Deadlocks can occur if a set of messages wait cyclically for each other to access a certain resource. Therefore it is related to topology. Some methods like message dropping [ScV96] or deadlock avoidance in Chaos routing [BoY94] work independently of the network topology and are not affected by faults. However they tend to waste network bandwidth by transmitting messages several times or using considerably longer message paths than necessary. A much better network usage results from incorporating knowledge about the topology which is done by the following two methods:

One popular method is *static deadlock prevention*. It uses the notion of virtual channels and avoids the occurrence of cyclically waiting messages by assigning a partial order on network resources and restricting their usage to increasing dependencies. Since also for adaptivity it is desirable to have a large set of paths available for a message, many faults can be tolerated. A good example for this strategy is NAFTA described above. Only in the presence of many faults the order has to be changed which in most cases is limited to few nodes near regions with faults [ChB95b].

Static deadlock prevention normally requires a larger number of virtual channels which are expensive in terms of hardware. Dependent on the topology the changes of the order near the faults can be negligible. For instance, using the negative hop scheme [BoC96] for which the number of virtual channels depends on the network diameter no changes to the deadlock avoidance are necessary at all.

Another group of deadlock avoidance concepts (e.g. [CyG94, PGF94]) can be called *dynamic* because the state of the system is incorporated. The basis of this scheme is the existence of a static deadlock prevention method. Links can be used as long as there is space available in a corresponding buffer. If no space is available, the static method has to be used. The dynamic extensions increases the performance and the static method needs only to provide a deadlock-free network without alternative

paths. But this scheme is very vulnerable to faults. For example the fault of one link[2] can separate several node pairs in the statically deadlock-free network which cannot be compensated by the dynamic extensions. Thus in this case already a single fault causes reconfiguration of some network nodes. A scheme described in [ChB95b] even requires additional virtual channels for bypassing faulty regions. The additional effort for the dynamic deadlock prevention to be made fault-tolerant is therefore higher; however in terms of virtual channels it will not exceed the static case.

Lifelock Avoidance

To ensure delivery of all messages the path length has to be finite, which is called lifelock avoidance. It is strongly affected by fault-tolerance since link faults can cause messages to use diversions and the path for a message is prolonged. If for instance a message is destined to a node which is directly connected by a faulty link instead of using just this link with distance 1 hop at least one intermediate node has to be passed.

Sufficient long paths have to be permitted to reach nodes via paths with many detours. The restriction on minimal paths (which is very effective and simple) in a fault-free network would disable a lot of connections in the presence of faults. To keep most of the simplicity of this scheme, messages on non-minimal paths due to faults can be marked and treated exceptionally [ChB95b] (e.g. by using a path length counter). This process of marking requires a modification of the message not found in many minimal routing algorithms. This is best done in the header, i.e. the first part of the message where routing information like the message destination is stored.

From an algorithmic point of view this is not expensive but the hardware has to be capable to support this. It is much more effort in the interface between the control portion and the data path than just copying some information from the message header. The message may be protected by checksums which in case of message modification have to updated, etc.

Purposiveness

The third important point is purposiveness which is strongly related to both previous points. This is the capability of the router to choose those outputs for a message that leads closer to its destination. In an incomplete topology this may be hard to decide. If for instance in a two-dimensional mesh a set F of links is faulty in the worst case some routers need $\Omega(|F|)$ memory to store enough information for the proper routing decision. This can be simply seen in the case where a chain of faulty links separates two regions of the network near a border (gray region in Figure 2). At the beginning of this chain a node has to decide to which side of the chain a message has to be forwarded. Therefore it needs enough information about the faults to know on which side the destination of the message is located.

[2] For example a link to the node with the highest inside-out order in algorithm 1 of [CyG94].

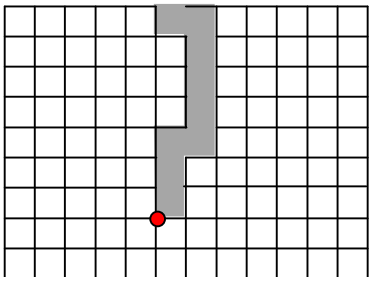

Figure 2. Situation where a specific node needs much fault knowledge

To access this information efficiently the routing algorithm has to apply suitable data structures. For instance a list of the fault coordinates in the mesh network is larger than the bound given above and implies a large computational effort. Approximations as shown in NAFTA, which use only a constant memory amount per node have to exclude some nodes from the network in awkward fault situations.

