Fall 2021

Multi-Objective Routing for Distributed Controllers

Konstantin Y. Rubin

MULTI-OBJECTIVE ROUTING FOR DISTRIBUTED CONTROLLERS

by

Konstantin Y. Rubin

Bachelor of Science
Smolensk Branch of Moscow Power Engineering Institute, 2013

Master of Science
University of South Carolina, 2017

Submitted in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy in

Computer Science

College of Engineering and Computing

University of South Carolina

2021

Accepted by:

Jason D. Bakos, Major Professor

Shrihari Nelakuditi, Committee Member

Marco Valtorta, Committee Member

Herbert Ginn, Committee Member

Yan Tong, Committee Member

Tracey L. Weldon, Interim Vice Provost and Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to thank my academic advisor, Dr. Jason D. Bakos, and my will power. Without them, this dissertation wouldn't exist.

ABSTRACT

A long-term goal of future naval shipboard power systems is the ability to manage energy flow with sufficient flexibility to accommodate future platform requirements such as better survivability, continuity, and support of pulsed and other demanding loads. To facilitate scalable, low-latency global distributed system control, each control module can include an integrated network interface connected through multiple channels onto a direct, multi-hop network topology. In this work, we focus on a 2D Torus, in which control nodes are arranged in a regular 2D grid, with each node connected through point-to-point connections to its four immediate neighbors. An important advantage of 2D Tori is their redundant topology where there is more than one minimal path between any source and destination as long as they do not share the same row or column in the grid. For the static, all-to-one traffic pattern used by a central controller, the number of minimal routing tables grows as O($N!^{N^2}$).

This dissertation presents a novel approach to generating routing tables that achieve two performance objectives: (1) minimal control period latency, the lower bound of which is the round trip latency of the messages exchanged between the controller and the node having the longest route, and (2) minimal latency jitter. Our approach relies on creating a large system of integer linear algebra equations describing (i) functionality of a network and (ii) constraints needed for perfect load balance and low jitter. We use Gurobi ILA solver to find a satisfying assignment of all boolean variables representing where packets are scheduled to be in a certain timeframe.

Experimental results show that our software pipeline generates routing tables that (i) are guaranteed to have perfect load balance regardless of shape and size of the network and (ii) lower jitter than any of randomly generated routing tables which we simulated. Our software also has an option of generating routing tables that allow packets to follow non minimum hop count paths as well as being held in the source nodes for some time instead of immediately rushing to the master node. That helps packets avoid congested areas, and, as the results show, achieves up to 2x improvement in jitter.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A long-term goal of future naval shipboard power systems is the ability to manage energy flow with sufficient flexibility to accommodate future platform requirements such as better survivability, continuity, and support of pulsed and other demanding loads [1]. Power Electronic Building Blocks (PEBBs) are considered to be a technology for combining modular solid state power converters to realize flexible, efficient power distribution at switching rates. That creates a need for new low-latency controllers which operate in 1-100μs timescale. Fast control coordination across the system using an appropriate communication architecture provides a degree of energy management not previously realizable in shipboard power systems.

To facilitate global distributed system control, each PEBB includes an integrated network interface and is connected through multiple channels to a direct, multi-hop network topology. A direct network was chosen because its cost, in terms of number of channels, scales linearly (e.g. as 4N for a 2D torus) while the message latency scales at $O(N^{1/d})$, where *d* is the dimension of the topology (note that this assumes a square topology; rectangular topologies have less favorable latency). In the case of the 2-ary *n*-cube topology or 2D mesh torus considered in this work, each node is associated with a 5-port crossbar switch, which includes the internal ingress/egress channel and connections to its immediate neighbors. Also, its routing latency is bounded by $O(\sqrt{N})$,

the number of minimal routes for a given source/destination pair scales at $O(4^N)$, and the number of possible static routing tables scales at $O(N!^{N^2})$.

Since this work targets a distributed closed-loop control system, we consider an all-to-one static traffic pattern, in which a single node is selected as the controller (also called "master node") and according to a given period, all nodes except for the controller send a message to the controller and then the controller subsequently sends a response message.

In a such controller network (i) each node (FPGA board) contains an integrated router, and (ii) physical channels (links between the nodes) are shared among logical source-destination transactions, (iii) the static traffic pattern avoids deadlock and livelock problems, (iv) congestion creates collisions within routers, causing non deterministic latency, creating jitter, but (v) properly created routing tables allow for the optimization of traffic flows as well as scheduling packets to be sent at different time slots uniformly distributing workload within the control period and avoiding collisions.

In these types of networks, the worst-case message latency is determined by the longest possible path between two control modules and a number of collisions occurring to each packet which also creates latency jitter. This worst-case latency serves as a constraint for the overall control system design. Load balancing is the key in achieving new low latency requirements as it ensures uniform utilization of the communication channels across the entire network.

Packet reception predictability is another factor determining stability of a controller network which also allows it to detect any potential problem with a network at an early stage. Thus, minimizing data packet latency jitter is the second target for this

dissertation. Also, since the entire power distribution system is controlled by FPGA boards, it should be suitable for an embedded processor.

In this dissertation, our goal is to develop an algorithm for generating routing tables which would service two purposes:

1. improve predictability of packets arrival (i.e. latency jitter) to the master node,
2. achieve the best load balance possible for different shapes of network.

Section 5.2 describes the challenges associated with these objectives: less than 1% of randomly generated routing tables have ideal load balance, while the search space is extremely large: there are $1.43*10^{26}$ routing tables for a 7x7 network. Additionally, those routing tables only allow packets to follow minimum hop count paths and not be delayed in any nodes. Such additional routing flexibility is helpful for collision avoidance and improving jitter.

Since the traffic pattern is static, we base our approach on global optimization using an integer linear programming (ILP) solver, specifically Gurobi. We design an objective function to represent the number of collisions between packets as well as a system of integer linear equations describing the network behavior in terms of (1) its possible routing decisions, (2) the routing constraints such as load balance and route length, and (3) the traffic pattern. This allows the ILP solver to construct the best routing tables in terms of load balance and jitter.

Using the software pipeline built in this dissertation this work makes the following contributions:

1. Our software supports routing table generation for any size and any dimensions of a 2D Torus Mesh network, including cases of using a sparse topology having

missing nodes, including the degenerate case of when one of the master node's neighbors has failed and can not send, receive or transfer any packets.

2. The output routing tables are guaranteed to have ideal load balance, meanwhile the random routing table generator (taken as a baseline) failed to create perfect balance for certain network dimensions even after generating a million routing tables.
3. Depending on a specific application, our software can generate routing table where packets either

a. rush to the mater node following minimum hop count paths only, or
b. follow non minimum hop count paths to avoid congested areas, or
c. are scheduled to be delayed in the source node and then follow minimum hop count paths (that way by the time packets reach congested areas, the links between the node already become available), or
d. follow minimum hop count paths and be delayed either in the source node or any other node along their paths.

4. For the approaches (b), (c) and (d), our software produces routing tables with lower jitter than randomized routing tables, and with no worse jitter than the best random routing tables for the first approach.

The rest of the document is organized as follows: Chapter 2 provides necessary background information about SiC PEBB, FPGA based systems, routing for control systems and problems which typically occur. This background is needed to understand the rest of the dissertation. Chapter 3 shows our literature search which has been done in this field. We have analyzed more than 120 research scientific papers in order to analyze

the most recent findings for the most efficient traffic routing. Chapter 4 describes all the details on how our software generates LP code, how it gets optimized, and how the ILA model is solved generating optimal routing tables. The major novelty here is we schedule the packets' paths to collide with each other as few times as possible. In order to achieve the lowest amount of conflicts between packets, we abstract out from cycles and introduce the concept of ticks - discrete time slots that gives bird-eye view on the control period: it takes packets only one tick to be transferred from a node to another one, although it may around 60 cycles in real hardware. Chapter 5 shows simulation results: we generated over a million of randomized routing tables, analyzed the load balance for them, and found that depending on the aspect ratio of the network, randomized approach produces 0-5% routing tables with ideal load balance while all the tables generated by the new approach have perfect load balance. We then describe how we test those routing tables using a custom made 2D Mesh Torus simulator: 200 random routing tables none of which show no better jitter than the new routing tables. Chapter 6 derives conclusion and describes future work. work.

CHAPTER 2

BACKGROUND

2.1 MESH TORUS NETWORKS

A mesh network (meshnet, shown at Figure 2.1) is a popular network topology in which all the nodes (or bridges, switches, etc.) connect directly, and, what's important, non-hierarchically such that as many nodes as possible can communicate with each other for efficient data routing. A torus interconnect network is also a very popular switch-less network topology for connecting processing nodes. It is also a topology that can be seen as a mesh interconnect with nodes arranged in a rectilinear array of N = 2, 3, or more dimensions, with nodes connected to their nearest neighbors, and corresponding nodes on opposite edges of the array connected.

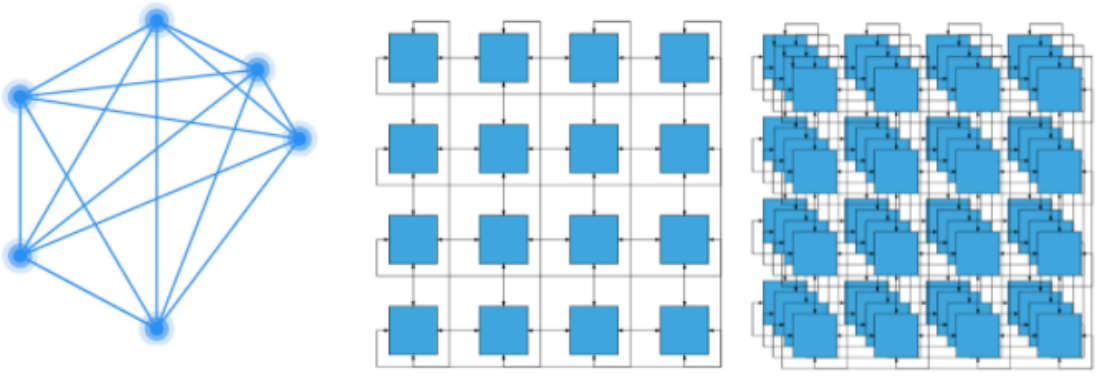


Figure 2.1 Graphical representations of a simple mesh network (left) and torus mesh network (middle) and 3D torus mesh network (right)[21].

A number of supercomputers on the TOP500 list use three-dimensional torus networks, e.g. IBM's Blue Gene/L and Blue Gene/P, and the Cray XT3. Among the

advantages of such apologies, we can highlight (i) higher speed and lower latency (as torus has connections of the opposite edges and data have more options to travel from one node to any other node which creates more paths and reduces network congestion), (ii) lower minimum hop count (e.g. for a 4×4 mesh network, the longest distance between nodes is six hops, but in a 4×4 torus mesh network, the longest distance between any two nodes is two hops because of the loop around links), and (iii) lower energy consumption (sending data packets through fewer amount links requires less energy).

The industry standard is to refer to any direct network as a k-ary n-cube. This term was first introduced by William Dally [20] in 2004. n refers to the number of dimensions in the network, k - radix, size of each dimension, or, in other words, how many nodes a network has in each dimension. That means any such network has kn nodes and a 10x10 2D torus mesh network would be called a 10-ary 2-cube.

As shown at Figure 2.2, each node can be labelled by an n-digit number of radix (base) k, and each node is connected to every node which has a label which differs in only one digit by one. The latter constraint requires the network to have loop-around links and excludes direct mesh (not torus) networks.

2.2 STORE AND FORWARD AND WORMHOLE ROUTINGS

Store and forward routing is a routing technique in which data packets are sent to an intermediate station (i.e. to the next node) where it is kept until the whole packet is fully received and verified and then sent at a later time to the final destination or to next node in the path.

Figure. 2.3 shows nodes A, B, C, D connected to each other. The packet contains multiple segments (words) which are transferred one at a time. Node B, for example, after receiving the first flit waits until all other flits are received before starting forwarding packet to C. The same thing happens with the node C when it receives the first packet from B.

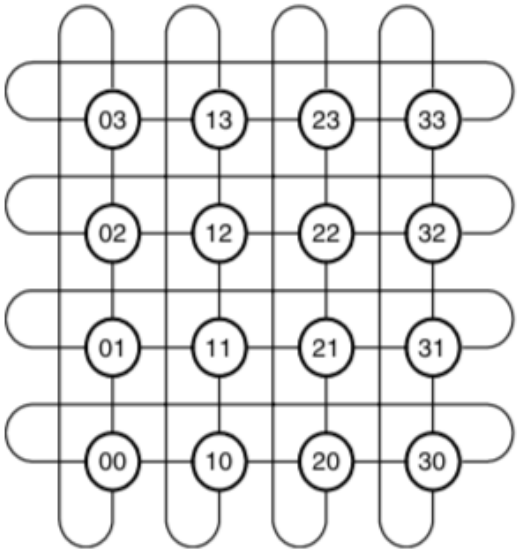


Figure 2.2 4-ary 2-cube network [22].

