TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Engineering

IAY70LT

Rando Rostok 111659

FAULT-TOLERANT 2D-ROUTING CONCEPT FOR NETWORK-ON-CHIP BASED MANY-CORE ARCHITECTURES

Master thesis

Supervisor: Thomas Hollstein

Professor

Dependable Embedded Systems

Co-Supervisor: Siavoosh Payandeh Azad

Research associate

Tallinn 2014

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAY70LT

Rando Rostok 111659

KAHEDIMENSIOONILINE VEAKINDEL MARSRUUTIMINE VÕRK-KIIBIL BASEERUVATEL MITMIKTUUM ARHITEKTUURIDEL

Magistritöö

Juhendaja: Thomas Hollstein
Professor
Dependable Embedded Systems

Kaasjuhendaja: Siavoosh Payandeh Azad
Teadur

Tallinn 2014

Author’s declaration of originality

Author’s declaration of originality is an essential and compulsory part of every thesis. It always follows the title page. The statement of author’s declaration of originality is presented as follows:

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Rando Rostok

11.06.2014

Abstract

Kahedimensiooniline veakindel marsruutimine võrk-kiibil baseeruvatel mitmiktuum arhitektuuridel

NoC based solutions are becoming more popular in industry, thus having an efficient routing scheme is important. Before the 90s, the main problem was adaptivity and many different algorithms were proposed since then to attack this problem. Today the main topics concerning NoCs are fault tolerance and resource efficiency. Since links and routers are dependent to each other in some way it is important to have some idea about their state. In this thesis we propose a method which enhances NoC routing algorithm's fault tolerance. This approach is beneficial, when each router gets information about faults in system and preventing packets to end up in dead end situations where further paths are blocked. Each router should have only limited necessary information, which helps keep memory occupation minimal. We also provide good metrics for calculating adaptivity and dependability in routing algorithm. The problem with current metrics is that faults are not considered. Adding fault tolerance can yield better reliability, performance and efficiency for chip.

The thesis is in English and contains 65 pages of text, 6 chapters, 41 figures, 1 table.

Annotatsioon

Tänu viimaseaja suurele arengule riistvara maastikul on sardsüsteemidel baseeruvad lahendused, kogumas aina enam popullaarsust. Suurenenud nõuded energia säästmisele, töö- ning vea kindlusele on tinginud olukorra, mis sunnib üha enam riistvara tootjaid vaatama tuleviku lahenduste suunas. Üks selliseid arhidektuure, mis järjest enam kõlapinda saab on kiipsüsteem (SoC). Kiipsüsteem on arhidektuur, kus kõik riistvara komponendid on paigutatud ühele kiibile. Kuna kiipsüsteemid on väga kinnised ning piiratud mäluga, on väga tähtis, et süsteemi töö- ning veakindlus oleksid tagatud.

Kiipsüsteemides kasutatakse komponentide omavaheliseks suhtluseks erinevaid kanaleid ning arhidektuure, millest märkimisväärseim on võrkkiip (NoC). Võrkkiip võib tunduda alguses kui tavaline võrk kuid tegelikult on nad suhteliselt erinevad. Võrkkiip on palju paremini kohandatav ning omab paremaid marsruutimis algoritme. Kuna võrkkiibil on mälu ning voolu piirangud, on vaja meetodeid, mis suudavad tagada optimaalse ressuriside kasutuse samal ajal ka süsteemi veakindluse ja võimsuse. Probleem hetkel pakutavate lahendustega seisneb selles, et ei suudeta pakkuda meetodit, mis oleks sama aegselt olla nii kohandatav, skaleeruv, veakindel kui ka suudaks toimida ka suure arvu vigade korral.

Selle töö eesmärk on pakkuda välja meetod ummiku vabadele ning hästi kohandatavatele marsruutimis algoritmidele. Pakutava meetodiga on võimalik tagada kõrge veakindlus aste, võrkiibil baseeruvatele lahendustele. Kuna marsruutimis algoritmide ligipääsetavus süsteemis esinevate vigade tõttu langeb, siis meie meetodiga on võimalik see tõsta saja protsendini. See ei tähenda seda, et vigu suudetakse parandada vaid seda, et vigaseid piirkondi suudetakse vältida. Kuna kohanevate marsruutimis algoritmide üheks probleemiks on informatsiooni lokaalsus on meie meetodi idee pakkuda lahendust, mis annaks ruuterile natuke rohkem globaalsemat teavet.

Pakutav meetod baseerub ideel kus otsitakse ligipääsmatuid piirkondi, mille kohta käiv informatsioon tagastatakse sellest sõltuvatele sõlmedele. Kui ruuteril on teave, ligipääsmatute piirkondade kohta, on võimalik neid kas sootuks vältida või kui antud piirkonnas asuvale sõlmele pole ühtegi alternatiivset teed, saab täielikult vältida sinna pakettide saatmist. Kuna informatsiooni maht on piiratud siis kasutatakse erinevaid funktsioone piirkondade optimiseerimiseks.

Meie meetod on kasutatav kõigi minimaalset teed kasutavate marsruutimis algoritmide korral, olles piisavalt skaleeruv tänu tema limiteeritud ruuteri tabelitele. Meetodit saab kasutada vaid võrgus, mis omab võrgusilma topoloogiat.

Testi tulemused näitavad, et meie poolt pakutud meetodiga on võimalik tõsta võrkiibil baseeruvate süsteemide vea- ning sellega seoses ka töökindlus astet. Ligipääsmatute piirkondade arvutamisel ei ohverdatud ühtegi tervet sõlme ning kõik võrgus paiknevate vigadega (katkised lingid) seotud ruuterid omasid vastavat informatsiooni.

Probleem, millele proovime lisaks veel tähelepanu juhtida seisneb selles, et puudub mõõtepuu, millega oleks võimalik määrata marsruutimis algoritmi efektiivsust juhul kui süsteemis esinevad vead. Hetkel pakutavad valemid suudavad küll näidata kui kohanev on algoritm aga ei anna adekvaatset teavet olukordade kohta kui süsteemis peaks esinema vead.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 65 leheküljel, 6 peatükki, 41 joonist, 1 tabel.

Table of abbreviations and terms

SoC	System on Chip
NoC	Network on Chip
DoA	Degree of Adaptivity
DoR	Degree of Reachability
BIST	Built in Self-Test
FPGA	Field-programmable Gate Array
FIFO	First in First out
VC	Virtual Channel(s)

Table of Contents

List of Figures

List of Tables

1. Introduction

Information technology is one of the most developing areas in the world today. As much as the software is important, we cannot forget the underlining hardware which is needed to run fancy, new and innovative software components. While software developing in general has made huge step forward, towards more efficient, better performance and more reliable solutions, hardware field have to catch up. For hardware different kind of new approaches are developed and researched, such as hardware-software co-design or different FPGA approaches. By doing so, we still have to maintain high performance, relatively small size and good thermal conductivity. What is the best approach to do it? Is like a million dollar question. It is a complex task with different dependencies, so there is no easy way of doing it. System on Chip architectures were developed by keeping in mind all those aspects. SoC is an embedded system architecture which combines all the major hardware components into one chip. Major hardware companies like AMD are already developing SoC architectures today. It is interesting and innovative architecture which has its place in industry.

In SoC many different communication architectures can be used. The most promising one seems to be Network on Chip architecture. Network on Chip is similar to regular network and routing will be done via packet communication as well. Because SoC has limited die and buffer space, it is important for those chips to have high fault tolerance and reliability. Since in NoC all the data communication between system components will be done over network, ideas based on regular networks can be used. Fault tolerance is one of the key topics for today's research projects and studies. To achieve good fault tolerance we need to have good and adaptive routing algorithm. But even if we get high adaptivity, there is a problem which does not have a good solution in current R&D projects yet, which is the problem with broken links in network. One broken link can disable a huge portion of the system and at the same time decrease reliability and fault tolerance rate. Broken links could be hard to discover. Ideally good fault tolerance method has to tolerate multiple faults, be scalable and have a minimal cost on both system and implementation.

Our purposed method in this thesis is to develop a fault tolerance method for the NoC based architectures. This new method has to be scalable and routing algorithm independent. It has to be applied on deadlock free algorithms and to be able to tolerate numerous faults.

1.1. Basic Properties of NoC Routing Algorithms

Network on Chip is a communication architecture used in SoC design. It is in constant developing cycle and has been getting more audience in recent years. Each component in network (i.e. IP core) has its own router which connects it to other components via fixed wires. NoC can use either enchanted or regular mesh topology. NoC has simple and more effective routing algorithms, less buffer memory, potentially more routing lines and fixed topology. NoC is also more scalable and flexible but has certain power and area limitations. It has two basic data transmission models: connection- and packet oriented [9]. Each one has its advantages and disadvantages.

Routing in NoC is simpler and has less complexity than other architectures, algorithms are also easier to implement, represent and test. Algorithms used in NoC can be either deterministic or adaptive ([8], chapter 8, 10). Adaptive algorithms can use different approaches, like routing tables or some mechanism to generate several alternative paths for the packet. Because NoCs are usually used in limited, specific and closed systems (i.e. SoC) one of the important factors which need to be considered carefully is **fault tolerance**. It can depend on several attributes (i.e. adaptivity).

1.1.1. Deadlock Problem

In packet routing, especially in NoC design, one of the biggest problems that we need to deal with is deadlock. Deadlock is a situation where packets do not reach their destination because each waiting for the other [9]. It causes a situation where all of the packets are stuck because all buffers are blocked. Deadlock scenario is shown on figure (Fig. 1-1). Deadlock is a dramatic problem and usually caused by circular routing configurations. Deadlocks can be evaded using either prevention/avoidance or recovery methods. One of the easiest resolutions to prevent deadlocks is to prevent cycles in routing ([9], chapter 14). Deadlocks can be also prevented by using virtual channels or layers but it will also increase hardware costs. Some algorithms are able to recover from deadlock situations.

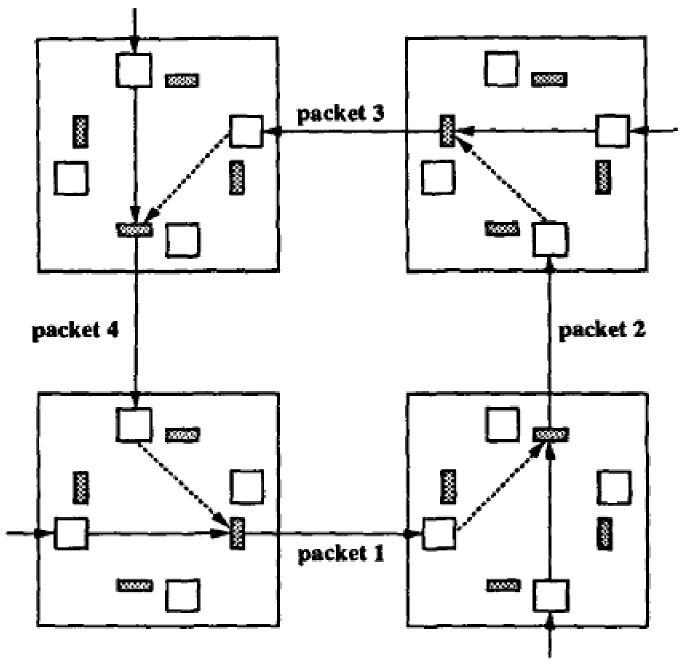

Fig. 1-1. A deadlock scenario © ACM 1992.

The idea behind deadlock prevention is simple; if cycles can be avoided then deadlock cannot occur. In order to perform this task we need to either avoid some turns in routing model or use planar-adaptive methods [8]. Turns can be disabled by using partially adaptive turn model algorithms, which prevent U-turns which can cause cycles. Planar-adaptive methods are more focused with use of virtual channels or different routing layers, these are also more expensive to implement than turn models ([9], chapter 4).

Turn Models

In the early 90's multiprocessors and multicomputer solutions became more popular, which was the motivation to wormhole packet switching which became available. Despite many wormhole packet switch advantages one of its major disadvantage is the problem with deadlocks. In year 1992, Glass and Ni developed a method called turn model, which they claimed to be deadlock and livelock free while not sacrificing much adaptivity of the algorithm [7]. They analyzed different packet routes and all possible turns on 2D-mesh and found that when by prohibiting some paths deadlocks can be avoided. They came up with an idea to prohibit certain (180 degree) turns to avoid cycles in routing model. In every turn model usually 6 turns are allowed and 2 are prohibited. The resulting routing algorithm is deadlock free while still being adaptive enough. There are 16 different ways to combine turns on 2D-mesh (Fig. 1-2), where 12 of them are deadlock free solutions. XY routing can also be represented with turn model, but it has only 4 turns allowed (Fig. 1-3). Common turn model based adaptive algorithms are: West-First, North-Last, Negative-First and Odd-Even are all deadlock free [7]. It is important to note that Odd-Even is the only algorithm in the list which is fully adaptive [13].

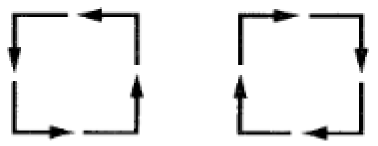
Fig. 1-2. The all possible turns and cycles in two-dimensional mesh © ACM 1992.