Adaptivity

The situation for adaptivity is more non-uniform than the topics above because the aim is not as clearly defined. The goal can be an equal balance of the traffic load or that some special kind of messages are preferred. Most times the aim can be achieved only approximately. While in a fault-free network this counteracts the inhomogenity of the application, in a faulty network also the unsymmetries due to missing connections can be compensated. Thus in this case adaptivity is even more important to supply graceful degradation. But an adaptivity scheme not aware of fault-tolerance could cause a very ineffective use of the network because faulty regions may appear lowly loaded and thus such a method may try to assign more traffic to it causing more detours. For this reason a very careful operation of adaptivity is required.

Local adaptivity which uses measures like link load can be easily modified to match fault-tolerance requirements: a faulty link just has to appear as maximally loaded. Non-local methods which distribute information about network load with messages can incorporate fault information in the same way.

But there are methods which require a lot more hardware effort if altered for fault tolerance. For example, if the load of a traffic pattern should be equally distributed onto several links, the break of a connection implies a more complicated arithmetic (division by variable number versus division by constant): the sum and the ratio of the link's bandwidth changes and thus the distribution of the traffic gets harder.

The integration of fault information into adaptivity measures is a cheap method which performs well for common cases. Altogether the complexity of the adaptivity may increase, which requires a more powerful hardware to keep the quality of methods applied for special situations.

Scheduling and Fairness

Applying fairness methods that guarantee a balance in resource granting avoid unlimited long delay of a message inside a router. They have to be considered for

scheduling, i.e. the assignment of resources to incoming and outgoing messages. Therefore it acts like adaptivity in the small.
It is only marginally touched by faults, cause the same situation occurs as in a node currently using only some of its links. However, it may be desirable to favor messages misrouted due to faults to compensate the double disadvantage of the longer path and higher loaded links.
In order to implement fault-tolerant routing algorithms a method describing these complex algorithms and a fast execution are necessary. A potential approach for this is described below.

4 Rule-based Routing

Traditionally routers with flexibility as required for fault-tolerant networks are controlled either by large tables or by software, i.e. a sequence of simple commands. While the first approach can be implemented really fast, feasible table sizes limit the versatility. Software descriptions on the other hand bear an inherent sequentialization which limits the execution speed. In-between these two the proposed approach of rule-based description is situated. It offers a wide flexibility for the intended class of problems and it allows an execution nearly as fast as a table-based solution or a dedicated hardware implementation of one selected algorithm. The concept uses ideas from both, the table and the CPU-like version: the rule skeleton can be a table and registers are updated by using arithmetic or logical units. All actions are controlled and synchronized by an event manager not found in this form in the other mentioned approaches. This flexibility is obtained by a combination of fixed and optimized design of common modules needed for every routing algorithm (e.g. portions of the data path) and integration of highly versatile blocks that can be configured to execute advanced algorithms.
Using the rule-based routing approach leads to the following advantages:

- The description of a routing algorithm is compact and intuitive allowing even non-experts to understand and modify the network behavior.
- A rule-based specification is semantically well based allowing the application of formal methods to routing algorithms, e.g. transformations.
- So far routing algorithms were published in pseudo-code with a nearly unlimited freedom of operation complexity (see for example the operation "match" which is a complex search for a message in a queue in the description of the chaos router [BoY94]). With the rule-based approach the comparison of different routing algorithms is permitted giving one universal language on a fixed abstraction level.
- It is possible to transform the rule base and apply a fast hardware interpreter (see Subsection 4.3) which is able to outperform software solutions and offers more complex realizations than table-based methods.

Especially the last point makes it possible to use complex (adaptive and/or fault-tolerant) routing algorithms within high-speed networks.
To get an overall impression of the factors for speed and hardware cost of a router, an overview of the architecture of a router for the rule-based approach refining

Figure 1 is presented in this chapter. Following this, the rule-based description form and its implementation by a rule interpreter is focussed.

4.1 Architecture

The overall performance of the router is determined by several factors, one of them is the routing algorithm itself [DLO97]. However, the efficiency of data transport is also important. It is not discussed in detail here, but for a complete view of the rule-based router its structure is outlined (Figure 3). In Section 1 the division of the router into a control unit and the data path was introduced. For the rule-based router obviously control is implemented by rule interpreters summed up in the block *Rule Bases*. The *Connection Unit* together with the *Input* and *Output buffers* form the data path. Both sections are coupled by the *Message Interface*. It enables the modification and generation of messages needed, as shown in Section 3. The *Connection Unit* is used to switch a fixed number of headers to the *Message Interface*. This makes header extraction independent from the node degree and thus enables scalability. Furthermore the message interface delivers header data to the rule bases.

The *Information Units* generate information about the links, like load (which can be measured by buffer exploitation) and faults. For instance they could produce and check heartbeat messages.