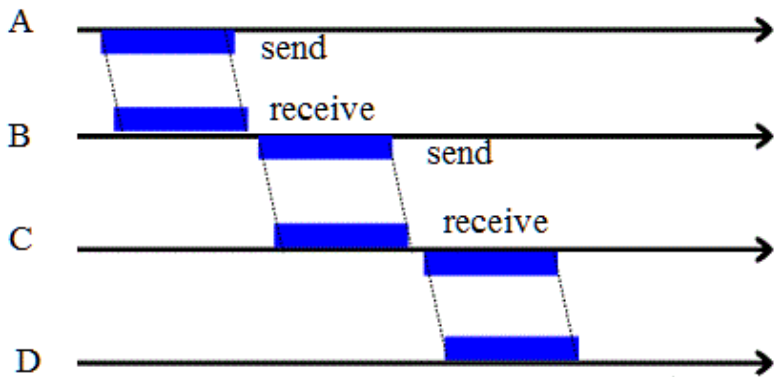


Figure 2.3 Store and forward routing illustration [23].

Another popular routing technique which is a special case of cut-through routing is wormhole routing. It splits the entire packet into subpackets called flits (flow control

digits). Flits are forwarded without waiting until the entire packet has arrived. Wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels of the routing node. The size of a flit depends on system parameters, in particular, the channel width. The first flit of a packet (the header flit) contains the routing information (addresses or source-defined route) for the entire packet. Once a node processes the header flit(s) of a data packet, it selects the direction to forward the entire packet to and begins forwarding flits down that channel. After the header gets transferred first, the remaining flits follow in a pipeline fashion. The timing diagram is shown at Figure 2.4.

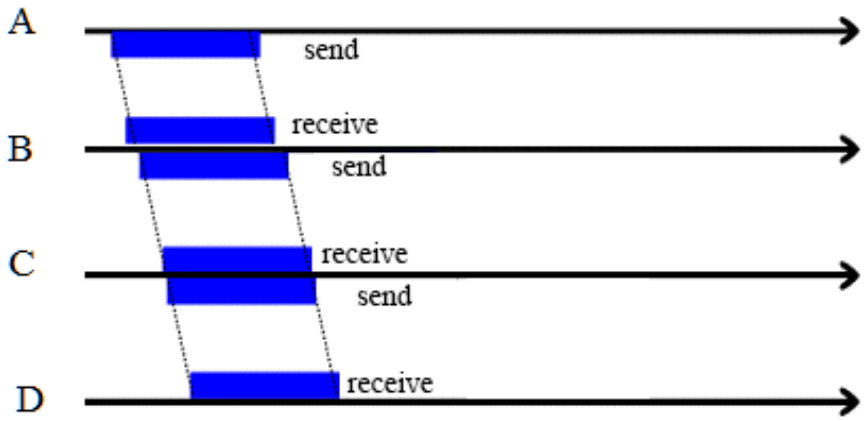


Figure 2.4 Wormhole routing illustration [23].

In store in forward routing, the packet latency is *h*size/bw*, where *h* - the hop count, *size* - the packet size, and *bw* - the channel bandwidth. In wormhole routing, the packet latency is *h*flitsize/bw* + *size/bw*, where *flitsize* = the size of a flit. Thus, wormhole routing offers a reduction in packet latency of at most *size/flitsize*.

Virtual channels are an optional feature of wormhole routers, in which multiple FIFOs are associated with each input and output port, allowing flits from one packet to bypass flits blocked due to upstream channel contention, thereby avoiding a possible

deadlock. Virtual channels allow different packets to share the physical channel on a flit-by-flit basis. Virtual channels are only needed for wormhole routers and carry a high cost due to the need for additional FIFOs and crossbar switch ports..

2.3 NUMBER OF ROUTING TABLES

The number of possible minimal-hop routes on a torus network scales exponentially as a function of the area of the right triangle formed by the source node, destination node, and the nearest node laying on the same dimension as the destination.

For example, for a 3x3 network, there are four nodes that are oriented diagonally relative to the control node, i.e. $|\Delta x|=|\Delta y| = 1$. Each of these has two possible minimal routes to the control node. There are, likewise, four nodes situated on the same horizontal or vertical plane as the control node and, thus, have only one minimal route. As a result, there are $2 \cdot 2 \cdot 2 \cdot 2 = 16$ possible routing tables for a 3x3 network.

A 5x5 network, on the other hand, has out of 24 slave nodes: 4 nodes with $|\Delta x|=|\Delta y| = 2$ each with $(\Delta|x| +|\Delta y|)!/|\Delta x|+|\Delta y| = 6$ minimal routes, 8 nodes with $\{|\Delta x|,|\Delta y|\}\in\{1,2\}$ each with 3 minimal routes, 4 nodes with $|\Delta x|=|\Delta y| = 1$, each with 2 minimal routes 8 nodes with $\Delta x=0$ or $\Delta y=0$ (four nodes having the same horizontal coordinate and four nodes having the same vertical coordinate as the controller) each with one minimal route, giving $6^4 3^8 2^4 = 136{,}048{,}896$ routing tables for 5x5 network, and $1.43*10^{26}$ routing tables for a 7x7 network.

We decided to base the majority of our experiments on 6x12 networks as such dimensions are the most realistic for a controller network which will be placed on a military ship. If we represent a number of routing tables as a function of number of

columns (W) and a number of rows (H), it will look like $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{(i+j)!}{i!j!}\right)^4$ if a network has odd number of rows and columns, $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}-1} \frac{(i+j)!}{i!j!}\right)^4 \cdot \frac{1}{2} \cdot \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=\frac{H}{2}}^{j=\frac{H}{2}} \frac{2(i+j)!}{i!j!}\right)^2$ if a network has an even number of rows and an odd amount of columns, $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}-1} \prod_{j=1}^{j=\frac{H}{2}} \frac{(i+j)!}{i!j!}\right)^4 \cdot \frac{1}{2} \cdot \left(\prod_{i=\frac{W}{2}}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{2(i+j)!}{i!j!}\right)^2$ if a network has an odd number of rows and an even amount of columns, and $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}-1} \prod_{j=1}^{j=\frac{H}{2}-1} \frac{(i+j)!}{i!j!}\right)^4 \cdot \left(\prod_{i=\frac{W}{2}}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}-1} \frac{2(i+j)!}{i!j!}\right)^2 \cdot \frac{1}{2} \cdot \left(\prod_{i=1}^{i=\frac{W}{2}-1} \prod_{j=\frac{H}{2}}^{j=\frac{H}{2}} \frac{2(i+j)!}{i!j!}\right)^2 \cdot \frac{\left(\frac{W}{2}+\frac{H}{2}\right)!}{\frac{W}{2}! \cdot \frac{H}{2}!}$ if a network has an even number of both rows and columns. Please refer to Chapter 4.1 for more detailed analysis.

2.4 TYPES OF ROUTING

Static routing (also called non-adaptive routing) is predefined before the network starts running. It follows a user-defined routing and routing tables can't be changed until the network administrator changes it either after stopping the network operation or on the fly. Such an approach obviously suffers from any changes in the network: node failure, link failure, buffer overflow, node addition, etc. Networks operating in a purely static environment are not able to choose a better route if the next link in a path becomes unavailable. Static routing can't receive and process any information about the network's state and change routing accordingly. The network designer has to reconfigure possibly all the routes and update routing tables in the nodes.

When the traffic pattern is known a priori and the node, channel, and router behaviors are deterministic, routes can be constructed with global knowledge of system state.

Dynamic routing, also called adaptive routing, creates a process for determining the optimal path every packet should follow through a network to arrive at its destination based on the current conditions of the network. Adaptive routing can be compared to a commuter following a different path to work after realizing that traffic on his usual route is clogged up and there is another path available which brings him to work in a shorter time. It creates a capability of a network to avoid running into a dead node or dead link, as long as other path choices are available. Dynamic routing allows as many routes as possible to remain valid in response to the change. The main purposes of adaptive routing are to prevent packet delivery failure, relieve network congestion, and/or improve network performance. However, static routing has its advantages. In static routing, the routes are entered manually and only once, meanwhile in dynamic routing the network needs to constantly be updated with the most recent information about all the nodes and packets. Such broadcast and multicast routing updates become an easy goal for cyber attacks. Additional configuration settings such as passive interfaces and routing protocol authentication are required to increase security.

2.5 NETWORK PERFORMANCE MEASUREMENTS

Network performance can be measured a few different ways: throughput, latency, jitter, failure rate, etc. The purpose of this subsection is to quickly introduce each one of them to make the rest of the dissertation easier to understand.

Network bandwidth is the measurement of the largest amount of data which can be theoretically transferred (sent and received) at a given time interval. It is measured in bits, megabits, or gigabits per second. Link bandwidth is defined as the largest amount of data that can be transferred through that link per second. The maximum bandwidth of a direct mesh network can only be limited by channel bandwidth and time it takes for a node to forward a packet. Throughput is also measured MB/s, GB/s, but refers to an actual data transfer rate that occurs in a network.

Latency is typically defined as the amount of time (measured either in seconds or cycles) it takes for some kind of data to be transferred. If a data packet left the origin at time t_1 and was received by the destination at time t_2, then the latency is calculated as (t_2-t_1).

It is important to distinguish between the packet latency and the network latency. In all-to-one traffic patterns, for example, packet latency refers to the time difference between when a packet was sent by the origin node and the moment when it was received by the master. For the purpose of the current proposal, network latency is measured as the time difference between when the first packet of a period was sent and when the last packet was received by the master node.

Jitter is described as the inconsistency of a packet latency, i.e. how unlikely it is that the packet will always arrive at the same time to the master node. The greater the jitter is, the more unlikely the packet will arrive at the expected time. It is calculated as standard deviation of packet latencies and calculated for every data packet separately. The resilience of a network in the presence of a fault(s) is called fault tolerance. A network which is able to send and deliver packets between non-faulty nodes no matter

which node or link stopped working is called single-point fault tolerant. Most often, being single-point fault tolerance is more than sufficient as the probability of one fault is very low which means two and more faults are extremely unlikely to occur.

2.6 PEBB AND MMC BASICS

PEBB (Power Electronics Building Block) is a strategic innovative engineering concept which is designed to reduce cost, weight, size, intellectual effort and standardize power management [2]. It usually involves power devices, gate drive, and other components with certain functionality and communication interfaces which may serve multiple applications in power electronics systems. Power Electronics Building Block concept is trying to create a modular and hierarchical design. Engineers designing PEBBs may take care of specific details of final intended application (such as switching speed, losses, thermal management, voltage and current measurements, control interfaces and protocols, and potential issues) based on the functional specifications of the power management system. Inspiration came from personal computers: any user knows what the building blocks of a personal computer are, their specification, types of connections, purpose of each block and its capabilities. This idea of a plug-and-play approach is very tempting as it simplifies a lot of procedures for both designers and users. The power electronics management system is aware of its PEBB specifications, its producer, and its operational requirements. The overall control architecture needs to be able to support these PEBBs, regardless of how they are configured. Every PEBB is in charge of operating in its own safety limits. It is possible that PEBBs, when plugged into a power electronic system, configure its operational settings automatically.

The main functionality of a PEBB includes the following:

- Power supply for gate drives and sensors
- Stack and module assembly including gate drives
- Switching control system, including pulse generation for gate drives, communication with control system, and primary protection
- Voltage, current and temperature sensors including A/D sensor signals conversion.

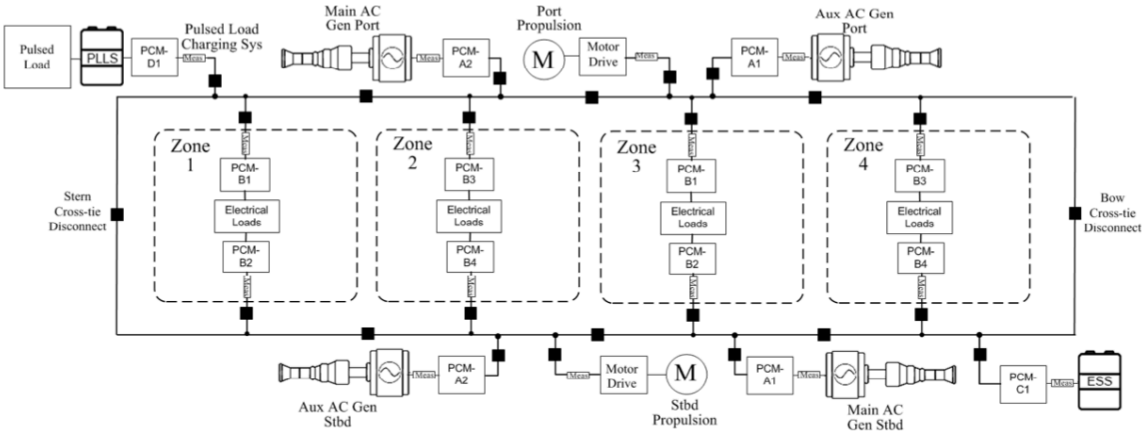


Figure 2.5 Simplified notional shipboard power system [1].