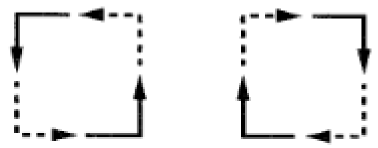
Fig. 1-3. XY routing with turn model representation © ACM 1992

Planar-adaptive methods

Another possibility to prevent deadlocks is using planar-adaptive methods. However it is more expensive than other approaches for avoiding deadlocks. Planar-adaptive methods will introduce virtual channels, different routing layers and dimensions ([9], chaper 4). This means, in order to avoid deadlock situation occurrences, packets will be routed to another layer or dimension. But this should be performed very carefully since multidimensional deadlocks can also happen which are much harder to detect than single dimension deadlocks. Planar-adaptive routing methods are usually both dead- and livelock free and usually with better performance than deterministic routing methods. They increase channel throughput by distributing load due to use of different layers and channels.As mentioned, planar-adaptive routing methods can be either:

1) Virtual layer based (usually maximum of 2 channels).

2) Multidimensional based (i.e. 3D Hypercube).

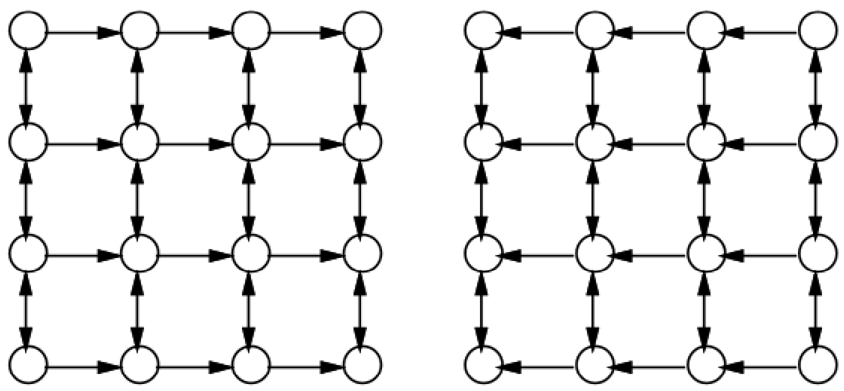

Fig. 1-4. Two virtual layers.

As shown on figure (Fig. 1-4), each virtual layer can have its own routing model (i.e. turn model) which can be applied to it. In each dimension a virtual channels can also be used, but we have to keep in mind that due to higher functionality and extra memory requirements, design cost will increase. Planar-adaptive methods can have a fixed number of VC-s which are dependent on networks size and topology.

1.1.2. Adaptivity

Adaptivity is one attribute to consider when choosing routing algorithm for design. It can provide us with information about reliability, fault tolerance and resource efficiency. The higher the adaptivity, more paths are available for routing, this provides possibility to avoid defected links, nodes and areas, also ability to shut down parts of the chip for better power efficiency. In practice, adaptivity is a measure parameter, which can be used also to compare different algorithms to see how well they perform. When routing algorithm can find new path with most of the occurring irregularities then this algorithm can be called fully adaptive. This however depends on different factors which can decrease adaptivity. Most problems rise with broken link(s). Broken links can decrease adaptivity value in real time and algorithms with high adaptivity can become less adaptive or even non adaptive because of it. Also we introduce a solution how to avoid or incapsulate this problem while not sacrificing algorithm adaptivity properties of the network. We have to keep in mind that algorithms also have to be deadlock free. Tradeoff between deadlock freeness and adaptivity is due to the fact that deadlock free approaches can decrease adaptivity while high or fully adaptive approaches can cause deadlocks.

One limitation with adaptivity is local information about the status of the network. Since each router knows only information about its links to direct neighbors then packets could end up on links with huge traffic on it. As shown on figure (Fig. 1-5) a node with red stripes is the central point in routing case. Since its north link has some traffic while east link is traffic free, packets will be routed towards east. It does not know that router I has a faulty link in north and a flooded by heavy traffic link in east. Since router I north link is blocked by fault, all the packets will be routed to east. Each packet on that link will have higher latency because of misrouting. Situations like this can be avoided if routers have more global and specified information about mesh.

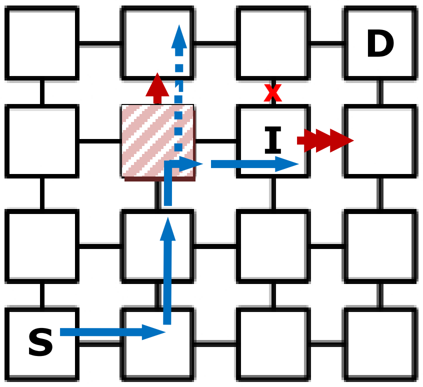

Fig. 1-5. Localization problem in adaptive routing.

One of the drawbacks with adaptive algorithms is that packets could be sent to nodes which in reality are unreachable. Since router does not know information about faults in the network it will be sending packet if it is possible. If the node which packet sent for is unreachable due to broken links or other faults in the system, packet(s) will never reach their destination and information will be lost.

In next chapters we will discuss adaptivity in more details. To give a brief introduction to adaptivity metrics, the basic idea behind it is calculate the combination of all possible paths. This method was introduced by Glass and Ni.

Assume that (s_x, s_y) is the corditation of sorce node and (d_x, d_y) is the cordiation of destination node, then adaptivity metric will be calculated by the folowing formula:

$$S_{algorithm} = \frac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!} \quad (1)$$

Where:

$\Delta x = |d_x - s_x|$

$\Delta y = |d_y - s_y|$

The bigger the S is the more adaptive algorithm is. We also did some research with routing algorithms and tried to find out what can be the impact on adaptivity with increasing number of faults in the system. The results have been described in next chapters.

1.2. Introduction: Analysis of Routing Algorithms with respect to dependability

Adaptivity and deadlocks are not only issues with routing algorithms. One of the today's major problems in routing comes when something (i.e. link, router or chip) breaks. In this thesis we will be considering matters with broken link(s). Now the main problem is how to evaluate our algorithm effectiveness and adaptivity after broken link occurs. Also we need to find out what is the maximum possible amount of broken links that system can tolerate while still being operational.

Before introducing dependability we need to first set few rules: 1) if possible, always go towards x-direction first 2) use only minimal paths. All links in mesh structure are somewhat dependable and adaptive routing algorithm can choose new link when the old one breaks. Mathematically this can be presented by using probabilities. There is no need for exact position where link is broken but instead we need to find out what is the percentage of all the links that still are operational. Therefore each algorithm can have its own dependability rating.

1.2.1. XY

It is most simplistic and basic NoC routing algorithm. Idea is to first go towards x-direction and after x is reached move towards y-direction. Problem with XY algorithm is that it is deterministic. There is no path diversity and there is always only one possible path towards destination. If any link breaks then some nodes could become inaccessible from that point. Any broken link especially on x-coordinate can potentially disable a huge part of the system.

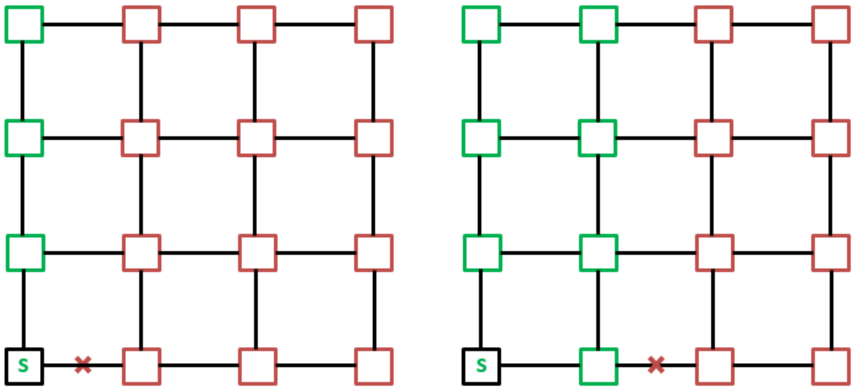

Fig. 1-6. Problem with XY routing fault tolerance.

As shown in figure (Fig. 1-6), the source loose access to the part of the network shown in red which describes the fact that XY does not have any fault tolerance. This means any broken link can cause real problems in system operation. Here we can see why deterministic algorithms are not suitable for systems with fault tolerance requirements in it.

Hybrid XY/YX

Thanks to XY deterministic structure, it can be easily used to combine different methods. We also tried to find adaptive algorithm by making modifications to XY. Idea is to change routing model to YX after fault occurred on path. It helps to overcome some of the XY bottlenecks (Fig. 1-7) but it requires two extra virtual channels to be deadlock free. There is the exception that a packet should never switch back to pervious layer. The proposed hybrid algorithm achieves better results in some cases but since we have changed the routing model to another XY, there is still same problem with adaptivity and fault tolerance in the end.

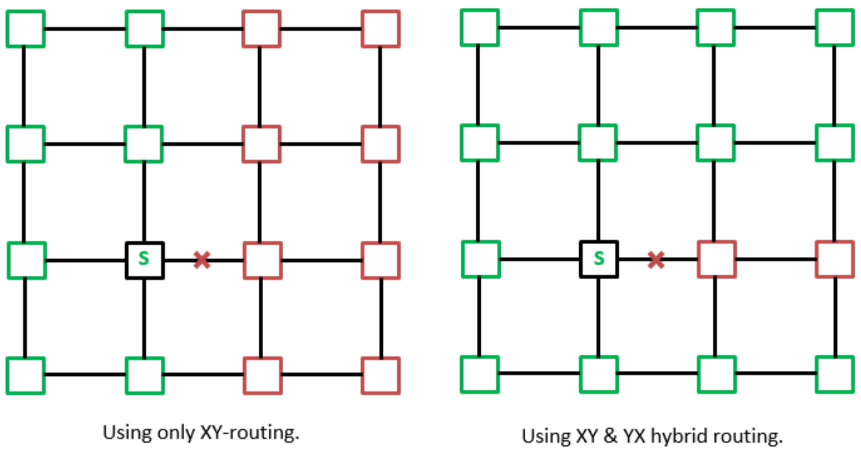

Fig. 1-7. Hybrid XY/YX and XY.

1.2.2. West-First

West first is one of the most commonly used turn model algorithms, idea behind it is to route packet first to west (if needed) and then adaptively N, E or S directions. It is proven that any turn model based algorithm including West-First is deadlock free [7]. West first, like any other turn model algorithm (except for the case of Odd-Even), is partially adaptive, which means that it can find a few but not all available alternative paths. One of the problems with West-First algorithm is when link on west path is broken then further routing on that destination cannot be made. This is the issue with all of the partially adaptive algorithms, that in some point (i.e. fault occurrences) they will find routing impossible.

1.2.3. Odd-Even

Odd-Even algorithm is the only fully adaptive turn model algorithm, which does not need to use virtual channels. It has cheaper implementation costs but its structure makes fault tolerance complicated. Odd-Even is interesting algorithm where mesh structure is divided into two classes: odd columns and even columns [13]. First column is usually even column with index number zero. Next to it is odd column with index number one and so on. Every column has its own turn model with different allowed and prohibited turns. Basic idea behind Odd-Even algorithm is to prevent U-turns. Finding dependability value for Odd-Even algorithm is difficult because of different cases needed to be considered. Odd-Even algorithm routing model is shown in figure (Fig. 1-8). It does not have any virtual channels in it.

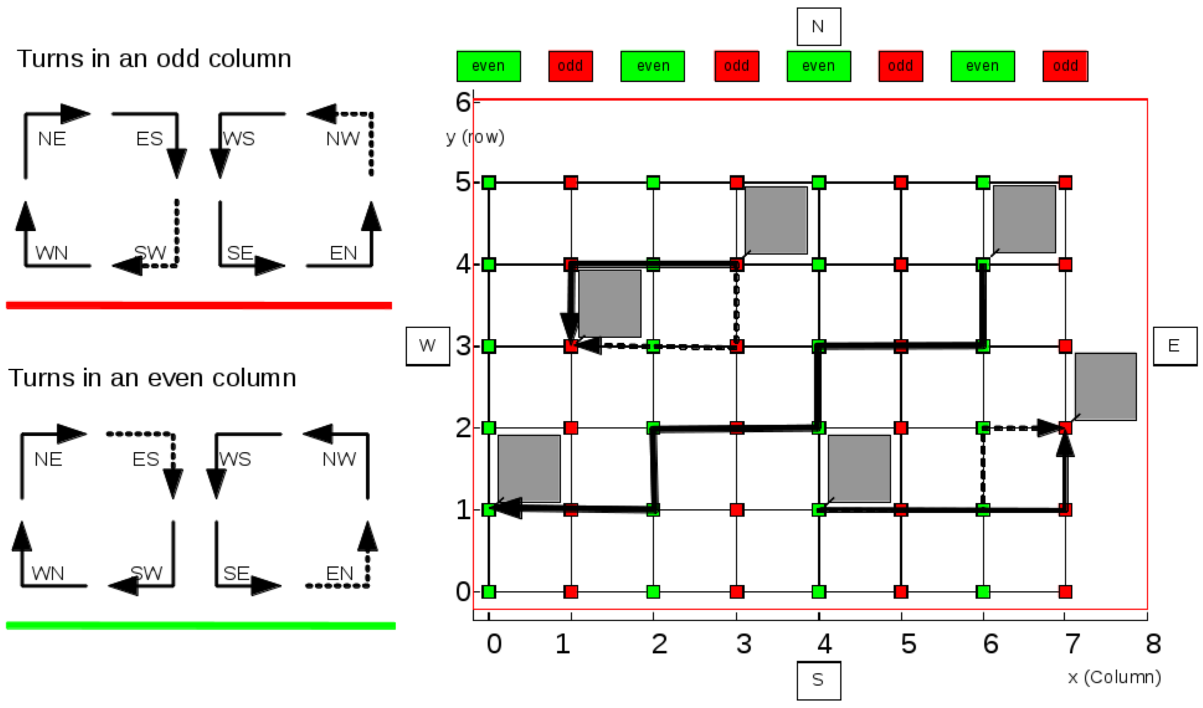

Fig. 1-8. Odd-Even algorithm representation. © Thomas Hollstein

1.2.4. DyXY

Dynamic XY is a NoC routing algorithm which is both deadlock and livelock free. It is similar to XY algorithm while being fully adaptive. DyXY is minimal routing algorithm, which means that packet only travels along the shortest path. If there is more than one path available, the least congested path will be taken. DyXY will use two virtual channels (also known as planes). Depending on destination node's x coordinate with respect to source (either east or west) packet will be routed to certain X sub-network or sub-plane. Figure (Fig. 1-9) shows complete network and its division into separate sub-networks b and c which are –X and +X sub-planes. On both sub-networks, routing will be done differently. On each sub-plane, all the turns are available with an exception of turning backwards (i.e. from E plane towards W). One of the constraints concerning adaptivity is that if packet reaches either x or y coordinate of destination node, it will go straight towards other direction. This might be a problem when some links are broken on that path [11], [15].