The buffers include the interface to the physical link and handle errors on the data link layer. Since these capabilities are needed for every configuration of the router, there is no need for much flexibility here. To implement fault-tolerant algorithms efficiently within this architecture the part *Rule Bases* is of key importance.

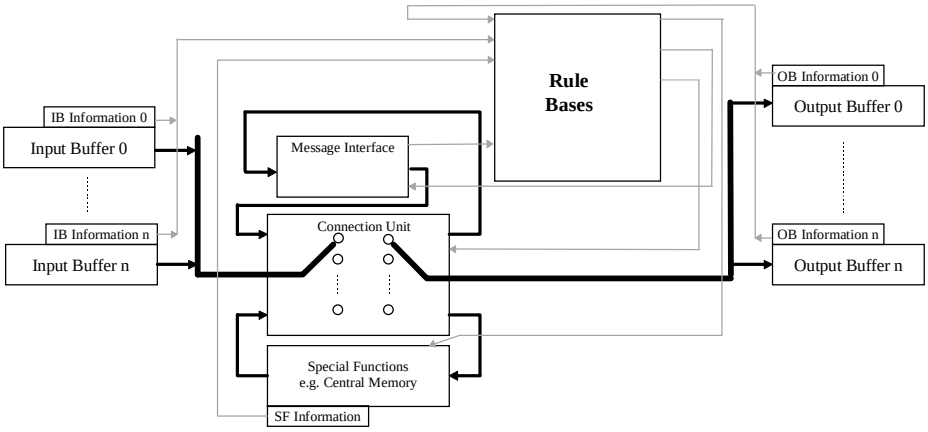

Figure 3. Overview of the router architecture

4.2 Rule-based Description

In general configuration data of hardware has a low abstraction level. Therefore especially for the router a higher level description method is needed. It should be

comfortable enough to allow non-experts to compose adequate routing algorithms for their problems. An appropriate tool ("Rule Compiler") generates the configuration data by translation.

An algorithm can be separated into the two parts state and a function computing results and state changes. Since the complexity of the algorithm is determined by this function, its suitable formalization is a basic precondition for the applicability of the approach. In order to describe routing algorithms clearly and concise, the form of the specification should support constructions typically found there. By the nature of the problem geometric concepts play an important rule, e.g. convex regions of the network. This results in frequent case distinctions which are well expressed by a set of rules. Furthermore, for a fast interpretation of the algorithm the underlying model has to be inherently parallel, which is fulfilled by the rule-based approach used here.

The basic form of a rule as used here is `IF <premise> THEN <conclusion>`, where the premise is a logic expression and the conclusion contains a list of commands.

In the simplest version the premise contains comparison expressions on input parameters while the conclusion is just one return command, e.g.

```
IF xpos<xdes AND ypos=ydes THEN RETURN(east);
```

This line from a routing algorithm for a two-dimensional grid like NAFTA shows, how the premise can be constructed from comparisons of the position of the router (xpos, ypos) and the destination of the message (xdes, ydes). If it applies it can be concluded that the only output for the message is the one labeled "east". For the other situations similar rules have to be written, giving a complete set. The sequence of the rules within a rule set has no effect. Checking whether a rule is applicable appears for all rules independent of each other, i.e. can be done in parallel. Only one rule is selected at one invocation; if more than one rule is applicable it is up to the implementation which one is taken. A routing algorithm normally consists of several such sets for different tasks like direction decision, adaptivity or fault state propagation. Complex routing algorithms consist of more than some functions. They have a state. Time and sequence of execution play an important role and the possible actions are much more complex. Hence, for the description of sophisticated routing algorithms further additions are necessary. They also ease the use by providing short forms for regular patterns. An example is given after the introduction of the extensions (Figure 4).

- *Variables*: can be written in conclusions with the value of an expression, and read in expressions within the premises and conclusions. In this form the state of an algorithm is modified explicitly. This can be seen in the example at the variable `number_unsafe`.
- *Indexed data accesses*: header information, status of hardware like buffer usage or link state have the form of arrays known from traditional programming languages; if variables can be indexed, the size of the rule base is reduced dramatically. In the example this is access to the message field `new_state`.
- *Data types*: Declaration of the used variables and functions allows a precise mapping to hardware and is necessary for completely defined semantics. The aim of mapping to hardware restricts the available data types to integers within finite

ranges, discrete symbols, the union of these two, and subsets of these. The declaration of symbolic names for the fault states (`fault_states`) is shown.