Figure 2.5 shows an example of a notional DC distribution system [3]. Here all major sources and load centers are connected to the distribution system by appropriate converters denoted as PCMs (power conversion modules). Such systems have both fuel based generators and an electrochemical energy storage system. The ESS (energy storage system) can operate both as source and load depending on the system need and battery state of charge (SOC) condition for the shipboard microgrid. The largest loads (e.g. pulsed load and propulsion loads) are interfaced to the main bus. There are also zones of utility loads and two power conversion modules share the zonal load demand for each zone from two main system buses.

The PEBB concept has driven advancements in highly modularized converter systems with many identical subsystems such as the Modular Multilevel Converter (MMC). They have become one of the most attractive multilevel converter topologies for medium-power or high-power applications, specifically, for voltage-sourced converter high-voltage direct current (VSC–HVDC) transmission systems. If we compare MMC with other multilevel converter topologies, the advantages of MMC become obvious:

1. MMC has a modular structure and scalability allowing to meet any voltage level requirements,

2. MMC has high efficiency, which is extremely important for high-power applications,

3. MMC also has superior harmonic performance, especially in high-voltage applications in which a big number of identical submodules (SMs) with low-voltage ratings are required to be placed on top of each other, therefore the size of passive filters can be reduced.

Figure 2.6 shows a schematic diagram of a three-phase MMC [3]. This MMC consists of two arms per phaseleg where each arm comprises N series-connected, identical submodules (SMs), and a series inductor Lo. SMs in each arm are controlled such that they create required AC phase voltage, and the arm inductor is needed to suppress the high-frequency components in the arm current. The upper (lower) arm of three phase-legs are represented by subscript “p” (“n”). Recent developments in SiC power devices are yielding PEBBs with greater switching frequencies than Si devices (which were previously used). That results in big reductions of the time scales compared to power converters utilizing IGBT based PEBBs. Moreover, overall MMC and PEBB

performance is affected by the delay between when the measurements (current, voltage at SMs) are taken and when updated reference signals are received from the controller. As PEBBs and submodules are spatially distributed and connected into local topologies, propagation delays between the nodes and control levels contribute to the overall network latency. Both of these reasons create a need of making controller systems faster and more stable, which is the purpose of this dissertation.

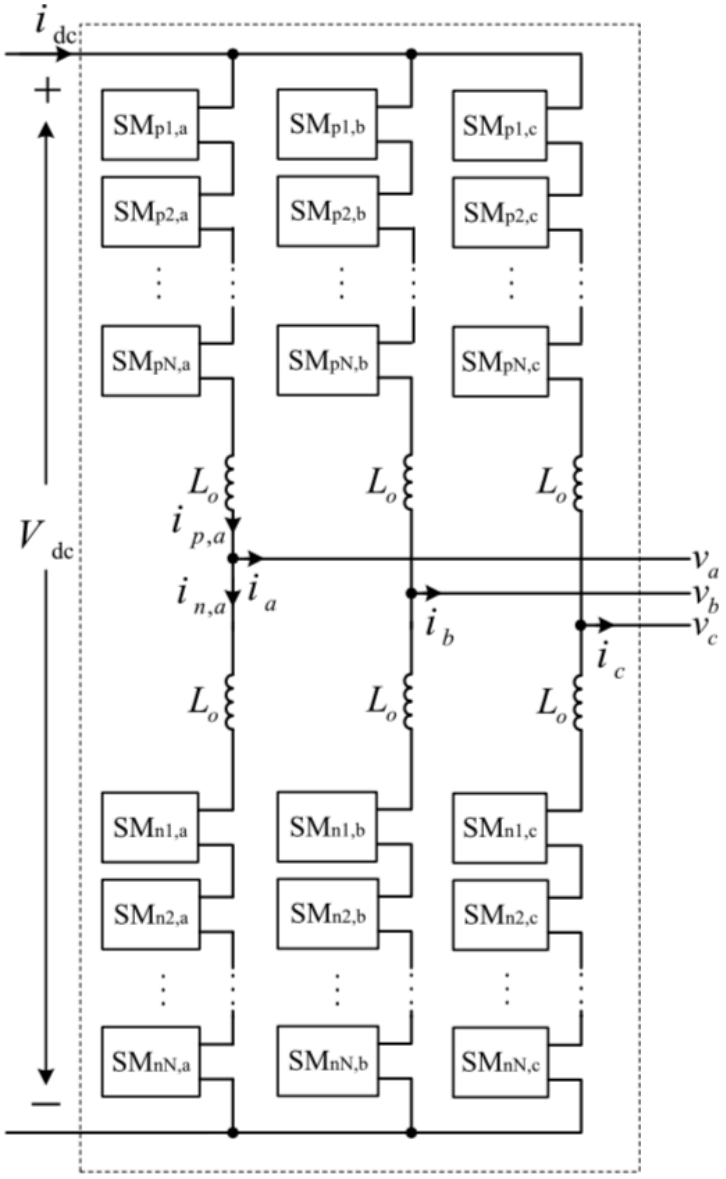


Figure 2.6 Modular Multilevel Converter [24].

CHAPTER 3

RELATED WORK

In this chapter, we summarize the prior work in the following categories: theoretical analysis of routing algorithms, introduction of new performance measurements, topology, and, new routing algorithms.

3.1 THEORETICAL ANALYSIS

Pham and Perreau [4] analyzed and compared reactive mutli-path and single-path routing with load balance mechanisms in the ad hoc networks in terms of connection throughput, traffic distribution (i.e. load balance) and overhead. The results showed that mutli-path routing mechanisms create more overhead as opposed to general single-path routing protocols. However, they provide better performance in terms of congestion. The authors were able to show that the amount of overhead increases with the number of multiple paths. The golden number turned out to be three: if there are more than three paths to route traffic, then the overhead is increased significantly. Three paths provide the best trade off. The authors also showed the formula for the upper bound on the average length of the multi-path routes which guarantees to decrease network congestion. This upper bound depends on a few variables: traffic intensity, number of nodes and processing power of each node which makes it easy to compute for every specific network. This bound allows to know whether load balancing will bring any value for a particular network. It also gives better idea which routes to select not to hit the upper

bound. Pham and Perreau were also able to prove that using multi-path routing always results in performance improvements as far as throughout is concerned.

Duato [5] has made a lot of progress in theoretical analysis of cut-through and store-and-forward networks. He created a necessary and sufficient condition for deadlock-free routing for such networks. Three theorems were proposed which verify that a given adaptive algorithm is deadlock-free, even when there are cyclic dependencies between routing resources: that is possible for a connected and adaptive routing function R for an interconnected network if there exists a subrouting function R1, that is connected and has no cycles in its extended resource dependency graph. More than that, the author provided a necessary and sufficient condition for deadlock free routing. It established that a routing algorithm is deadlock-free if and only if there exists a restricted routing function that is able to deliver all the packets and it doesn't have any cyclic dependencies between the queues supplied by it. Based on those theorems, a network design methodology was shown which always creates fully adaptive, minimal and non-minimal routing algorithms that are guaranteed to be deadlock-free.

3.2 NEW ROUTING ALGORITHMS PROPOSALS

Over the last decades, numerous routing algorithms were proposed for fixed and dynamic traffic patterns, for various topologies, for wired and wireless networks. However, they are all different in terms of what they are trying to improve: some focus on latency and throughput, some try to minimize the hop count, some try to maintain the load balance, and others try to make sure there will be no deadlocks and livelocks throughout the execution of the algorithm. In this section we summarize several relevant and impactful papers.

3.2.1 FOCUS ON THROUGHPUT AND/OR LATENCY

Glass and Lionel [6] introduced several routing algorithms found that previous routing algorithms had had a few flaws: they were either deadlock free and nonadaptive or deadlock free and adaptive but with the necessity to add extra physical or virtual channels. In adaptive routing (as opposed to static routing) the data packets “adapt” to the state the network is currently in and take a peth which depends on other packets or events taking place in the network during its travel. Virtual channels (VC) is a buffer management flow control mechanism which divides buffer storage associated with a physical channel into several small queues rather than a single deep queue. A few virtual channels are associated with one physical channel and compete with each other for physical bandwidth. The authors introduced a few simple partially adaptive routing algorithms for 2D and 3D meshes and were able to simulate them under different conditions: uniform distributions, matrix-transpose and merge-south. As a result, partially adaptive algorithms were shown to perform better than the nonadaptive xy algorithm for many nonuniform patterns of message traffic. Partially adaptive algorithm is a compromise between deterministic routing and fully adaptive algorithm where a data packet selects a path from a limited portion of all possible choices. Those algorithms stand in the middle between the nonadaptive xy algorithm and fully adaptive algorithms which add extra channels to meshes. They became a compromise between less hardware and more adaptiveness with fault tolerance.

Basu, Lin, et. al. [7] presented another innovative routing algorithm which they called PB routing. This algorithm utilizes the steepest gradient search method to route packets of data. In fact, PB-routing is able to calculate and assign scalar potentials to

network elements and forward packets in the direction of maximum positive force which represent the most optimal routing direction for each packet. The authors prove that this algorithm is loop free. They also show how to create and use the function which accounts for traffic conditions inside a node. The resulting algorithm is non minimal in terms of hops and routes traffic around congested areas. During the simulations, such traffic aware algorithms are shown to have significant performance in end-to-end delay and jitter when compared to standard algorithms with shortest path routing. Originally, this concept was created for IP routing, but it was shown that the main concept can be applied to a variety of applications as the main idea is platform independent: using steepest gradient search to find the best direction of a packet.

Singh, Dally,et al. [8] proved mathematically two claims: (i) no routing algorithm can guarantee a throughput greater than half of the network capacity on k-ary n-cubes and (ii) no minimal routing algorithm can guarantee a throughput asymptotically greater than 25% of the network capacity on k-ary n-cubes. That brought them to the conclusion that any minimal routing algorithm can have worst-case performance at best half as good as a well balanced non-minimal algorithm on k-ary n-cubes. Based on that, they introduced a load-balanced routing algorithm for torus network called GOAL which stands for Globally Oblivious Adaptive Locally. This algorithm achieves high throughput on adversarial traffic patterns while preserving locality on bening patterns. GOAL also has global load balance by randomly choosing the direction to route in each dimension. Local load balance is also achieved by routing in the predefined directions adaptively. GOAL was compared to other popular oblivious and adaptive algorithms: VAL, DOR, ROMM and RLB, CHAOS and MIN AD in terms of throughput, latency, stability, and hot-spot

performance. In fact, GOAL achieves the highest throughput among the four advertiser patterns and on the average and worst-case of 1,000 random permutations while showing only 58% and 76% of the throughput of minimal algorithms on nearest neighbor traffic and uniform traffic respectively. However, because of the oblivious misrouting, GOAL also has 40% higher latency on random traffic than the minimal algorithms.

3.2.2 FOCUS ON PACKETLOSS

Kimura and Muraguchi [9] concluded that in order to avoid buffer overflows, any store-and-forward routing network should have a buffer management policy when the buffer size is limited. That would minimize the number of messages undelivered to destination nodes. They proposed a new buffer management algorithm based on the data packet rarity. The main idea is to give high priority to rare packets (messages with very few or even only copy existing in the network) and assign low priority to the messages which have a lot of copies walking around the network. When the number of copies of any data packet in the network is only one, the algorithm regards the packet as “rare”. If a rare message is discarded by a receiving node, it vanishes from the network and guarantees to be lost. If the buffer of the receiving node is full, the new incoming data packet can be completely disregarded if it has low priority and can replace another node with lower priority otherwise. More than that, to improve message delivery probability, the proposed algorithm also takes distance between the source and destination into account: the closer they are, the higher the priority of the data packet. What’s important to note is that in the proposed policy, it is not necessary to know the exact number of copies of a data packet to calculate the priority (they use a delegation mechanism) to do that. The proposed algorithm was simulated and compared to four other previously

described algorithms and showed up to 60% improvement in undelivered probability. Although, conventionally, 2D Torus Mesh networks have only one copy of each packet, our system operates way below the network saturation point, so this algorithm can be useful for the current dissertation, because creating data packet copies can be a good option.

Ramesh, Gururaj, et. al. [26] evaluated the performance of Torus interconnection and modified mesh interconnect (specifically, a mesh network with two additional nodes) on the basis of packet loss with source routing with the use of different traffic generation mechanisms for parallel transmission and comparing the results. They concluded that torus interconnect has better packet loss as compared to modified mesh topology as it on average traverses a lot fewer hops to reach the destination. They did it by setting up practical experiments where in different network topologies for different traffic patterns, node pairs were sending packets to each other, the number of generated packets was compared to the number of received ones. Unfortunately, the authors analyzed at most 149 packets per pair which does not seem statistically significant.