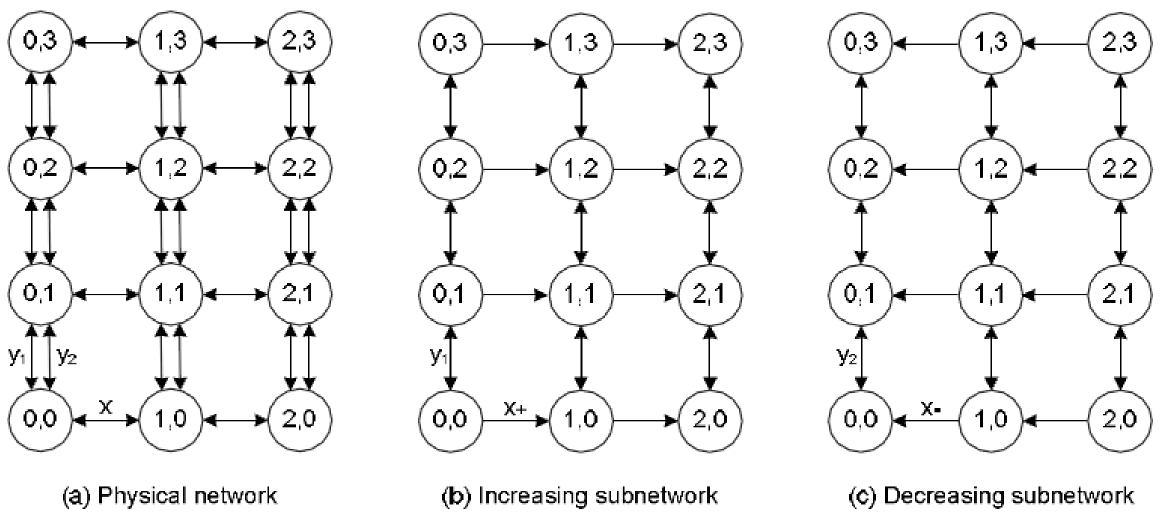

Fig. 1-9. DyXY network representation © Danesthalab.

1.3. Metrics for Reachability

As already mentioned in pervious sections there is a great need for descriptive metrics for both adaptivity and dependability. In this section we will take a closer look into it and describe a method to calculate this metric while considering link faults.

Glass and Ni developed adaptivity metrics based on their turn model. They proposed metrics called Degree of Adaptivity (DoA) [7], which calculates the amount of accessible paths divided by amount of total possible paths. Glass and Ni found out that larger the amount of shortest paths that algorithm allows, the more adaptive is the algorithm (see 1.1.2 for Glass and Ni, DoA formula).

As shown in pervious chapter, in XY algorithm, half of the system can be inaccessible because of faults (Fig. 1-6). Even some XY modified variants like surrounding XY [10] or our own proposed Hybrid XY/YX (Fig. 1-7) does not deliver adaptivity which will be satisfying for us.

Deterministic algorithms also introduce a new problem which is congestion. Great amount of the traffic will be routed through same paths because of broken links and limited router information. However even adaptive algorithms suffer from bottlenecks caused by faulty links, which can decrease their adaptivity. For example in West-First algorithm you will need to go first towards west but when west path is faulty then further routing cannot be made. DyXY and Odd-Even are probably most adaptive algorithms. Calculating adaptivity value for Odd-Even is more complicated than for other tested algorithms. Our experiments showed that DyXY was best algorithm with satisfying adaptivity and reachability factor. DyXY is also more simple than Odd-Even, however two virtual channels are still needed.

1.3.1. Static Reachability Metrics Calculation

Dependability is another attribute which can be compared and considered for routing algorithm. Dependability has similarities with adaptivity but in the end they are different. The idea is also based on probability theory. Assume that probability chance of event A is P1 which occurs when the link is working. Probability chance of event B is P2 which occurs when the link is not working. This can be used to construct formula.

There are two assumptions which have to be applied first:

1) Always use minimal path routing.

2) Always try to go towards x-direction first, if the link in x-direction is defect, go to y-direction.

With those constraints we can construct basic idea behind this.

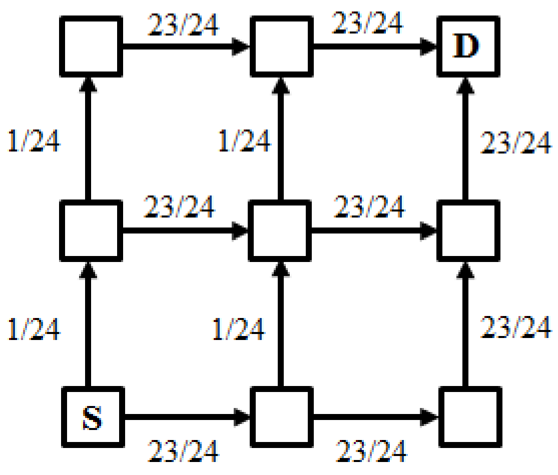

Fig. 1-10. Dependability metrics idea.

To explain it better, take a look at figure (Fig. 1-10). We assume that the probability of a link to be broken, **P**, is 1/24. Assuming we have 1 broken link, the probability of going towards x direction in our routing algorithm is 23/24 (because we still have 1/24 chance that link is broken). When we need to go towards north the probability is 1/24 (because

23/24 chance that we are going towards favorable path and 1/24 we choosing second path because link is broken).

If we have a fully adaptive algorithm like in our case then we can make following calculation:

$$\left(\frac{23}{24}\right)^4 + 2\cdot\left(\frac{1}{24}\right)\left(\frac{23}{24}\right)\left(\frac{23}{24}\right)^3 + 3\cdot\left(\frac{1}{24}\right)^2\left(\frac{23}{24}\right)^2\left(\frac{23}{24}\right)^2 = \left(\frac{23}{24}\right)^4\left(1 + 2\left(\frac{1}{24}\right) + 3\left(\frac{1}{24}\right)^2\right) = 0.918$$

What we can see here is that if we assume we have a broken link in our system then even with fully adaptive algorithm that will use only minimal (fixed) path lengths we will never get 100% dependability value.

For simplicity of calculations, we ignore the negligible terms (from 2nd term on) in the summation and we can construct a general basic formula for adaptive algorithms which we will call, Degree of Reachability (DoR):

$$(1-p)^{\Delta x+\Delta y}[1+(\Delta x-1)p] \qquad (2)$$

Where:

$$\Delta x = (X_D - X_S)$$
$$\Delta y = (Y_D - Y_S)$$
$$Path\ length\ (L_{path}) = \Delta x + \Delta y$$
$$p: the\ probability\ of\ a\ link\ being\ broken$$

This formula provides us with some rough estimation of the networks state, while we cannot use probability calculations as a base for our routing decision. For a routing decision we need to know exactly which node is unreachable before we take a routing decision.

2. State-of-the-art in Dependable Routing

A lot of the research projects, case studies and papers have been written in this field, however quite a few projects describe this problem with fault tolerance more closely. The problem with most of the researches is that they offer solutions which are limited, have very specific use in field or have design flaws (for example, they are not deadlock free). Also most of the solutions are using fault regions and/or many virtual channels which will make routers more complex. For us, easy implementation, high performance, deadlock freeness and multiple fault tolerance are the most important factors. In general, papers can be divided into two main topics: local circumfusion strategies and global approaches. Following I will give a brief overview on some of the promising methods and approaches proposed in papers.

2.1. Dependable Routing in Multi-Chip NoC Platforms

Yoneda and Imai made approach specifically for automotive industry. They propose two solutions, first a different NoC architecture a multi-chip NoC for automotive industry and an algorithm. In automotive industry different car types has to have different size of NoCs which is expensive. Multi-chip NoC combines different NoCs together via off-chip links, in this way network is extended and chip is more fault tolerant. One of the huge disadvantages with such approach is that it cannot preserve full on-chip topology and missing links have to be considered as faulty links.

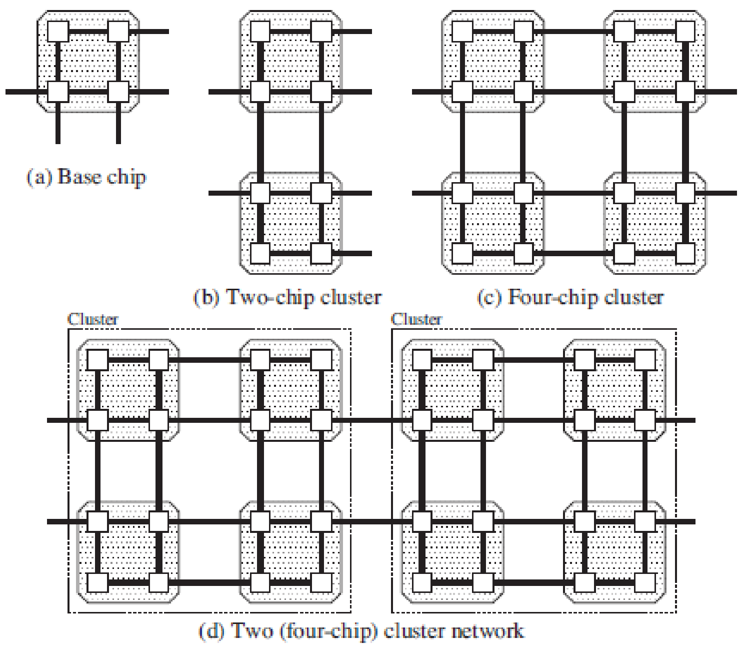

Fig. 2-11. Multi-Chip NoC and its configurations © 2012 IEEE.

Two routing methods proposed in this paper are gateway method and position-route method. The concept behind these methods is to divide systems into base chips (2x2 meshes) with each one can have total 5 off-chip links. Two or four base chips can make one cluster. Different clusters are connected with each other, as shown in figure (Fig. 2-11). Only one single component considered faulty, it can be link, router or chip. Each cluster contains routers which communicate with neighbor clusters, called gateways. If packet has to be routed outside of the current cluster, one of the gateways will be chosen. Otherwise it will be routed via in-cluster routing. Some cluster configuration can cause deadlock. In-cluster routing can tolerate single link or router fault. Other method mentioned called Position-Route method, uses faulty rings, which are rectangles which make alternative path around faulty component. Disadvantage of this method is that non-faulty (healthy) nodes will be masked out [1].

2.2. Multiple-Round Routing for 2D Network-on-Chip Meshes

Method by Fu, Han et al. called NMR-DOR helps to reduce amount of sacrificed non-faulty routers. It does not forbid any turn taken by the DOR routing. Main idea behind this algorithm is to take advantage from turn-legally intermediate nodes. It finds out turn legally intermediate node and skips non-legal node.

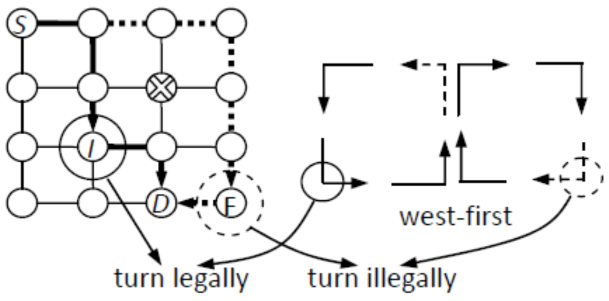

Fig. 2-12. NMR-DOR algorithm, Example of turn legally and turn illegally intermediate nodes. © IEEE 2009.

It uses adapted turn model only on turn legally nodes. At the beginning it is using DOR turn-model and when intermediate node occurs a new turn model is adapted, basically combining two turn models this way. On figure (Fig. 2-12) a basic behavioral scenario is shown. Authors proofed that NMR-DOR method is deadlock free. With low fault rate, a number of sacrificed routers can be kept fairy low. With an increase in faults, sacrificed router rate will expand rapidly. Authors claimed that with maximum of two virtual channels, increasing fault rate can be tolerated, but this is imposes extra cost for the system [2]. The other drawback of this approach is sacrificing healthy routers.