- *Quantors* from predicate logic: The existence and all quantors are applied only on finite sets and therefore no problem about decidability arises. Their usage is just a short form for propositional logic expressions in a regular pattern, yet powerful. Note that the "∃"-Sign is used often in the pseudonotation of routing algorithms [CyG94, ChW96].
  An example from NARA [CuA95] with Quantors in the premise is given here:

```
IF EXISTS i IN minimal(dx,dy): outchan(i,vc)=free AND
     (FORALL j in minimal(dx,dy): outchan(j,vc)=free AND
       mean_queue(i,vc)<=mean_queue(j,vc))
  THEN !send(indir,vc,i,vc),
       out_queue(i,vc)<-message_length(indir,vc);
```

  The nested expression shows a selection of a direction which is minimal (shortest to destination), deadlock-free and available. The message handled is sent there (generation of `!send` event, see below) and a variable holding the amount of data assigned for this output is updated.
- *Subbases*: Algorithms are modularized by using sets of rules forming a subbase, which can be used like a function or a command in premise or conclusion of other rules. The invocation of a subbase does not imply a sequential processing order because of the fully functional interpretation.
- *Functions* allowed in premise and conclusion expressions include standard arithmetic operations and comparisons, set union and join and membership testing. Note that the set operations are simple to implement in hardware for the small sets encountered in routing algorithms.
- *Events*: The temporal behavior is based on the notion of discrete events that are generated by conclusion commands or the switch hardware, e.g. when a message arrives. An algorithm consists of rule bases which are associated with an event (or set of events) by a line of the form: ON <event> <parameterlist>
  The parameter list allows processing events that differ only in a parameter (like the input an arriving message comes from) with the same rule base. In the example the `<event>` is `update_state` and one parameter `dir` is present.
  The model of event-triggered execution of rule bases is inherently parallel as the rule bases are independent from each other. For proper synchronization the execution of a rule has to appear atomically. Asynchronity can be explicitly allowed by the generation of internal events. The commands of the conclusion are executed in parallel. The generation of an event can be seen in the example below (!send_message). It is used to generate a message to neighboring nodes.

To illustrate the concept an excerpt from a rule base of ROUTE_C is shown in Figure 4 which implements the fault state generation and propagation.

```
-- it is assumed that the event update_state occurs
-- if a neighboring node fails, or the neighbor's
-- state changes, or a link to it

-- this constant is a type declaration: set of states
CONSTANT fault_states={safe,faulty,ounsafe,sunsafe,lfault}
VARIABLE number_unsafe IN 0 TO dirs
-- in opposite to the paper [ChW96] number_unsafe is the #
of
-- neighbors that are not safe, that is unsafe (strongly
-- or ord.) or link fault or faulty

ON update_state(dir)
-- the first neighbor gets faulty, just note it
 IF  new_state(dir) IN {faulty,lfault} AND number_faulty=0
  THEN neighb_state(dir)<-new_state(dir),
       number_faulty<-number_faulty+1,
       number_unsafe<-number_unsafe+1;
--  now  too  much  n'bors  are  unsafe,  change  state  and
propagate
 IF new_state(dir) IN {sunsafe,ounsafe} AND state=safe
    AND number_unsafe=2
  THEN state<-ounsafe, number_unsafe<-number_unsafe+1,
       FORALL i IN dirs: !send_newmessage(i,ounsafe),
       neighb_state(dir)<-new_state(dir);
-- more rules ommitted here for brevity
END update_state;
```

Figure 4. Some rules for state update in ROUTE_C

While the premises manage with simple constructs the conclusions contain several commands including the use of a quantor. This way the rule base is independent of the node degree. Rules for the update of the node state are shown to be evaluated when a neighboring link or node breaks or notifies a state change. The different actions (update of counters about number of nodes in unsafe states and so on) are written in the conclusion.

Overall the rule-based description has the properties of a modern programming language. Since it is similar to the notation used in publications about routing algorithms it can be used as representation method to compare and assemble routing algorithms. Its most important advantage is the possibility of fast interpretation by a hardware structure embedded in the router architecture with sufficient freedom for the routing algorithm (e.g. generation of messages, buffer structures).