Youjun, Binqiang, et. al [10] concluded that conventional single-next-hop routing algorithms, on one hand, always choose the best path for routing, and, on the other hand, bring traffic congestion. They can not be flexible about distributing traffic effectively and utilizing network resources efficiently. As traffic gets increasingly congested, the packet loss increases. The authors proposed a new multi-next hop routing algorithm named STSA (Spanning Tree Searching Algorithm) which is based on analyzing the network and the direction of the next hop which leads to efficient traffic regulation by building a spanning tree with the destination node as the root which creates multiple possible next

hops for the original node. During the simulation experiments it was found that STSA algorithm is more efficient as compared to SPT. It avoids routing in loops, it can fully utilize the network resources and provide basis for whole network balance as well as congestion avoidance. At the rate of 130 Mb/s the packet drop rate of STSA was proven to be 18% while being 23% for the traditional SPF algorithm.

3.2.3 FOCUS ON MINIMUM HOP COUNT

Ramanujam and Lin [11] created a new closed-form oblivious routing algorithm which they called W2TURN that is worst-case throughput optimal for 2D torus networks. In oblivious routing, a set of optional paths is chosen in advance for every source-destination pair, and every packet for that pair must travel along one of these optional paths. Thus, the path a packet takes only depends on its source-destination pair (and maybe a random choice to select one of the options). Oblivious routing is opposed to adaptive routing, where the path taken by a packet may also depend on other packets or events taking place in the network during its travel. This novel technique is based on a weighted random selection of paths that contain at most two turns. The authors were able to prove that W2TURN performs better than the best previously known closed-form worst-case throughput optimal routing algorithm known as VAL in terms of average hop count. The idea of using was already proposed before by Towles, but did not have a closed-form algorithmic description. That algorithm only provided closed-form description of generation possible paths that data packets can take, and still required solving a complicated linear programming problem to determine the path distribution for each given network radix. The proposed algorithm describes the routing in terms of XYX segments where particular turns depend on if k is even or odd and the location of source

and destination nodes. As a result, if a network radix is even, W2TURN performs almost as good as optimal-2TURN, within just 0.72% of average hop count when k goes up to 12. When k is even, however, W2TURN comes very close to optimal routing, within just 1.4% for k=10. In general, though, W2TURN is shown to outperform IVAL which is the best previously known closed-form worst-case throughput optimal algorithm.

Olson and Shin [12] created an algorithm which tries to minimize the hop count in case of any number of faulty links. Their fault-tolerant routing algorithm guarantees the delivery of any message provided that there exists a path between a data packet's source and destination. This technique has been proven to work on many conventional mesh architectures such as hexagonal mesh as well as torus mesh. The algorithm can also determine that there is no path existing between the source and destination. The algorithm assigns all packets one of two different modes: detour mode for the packets whose shortest paths are blocked by faulty regions and free mode for the ones which should be on their shortest path to the destination. If the packet is in the detour and it can not proceed, it should be transmitted following the link which is immediately counterclockwise of the link by which the message entered the node. In the case of being in the detour mode, but if the current node is located closer to the destination than the node the packet came from, then mode is changed from detour to free. Data packets in free mode just proceed in a regular fashion following the minimal routing path. The simulations showed very good results: the number of extra hops increase significantly only when the number of faulty links get above 40% and 55% for big networks. That is logically reasonable: the larger the mesh, the more ways we can find to route packets

around faulty regions. The algorithm was tested and proved to be working for wrapped and unwrapped, square, hexagonal meshes, but should work for other meshes as well.

3.2.4 FOCUS ON LOAD BALANCE

Yuana, Mahapatra, et. al. [13] proposed a static load balance routing algorithm for 2 and 3 level extended generalized fat-trees (XGFT) called Round-Robin Routing (RRR). This algorithm achieves near perfect load balancing. Simulation results showed that on many slimmed fat-trees, RRR performs much better than D-mod-k for dense traffic patterns due to its better load-balancing property, but performs worse for sparse traffic patterns. D-mod-k makes all up-links balanced but not down-links. With S-mod-k, the traffic with one destination is spread evenly across all available links making all down-links balanced and up-links imbalanced. Hence, the idea behind RRR is to route SD pairs such that, on one hand, all SD pairs whose destinations are in lower level switch user top level switches in a round-robin fashion. That will guarantee all downlinks to each lower level switch carry a similar number of SD pairs. On the other hand, all SD pairs that have sources sources in a lower level switch, use the top level switches in a round-robin fashion. That will guarantee that all uplinks from each lower level switch carry a similar number of SD pairs. The authors also integrated D-mod-k, S-mod-k and RRR in one routing algorithm and proposed so-called Combined routing which pulls together the best parts from all algorithms. While D-mod-k may result in imbalance when applied to SD pairs, if applied only to a part of SD paris while maintaining perfect load balance, so is S-mod-k. The combined mode first uses D-mod-k to route as many SD pairs as possible while maintaining load balance, after which S-mod-k is applied. After that, for the rest of the SD pairs which can not be scheduled using either D-mod-k or S-mod-k are routed

using RRR. Such Combined routing scheme showed 20-80% improvement over D-mod-k alone. Although this algorithm is designed for fat-trees, we may probably consider a mesh network to be a special case of a tree with the root being a destination for all packets by removing necessary links from the mesh.

Manevich, Cidon, et. al. [14] noticed that NoC encounters diverse and time dependent traffic as the number of applications in CMPs and MPSoCs increased. They introduced a new NoC routing scheme, called Adaptive Toggle Dimension Order Routing (ATDOR) focusing on constant load rebalancing of the network. This routing technique is adaptive and achieves higher throughput as compared to conventional oblivious routing schemes that are perceived to be easier in terms of hardware implementation. In ATDOR. For every source-destination pair, paths are adaptively switched between XY and YX. The core idea is to split the network traffic to XY and YX routes coming from O1TURN and TXY. ATDOR is composed of two modules : a feedback module and a control module. The first one is constantly monitoring the traffic across the network and aggregates the data about the most congested locations and updating Traffic Load Matrix (TLM). After each iteration, the routing control module a set of source-destination pairs and computes the updated least congested route for each pair. After evaluating all destinations for a particular source, it updates the corresponding routing tables. Although it is a simple centralized routing system for 2D meshes that adaptively toggles between Xy and YX rimensions-ordered routes for all source-destination pairs, ATDOR outperformed distributed adaptive routing scheme in terms of average latency and load balancing for a variety of traffic loads by 30%.

Firreira, Silveira, et. al. [15] introduced a new load balancing algorithm trying in order to find an opportunity to reduce latency for NoC. They called it Specific balancing method, and it allows generating deterministic routing algorithm with simplistic implementation and low latency. First, for each communication pair of PEs, it selects the path with the lowest communication weight. It then updates the weights of all links in the chosen path with upcoming communication volume. That of course changes weights of selected links and paths overall which has to be taken into account while routing other data packets. The algorithm goes through the same procedure iteratively until the convergence of the standard deviation of the average of the NoC resources' weight. This convergence creates a set of deadlock free paths which are then used to routing tables for each NoC router. Specific balancing algorithm is simulated and compared to Random, Dynamic and Uniform balancing methods. Uniform balancing is shown to be the best out of those three, and Specific balancing beats it in terms of latency by more than 30% with 100% traffic injection rate.

3.2.4 FOCUS ON DEADLOCK AND LIVELOCK FREE

Kinsy, Cho, et. al. [16] developed a completely new application-aware framework that assures deadlock free routing by forcing routes to go along acyclic dependency graphs. They presented a MILP (mixed integer-linear programming) approach and a heuristic approach for creating a deadlock-free routings which also minimize channel loads for a network which bandwidth is known beforehand. This Framework first creates a new acyclic CDG by deleting some edges from a channel dependence graph, then transforms it into a flow network and creates a set of flows. After that, it performs application aware routing using these flows and, finally, selects the best set of routes

using Dijkstra's algorithm. Routing tables generated by this framework were simulated and compared to dimension order routing, ROMM and Valiant on a set of standard synthetic traffic patterns such as transpose, bit-complement, shuffle and H.264 decoding. Such traffic patterns represent a variety of bandwidth demands for flows. Simulation results showed that the proposed framework generated routing algorithm which performs better than dimension order routing by having a 40% bigger throughput saturation threshold for transpose and shuffle benchmarks, and has exactly the same results for bit-complement traffic. The primary feature of the framework became its biggest drawback: it needs to have some kind of knowledge of the target application which doesn't necessarily have to be bandwidth demands.

Blazewicz, Bovet, et. al [17] concluded that a problem of deadlock avoidance in store-and-fowrward networks with one buffer per node is NP-complete, and, unlikely to lead to any polynomial time algorithm. Thus, authors introduced a second buffer in each node and considered the problem of deadlock avoidance for fixed and dynamic traffic routing. They created a graph-theoretical condition to avoid deadlocks specifically for store-and-forward networks which says that the network is deadlock free if the state S=(V, A) it is in is so called free-dominated which happens when free(S) is a dominating set for the transitive closure S* of S. So in order to avoid deadlocks, the algorithm simply needs to check whether the network state S' resulting from the transition of a message would be free-dominated. After concluding that, authors introduced and discussed control flow procedures assuring the deadlock avoidance for both fixed and dynamic traffic routing. For dynamic routing, there are four conditions which need to be checked using depth-first search strategy from the node which is sending a message. For fixed routing,

where all the information about data packets transactions is present beforehand, it is easy to check if a possible realization of the transmission to the destination nodes will lead to a free-dominated state S". The transmission would be allowed only in that case.

3.2.5 FOCUS ON FAULT TOLERANCE

Zhang and Greiner [18] presented a reconfigurable routing algorithm for a 2D-Mesh Network-on-Chip (NoC) dedicated to fault- tolerant, Massively Parallel Multi-Processors Systems on Chip (MP2-SoC). The main idea of their algorithm is to route all the packet through so-called cycle free contours which are created around faulty regions. For a 2D-Mesh network considered in the paper, a natural contour has nine possible shapes corresponding to the nine types types of faulty node locations: at each corner (4) where the contour consists of three nodes surrounding the faulty one, at each side (4) in which case there are five nodes surrounding the faulty one, and at all other positions where the contour will consist of eight nodes surrounding the faulty one. The last type of contours is the only one which has cycles, so a turn-based fault tolerant approach was adopted to break them - the two NE turns were prohibited. The authors measured performance as a latency threshold and concluded that given an average workload, the impact of the modified routing algorithm on the average latency is negligible, but the saturation threshold becomes much worse, when the hole is situated at the center of the mesh. Implementation of this algorithm requires only a 4bits configuration register per router. The silicon area penalty is only 8% of the router footprint, and about 0.2% of the total chip area.

Schonwald, Zimmermann, et. al. [19] presented a novel fully adaptive and fault-tolerant algorithm for Network-on-Chips called FDWR (Force-Directed Wormhole

Routing). This algorithm is implemented in the switches of a TLM (Transaction Level Model) packet switching NoC using SystemC. The paper shows how the algorithm distributes all the network traffic uniformly avoiding overloaded links by using forces and simulation results show that the proposed FDWR algorithm routes data packets even in the case of faulty links or nodes. The authors conclude that the scalability of a global routing table declines with the size of the network because every component needs to use that global routing table to look up the distance and the path to the packets destinations. That is why the proposed algorithm uses a completely different approach. It first divides the packet into flits, and the first flit each packet is called ADDRESS or DISCOVERY flit which contains routing information like destination and source addresses. This flit is sent to all four neighbors of any node. If the neighbor nodes are available, they answer with the ANSWER flit containing the shortest available distance to the destination. Whoever proposes the shortest path, receives the data packet next. Obviously, if a node or a link is faulty, no response will be received. This algorithm may work even better for store-and-forward routing, because if a link or node becomes faulty during communication using wormhole routing, some flits can not be routed along the established paths to the receiver and these flits get lost in the network. That, of course, is impossible to happen in store-and-forward routing as there are no flits.

In conclusion, we found that almost no scientific research was carried out which would focus on what our practical application requires which is minimizing jitter for a 2D Mesh torus network with a static traffic pattern and store-and-forward routing forcing data packets to follow minimum hop count paths.

CHAPTER 4

APPROACH

4.1 INTRODUCTION TO LOAD BALANCING AND SIMULATOR

The designed controller is based on a 2D torus mesh network with 60-120 nodes where every node is an FPGA board located in different parts of the ship and controlling power in that particular area. Each node is connected to four neighbors with optical fiber links using which it is possible to exchange data describing the functionality of the current state of the controller and spread the controller commands. One of the nodes is called a "master node". Every control period, the master node collects data packets from every other node containing the latest information on every node (i.e. information about electrical power in a specific part of the ship). Thus, such a network executes an all-to-one routing pattern.

The number of minimal paths for a node (excluding the around links) located at the i^{th} row and j^{th} column can be calculate as $\frac{(i+j)!}{i!\,j!}$. If a node is located at the diagonal (i=j), then the number of minimum paths grows is in $O(4^N)$, where N is a row (or column) number the node is at.

$$f(N) = \frac{(N+N)!}{N!\,N!} = \frac{(2N)!}{N!\,N!} \approx \frac{\sqrt{2\pi 2N}(2N)^{2N}}{\sqrt{2\pi N}N^N \cdot \sqrt{2\pi N}N^N} = \frac{\sqrt{2\pi 2N}2^{2N}}{\sqrt{2\pi N}\cdot\sqrt{2\pi N}} = \frac{4^N}{\sqrt{2\pi N}} \in O(4^N)$$

The number of routing tables for such networks grows exponentially as the dimensions of the network increase. Even if we force all the packets to be routed via the routes with the minimum hop count, there are 16 possible routing tables for 3x3 network,

136,048,896 routing tables for 5x5 network (typical 2D Torus Mesh network is shown at Figure 4.1), and $1.43*10^{26}$ routing tables for a 7x7 network. We decided to base the majority of our experiments on 6x12 networks as such dimensions are the most realistic for a controller network which will be placed on a military ship, $2.13*10^{47}$.