2.3. A Fault-Aware, Reconfigurable and Adaptive Routing Algorithm for NoC Applications

Valinataj and Mohammadi adaptive routing method, which uses only 2 virtual channels to provide fully adaptive and fault-tolerant solution, is proposed here. Authors say that methods with 3 or more virtual channels are not efficient in NoC due to power limitation and area overheads. Two methods are proposed in this work, one Basic-RAFT (The Reconfigurable, Adaptive and Fault-Tolerant method) which can tolerate only single faults and Main-RAFT which is pervious modification and can tolerate multiple faults. Idea behind this method is to route through reconfiguration which means calculate contour with routers which are surrounding broken link and then define new routing algorithm for each of those. Like shown in figure (Fig. 2-13) reconfigurations are made based on different fault cases. Cases C2 and C5 will cause deadlock if virtual channels are not used.

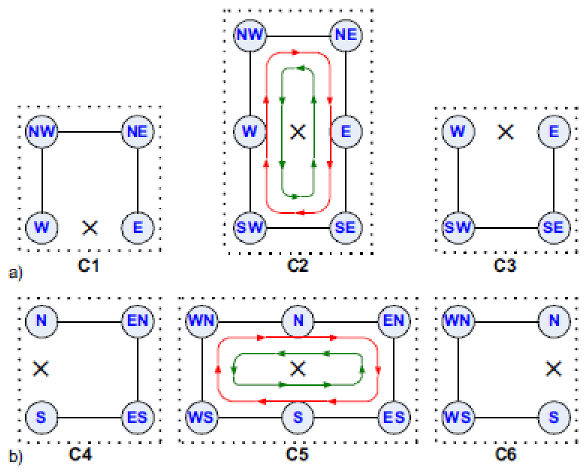

Fig. 2-13. Basic-RAFT different fault case reconfigurations © IEEE 2010.

Each router also has configurable register which stores local fault information. Router makes decision based on fault- and traffic information and selects shortest path to route packet. Main-RAFT uses similar ideology as Basic-RAFT but instead of 4 bit registers it needs more (i.e. 12 bit register). It can also tolerate faulty routers and/or regions but needs extra functionality in order to do that [3].

2.4. High Performance Fault-Tolerant Routing Algorithm for NoC-based Many-Core Systems

Algorithm from Ebrahimi, Daneshtalab and Plosila which is based on fully adaptive routing algorithm, core idea is to tolerate faults without losing performance. It requires collecting the fault information from only four direct neighbor nodes. It is deadlock free and uses two virtual channels, one for each dimension X and Y. This algorithm is able to tolerate faults, using only available shortest paths. However when packet is either on east-, west-, north- or south-ward, a non-minimal path will be used to bypass fault. This method is mainly about hopping between different virtual channels, packets will be routed first into one channel and when fault is bypassed then they will be switched to another. Algorithm has to do constant calculations for values to minimal destination between nodes and destination remaining [4]. This algorithm can tolerate up to 6 faults.

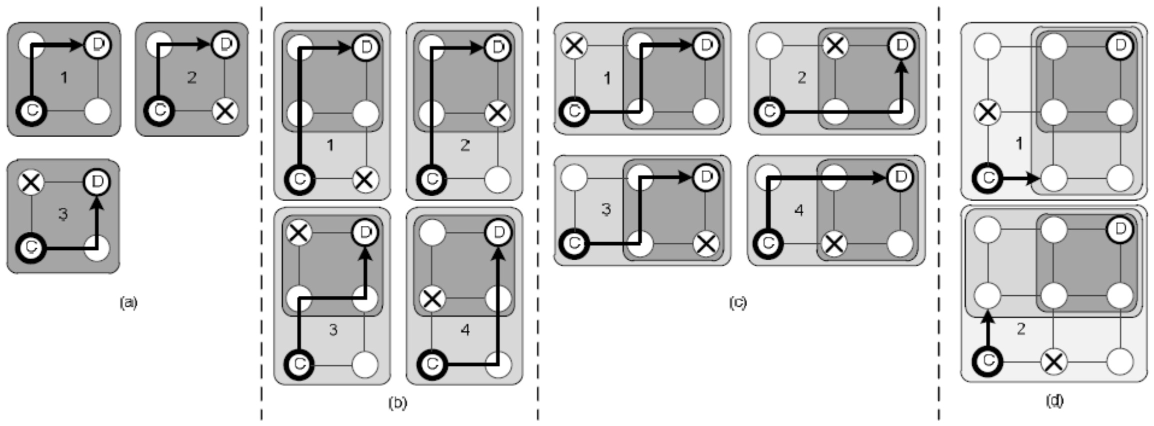

Fig. 2-14. Packet routing examples in algorithm, with fault cases © IEEE 2013.

2.5. Topology-Agnostic Fault-Tolerant NoC Routing Method

Wachter, Erichsen et al. promises features like topology-agnostic, full adaptivity, fault-tolerance and scalability. Full operational diagram for algorithm is shown on figure (Fig. 2-15). It has three steps to be made: seek, backtrack and clear & compute. Their algorithm initiates seek mechanism when lastly sent packet has not reached its destination. Seek will try to search for a new path and it will be done only once for each missed packet. Seek is based on hop counter and every router stores seek requests in their internal tables. Tables are not dependent on mesh size, instead table entries are only related to maximum number of simultaneous seeks. When target router is found, backtrack process will start. Backtrack packet will be sent towards source router, based on payload information. When backtrack packet is not reached then target node is unreachable or maximum number of simultaneous seeks are exceeded. Last step is to check for invalid turns (which can cause deadlock) and if any occurs, then the algorithm forces packet to use virtual channel(s) [5]. This approach uses routing tables and is not scalable.

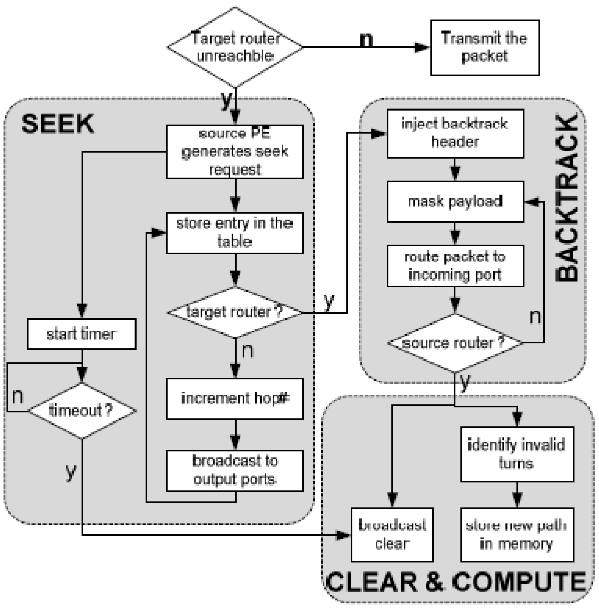

Fig. 2-15. Algorithm for Topology Agnostic routing method.

2.6. C-Routing: An Adaptive Hierarchical NoC Routing Methodology

Puthhal, Singh, et al. proposed algorithm is partially adaptive routing algorithm which prevents deadlock without using virtual channels. C-Routing combines XY and partly adaptive routing depending on source and destination nodes. It check routing table for a neighbor with minimum congestion and starts sending the packet. Neighbor node will be chosen either randomly or with minimal cost. Algorithm has two different parts: inter-cluster and intra-cluster routing. Depending on destination router cluster location, different routing techniques will be applied [6]. This algorithm also use routing tables and is not suitable for large networks.

Comparison between different routing algorithms is shown in Table 1. We compared different aspects of proposed algorithms: virtual channels, scalability, deadlock freeness etc.

Table 1. State of the Art comparison.

Method	**Number of Virtual Channels**	**Number of tolerable faults & Fault model**	**Deadlock freeness**	**Pros**	**Cons**
Dependable Routing in Multi-Chip NoC Platforms	0	1* link, router or chip.	Yes	+ Can distribute traffic. + Provides little HW solution.	- Can mask non-faulty routers.
A New Multiple-Round DOR Routing for 2D NoC Meshes	0 if low fault rate. If high fault rate 2 VC needed.	Many faulty links or routers	Yes	+ Can reduce number of sacrificed non-faulty routers.	- Decreases performance.
A Fault-Aware, Reconfigurable and Adaptive Routing Algorithm for NoC Applications	2	Basic-RAFT for single- and Main RAFT for multiple faulty links.	Yes	+ Supports irregular topologies. + Low cost + Able to balance traffic + Does not disable any faulty component.	- Implementation complexity
High Performance Fault-Tolerant Routing Algorithm for NoC-based Many-Core Systems	2	Up to 6 faulty nodes.	Yes	+ Supports High Performance with 98% reliability.	- Not scalable.
Topology-Agnostic Fault-Tolerant NoC Routing Method	2 for mesh topology more for other topologies.	Many links and/or routers.	Yes	+ Topology-Agnostic + Provides decent HW realization.	- Uses tables, not scalable.
C-Routing: An Adaptive Hierarchical NoC Routing Methodology	0	Unknown	Yes	+ Can balance traffic.	- Partially adaptive. - Uses routing tables.. - No information about fault tolerance

** Router contains 4 links and chip contains 4 routers.*

In conclusion, there are flaws and bottlenecks which specially standing out from the comparison table (Table 1). We can classify these flaws and drawbacks in the following groups:

- **fault tolerance**:
    - The algorithms are either indistinct or cannot tolerate satisfied number of faults.
    - Also there are serious issues with some of the methods which can sacrifice healthy nodes.
- **Scalability**:
    - In some methods maximum network size is predefined.
    - Some methods use **routing tables**, which will decrease both scalability and performance.
- **Virtual channels**: While keeping VC amount low they can be beneficial, however implementing them will require extra cost.
- **Performance**: this property of NoC is also an issue, especially with those complex and table based methods. There was only one method [4] which contributed into high performance while having problem with scalability.

None of the methods proposed can address all those problems, so it is necessary to investigate a method to coupe with these problems all together. Our motivation for developing new method was to avoid these flaws and have a more flexible solution which can be applied in different cases.

3. Methodology

Our goal is to introduce a method which will enhance the routing algorithm with the following properties:

- it should be able to tolerate any number of faulty links
- the amount of information stored in the each node should be limited and does not grow with increase in size of network
- the amount of communications on network links should be limited
- the adaptivity of the routing algorithm should be preserved
- our method should cover all adaptive minimal path routing algorithms

In the following sections we will discuss the algorithm description and implementation scheme.

3.1. Limitations and assumptions

In this project we assume:

- The Topology of the network is Mesh
- We have packet oriented transfer model due to its efficient resource management
- Our chosen routing algorithm is Dy-XY
- Our network has a worm-hole flow control
- Each input port has 2 virtual channels.
- Our chip has an efficient testing mechanism built in it, for example BIST [12].

3.2. Dependable routing concept

While considering different papers and proposed methods we came up on the conclusion that to achieve and keep high adaptivity value, routing algorithm needs some enchantments. Our solution is a mixture of using virtual channels along with some routing information tables. The main idea is to back propagate fault information from the node with the faulty link to all routers which depend on it. This information can be stored in a vector which will be stored on every router output channel. Each vector will contain all the necessary fault information about the mesh. If the router wants to make a routing decision, the routing algorithm can pre-check router “table” and use provided information to act accordingly.

Each router should know only local information about their neighbors. There is no point to back propagate all the information to every single router in mesh. With huge size mesh structure it takes time while not being beneficial enough. The back propagation will start if any fault occurs on a router link with the assumption that other links of the router are healthy nodes. A rectangle will be calculated based on information about working and faulty links. After all calculations are done, final list will be optimized due to limited size of memory in routers. Optimizing aims to merge rectangles and remove duplicates. Rectangle merging will be done using different mathematical methods, including graph theory. Finally whole list will be back propagated to neighbor nodes.

3.2.1. System Health Map

Health map gives a good synopsis over system health status. It shows which links and/or nodes are faulty and which are not. From there information about system health status in general can be gathered.

Main problem with source routing based local accumulation methods proposed in papers are with fault tolerance [1], [4]. The proposed methods are either having limitations on the number of the faults that they can tolerate or they are not scalable. This is actually problematic and our motivation with new method is to avoid such things to happen. Our method’s aim is to improve this property and tolerate more faults in the network. Concerning system health map is still required but we have to set more flexible boundaries and toleration level before judging system either to be healthy or not healthy.

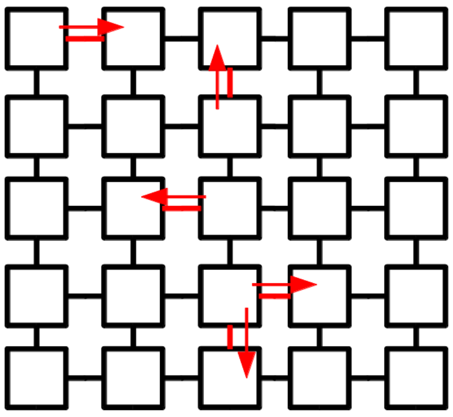

Fig. 3-16. A health map

In our health map (Fig. 3-16), Gate-Level Fault model have been used, based on outgoing links in network. If a link is broken, it is considered to be faulty. Stuck-at faults, fault redundancies and delayed faults were not considered [14]. Parallel fault occurrences, faults in the router control unit etc. are not considered.

3.2.2. Reachability Attributes

In our method we will be looking into **non-reachability**. We will store information about regions that are not reachable via each list and we call the non-reachable rectangles.