4.3 Rule Interpreter

As noted before, software solutions would limit the network performance drastically and therefore an optimized execution of routing algorithms in hardware based on the rule description is preferable. The execution of the routing algorithm is done in the block "Rule Bases" of Figure 3. This block contains registers, several rule

interpreters and an event manager coordinating the parallel processing in the rule interpreters.
Rule evaluation by the rule interpreter means that, initiated by an event, the parameters and register values are used to select the rule whose premise applies and to perform all commands of its conclusion. It is clearly prohibitive to search for the appropriate rule in a sequential manner.
Given a rule-based representation of a routing algorithm, a fast hardware rule interpreter can be implemented with the ARON[3] approach [BDK96, BKM96, DLO97]. Its main concept is the generation of an unique index to a table in which the conclusions of the rules are stored. This index is computed from the input values and has a much smaller range than the input space. The rule base itself is compiled off-line to a completely filled rule table where conflicts are resolved and gaps are eliminated, i.e., for each possible combination of input values there is exactly one table entry which holds the corresponding conclusion. The principal structure of this rule interpreter is shown in the following figure.
Interpreting a rule base can be divided into three steps (Figure 5):

- evaluation of premise expressions (premise processing),
- selection of the appropriate rule within the RBR-kernel and
- execution of the corresponding commands (conclusion processing).

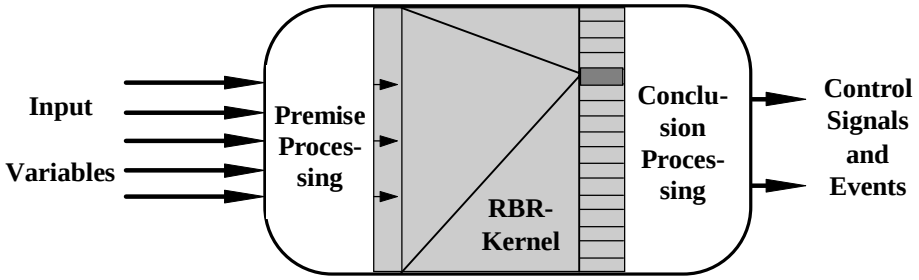

Figure 5. Principle structure of the rule interpreter

Inputs for a rule base are e.g. network conditions and information about a message (destination, source, etc.). The relevant features of the input variables are extracted in the premise processing unit such that rule interpretation is reduced to a simple table lookup in the RBR-kernel. Within the conclusion processing resulting values of internal and event parameters are determined.
A flexible rule interpretation has to handle predicates and functions in the premises and performing calculations within the conclusions. Further requirements are selection and evaluation units for the rules itself. A hardware implementation for the rule interpreter is shown in Figure 6.
The predicates and functions (e.g. subtraction, addition, priority detection etc.) in the premise processing unit are implemented as configurable hardware units (*Free Configurable Function Block* = FCFB) with flexible interconnections. Since the

[3] Alternatives Regularly Organized and Numbered

expressions in the conclusions could contain the same kind of functions as premises it is suggesting to use a common pool of resources for their computation.
The three steps of Figure 5 are found in this structure: the premise processing uses the *Input Configuration* and some of the FCFBs, while the RBR kernel is made up of the *Rules* and the *Premise Configuration* modules. The conclusion processing also uses the FCFBs to share these area-expensive blocks. Furthermore it incorporates the *Conclusion* and *Result Configuration*.