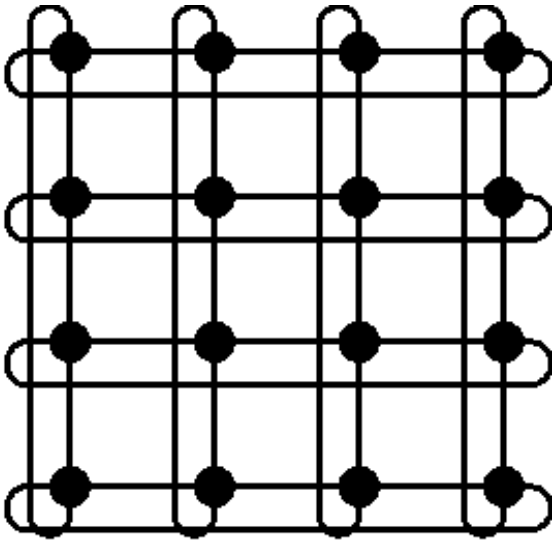


Figure 4.1 5x5 Torus Mesh Network.

As mentioned in Chapter 2.3 of the current dissertation, if we represent a number of routing tables as a function of number of columns (W) and a number of rows (H), it will look like this: $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{(i+j)!}{i!j!}\right)^4$ if a network has odd number of rows and columns, $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}-1} \frac{(i+j)!}{i!j!}\right)^4 \cdot \frac{1}{2} \cdot \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=\frac{H}{2}}^{j=\frac{H}{2}} \frac{2(i+j)!}{i!j!}\right)^2$ if a network has an even number of rows and an odd amount of columns, $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}-1} \prod_{j=1}^{j=\frac{H}{2}} \frac{(i+j)!}{i!j!}\right)^4 \cdot \frac{1}{2} \cdot \left(\prod_{i=\frac{W}{2}}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{2(i+j)!}{i!j!}\right)^2$ if a network has an odd number of rows and an even amount of columns, and $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}-1} \prod_{j=1}^{j=\frac{H}{2}-1} \frac{(i+j)!}{i!j!}\right)^4 \cdot$

$\left(\prod_{i=\frac{W}{2}}^{i=\frac{W}{2}}\prod_{j=1}^{j=\frac{H}{2}-1}\frac{2(i+j)!}{i!j!}\right)^2 \cdot \frac{1}{2} \cdot \left(\prod_{i=1}^{i=\frac{W}{2}-1}\prod_{j=\frac{H}{2}}^{j=\frac{H}{2}}\frac{2(i+j)!}{i!j!}\right)^2 \cdot \frac{\left(\frac{W}{2}+\frac{H}{2}\right)!}{\frac{W}{2}! \cdot \frac{H}{2}!}$ if a network has an even number of both rows and columns.

As you can see, the first factor of all the formulas above is the same. Any rectangular 2D torus network with an odd number of columns and rows can be split into fours parts (quarters) where every node will have only direction to follow to reach the master node assuming there is only the minimum hop count paths are allowed as shown on the Figure 4.2 - 5x5 2D direct mesh torus network virtually split into four quarters with the master node located at the top left corner.

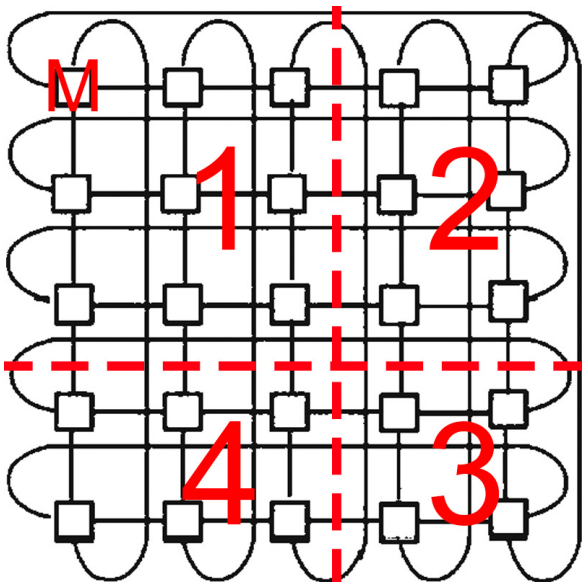


Figure 4.2 5x5 2D direct mesh torus network.

All the nodes from the first quarter will pass their messages in north-west direction, second - north-east direction, third - south-east direction, fourth - south-west direction. All the nodes located at the same row or column as the master node don't have any path diversity - there is only one path with the minimum hop count transferring the data packet to master. Apart from that, in each quarter we have H*W-H-W+1 nodes split

into four equal groups. Each one of those nodes has a choice of $\frac{(\Delta x+\Delta y)!}{x!y!}$ possible paths where Δx and Δy are minimal distances between the current node and the master in horizontal and vertical directions respectively. Since we only consider the paths with the minimum hop count, the lengths of all of those paths are the same - Δx+Δy. However, if we take into account the loop-around links, the paths length changes depending on which quarter the node is located in: (Δx+Δy) for the first quarter, ((W-Δx)+ Δy)for second, ((W-Δx)+(H-Δy)) for third, and (Δx+(H-Δy)) for the fourth quarter. More than that, The formula for path diversity should have an additional factor of 2 (if a node is located right between two quarters) or 4 (if a node is located right between four quarters) Note that the minimal distance may take advantage of loop-around links. Multiplication of all paths of all of those nodes gives us the number of routing tables and can be represented as $\left(\prod_{i=1}^{i=\frac{W}{2}}\prod_{j=1}^{j=\frac{H}{2}}\frac{(i+j)!}{i!j!}\right)^4$. If we add one more column (or row) to the described network (Figure 4.3) and follow the same logic, we will see that the nodes in the column (row) in the middle between the quarters have twice as many available routes as they would if they were not on the border. In this case, the formula for the amount of routing tables needs to have an additional factor of $\frac{1}{2}\cdot\left(\prod_{i=\frac{W}{2}}^{i=\frac{W}{2}}\prod_{j=1}^{j=\frac{H}{2}}\frac{2(i+j)!}{i!j!}\right)^2$. If the network has odd number of columns and even number of rows, we can follow the same logic and derive $f(W,H)=\left(\prod_{i=1}^{i=\frac{W}{2}}\prod_{j=1}^{j=\frac{H}{2}-1}\frac{(i+j)!}{i!j!}\right)^4\cdot\frac{1}{2}\cdot\left(\prod_{i=1}^{i=\frac{W}{2}}\prod_{j=\frac{H}{2}}^{j=\frac{H}{2}}\frac{2(i+j)!}{i!j!}\right)^2$. If a network has an even number of both columns and rows (Figure 4.4), then we have to take into account the node which is located right on the border of all four quarters. Because of the loop-around links, that node has four times as many paths as it would have without the links and can

be described as $4 \cdot \frac{(\Delta x+\Delta y)!}{x!y!}$. Comparing the first and last formulas, we can see that, just by adding another row and column making W and H to be even, we are significantly increasing the number of possible routing tables. Estimating order of growth of the function above turned out to be more difficult than expected. Assuming that the order of growth is approximately the same regardless of whether the number of rows and columns are odd or even, we will only estimate the first case, in which a network has odd number of rows and columns: $\frac{(i+j)!}{i!j!} = \frac{i^j}{j!}$ $(for\ N \to \infty)$. So $f(W,H) = \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{(i+j)!}{i!j!}\right)^4 \approx \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{i^j}{j!}\right)^4$ which is a lot easier to analyze in terms of big-O notation using automated math software and turns out to be $O(N!^{N^2})$. Further analysis showed that, technically, $f(W,H) \approx \left(\prod_{i=1}^{i=\frac{W}{2}} \prod_{j=1}^{j=\frac{H}{2}} \frac{i^j}{j!}\right)^4 \in \Theta\left(\left(\frac{1}{BarnesG\left(2+\frac{N}{2}\right)}\right)^{2N} \left(\frac{N}{2}!\right)^{\frac{N}{2}\cdot(2+N)}\right)$.

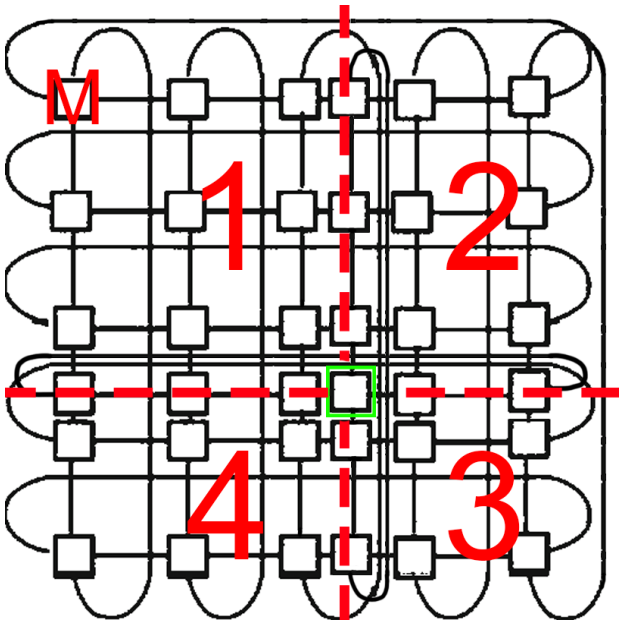


Figure 4.3 6x6 network split into four quarters.

To make this controller network as efficient as possible, it was decided to create an algorithm which would find the best routing table out of a large pool of possibilities. The best routing table is the one which balances the load for four channels attached to the master node, i.e. amount of packets traversing each of the four links should be the same and can be described as $L_{ideal}=(W*H-1)/4$. For example, for a 10x10 network, the number of packets traversing each link attached to the master node is $\left\lceil \frac{10 \cdot 10-1}{4} \right\rceil = 25$.

4.2 NEW APPROACH OVERVIEW

The main goal of this dissertation is to develop an algorithm for generating routing tables which would serve two purposes: (i) improve predictability of packets arrival (i.e. latency jitter) to the master node, (ii) achieve the best load balance possible for different shapes of network. To achieve this, we have built a software toolkit shown on Figure 4.4. The input parameters are the dimensions of a torus mesh network for which routing tables are being created. The outputs are packet latency jitter and load balance for the generated routing table(s).

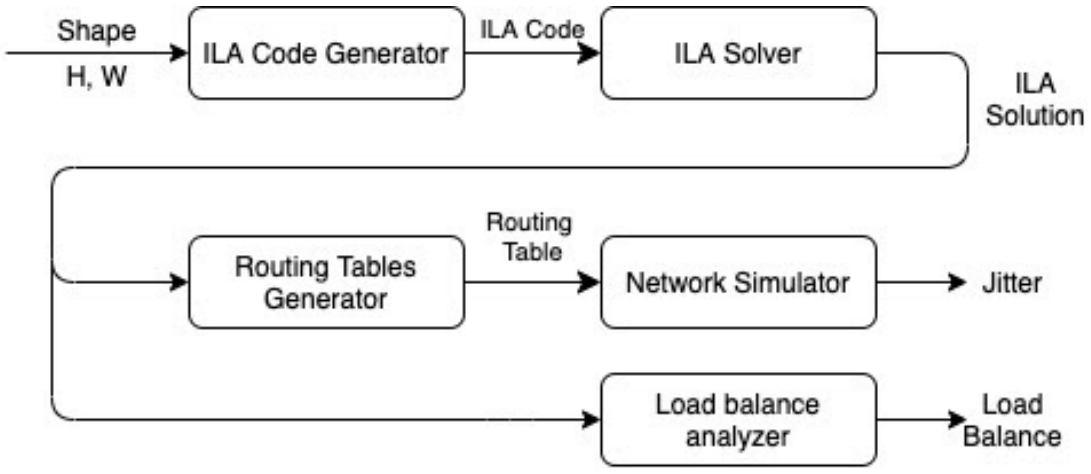


Figure 4.4 Block-diagram explaining the software toolkit which will be developed.

ILA Code Generator - Python code which automatically generates linear programming (.lp) file. This file contains a big set of linear integer equations describing all the mechanics of a torus mesh network. Such equations guarantee that at every moment each packet can only be in only one node, packets can be transferred only between neighboring nodes, at the beginning of each period all the packets are located in the corresponding source nodes, at the end of each period all the packets reach the master node, throughout the period master node receives no more than is $\left\lceil \frac{W \cdot H - 1}{4} \right\rceil$ packets on each one of four links (which guarantees the load balance), etc. The process of creating such equations is long and tedious, and, thus, has to be automated. For example, the .lp file describing the mechanics of 6x12 torus mesh networks has 6,155,753 constraints and is 350MB of size. Please see a more detailed explanation of how Integer Linear Programming applied to our problem works in Chapter 6.