The main issue is; while it is good to have global information about whole network, in the end there is a lot of information which 1) requires huge memory size 2) consists of data which is not needed at that node.

Instead of doing global reachability analysis, in our approach it will be done locally in routers. This way routers near faulty location have more information about faults but routers further away from it will have very little or no information at all. This helps to save memory and keeps only necessary information so performance will be also better.

Adaptivity has also an important part in reachability, especially when concerning fault tolerance. Adaptive algorithm can tolerate more faults than deterministic.

3.2.3. Local Information Accumulation

As some of the research studies noted in previous chapter, to achieve highly adaptive degree for the algorithm, some local information is needed which should be stored in some tables in the node. If tables are too big then it consumes more buffer space therefore such solution will be much more expensive. Another problem comes from routing via routing table itself. If tables have huge amount of information in it, when performing full analysis it will have impact on routing performance.

In our approach each router gets only limited information about its neighbor nodes. If a link breaks then these nodes (or areas) impacted by these faults can be avoided, which increases the adaptivity. The only information that is actually needed is description of unreachable area. While 2D mesh is basically a matrix with x and y coordinates it is possible to describe area with only few bits of information, while using basic mathematic calculations. Another important constraint is about table size which should be fixed. Table size should be as small as possible, because of NoC's limited amount of resources.

Using area and table optimization (see section 3.3.3) techniques enables us to reduce and keep table sizes fairy small. But we have to understand that some of the information can be lost by applying some of these techniques; if the tables are full and records can't be merged then we potentially can lose some healthy nodes.

3.3. Fault propagation for dependable routing

Our proposed method can be applied using any deadlock free, minimal path routing algorithm and helps to recover some of its lost adaptivity. Basic idea behind this method is, when a link breaks then a rectangle describing unreachable area will be generated and back propagated to neighbor nodes. Rectangle propagation will continue until there are no more routers which will depend on it.

3.3.1. Rectangle

What is a rectangle? It is a representation of area which is unreachable via current link. Idea behind rectangle representation is that this needs to be easily understandable and compact enough for the buffers to store. We try to propose a solution which is cost efficient as possible but we found that some constraints had to be made. To achieve high adaptivity and fault tolerance in NoC, some local information is needed. Our rectangle representation is based on Cartesian location representation. We present each rectangle by its upper left and lower right corner location (Fig. 3-17).

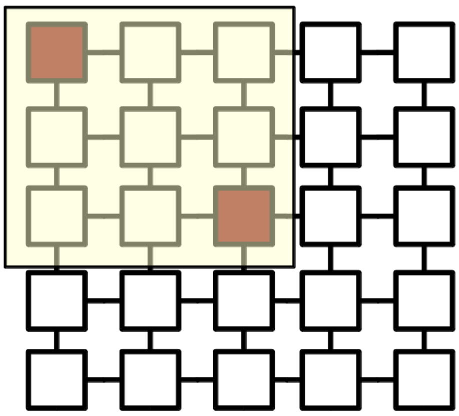

Fig. 3-17. Our rectangle presentation

3.3.2. Rectangle list

Each node's outgoing link has one table associated to it, which holds predefined fixed number of entries which is called rectangle list. This table stores rectangles that are unreachable via current link. Routing algorithm has to check each link's rectangle list to get information about reachability on current path. Any node in faulty area (rectangle list) can be avoided and a new path around it would be created.

To keep rectangle list minimal, some minimization and optimization functions has to be applied. It is normal that duplications are not allowed and will be removed but merging rectangles. Only optimized rectangle list will be back propagated to neighbors.

3.3.3. Rectangle optimization

Optimization is critical step needed to be performed in order to achieve good results. It is necessary for meeting buffer memory requirements and keeping costs low.

Following cases are the main three optimization methods, which are preformed to optimize rectangle list. First two optimization methods will be performed in any case but third one is only for extreme cases when either rectangle list is full or no more memory can be shared.

- Merging: if rectangles are right beside each other and have border with same width or height, or one rectangle is located inside the other one, we can merge them. Figure (Fig. 3-18) shows situation where two rectangles are merged together.

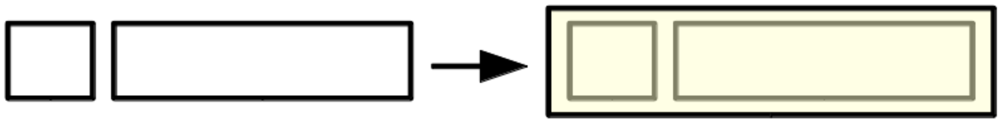
Fig. 3-18. Rectangle optimization, Expansion.

- Lossless expansion: this optimization task is preformed to make list search process faster. When two rectangles are beside each other and they can’t be merged, smaller rectangle will be expanded into another. This will increase rectangle area, while not sacrificing any healthy nodes. When routing algorithm has to search the rectangle list, it will have higher probability to hit an entry containing the destination node, because there are several rectangles with similar information. This gives boost in performance. On figure (Fig. 3-19) a basic idea behind this optimization is shown. This algorithm gives more benefit if we are dealing with large rectangle lists.

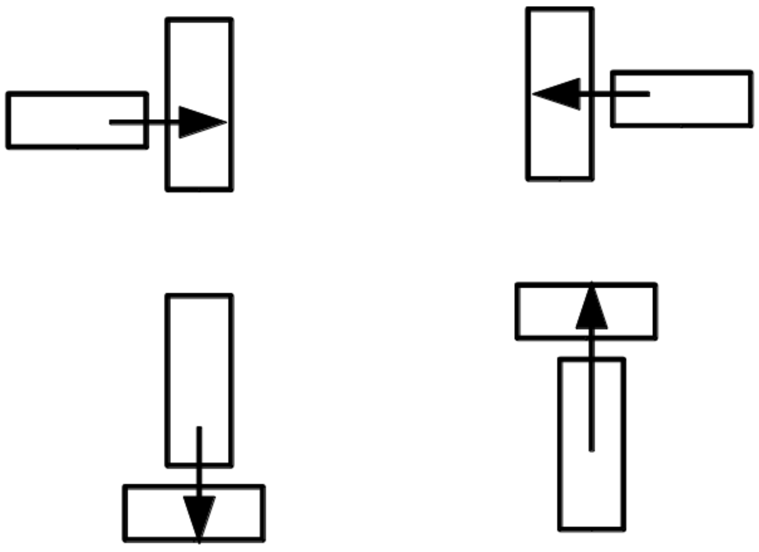
Fig. 3-19. Rectangle optimization, lossless merging.

- Lossy expansion: this method, combines several smaller rectangles into bigger ones. As can be realized from the methods name, we will lose some healthy.

This method is only used when it is necessary, when router memory is full or rectangle list size limitations are met.

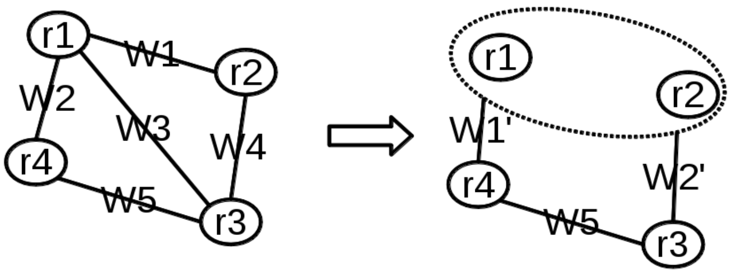

Fig. 3-20. Lossy merging, a graph representation.

At first, a graph will be made where each node on the graph represents a rectangle. Weights on graph links will show number of cells which will be lost when merging occurs between two rectangles. Nodes with lowest link weight will be combined into a cluster (Fig. 3-20). Process will continue until router memory requirements are met. On figure (Fig. 3-21) is shown, how much information actually can be lost by combining two rectangles into bigger one.

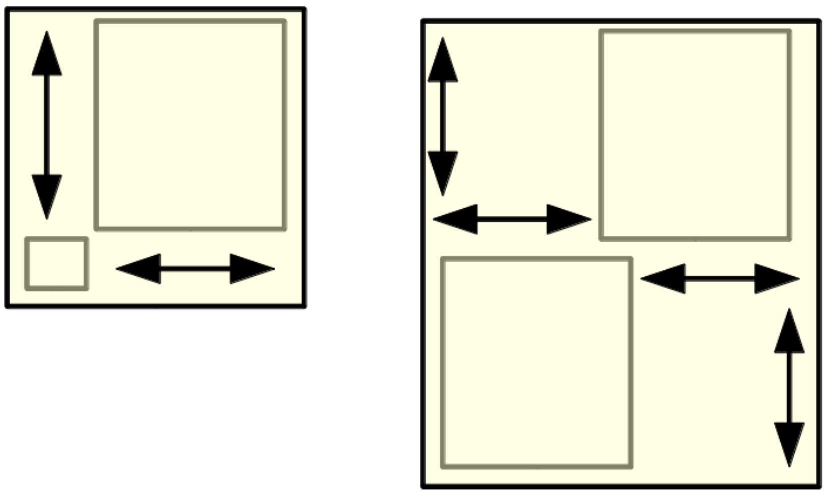

Fig. 3-21. Lossy merging, making bigger rectangles.

3.3.4. Back propagation

Backtracking is an important component in our approach which takes care of sending information received from neighbors, to neighbor nodes which are dependent on that information. If routing component has information about a routing direction that leads to a dead end with respect to a concrete path destination, then it can avoid this routing decision completely and automatically calculate new and presumably routing decision.

After fault list is corrected and optimized then it will be propagated back to all neighbor nodes. Propagation will be done depending on direction from where fault occurred. At first rectangle list will be back propagated from node with broken link, to its neighbor nodes. If some of them have dependences on path where fault occurred their lists will be updated. In theory neighbors of neighbors are not affected (except for the case that, the faulty region that they receive is on the node's row or column) by fault because they can route towards other paths and therefore they do not have to know unneeded information. Our back propagation algorithm is designed for minimal path adaptive routing. Back propagation scenario in our method is described below.

In our algorithm, for any given adaptive minimal path routing algorithm, we define:

$\forall Dir_i$: *list of reachable rectangles {R_i}*

R_i: *old list of rectangles in Direction* i

R_j: *list of rectangles in Direction* j

$R_{i,new}$: *new list of rectangles in Direction* i

r_{ik}: *rectangles in* i*direction*

r_{jk}: *rectangles in* j*direction*

R_{final}: *final set of rectangles (this is how the node sees the network)*

To describe our algorithm for fault propagation, we present its pseudo-code representation as follows:

```
∀outgoing direction $i$(Dependant on routing
algorithm):
 $R_{i,new}$= ∅
∀ entries $r_{ik}$in  $R_i$(forbidden rectangle list):
        $r_{ik,new}$= $r_{ik}$-  part of $r_{ik}$which can be reached
via other directions
        $R_{i,new}$=  $R_{i,new} \cup r_{ik,new}$
       ∀outgoing directions $j$> $i$:
             ∀ entries $r_{jk}$in list $R_j$:
                     $R_{i,new}$  =  $R_{i,new} \cup (r_{ik} \cap r_{jk})$
minimize $R_{i,new}$.*
∀outgoing direction $i$: compute $R_{final}$as the set union
of all $R_{i,new}$.
  minimize$R_{final}$.*
  If $|R_{final}|$ > # link entries, then:
compress $R_{final}$
  back propagate $R_{final}$
```

Algorithm implementation is done a bit different, in Python language. The network consists of two dimensional matrices and a health map. In back propagation algorithm it is always assumed that network is faulty free at the beginning. Simultaneous faults occurrence are not possible, and faults will be checked and processed one at a time. While back propagating, new faults are not considered. Propagation process is done using 2 stacks. First stack contains all the faults. For each fault, back propagation will be done separately. Second stack holds nodes which require propagation rectangle. The propagation process will be handled like a tree structure; we start from one fault, and follow the neighbors until there is no more affected nodes and then we start from a new branch and investigate it top-down. The details of implemented method is described below.

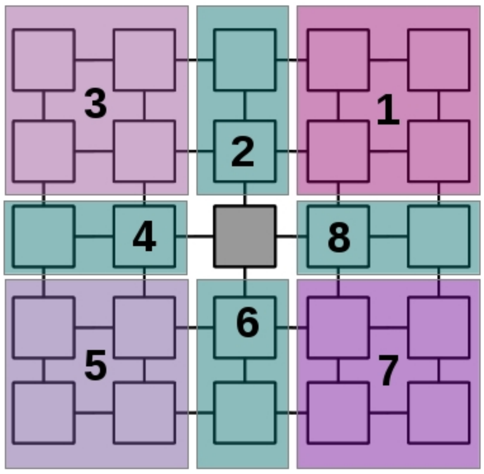

Fig. 3-22. Different areas in back propagation.