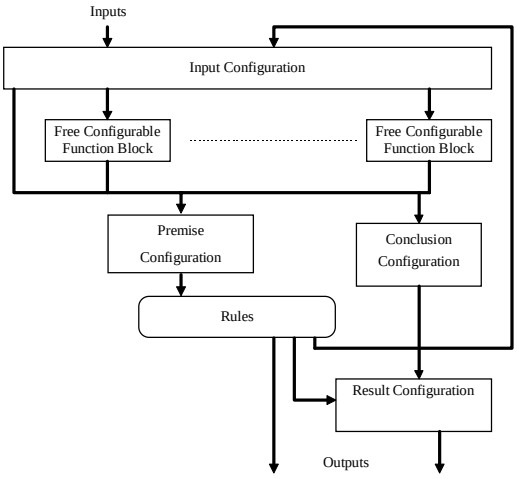

Figure 6. Structure of the rule interpreter

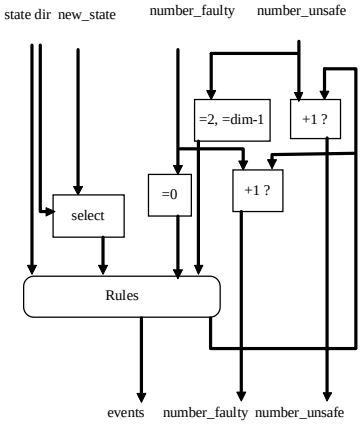

Figure 7. Example configuration

The rule interpreter is divided into three main parts, the interconnection structure *(Input, Premise and Conclusion Configuration*), the functions (FCFB) and the rule skeleton, labeled *Rules*. The interconnection structure resembles the "routing pool" found in modern programmable logic devices (FPGAs, CPLDs) and allows a very fast wiring of a set of inputs to a set of outputs, e.g. for the Input Configuration from the rule interpreter's inputs to the FCFBs and the Rules. Note that most of the signals are actually wire bundles since they carry values from a larger domain. These bundles are routed together which gives a simpler and more regular structure than in the FPGA/CPLD case. A detailed study of interconnection systems can be found in [Hon96]. The FCFBs have to be able to implement all expressions occurring in premises and conclusions. However the kind of expressions found in routing algorithms is very typical and therefore only few universal blocks are necessary. Most of them can be a special form of arithmetical units, tailored to network topologies or arithmetic for adaptivity. For instance one very common function is the selection of a minimal value.
To explain the rule interpreter's structure in detail, Figure 7 shows an example configuration for the rule base `update_state` of ROUTE_C. A part of this rule base was shown in Figure 4 above. Since for `state` and `new_state(dir)` all individual values occur in the premisses of the rules, no comparison is needed and their current values are used as part of the table index directly. Variables

`number_faulty` and `number_unsafe` are compared to the value of 0 and the values of 2 as well as dimension-1 respectively (note that not all rules are shown in Figure 4). These variables are implemented by using registers and incrementers as FCFBs. Since only some rules count up `number_unsafe` or `number_faulty`, the incrementers have to be controlled by the conclusion (conditional increment). To enable this combination of rule evaluations and input values, a connection between rules and the input configuration has been provided. It has to be configured in a way that backward influences of conclusions to the premises of the rules are excluded. In this way the flow through the rule interpreter is straight and pipelining can be applied to increase throughput.

The rule unit can be implemented as RAM or PAL-structure. The advantage is that the routing decision is done in a very short time given by the sum of the delays in the configurable wiring (negligible), two times the FCFBs and *one* memory access or way through a PAL. It has to be considered as an implication of this method that the amount of required RAM can grow exponentially with the number of input values. It turned out that routing algorithms have enough regularity that structuring and using the premise configuration allow small rule tables even for complex algorithms. This can be seen also from the application of the architecture to the two examples of Subsection 2.2 in the next section.

5 Evaluation of the Examples

To illustrate the architecture and to give concrete numbers for the fault tolerance overhead of the two introduced algorithms NAFTA and ROUTE_C the result of mapping them to the rule-based router hardware model is shown. The data given is derived from example implementations. The rule-based router can execute them efficiently, because their model of data transport and buffering does not impose special adaptations. Thus no special functions are needed. The adaptivity method of the second algorithm is not specified in detail. Therefore it is assumed that the generation of the adaptivity criterion is done separately. This is valid for many routing algorithms and ROUTE_C can be completed by any of the methods used there. This can be done simply by using an appropriate rule base. However this assumption has no implications on the overhead generated by fault tolerance.

Both algorithms require more than one consecutive rule interpretation in the presence of faults. While NAFTA in the fault-free case proceeds with one step and in the worst case needs three, ROUTE_C always needs two steps. In both cases this overhead in time accounts to fault-tolerance. The non-fault-tolerant routing algorithm NARA and a stripped down variant of ROUTE_C can be implemented with only one interpretation per message. Of course it is possible to integrate several steps into one, but this would result in very large rule bases with many complex FCFBs. For instance the combination of the two rule bases of ROUTE_C decide_dir and decide_vc (see below) requires a rule interpreter configuration with $1024*2^d \times (d+1+a)$ bits rule table, where d stands for the dimension of the hypercube and a additional bits are needed for adaptivity.

Name	Size (Bit)	FCFBs	Meaning	nft
incoming_message	1024 × 8	2 × magnitude comparator, minimum selection, mesh distance computation, membership testing	handling of an incoming message	*
in_message_ft	256 × 7	logical unit, minimum selection	routing decision in ft mode	
update_dir_table	64 × 28	set subtraction	new fault states require update of data	
message_ finished	64 × 8	minimum selection, 4 decrementors	fair output scheduling	*
calculate_new_ node_state	64 × 9	computation in a finite lattice, set difference, state comparison	status from a neighbor node or change of a link state	
test_exception	32 × 9	membership testing	handling of messages in a special situation	
tell_my_ neighbors	16 × 4	no FCFB needed	generation of messages to adjacent nodes	*
flit_finished	4 × 4	decrementor, adder, comparator	update adaptivity criterion	*
fault_occured	3 × 4	2 × membership testing, set union	update of node state on failure	
message_from_ info_channel	2 × 3	no FCFB needed	update of adaptivity or fault information	*
consider_ neighbor_state	2 × 7	incrementor, computation in a finite lattice, integer comparison with const.	consistency of neighboring states	