ILA Solver - (Integer Linear Algebra Solver), also Python code which reads the .lp file produced in the previous step, and runs a Mixed Integer Linear Programming solver. Out of all the solvers available on the market we chose Gurobi for the following reasons:

1. It is commercial software, i.e. very reliable and has good support system and documentation,
2. Gurobi has free academic license available for one year with an opportunity to be extended,
3. It supports API in multiple programming languages such as Python, C, C++, Java, MATLAB, R, .NET.

4. Gurobi supports solving external .lp files, which is convenient as we can always switch to another solver that supports reading external .lp files in case Gurobi doesn't fulfill our expectations.

This ILA Solver block generates a set of values satisfying all the constraints in the .lp file, which describe which link or node every packet needs to be at every time slot. This data is fed to the next two blocks.

Routing Table Generator - Python code that parses and transforms the solution to the satisfiability problem into a routing table. The routing table usually tells a router that the final destination (master node in our case) can be optimally reached by sending the packet to a specific router (next node's router in our case) that represents the next hop on the way to the final destination. The output of this block is a routing table which is then fed to the network simulator.

Network Simulator - JavaScript code which uses Cytoscape library to simulate all the processes in the torus mesh network. The simulator presents the network as a graph with vertices being FPGA boards distributed over the ship and edges being fiber optical links connecting the boards. The biggest advantages of Cytoscape being a third party library are (i) graphical representation of a graph including various animations and (ii) set of predefined graph functions such as Dijkstra's Floyd–Warshall, Bellman–Ford, etc. algorithms. The simulator has the following features:

1. It is capable of defining the clock jitter (i.e. how much FGPAs' clocks differ from one another), number of flits per packet, number of cycles required to transmit a packet over a channel, number of cycles required for an FPGA to process a packet and send it to the next node in a path, number and frequency of clock

synchronization cycles. These cycles are introduced by the hardware manufacturer and necessary for successful communication. In those cycles links don't transfer any data from one node to another as they send special characters that help the receiving and transmitting sides read and write data with the same speed.

2. Number of clock cycles per period and number of periods a network is simulated for are custom set.
3. It supports simulating multiple routing tables and multiple clock jitters for those routing tables.
4. All the code is written in HTML, CSS and JavaScript, thus, can be deployed to the web server, and then run in the user's browser from any location.

The network simulator outputs the latencies of every packet which are accumulated into arrays. Latency jitter is calculated as the standard deviation of latencies for every packet individually.

Load Balance Analyzer - Python code that analyzes load balance of the network for a generated routing table. One of the objectives of distributed control of high-frequency power electronic systems is to achieve the minimal control period for a given network size, packet size, and channel bandwidth, a lower bound for which is shown in the following equation, where *width* and *height* define the size of the torus topology, *size* - packet size in bits, and *bw* - channel bandwidth in bits/sec.:

$$period \geq max\left(2 \cdot \left(\left\lfloor \frac{width}{2} \right\rfloor + \left\lfloor \frac{height}{2} \right\rfloor\right) \cdot \frac{size}{bw}, \frac{size \cdot \left\lceil \frac{width \cdot height - 1}{4} \right\rceil}{bw}\right)$$

Based on the ILA Solution, load balance analyzer looks at every path of every packet at every time frame, and counts packets that are in a master's neighbor node at a timeframe N and in the master node at the timeframe (N+1). Please note that the "timeframes" above are not cycles (as it may take take up to 60 cycles to transfer a packet over a link). Please see Chapter 4.3.1.1 for further discussion on what measurement of time is being used in the current dissertation.

4.3 GUROBI DESIGN

4.3.1 SCHEDULE TIMING ABSTRACTION (TICKS)

The ILA Code Generator block (from Figure 5.1) is responsible for generating a set of linear inequalities describing the mechanics of a torus mesh network. For example, the following statement consisting of boolean variables guarantees that a packet from a node with coordinates (0,2) in a 3x3 network is located only in one node at a time slot 0.

$$P\_0\_2\_T\_0\_N\_0\_0 + P\_0\_2\_T\_0\_N\_0\_1 + P\_0\_2\_T\_0\_N\_0\_2 + P\_0\_2\_T\_0\_N\_1\_0 + P\_0\_2\_T\_0\_N\_1\_1 + P\_0\_2\_T\_0\_N\_1\_2 + P\_0\_2\_T\_0\_N\_2\_0 + P\_0\_2\_T\_0\_N\_2\_1 + P\_0\_2\_T\_0\_N\_2\_2 = 1$$

Here and in all further code examples in the current dissertation, a boolean variable $P\_i\_j\_T\_k\_N\_l\_m$ is True if and only if a packet from node (i,j) is located in a node (l,m) at a time slot k.

The most intuitive way of describing a network would be to create such boolean equality for every clock cycle. However, that approach is intractable, because (i) the number of clock cycles for one period of network operation may reach thousands for big networks, (ii) according to transceivers documentation and our practical experiments the propagation delay has a variance (i.e. if a packet starts its transmission from a node (i,j)

to a neighbor node (l,m) at a cycle i, there is no guarantee that the packet will be received by (l,m) at a cycle (i+delay), but may be received a few cycles earlier or later), and (iii) it takes only a few cycles for a router to make a decision which way to send each packet, but the packet transfer can take up to 60 cycles (depends on the hardware used), which means that packets spend the majority of time not in the nodes, but in the links. Thus, packets' statuses and locations remain fixed for 96-98% of clock cycles.

For those reasons we introduce *ticks*, an abstraction for representing discrete time slots over which the network state changes. For example, if a node (2, 0) in a 10x10 network has a path of first being transferred to (1, 0) and then to (0, 0) which is a master node, then at the zeroth tick, it's position is (2, 0), first tick - (1, 0) and the second one - (0, 0). Each tick can be considered a rough equivalent to a number of cycles needed for a packet transfer, allowing it to be platform independent. Each packet will be stored in a different location in each tick, which is necessary and sufficient for building a system of boolean equations, a solution to which (i.e. an assignment of values to variables) will allow us to build an optimal routing table. Thus, a boolean constraint like the one below guarantees that a packet from node (0,2) is located in only one node at a tick 0.

$$P\_0\_2\_T\_0\_N\_0\_0 + P\_0\_2\_T\_0\_N\_0\_1 + P\_0\_2\_T\_0\_N\_0\_2 + P\_0\_2\_T\_0\_N\_1\_0 + P\_0\_2\_T\_0\_N\_1\_1 + P\_0\_2\_T\_0\_N\_1\_2 + P\_0\_2\_T\_0\_N\_2\_0 + P\_0\_2\_T\_0\_N\_2\_1 + P\_0\_2\_T\_0\_N\_2\_2 = 1$$

Using the concept of ticks, the total number of boolean variables ILA Code Generator will use is $\#VARS_{total} = (N*M-1)(N*M)*Ticks,$ because there are $N*M-1$ packets, each of which can be in any of $N*M-1$ nodes at any tick.

4.3.1.2 BASICS MECHANICS OF 2D TORUS MESH NETWORK IN ILA CODE

We begin with the basic packet movement possibilities in a form of a system of boolean equations or inequalities. Specifically, we need to make sure that ILA solver won't be able to find a satisfying assignments to $\#VARS_{total}$ boolean variables that would violate the following four types constraints:

1. Every packet can only be in one node during each tick. For each packet coming from a node (i,j), and for every tick k such a constraint would look like $\sum_{l,m=0}^{Rows,Cols} P\_i\_j\_T\_k\_N\_l\_m = 1$. The number of such constraints needed to describe an NxM network with Ticks ticks in a control period is *(N*M-1)*Ticks*.
2. Every packet which didn't reach the master node yet must move to one of the four neighbor nodes every tick. For each packet coming from a node (i,j), located at a node (l,m) at a tick k neighbors of which are (l1,m1), (l2,m2), (l3,m3), (l4,m4) such a constraint would look like

$$P\_i\_j\_T\_(k+1)\_N\_l1\_m1 \; + \; P\_i\_j\_T\_(k+1)\_N\_l2\_m2 + P\_i\_j\_T\_(k+1)\_N\_l3\_m3$$
$$+ P\_i\_j\_T\_(k+1)\_N\_l4\_m4 \; - \; P\_i\_j\_T\_k\_N\_l\_m \; \geq 0.$$

The first four terms represent packet (i,j) being in one of the neighbor nodes at the tick k+1, the last term has a negative sign and represents packet (i,j) being in node (l,m) at tick k. If the last term is True, then one of the previous four terms has to be true in order to make the whole left hand side be greater or equal to zero, which means the packet (i,j) has to move to another node on the next tick. If the last term, however, is false (i.e. packet (i,j) was not in a node (l,m) at a tick k), then the values of the first four terms don't matter: they can be either True or False, the inequality will always be satisfied. The number of such constraints needed to describe an *NxM* network with *Ticks* ticks in a

control period is *(NxM-1)*(NxM-1)*(Ticks-1)*: every packet can be at any node at any tick apart from the last one since by that time all packets need to reach the master node.

3. Every packet should start at its source node at the very first tick. Such constraints should look like this *P_i_j_T_0_N_i_j =1* and there are *N*M-1* such constraints.
4. Every packet should reach the master node at the final tick. Such constraints should look like this *P_i_j_T_(Ticks-1)_N_0_0 =1* and there are *N*M-1* such constraints.

4.3.1.3 LOAD BALANCE CONSTRAINTS IN ILA CODE

One of the problems our algorithm needs to solve is to create a routing table that achieves load balance. One of the first hypotheses of creating a perfect routing table is to achieve the load balance: the routing table would need to spread the packets across the network such that all the links would have to transfer the lowest amount of packets. With an all-to-one pattern and forcing minimum hop-count, the most congested area of the network is around the master node. For a 5x5 network, for example, each of the four links connected to the master node would ideally transfer (25-1)/4 = 6 packets. In other words, for a network with dimensions NxM, the ideal number of packets transferred by the links attached to the master node is $L_{ideal} = \left\lceil \frac{N \cdot M - 1}{4} \right\rceil$. The ILA Code Generating block knows the dimensions of the network and how many ticks are included in a control period. Provided that a node (l,m) is a neighbor of the master node (0,0), a product of two boolean variables *P_i_j_T_k_N_l_m * P_i_j_T_k+1_N_0_0* is True if and only if a packet (i,j) was in a neighboring node at a tick k and then reached the master node at a tick *k+1*. Such a situation should repeat no more than L_{ideal} times within a control period

for each master's neighbor. Thus, in order to achieve the load balance, the ILA should satisfy the following four constraints.

$$\sum_{k=0}^{Ticks-1} \sum_{i=0,j=0}^{Rows,Cols} P\_i\_j\_T\_k\_N\_0\_1 \cdot P\_i\_j\_T\_(k+1)\_N\_0\_0 \leq L_{ideal}$$

$$\sum_{k=0}^{Ticks-1} \sum_{i=0,j=0}^{Rows,Cols} P\_i\_j\_T\_k\_N\_1\_0 \cdot P\_i\_j\_T\_(k+1)\_N\_0\_0 \leq L_{ideal}$$

$$\sum_{k=0}^{Ticks-1} \sum_{i=0,j=0}^{Rows,Cols} P\_i\_j\_T\_k\_N\_(Rows-1)\_0 \cdot P\_i\_j\_T\_(k+1)\_N\_0\_0 \leq L_{ideal}$$

$$\sum_{k=0}^{Ticks-1} \sum_{i=0,j=0}^{Rows,Cols} P\_i\_j\_T\_k\_N\_0\_(Cols-1) \cdot P\_i\_j\_T\_(k+1)\_N\_0\_0 \leq L_{ideal}$$

Please note that the ILA code will always have only four such constraints no matter what the dimensions of the network are and how many ticks are included in a control period, because in a 2D torus mesh network, the master node has only four links.

4.3.1.4 LOAD BALANCE CONSTRAINTS IN ILA CODE

Suppose we have a part of some torus mesh network shown on Figure 4.5. At the very beginning of a control period nodes (1,3) and (Rows-1, 3) send their packets to the node (0,3). Both packets arrived on different channels so no conflict occurred. In the next tick, both packets must be transferred to (0,2) since the master node is located at (0,0). The link between (0,3) and (0,2) can't send two packets at the same clock cycles, so this will lead to one packet being temporarily blocked by the other. The packet which reaches (0,3) second will have to wait until the router finishes sending the packet which came to (0,3) first.

In an ideal deterministic and fully predictable scenario, the control period should start at the same time, and (0,3) would receive both packets at the same clock cycle and always let one of them go to (0,2) first. If the same scenario repeated every control period, then both packets would predictably arrive at (0,0) at approximately the same clock cycles which means the jitter would be low.