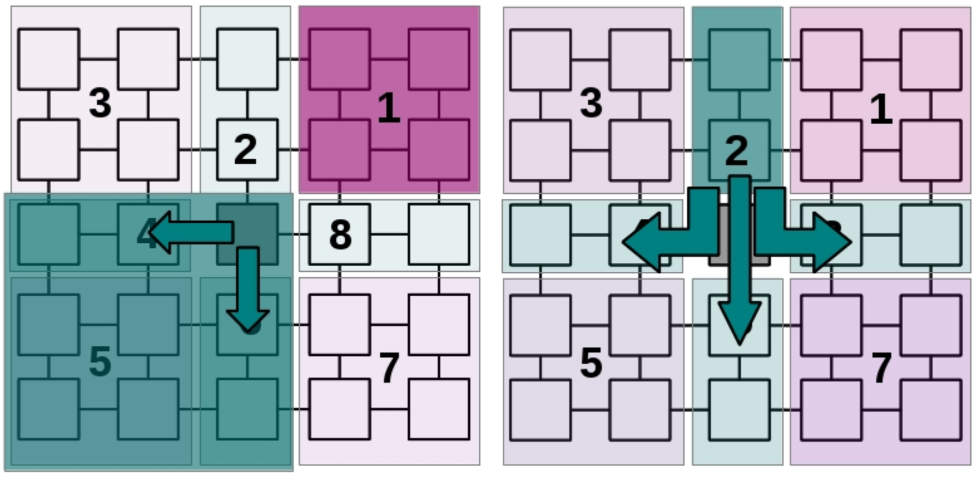

Fig. 3-23. Back propagation examples on different mesh areas.

As you see on figure (Fig. 3-22) 8 different areas are shown on the mesh, from node point of view. A rectangle in each area has different back propagation scenario. Areas 2, 4, 6 and 8 are acting similarly while others are different. If rectangle is in area 2, 4, 6 or 8, it has to be back propagated to all its neighbors except for the neighbor on that side (Fig. 3-23, right). For example rectangle in area 2 has to be back propagated to all node neighbors in W, E and S directions. Now if rectangle is in other areas (so called odd areas) propagation has some differences. For example on figure (Fig. 3-23, left) rectangle in area 1 has to be back propagated to node E and S neighbors only because other neighbors are not dependent on it. Things are getting more complicated if rectangle covers multiple areas. In that case intersection between different rectangles in area has to be found and back propagation will be done only with intersections. To find out actually what rectangle falls in odd areas we have to make intersections of rectangles in two adjacent lists with the half-plane. As an example to find rectangles located on North-East region, we make intersection between each two rectangles on North and East lists and with the north east half plane (see figure Fig. 3-24). This process should be performed for each half plane in odd areas.

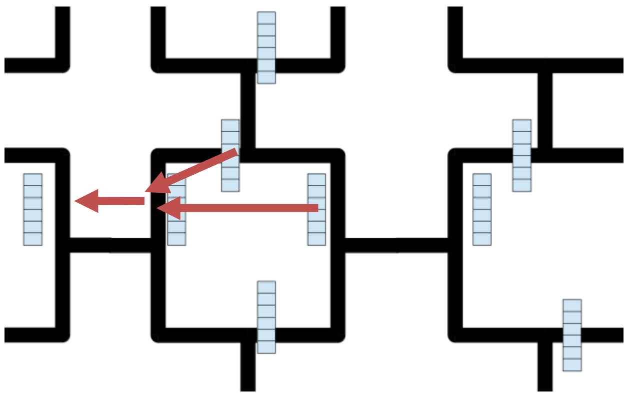

Fig. 3-24. Case when back propagation is needed.

After intersections have been performed, we can initiate back propagation process. Figure (Fig. 3-25) represents basic back propagation situation. Node marked with red is unreachable due to faulty link. Its west neighbor eastern link is faulty so each of its direct neighbors will be getting rectangle propagated to them. Now each of those nodes will check do they have alternative paths to red node or not. If they do not have other paths then they will propagate back to their neighbors. Nodes on the figure with red arrows need to propagate information to their neighbors because they do not have any alternative path and packet will be misrouted through them. If something is back propagated to node which has faulty outgoing link then its list needs to be updated and back propagated to its neighbors again.

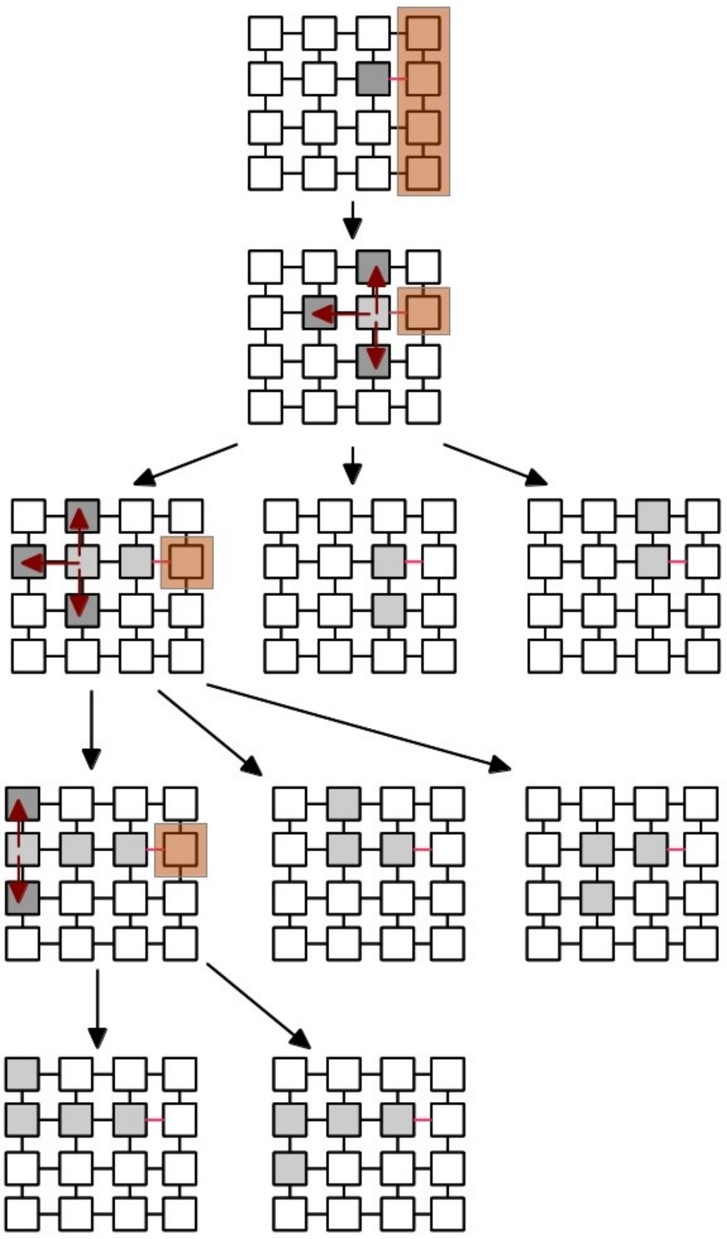

Fig. 3-25. A back propagating scenario.

3.4. Elaborative examples

In order to demonstrate better our method behavior, some examples had to be made. In this section some interesting examples are presented, these show our method strong points. DyXY routing scheme (Fig. 1-9) will be used in simulation with an exception that only one set of rectangle lists will be used on both planes. “ListOfSuccessors” is a list which contains all the nodes which are dependent on fault. On each node a list of potential neighbors should be calculated, which will be then visited and rectangle will be propagated to them.

3.4.1. Example 1 – Short wall of broken links

In first example on 3x3 mesh network has two faulty links; North link on nodes 3 and 4. Health map is shown on figure (Fig. 3-26). This example shows the case when list information is needed to be updated. A scenario will be presented next.

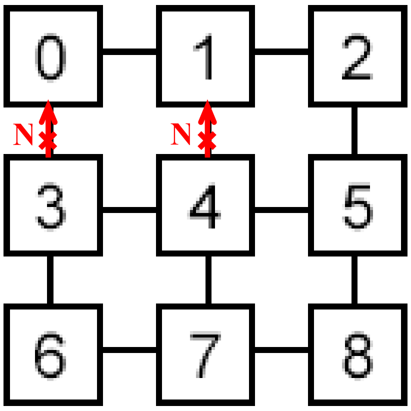

Fig. 3-26. Example 1, health map.

```
Starting from Faulty node: 3
Nodes initial rectangle lists:
     north list: [[0, 2]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors [4, 6]
----->SuccessorsList: [4, 6]
```

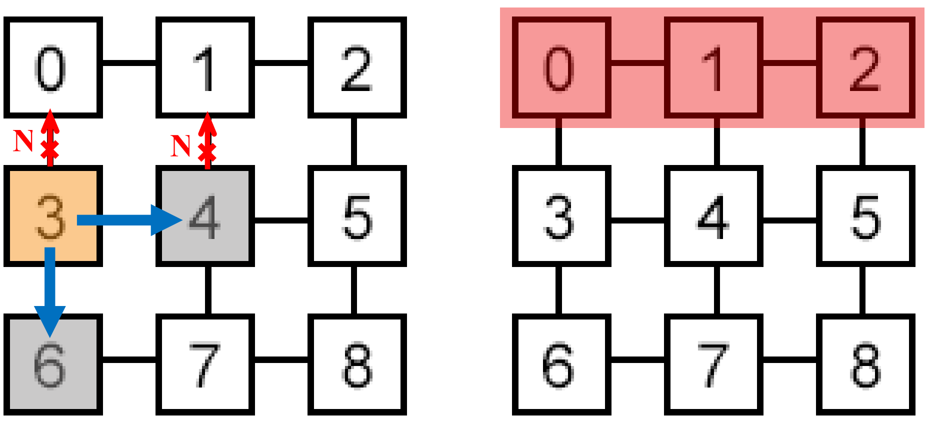

Fig. 3-27. Example 1, propagation case 1.

Node 3 has a broken link on north direction. A rectangle [0, 2] to be added its north list (Fig. 3-27, b) which is the whole half plane in the north of node 3. Intersection between North plane [0, 0] and rectangle [0, 2] is [0, 0] which will be back propagated to nodes 4 and 6. (Fig. 3-27, a) according to our plane presentation (Fig. 3-22).

```
chosen node: 6
Nodes initial rectangle lists:
     north list: [[0, 0]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors [7]
----->SuccessorsList: [4, 7]
```

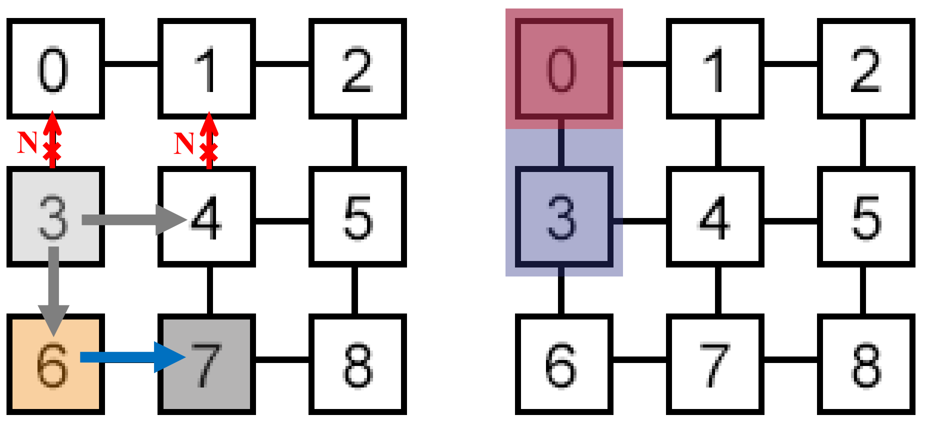

Fig. 3-28. Example 1, propagation case 2.

Next node 6 will be selected from stack. Intersection between its North plane [0, 3] and rectangle [0, 0] is [0, 0] (Fig. 3-28, b) which will be propagated back to node 7 (Fig. 3-28, a).

```
chosen node: 7
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: []
     West list: [[0, 0]]
ListOfSuccessors []
----->SuccessorsList: [4]

chosen node: 4
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: []
     West list: [[0, 0]]
ListOfSuccessors []
```

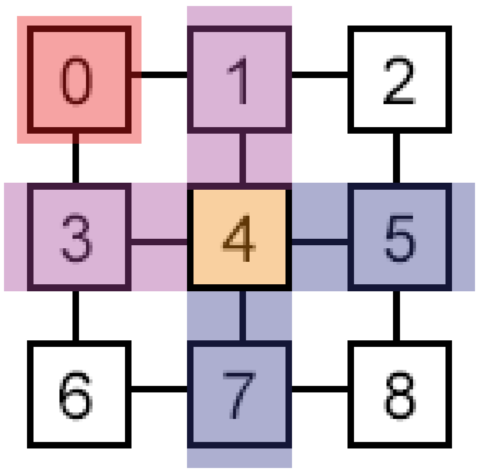

Fig. 3-29. Example 1, no back propagation presentation.