Table 1. Rule bases of NAFTA

For each algorithm in Table 1 and 2 respectively characteristics of the rule bases are given. There it is marked (*) whether a rule base is needed for a non-fault-tolerant algorithm or not in the column "nft". This non-fault-tolerant algorithm results from the demand that it should behave exactly like the original algorithm in a fault-free network. Clearly a lot of resources are not used in this case. For NAFTA the non-fault-tolerant version is simply NARA. The implementation used for NAFTA (Table

1) uses many rule bases to get high parallelity and to reduce the operations needed for every message. The size is the dimension of a RAM necessary for the compiled rule base.
ROUTE_C (Table 2) needs only four rule bases, but requires five virtual channels as already mentioned. The total size of 2960 bits of rule table memory for a 64-node hypercube and a=2 is really small. Also the requirements on the FCFBs are achievable. The most complex device is the minimum selection, but since this operation is found in most adaptive routing algorithms the existence of a rule interpreter that provides this function as FCFB is suggesting. Since the virtual channels are only needed for fault tolerance the rule base which selects the right channel accounts clearly for fault tolerance. Obviously the state update is also not needed in the fault-free case.

Name	Size (Bit) d = dimension a = adaptivity command	FCFBs	Meaning	nft
decide_dir	512 × 4	6 logical units d bits wide: AND, zero check, input negate	decides which outputs can be taken	*
decide_vc	(4*d) × (1+a)	minimum selection (same as NAFTA), compare with constant,	decide output and virt. channel, update adaptivity ("a" Bits for that)	
update_state	180 × 7	conditional increment, compare with constant	state update requires counting of unsafe or faulty neighbors	
adaptivity			create adaptivity criterion, no details given, since this is not specified.	*

Table 2. Rule bases of ROUTE_C

Besides of the rule bases the hardware effort is determined by the registers needed. In this regard it is interesting to see how many rule bases compete for concurrent access, since this forces partial sequentialization. The term register refers to the variable declaration of the rule base and is not necessarily the granularity of mutual exclusion.
For the NAFTA implementation 159 bits are organized in 8 registers, where some of them are modified by several rule bases; only 47 bits account for fault-tolerance.

In contrast for ROUTE_C the amount of registers needed depends on the node degree d. In total 15d+2log d +3 register bits are needed, organized as nine registers, where one of them is constant and the other are modified only by one rule base. 9d register bits are needed in the non-fault-tolerant case too.
Altogether both algorithms show that the additional effort in hardware size and speed for fault tolerance is not negligible. While NAFTA shows an increase mainly in the complexity for updating states and choosing the right output, the additional hardware cost for ROUTE_C is dominated by the fivefold virtual channel demands. The degree of fault tolerance reached by both algorithms cannot be compared because two different topologies are used.

6 Conclusion

In this paper the reason and amount of overhead (complexity, hardware resources) for provision of fault tolerance in routers for high-speed networks are investigated. Two algorithms NAFTA [CuA95] and ROUTE_C [ChW96] were used to show the concept of structuring a routing algorithm according to several subgoals. The derivation of concrete data for the overhead was done using the hardware architecture of a flexible router. It turned out that fault tolerance implies a considerable overhead in hardware cost and in the time required for a routing decision. This architecture is based on a description of the algorithm using a set of rules. Since this rule-based model for the execution of routing algorithms is universal and hardware implementation oriented, the results are applicable for the design of dedicated routers too.

7 Acknowledgment

This work was sponsored by the German Research Council DFG (MA 1412/3-1) in cooperation with SFB 376 „Massive Parallelität“ at the University of Paderborn, Germany.

8 References

[BCF95] N. J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, W.-K. Su: Myrinet: A Gigabit-per-Second Local-Area Network. IEEE Micro, Vol. 15, No. 1, 29-36, 1995

[BDK96] W. Brockmann, A.C. Döring, T. Kosch, G. Lustig, E. Maehle: Rule-Based Routing for Fault-Tolerant Parallel Computers. Proc. EDDC-2 Companion Workshop on Dependable Computing, 63-72, AMK-Press, Gliwice 1996

[BKM96] W. Brockmann, T. Kosch, E. Maehle: Rule-based Routing in Massively Parallel Systems. Proc. 4th Euromicro Workshop on Parallel and Distributed Processing - PDP'96, 154-161, IEEE Computer Society Press 1996

[BoC93] R.V. Boppana, S. Chalasani: A Comparison of Adaptive Routing Algorithms. Proc. 20th Symposium on Computer Architectures, 351-360, IEEE Computer Society Press 1993