However, since (i) clocks of different nodes are not synchronized (which means nodes start sending packets to the master not at the same clock cycle at the beginning of each control period) and (ii) the packet transmission delay also has certain a variance (according to transceivers documentation and our practical experiments), this means that in a period i node (0,2) can receive packet from (0,3) first and immediately forward to (0,2), in which case the packet from node (ROWS-1, 3) will be significantly delayed, and in the next period i+1, node (0,2) can first receive packet from (ROWS-1, 3), and then the other one will be delayed. That creates packet latency jitter, i.e. unpredictability of when a certain packet arrives at a master node.

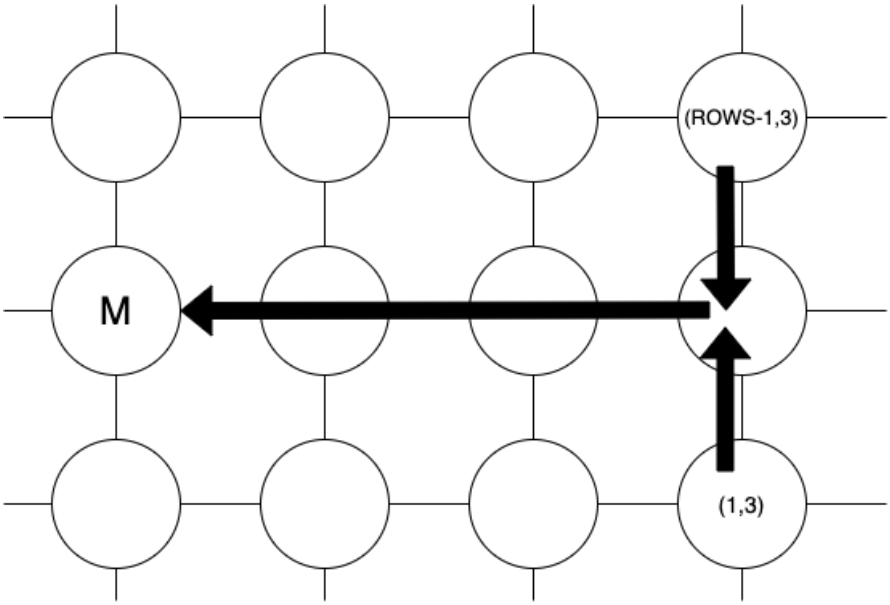


Figure 4.5 Two packets colliding illustration.

If we abstract out cycles and consider ticks, such a situation occurs whenever any two packets are scheduled to be in the same node at tick i, and then transferred to the same neighbor node at tick i+1. We will call such situations ***conflicts***. In order to minimize the jitter, ILA Code Generator needs to include constraints that will minimize (or even eliminate) such conflicts. In general terms, such constraints look the following way:

*P_i1_j1_T_k_N_l1_m1*P_i2_j2_T_k_N_l1_m1*P_i1_j1_T_(k+1)_N_l2_m2*P_i2_j2_T_(k+1)_N_l2_m2=0.*

The product on the left-hand side is True if and only if two packets are coming from (not necessarily neighbor) nodes (i_1,j_1) and (i_2,j_2) and both are located at a node (l_1,m_1) at a tick k, and then both packets are scheduled to be in a node (l_2,m_2) which is a neighbor of (l_1,m_1) at the tick k+1. Creating such constraints for all pairs of packets *((N*M-1)*(N*M-2)/2),* for all the *(N*M-1)* nodes where those packets can meet, for all the neighbors of those nodes (4), for all two consecutive ticks should eliminate conflicts, and thus, minimize the jitter. The number of such constraints is $\frac{(N\cdot M-1)(N\cdot M-2)}{2}\cdot(N\cdot M-1)\cdot 4\cdot(Ticks-1)$.

To prove that conflicts are the reason for packet latency jitter, we ran the following experiment: we generated a routing table for an 8x8 network with one conflict only and simulated it counting the number of stalls for each link. Here by stall we mean a cycle at which a link had to delay a packet because another one is being sent. The conflict was in the link connecting nodes (0,1) and (0,0). Such stalls were counted for every link. Most of the links had no stalls, all other links are shown in Table 4.1 that shows correlation between the number of conflicts shown by the ILA Solver and the number of

stalls counted by the Simulator. We can clearly see that the link which is expected to have more collisions than other links has to delay packets for the biggest amount of cycles.

Table 4.1 Number of conflicts and stalls for all links of an 8x8 network.

Link	Number of Conflicts	Number of Stalls
(1,0)-(0,0)	1	1896
(7,0)-(0,0)	0	1189
(0,0)-(0,1)	0	972
(0,7)-(0,0)	0	856
(1,0)-(2,0)	0	423
(0,1)-(0,2)	0	422
(6,0)-(7,0)	0	323
(0,2)-(0,3)	0	226
(2,0)-(3,0)	0	119
(0,1)-(1,1)	0	112
(6,7)-(7,7)	0	105
(1,3)-(2,3)	0	101
(0,7)-(1,7)	0	65
(5,0)-(6,0)	0	46
(1,6)-(2,6)	0	39
(6,0)-(6,1)	0	37

(6,6)-(6,7)	0	19

Another experiment we ran in order to prove that packet latency jitter increases as the number of conflicts increases is that we generated six routing tables with different amounts of conflicts for a 7x14 network. Each routing table was simulated, and the jitter was calculated. The results are shown in Table 4.2. Here we can clearly see that the routing table with no conflicts has the best jitter, and as the number of conflicts increases, the jitter gets bigger.

Table 4.2. Correlation between number of conflicts and jitter.

Number of conflicts	Jitter
0	1.297
28	1.318
56	1.749
128	3.899
168	4.148
208	6.869

4.3.2 ILA CODE OPTIMIZATIONS

While working on the routing table generation, we have noticed that the text files with ILA code may reach up to 4GB of size, and the ILA solver may take hours to find proper routing tables. At the same time, the constraints we described can include redundancies and room for optimization, e.g. at the 0th tick all the boolean variables have obvious values: all variables in a form of *P_i_j_T_0_N_i_j* are definitely True and all variables *P_i_j_T_0_N_l_m*, where i≠l or j≠m (or both) are definitely False. Thus, we decided to introduce the following optimizations:

1. Do not allow packets to move away from the master node. Since we want packets to follow minimum hop count paths, they should always move towards the master node. If a packet is located on the same row or column as the master node (e.g. (0,2) or (2,0) nodes), then there is only one way they should be able to move (to (0,1) and (1,0) respectively). If a packet is positioned anywhere else, then, with a few exceptions, there are two nodes a packet should be able to go to, e.g. a packet from (2,2) should be transferred to either node (1,2) or node (2,1), but not to (3,2) or (2,3).
2. Do not include conflicts in which packets can go away from the master node. Since we don't allow packets to go away from the master, it is impossible to have conflicts moving those directions. For example, two packets (3,2) and (2,3) can meet at the node (2,2), but then, since they can't be transferred to (3,2) or (2,3), ILA Code Generator doesn't have to generate constraints eliminating those conflicts.
3. Do not account for conflicts between the packets that come from sources located in different "zones". For every packet ILA Code Generator defines its "zone" - a nonempty finite set of nodes which a packet can possibly visit on its way to the master node. For example, a packet from a node (2,2) would have a "zone" consisting of nodes (1,2), (2,1), (2,0), (0,2), (1,1), (1,0), (0,1). Those are the only nodes which the (2,2) packet can ever visit no matter what minimum hop count path it takes. But it can never be in a node outside of its zone, like (*Rows*-1, 0), which means it can never have a conflict with any other packet there.

4. Eliminate constraints preventing impossible conflicts outside of packets' zone. Even if two packets are in the same "zone", they can't meet for a conflict in a node in a different zone. For example, packets (2,3) and (3,2) are in the same zone, so they can for sure collide, but not in nodes outside of their "zone", like (*Rows*-1, 0).
5. Do not include certain constraints allowing packets to move too far at the beginning of a control period. For every packet, it is possible to calculate what nodes it can reach at a certain tick. For example, a packet (2,2) can't be in nodes like (0,2) at the first tick, because it takes two hops to get from (2,2) to (0,2). Thus, *P_2_2_T_1_N_0_2* is definitely False, and constraints described in Chapter 4.3.1.2 are not needed.
6. Eliminate constraints preventing impossible conflicts at the beginning of a control period. Since a packet (2,2) can't be in nodes like (0,2) at the first tick, it can't meet any other packet there to have a conflict with. Thus, constraints avoiding such conflicts don't need to be generated.
7. Do not include certain constraints allowing packets to move at the end of a control period. For example, if a control period consists of 8 ticks (0th tick being the very first one and 7^{th} tick being the last one), then we definitely know that all the packets should be in the master node at the 7^{th} tick. That's why on the 6th tick packets can be no further away from the master than just one hop, on the 5th tick - two hops away, etc.
8. Eliminate constraints preventing impossible conflicts at the end of a control period. Similarly to previous optimizations, conflicts can't happen in nodes that

are more hops away from the master than the number of ticks left before the period ends. For example, since no packet can't be in node (0,2) at the 6th tick, no conflicts are possible at node (0,2) at the 6^{th} tick.

9. Eliminate constraints allowing packets to be "behind" its source. Since packets are required to follow non minimum hop count paths and not to go away from the master, they can never be in nodes "behind" its source. For example, a packet (0,1) can never be in the node (0,2), and variables like *P_0_1_T_k_N_0_2* are always False and constraints described in Chapter 4.3.1.2 are not needed.
10. Eliminate constraints preventing impossible conflicts for packets taking place behind their sources. Similar to the optimizations described before, if a packet can't be in the nodes behind its source, then it can't conflict with any other packet there. For example, packets (2,3) and (3,2) are in the same zone. The node (3,3) is behind both of them, so they can't meet there for a conflict.
11. Hardcode all variables at the very first tick and at the last ticks. At the 0th tick all packets are located in the source nodes, and at the last tick, all packets are required to be located in the master node. For example, *P_2_2_T_0_N_2_2* is definitely True, and all variables of the form *P_2_2_T_0_N_l_m* where $l \neq 2$ or $m \neq 2$ (or both) are definitely False. Similarly, *P_2_2_T_(Ticks-1)_N_0_0* has to be True, and all variables of the form *P_2_2_T_(Ticks-1)_N_l_m* where $l \neq 0$ or $m \neq 0$ (or both) are definitely False.
12. Hardcode all the variables describing the four closest to the master node packets. Obviously, packets (0,1), (1,0), (*Rows*-1,0), (*Cols*-1,0) have clear paths to the master node, and it's best to set *P_i_j_T_1_N_0_0* to True for all of them, and

P_i_j_T_k_N_l_m (where k>1 and l,m≠0,0) to False. Those four packets should not conflict with any other packets, and no ILA constraints described in Chapter 4.3.1.2 are necessary.

4.3.3 GENERATING ROUTING TABLES WITH NODE FAILURE

One of the objectives of this work is to give the ILA Code Generator the ability to create routing tables with a dead node. A dead node is not allowed to receive or transmit any packet which means four links attached to it are useless. We had to make the following changes to our software pipeline:

1. Both ILA Code Generator and ILA Solver now have an additional command line parameter: either "e" or "s" that stand for east and south respectively. Here we want to test two options: either the node (1,0) (to the south from the master node) or the node (0,1) (to the east from the master node) can be dead.
2. ILA Code Generator adjusts all the constraints that involve four nodes around the dead node. One of them is the master node, but the other three can't send any packets towards the dead node and are not allowed to have any conflicts following the broken links. For example, if the node (0,1) is broken, then nodes (0,2), (1,1), (Rows-1, 1) can't send packets to (0,1) and can't have conflicts where packets would go to the node (0,1) at the same tick. Those ILA constraints need to be removed.
3. Constraints for load balance also needed to be adjusted. The master node now only has three neighbors, so $L_{ideal} = \left\lceil \frac{NM-2}{3} \right\rceil$. Depending on which node is declared to be dead, one of the four constraints from Chapter 4.3.1.2 need to be removed.

4.3.4 ROUTING TABLE OPTIMIZATIONS

On top of optimizing the ILA code and getting rid of various redundancies to create fewer constraints and allow Gurobi spend less time solving our model, we found that for larger networks, if we allow the packets to only follow minimum hop count paths, it is impossible to avoid conflicts on the tick level which will turn into inconsistent packets latencies (i.e. jitter) on the cycle level. Thus, we decided to allow the ILA Code Generator to have the following two options:

1. Allow packets to follow non minimum hop count paths, i.e. if the packet is coming from the node (0,2), it will proceed to the master node (0,0) via node (0,1). Now, however, it may also proceed to the opposite direction for one or multiple hops: to (0,3) first (or, possibly to (1,2) or (Rows-1, 2)) and then turn back to get on a minimum hop count path from there. This will allow packets to avoid congested areas or time when certain areas get congested as well as use the links which are otherwise never used.
2. Allow the packets to be held in the source nodes for one or more ticks, and then follow only minimum hop count paths. For example, a packet from the node (0,2) doesn't have to rush to the master node as soon as the period starts, but it can be scheduled to wait for a few ticks in the source node and then go to (0,1). This way, packets are scheduled to avoid congested areas by being delayed in the source node.