Since next elements in stack, nodes 7 and 4, do not have any intersections between lists, no further propagation is needed. For exmple on figure (Fig. 3-29) is a case for node 4. It has rectangle [0, 0] in its west list and nothing in its north list. According to our plane presentation, rectangle is on node 4 North-West plane which means back propagation has to go towards east and south (blue) and there hase to be intersection between rectangles on north list and west list (purple).

```
Starting from Faulty node: 4
Nodes initial rectangle lists:
     north list: [[0, 2]]
     South list: []
     East list: []
     West list: [[0, 0]]
ListOfSuccessors [5, 3, 7]
----->SuccessorsList: [5, 3, 7]

chosen node: 7
Nodes initial rectangle lists:
     north list: [[1, 1], [0, 0]]
     South list: []
     East list: []
     West list: [[0, 0]]
ListOfSuccessors [8, 6]
----->SuccessorsList: [5, 3, 8, 6]
```

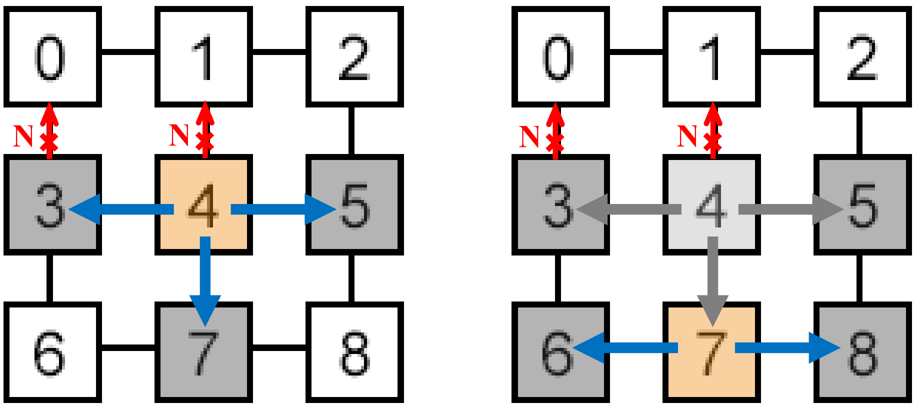

Fig. 3-30. Example 1, propagation case 3.

Here we have reached at the end of one branch in our tree, so we start with another faulty node. Next fault in the network is on node 4's north link. Rectangle [0, 2] will be added into node 4's north list. Rectangle [1, 1] will be calculated and back propagated to nodes 3, 5 and 7 (Fig. 3-30, left). Node 7 will be selected from the stack, rectangle [1, 1] will be added to its north list and back propagated to nodes 6 and 8.

```
chosen node: 6
Nodes initial rectangle lists:
     north list: [[0, 0]]
     South list: []
     East list: [[1, 1]]
     West list: []
ListOfSuccessors []
----->SuccessorsList: [5, 3, 8]

chosen node: 8
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: []
     West list: [[1, 1], [0, 0]]
ListOfSuccessors []
----->SuccessorsList: [5, 3]

chosen node: 3
Nodes initial rectangle lists:
     north list: [[0, 2]]
     South list: []
     East list: [[1, 1]]
     West list: []
ListOfSuccessors [6]
----->SuccessorsList: [5, 6]

chosen node: 6
Nodes initial rectangle lists:
     north list: [[0, 0], [1, 1]]
     South list: []
     East list: [[1, 1]]
     West list: []
ListOfSuccessors []
----->SuccessorsList: [5]

chosen node: 5
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: []
     West list: [[1, 1], [0, 0]]
ListOfSuccessors []
```

Next in the stack is node 6 which rectangle [1, 1] will be added to its east list. Similar process will happen to other nodes in the stack: 8, 5 then 3 which will be propagated to 6 also. No more further propagation is needed because same idea as shown on figure (Fig. 3-29) applies.

It was just an easy and basic description about algorithm's behavior. The main ideas works on much complex cases as well.

3.4.2. Example 2 – An island

Our second example showing an interesting case, we call it island case. Island is the situation where one node or area is inaccessible, because it is surrounded by faulty links. In our case on a 3x3 network, upper right corner (node 2) is an island because node 1 outgoing east link and node 5 outgoing north links are faulty.

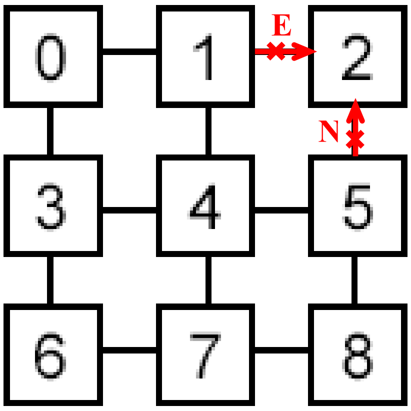

Fig. 3-31. Example 2, an island case via health map representation.

```
Starting from Faulty node: 1
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: [[2, 8]]
     West list: []
ListOfSuccessors [0, 4]
----->SuccessorsList: [0, 4]
```

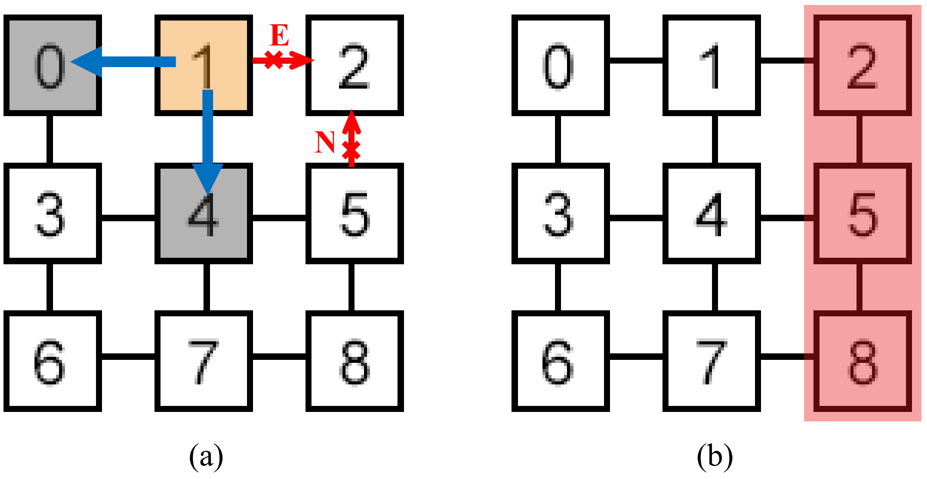

Fig. 3-32. Example 2, propagation case 1.

At the beginning a fault on node 1 east link is dectected and rectangle [2, 8] to be added into its list (Fig. 3-32, b). Propagations towards west (node 0) and east (node 4) has to be made (Fig. 3-32, a). Rectangle which will be propagated is [2, 2], because node 4 rectangle [2, 8] in east list and its East plane [2, 2] intersection is [2, 2]. Nodes 4 and 0 will be added to the stack.

```
chosen node: 4
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors []
----->SuccessorsList: [0]

chosen node: 0
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors [3]
----->SuccessorsList: [3]
```

```
chosen node: 3
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors []
```

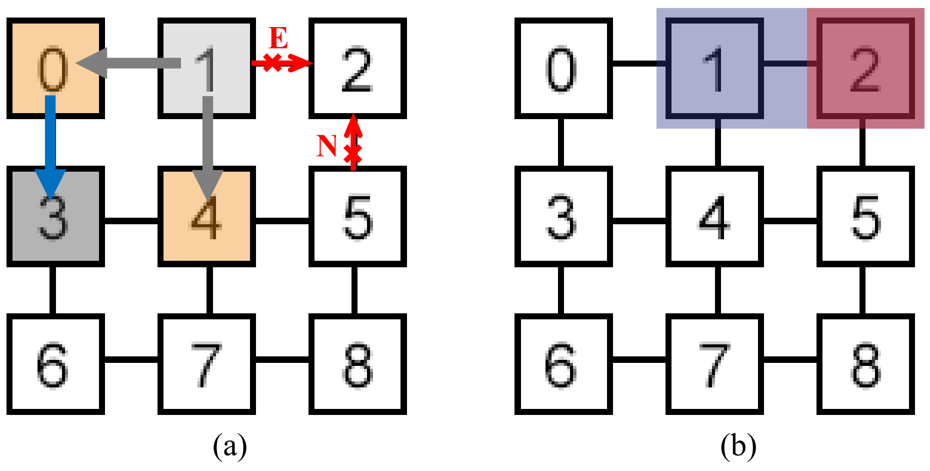

Fig. 3-33. Example 2, propagation case 2.

Node 4 and 0 are next in stack. For node 4 no further propagation is required, but node 0 will be propagating back to node 3 (Fig. 3-33, a). Rectangle [2, 2] will be propagated because intersection between node 0's East plane [1, 2] and east list rectangle [2, 2] is equal to [2, 2] (Fig. 3-33, b). Node 3 will not propagate this list further since it has no entry on its east list.

```
Starting from Faulty node: 5
Nodes initial rectangle lists:
     north list: [[0, 2]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors [4, 8]
----->SuccessorsList: [4, 8]
```

```
chosen node: 8
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors [7]
----->SuccessorsList: [4, 7]

chosen node: 7
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors []
----->SuccessorsList: [4]

chosen node: 4
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors [3, 7]
----->SuccessorsList: [3, 7]
```

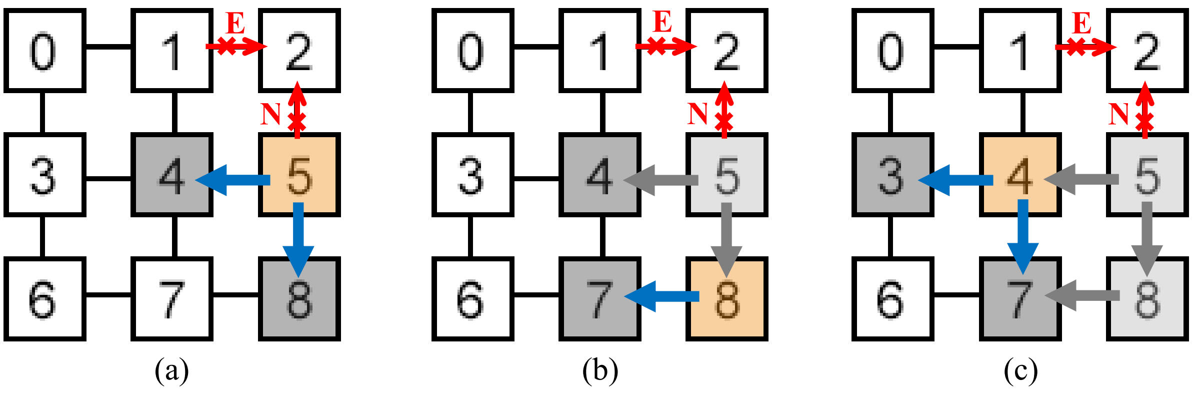

Fig. 3-34. Example 2, propagation case 3.

Faulty north link on node 5 will be discovered next. Back propagation to nodes 4 and 8 will be made (Fig. 3-34, a). Node 8 will be chosen from stack and propagation towards node 7 will be performed (Fig. 3-34, b). Next in the stack is node 7, which however do not have to propagate anything because its north link is still empty. Node 4 will be chosen and propagates towards node 3 and 7 (Fig. 3-34). Rectangle [2, 2] will be propagated, which is intersection between node 4 north and east lists.

```
chosen node: 7
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors [6]
----->SuccessorsList: [3, 6]

chosen node: 6
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: [[2, 2]]
     West list: []
----->SuccessorsList: [3]

chosen node: 3
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors [6]
----->SuccessorsList: [6]

chosen node: 6
Nodes initial rectangle lists:
     north list: [[2, 2]]
     South list: []
     East list: [[2, 2]]
     West list: []
ListOfSuccessors []
```

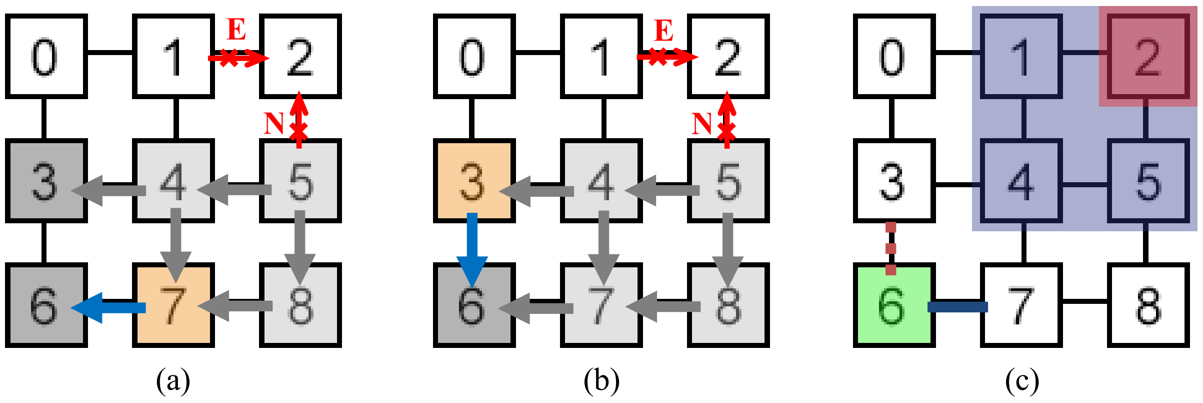

Fig. 3-35. Example 2, propagation case 4.

Node 7 will be chosen and rectangle [2,2] will be propagated towards node 6 (Fig. 3-35, a). Node 6 will be chosen from stack but nothing will be backpropagated because no intersection between node 6 east list and north list can be made. (Fig. 3-35, c). Next in the stack is node 3 which will be chosen and its propagating back to node 6 (Fig. 3-35, b). Lastly, a node 6 is in the stack, but again nothing well be back propagated.

3.4.3. Example 3 – Corners

Third example is showing a case where links are broken in such way, that they compose a complex path. What we want to show here is that in many cases on such network configuration, routing algorithms will be sacrificing healthy nodes. Our method will not do it, instead it will mask out only unreachable nodes.