[BoC96] R.V. Boppana, S. Chalasani: A Framework for Designing Deadlock-Free Wormhole Routing Algorithms. IEEE Trans. on Parallel and Distributed Systems, Vol. 7, No. 2, 169-183, 1996

[BoY94] K. Bolding, W. Yost: Design of a Router for Fault-Tolerant Networks. Proc. PCRCW'94 Parallel Computer Routing and Communication '94, 226-240, Springer 1994

[ChB95a] S. Chalasani, R.V. Boppana: Fault-Tolerant Wormhole Routing in Tori. Proc. Computers and Digital Techniques, Vol. 142, No. 6, 386-394, IEEE Computer Society Press 1995

[ChB95b] S. Chalasani, R.V. Boppana: Communication in Multicomputers with Nonconvex Faults. Proc. Conference on Parallel Processing, Lecture Notes on Computer Science 966, 673-684, Springer 1995

[ChK92] A.A. Chien, J.H. Kim: Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. Proc. 19th Annual Int. Symposium on Computer Architecture, 268-277, ACM Press 1992

[ChW96] G.-M. Chiu, S.-P. Wu: A Fault-Tolerant Routing Strategy in Hypercube Multicomputers. IEEE Trans. on Computers, Vol. 45, No. 2, 143-155, 1996

[Coh96] J. Cohen: Gigabit Ethernet vs. ATM. Network World Magazin, 1996

[CuA95] C.M. Cunningham, D. Avresky: Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes. Proc. First Int. Symposium on High Performance Computing Architecture, 122-131, IEEE Computer Society Press 1995

[CyG94] R. Cypher, L. Gravano: Storage-Efficient, Deadlock-Free Packet Routing Algorithms for Torus Networks. IEEE Trans. on Computers, Vol. 43, No. 12, 1376-1385, 1994

[DaS87] W.J. Dally, C.L. Seitz: Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. IEEE Trans. on Computers, Vol. 36, No. 5, 547-553, 1987

[DLO97] A.C. Döring, G. Lustig, W. Obelöer: The Impact of Routing Decision Time on Network Latency. Proc. 4th PASA Workshop on Parallel Systems and Algorithms, 67-83, World Scientific Publishing 1997

[Dua97] J. Duato: A Theory of Fault-Tolerant Routing in Wormhole Networks. IEEE Trans. On Parallel and Distributed Systems, Vol. 8, No. 8, 790-802, 1997

[Gal97] M. Galles: Spider: A High-Speed Network Interconnect. IEEE Micro, Vol. 17, No. 1, 34-39, 1997

[GlN92] C.J. Glass, L.M. Ni: The Turn Model for Adaptive Routing. Proc. 19th Annual Int. Symposium on Computer Architecture, 278-287, ACM Press 1992

[GaY95] P. Gaughan, S. Yalamanchili: A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks. IEEE Trans. on Parallel and Distributed Systems, Vol. 6, No. 5, 482-497, 1995

[Gus92] D.B. Gustavson: The Scalable Coherent Interface and Other Related Standards Projects. IEEE Micro, Vol. 12, No. 1, 1992

[Hon96] André DeHon: Reconfigurable Architectures for General-Purpose Computing. PhD-Thesis at Massachusetts Institute of Technology, Artificial Intelligence Laboratory, available as Technical report No. 1586, 1996

[KiS93] J. Kim, K.G. Shin: Deadlock-Free Fault-Tolerant Routing in Injured Hypercubes. IEEE Trans. on Computers, Vol. 42, No. 9, 1078-1088, 1993

[Kon90] S. Konstantinidou: Adaptive, Minimal Routing in Hypercubes. Proc. 6th MIT Conference on Advanced Research in VLSI, 139-153, MIT Press 1990

[MLS96] A. Mu, J. Larson, R. Sastry: A 9.6 GigaByte/s Throuput Plesiochronous Routing Chip. Proc. IEEE Int. Computer Conference ’96, 261-266, IEEE Computer Society Press 1996

[PGF94] G. Pifarré, L. Gravano, S.A. Felperin: Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations. IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 3, 247-263, 1994

[ScV96] C. Scheideler, B. Vöcking: Universal Continuous Routing Strategies. Proc. 8th ACM Symposium on Computer Architecture and Algorithms, 356-365, ACM Press 1996

[SuS95] C.-C. Su, K.G. Shin: Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes. IEEE Trans. on Computers, Vol. 45, No. 6, 666-683, 1995

[SuW97] P.-H. Sui, S.-D. Wang: An Improved Algorithm for Fault-Tolerant Wormhole Routing in Meshes. IEEE Trans. on Computers, Vol. 46, No. 9, 1040-1042, 1997