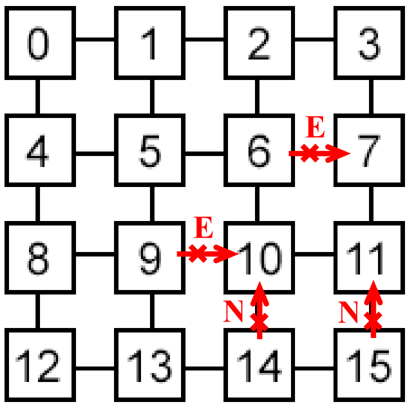

Fig. 3-36. Example 3, health map representation.

```
Starting from Faulty node: 6
chosen node: 5
Nodes initial rectangle lists:
     north list: []
     South list: []
     East list: [[7, 7]]
     West list: []
ListOfSuccessors [4, 9, 1]
----->SuccessorsList: [4, 9, 1]
```

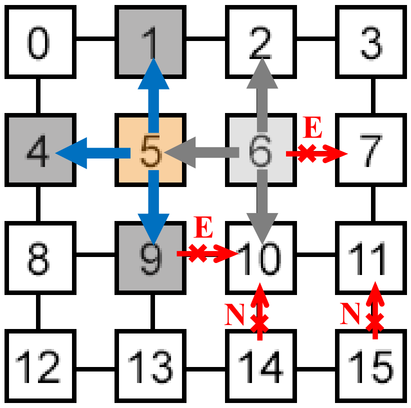

Fig. 3-37. Example 3, propagation case 1.

Starting from node 6 which has faulty east link. Back propagation will be done similar way like in pervious examples. Node 6 will be getting unreachable rectangle [3, 15] in its east link. Intersection which will be back propagated to nodes 1, 4 and 9, is [7, 7] (Fig. 3-37).

```
Starting from Faulty node: 9
Nodes initial rectangle lists:
     north list: [[7, 7]]
     South list: []
     East list: [[2, 15]]
     West list: []
ListOfSuccessors [8, 13, 5]
----->SuccessorsList: [8, 13, 5]
```

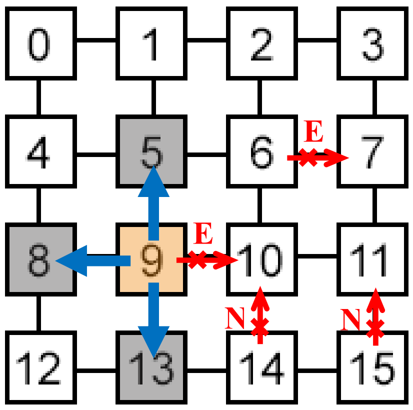

Fig. 3-38. Example 3, propagation case 2.

Also the case with fault on node 9 east link, will be handled similar way. (Fig. 3-38) Node 9 unreachable rectangle via east link is [2, 15]. Rectangle [10, 11] will be back propagated to nodes 5, 8, 13 lists.

```
Starting from Faulty node: 14
Nodes initial rectangle lists:
     north list: [[0, 11]]
     South list: []
     East list: []
     West list: []
ListOfSuccessors [15, 13]
----->SuccessorsList: [15, 13]
```

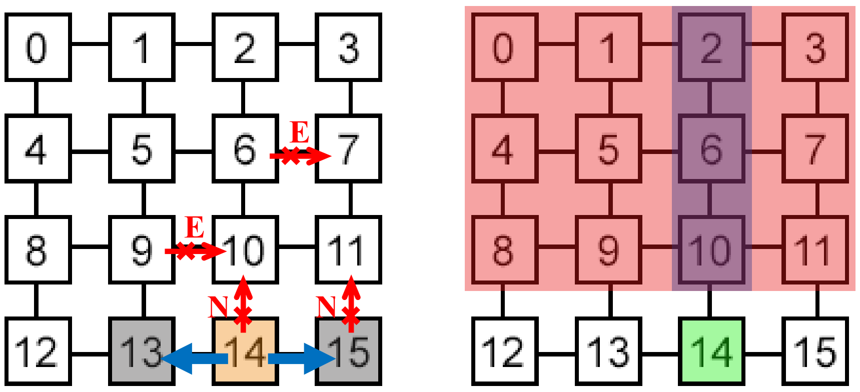

Fig. 3-39. Example 3, propagation case 3.

Third case with node 14 which has faulty north link is quite different. First of all, node 14 will get rectangle [0 ,11]. Rectangle [2, 10] will be back propagated, because it is the intersection between [0, 11] and [2, 10] (north plane)(Fig. 3-39, right). [2,2] will be back propagated to nodes 13 and 15 (Fig. 3-39, left).

```
chosen node: 13
Nodes initial rectangle lists:
     north list: [[7, 7], [10, 11]]
     South list: []
     East list: [[2, 10]]
     West list: []
ListOfSuccessors [12]
----->SuccessorsList: [15, 12]
```

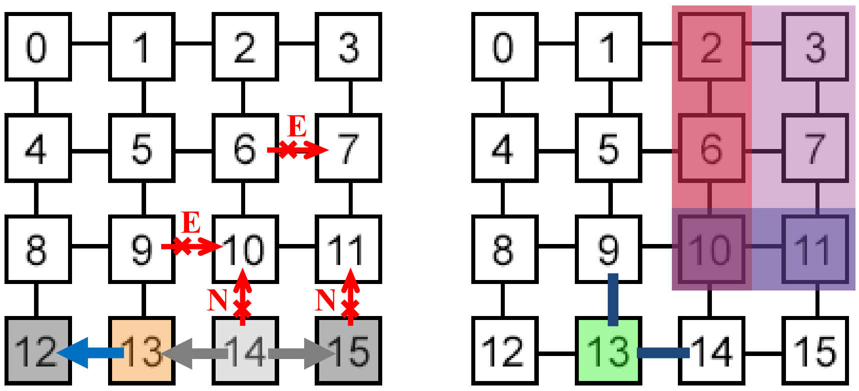

Fig. 3-40. Example 3, propagation case 4.

When node 13 is selected, back propagation with rectangle [10, 10] will be propagated to node 12 only (Fig. 3-40, left). Unreachable rectangle in node 13 north list [10, 11] is on its North-East plane (purple). Since east list also has rectangle [2, 10] which is on the same plane, their intersection is [10, 10] (Fig. 3-40, right) will be back propagated to west (according to planes). Futher propagation from nodes 12 and 15 will not happen because they do not have any intersections between rectangles.

```
Starting from Faulty node: 15
Nodes initial rectangle lists:
     north list: [[0, 11]]
     South list: []
     East list: []
     West list: [[2, 10]]
ListOfSuccessors [14]
----->SuccessorsList: [14]

chosen node: 14
Nodes initial rectangle lists:
     north list: [[0, 11]]
     South list: []
     East list: [[3, 11]]
     West list: []
ListOfSuccessors [13]
----->SuccessorsList: [13]

chosen node: 13
Nodes initial rectangle lists:
     north list: [[7, 7], [10, 11]]
     South list: []
     East list: [[2, 10], [3, 11]]
     West list: []
ListOfSuccessors [12]
----->SuccessorsList: [12]

chosen node: 12
Nodes initial rectangle lists:
     north list: [[7, 7], [10, 11]]
     South list: []
     East list: [[10, 10], [7, 7], [11, 11]]
     West list: []
ListOfSuccessors []
```

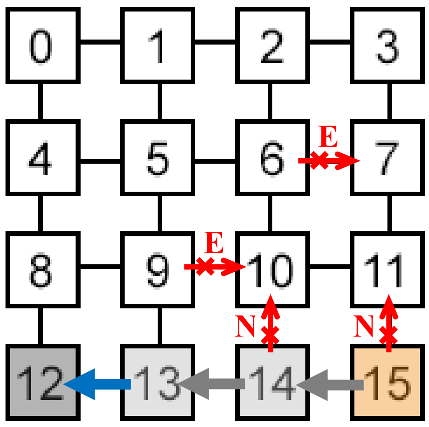

Fig. 3-41. Example 3, propagation case 5.

Last interesting case is with node 15 faulty north link. First node 15 north link will get rectangle [0, 11]. Intersection between its East plane [3, 11] and rectangle [0, 11] is [3, 11] which will be back propagated to node 14, 13 and 12 (Fig. 3-41). From node 14 back propagation goes to node 13 because rectangle on its east list [3, 11] has intersection between its north list rectangle [0, 11]. From node 13 also back propagation to node 12 takes place. This case is more special than pervious because two rectangles will be pack propagated. First rectangle [7, 7] which is intersection between north [7, 7] and east [3, 11] and second rectangle [11, 11] which is intersection between north [10, 11] and east [3, 11]. From node 12, no further back propagations will be made.

4. Results

Our method can discover unreachable areas and routing will not be made if node is in that area. In case we need to send a packet to a node in unreachable area, remapping of the algorithm is required. By implementing this method we can increase reachability to 1; if node is reachable then local neighbors will have this information already. If node is on an island -an area which is surrounded by faulty nodes- then information will be back propagated to all over the network. In this way, sending packets to unreachable destinations cannot happen. This property also provides us with a tool to recover the lost adaptivity for the routing algorithm.

Our method helps to achieve higher level of fault tolerance. For testing our algorithm we used our own environment, described in pervious chapter. Different tests using different mesh sizes were performed according to DyXY routing algorithm. The back-propagation process takes longer time when either network size or number of faults increase.

In the end our concept indeed imposes some extra cost to system. Increasing cost comes from each individual outgoing link which has to have a fixed table in it. But it also should give noticeable adaptivity boost which helps to increase reliability and fault tolerance in the chip. It also should be more flexible and scalable because it is not dependent on routing algorithm and fault rate. In some cases some information can be lost due to lossy merging optimization.

5. Conclusions & Future Work

Our methodology described in this to develop a method which is scalable, routing algorithm independent, fault tolerant (even with multiple faults). Our method is based on the idea that reachability analysis will be done for each router separately so routers will have only local information about the network state. This keeps table size and data communication minimal.

One important task of our proposed algorithm is back propagating the information to neighbor nodes. Not all the information is required. If node has alternative paths to route, no back propagation process will be initiated.

Optimization is also important from resources stand point, mainly because of limited memory in NoC. Some optimization tasks will be performed but one of the downside in our approach is with area optimization when tables are full, then some of the healthy nodes has to be sacrificed. We also proposed metrics, for calculating reachability and dependability on routing algorithm, which takes into account occurring link faults.

Our proposed method enhances adaptive minimal path routing algorithms to tolerate more faults. Experiment results show huge impact of our method on network reachability and dependability. Performance is not affected and fault tolerance is increased compared to regular routing algorithms.

5.1. Future Work

While in the end a lot were achieved, there is still room for improvements for the future. Following topics will improve our method, and can be investigated in the future.

Reachability analysis versus other algorithms

We have proposed some metrics which can be improved and then used to calculate reachability and adaptivity for the algorithm by taking all the faults into account. It is necessary to adapt these metrics to different algorithms and see how results differ when number of faults increase. Also comparison between results by using our developed method and without it should also be performed.

Implications for Risk-minimized Task Mapping

Another area which also should be considered in future works is task mapping. Since faults in network decrease number of all possible paths for the adaptive algorithm, information about possible paths after some faults occurred is required. This information can be used to map tasks according to system health map. If tasks are be mapped on fault free areas as possible then this would give benefit to cost and efficiency. While faulty areas will decrease efficiency and increase cost they should be avoided as much as possible.

Another problem is with so called islands (shown in example 2). Island is area on the mesh which is completely inaccessible. With our methodology it is possible to define those areas and this information can be used to preform dynamic task mapping in system. However this topic will need much more research in the future.

6. Bibliography

[1] T. Yoneda and M. Imai "Dependable Routing in Multi-Chip NoC Platforms for Automotive Applications", 2012

[2] B. Fu, Y. Han, H. Li and X. Li "A New Multiple-Round DOR Routing for 2D Network-on-Chip Meshes", 2009

[3] M. Valinataj and S. Mohammadi "A Fault-Aware, Reconfigurable and Adaptive Routing Algorithm for NoC Applications", 2010

[4] M. Ebrahimi, M. Daneshtalab and J. Plosila "High Performance Fault-Tolerant Routing Algorithm for NoC-based Many-Core Systems", 2013

[5] E. Wachter, A. Erichsen, A. Amory and F. Moraes "Topology-Agnostic Fault-Tolerant NoC Routing Method", 2013

[6] M. K. Puthhal, V. Singh, M.S. Gaur and V. Laxmi "C-Routing: An Adaptive Hierarchical NoC Routing Methodology", 2011

[7] C. J. Glass and L. M. Ni "The Turn Model for Adaptive Routing", 1992

[8] W. J. Dally and B. P. Towels "Principles and Practices of Interconnection 3Networks", 2004

[9] J. Duato, S. Yalamanchili and L. Ni "Interconnection Networks: An Engineering Approach", 2002

[10] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, J. van der Veen "DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices," International Conference on Field Programmable Logic and Applications, 24–26 August 2005, pp. 153–158.

[11] M. Li, Q. Zeng, W. Jone "DyXY - A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip", 2006

[12] L. Wang, C. Wu and X. Wen "VLSI Test Principles and Architectures Design for Testability", Elsever, 2006

[13] Ge-Ming Chiu "The Odd-Even Turn Model for Adaptive Routing", IEEE Trans. Parallel and Distributed Systems 11(7), 729–738, 2000

[14] O. Novak, E. Gramatova and R. Ubar "Handbook of testing electronic systems" CTU Printhouse Prague, 2006

[15] Daneshtalab Masoud "Exploring Adaptive Implementation of On-Chip Networks", PhD Thesis, 2011