4.4 ROUTING TABLE OPTIMIZATIONS

In general, ILA solvers allow a programmer to specify one or multiple objective functions and a set of constraints. The previous sections included examples of

constraints: a mathematical expression involving integer (boolean, in our case) variables on the left-hand side needs to be equal (=), less than (<), less than or equal (≤) greater than (>), greater than or equal (≥) to a fixed number on the right-hand side.

An objective function is the function on the decision variables that one wishes to minimize or maximize. Our objective pursues two goals: minimize packet latency jitter and ensure the load balance.

Three possible approaches arise: (i) set load balance as an objective function and let jitter minimization be constraints, (ii) set jitter minimization as an objective function and let load balance be constraints, and (iii) set both jitter minimization and load balance as objective functions. In the first method, Gurobi will first create a pool of solutions that make sure there are (exactly) 0 conflicts, then will start looking for the most optimal solution that has a perfect load balance. In the second approach, Gurobi will first create a pool of solutions that make sure there is a perfect load balance, then will start looking for the most optimal solution that has the lowest possible amount of conflicts. In the last method, Gurobi will explore the tradeoffs between the two objective functions. Gurobi allows you to blend multiple objectives, to treat them hierarchically, or to combine the two approaches. In a blended approach, we can optimize a weighted combination of the individual objectives [27].

Practical experiments showed that since load balance involves only 4 constraints while jitter minimization involves $\frac{(N\cdot M-1)(N\cdot M-2)}{2}\cdot(N\cdot M-1)\cdot 4\cdot(Ticks-1)$ ones, ILA solver has no issues with satisfying load balance constraints, but jitter minimization is much more challenging. Thuis, we made a decision to execute the second approach and

set the amount of conflicts as an objective function which the ILA solver needs to minimize.

4.5 ROUTING TABLE OPTIMIZATIONS

When the ILA solver is run, it first reads the .lp file with all the constraints and objects function(s), and then tries to solve the model giving an extensive output to the console. For our purposes, the most important part is a table shown on the Figure 4.6.

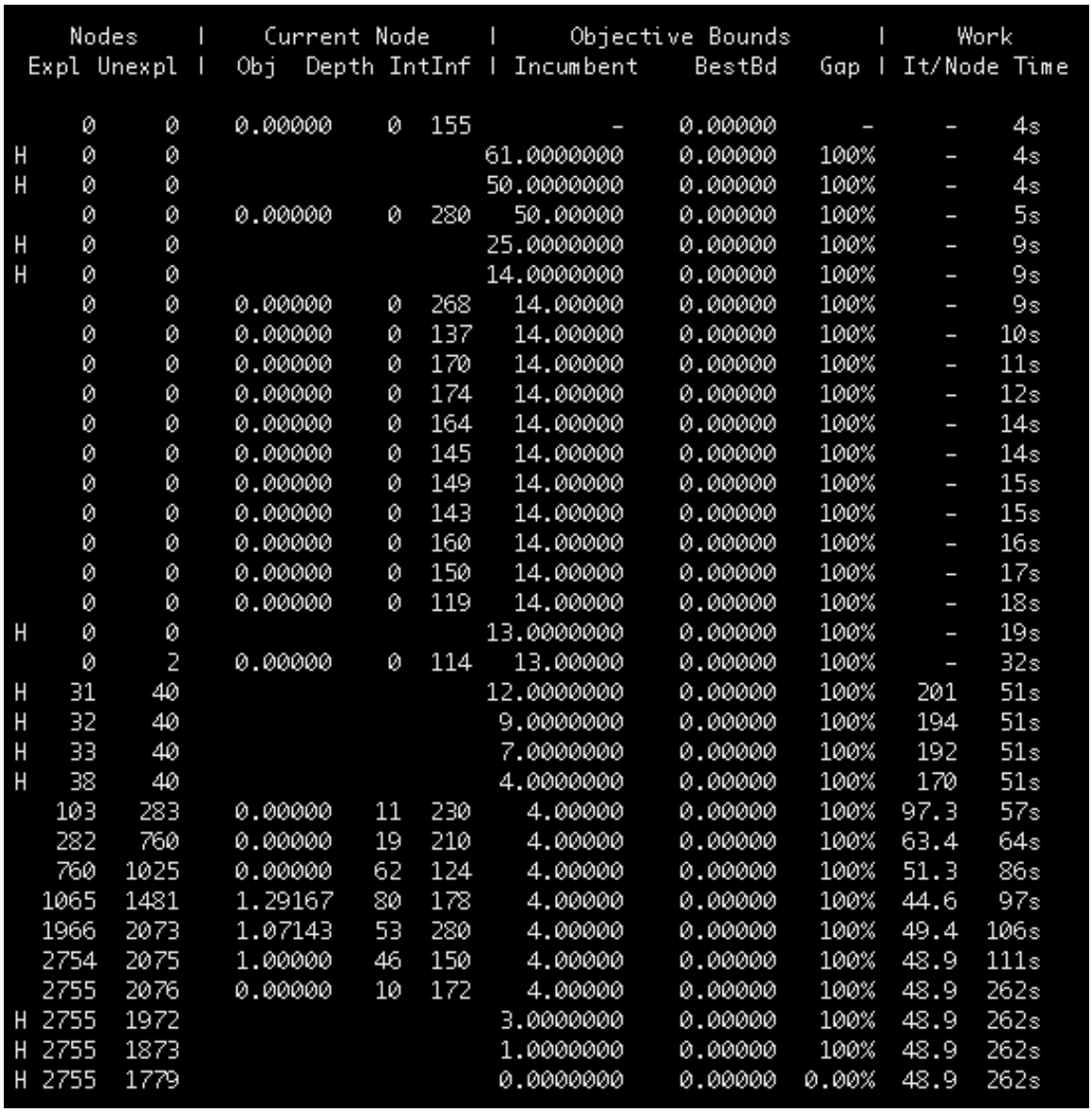
Here we are mostly interested in following columns:

1. Incumbent - the best value of the objective function achieved so far,
2. BestBd (best bound) - best known objective value for a feasible solution, i.e. the best possible value of the objective function,
3. Gap - relative gap between the two values,
4. Time - amount of seconds elapsed since the moment when the Gurobi started solving the current model.

One of the advantages of the Gurobi solver is that it has a lot of customized parameters which can be set to fit the user's particular purpose. Here are the most useful for our problem:

1. MIPGap - defined as |Zp - Zd|/|Zp|, where Zp is the incumbent objective value (i.e. the upper bound for minimization problems), Zd - the lower bound for minimization problems. When set to 1.0, Gurobi solver will stop once it found the first feasible solution no matter what the value of the objective function is. When set to 0 or 1e-4 (which is a default value), it stops the solver only when the best objective value is achieved. That can be useful in case we need to grab the first available routing table (regardless of how many conflicts it contains). When set to

0.5, we can potentially achieve a good balance between time spent deriving a routing table and amount of conflicts in that routing table. For all the experiments discussed further in this dissertation it was set to 0.



	Nodes		Current Node			Objective Bounds			Work	
	Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
	0	0	0.00000	0	155	-	0.00000	-	-	4s
H	0	0				61.0000000	0.00000	100%	-	4s
H	0	0				50.0000000	0.00000	100%	-	4s
	0	0	0.00000	0	280	50.00000	0.00000	100%	-	5s
H	0	0				25.0000000	0.00000	100%	-	9s
H	0	0				14.0000000	0.00000	100%	-	9s
	0	0	0.00000	0	268	14.00000	0.00000	100%	-	9s
	0	0	0.00000	0	137	14.00000	0.00000	100%	-	10s
	0	0	0.00000	0	170	14.00000	0.00000	100%	-	11s
	0	0	0.00000	0	174	14.00000	0.00000	100%	-	12s
	0	0	0.00000	0	164	14.00000	0.00000	100%	-	14s
	0	0	0.00000	0	145	14.00000	0.00000	100%	-	14s
	0	0	0.00000	0	149	14.00000	0.00000	100%	-	15s
	0	0	0.00000	0	143	14.00000	0.00000	100%	-	15s
	0	0	0.00000	0	160	14.00000	0.00000	100%	-	16s
	0	0	0.00000	0	150	14.00000	0.00000	100%	-	17s
	0	0	0.00000	0	119	14.00000	0.00000	100%	-	18s
H	0	0				13.0000000	0.00000	100%	-	19s
	0	2	0.00000	0	114	13.00000	0.00000	100%	-	32s
H	31	40				12.0000000	0.00000	100%	201	51s
H	32	40				9.0000000	0.00000	100%	194	51s
H	33	40				7.0000000	0.00000	100%	192	51s
H	38	40				4.0000000	0.00000	100%	170	51s
	103	283	0.00000	11	230	4.00000	0.00000	100%	97.3	57s
	282	760	0.00000	19	210	4.00000	0.00000	100%	63.4	64s
	760	1025	0.00000	62	124	4.00000	0.00000	100%	51.3	86s
	1065	1481	1.29167	80	178	4.00000	0.00000	100%	44.6	97s
	1966	2073	1.07143	53	280	4.00000	0.00000	100%	49.4	106s
	2754	2075	1.00000	46	150	4.00000	0.00000	100%	48.9	111s
	2755	2076	0.00000	10	172	4.00000	0.00000	100%	48.9	262s
H	2755	1972				3.0000000	0.00000	100%	48.9	262s
H	2755	1873				1.0000000	0.00000	100%	48.9	262s
H	2755	1779				0.0000000	0.00000	0.00%	48.9	262s

Figure 4.6 Gurobi logging progress information on the branch-and-cut tree search.

2. TimeLimit - amount of seconds Gurobi is given to solve the model. Once that much time has elapsed, the ILA solver has to stop and show the best solution it was able to achieve (if any). For all the experiments discussed further in this

dissertation this parameter was not used, i.e. solver took any time it needed to produce the best routing table.

3. BestObjStop. Gurobi solver will terminate as soon as the engine finds a feasible solution whose objective value is at least as good as the specified value [28]. That is an extremely useful option when we want to test performance of routing tables with, let's say, 1, 10, or 100 conflicts. For all the experiments discussed further in this dissertation this parameter was not used either.

4.6 PACKET LATENCY JITTER MEASUREMENT

One of the challenges we encountered is to objectively measure jitter for a routing table, ideally, as one number which would make it easy for the performance comparison. Initially, we wanted to split packets into groups depending on how many hops away they are from the master node, and calculate jitter for every group individually.

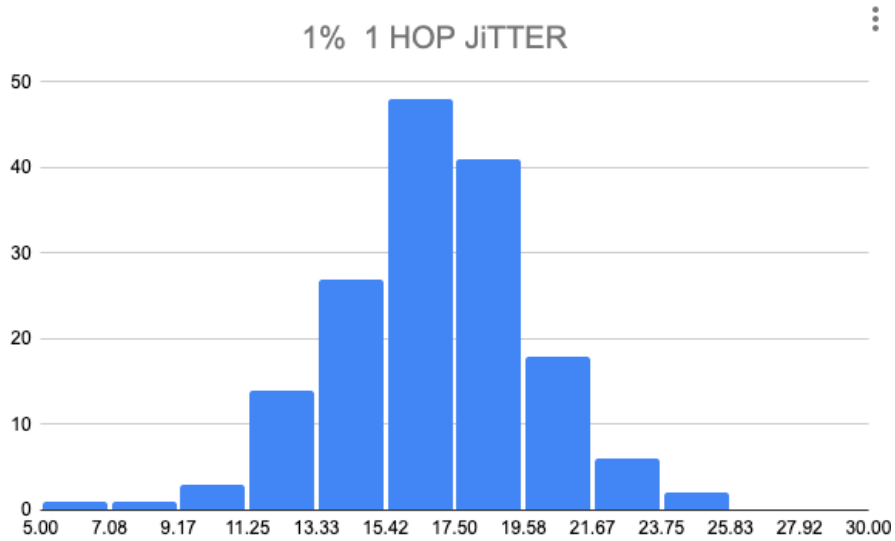


Figure 4.7 Simulation results for packets closest to the mater node

Figure 4.7 shows an example of such an experiment: 170 routing tables were simulated, jitter was calculated for all the packets, but only packets that are one hop away from the master node are included in the distribution. However, it makes routing tables

performance comparison cumbersome, because if one routing table is better in half of the groups, but the second routing table is better in the second half, it is difficult to determine a winner. Also, after allowing packets to be held in the source nodes for one or more ticks, dividing packets into groups solely based on their positions is unfair as it is possible two packets are located, let's say, one hop away from the master node, but one of them is scheduled to be in the network fo five ticks (wait for four, and then get transferred to the master node), and the other one is scheduled to go to the master node immediately. Thus, we made a decision to calculate the jitter of the entire network as a sum of packets jitters divided by their average latencies: $NetworkJitter = \sum_{i=1}^{N*M-1} \frac{Ji}{AvgLatencyi}$, where Ji – jitter of the packet i, and $AvgLatency_i$ – average latency of packet i.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 BASELINE

All the baseline work has been done in our own simulator written with Python and JavaScript using the Cytoscape JS library, which offers graphical representation of a graph including various animations and predefined graph functions such as Dijkstra's Floyd–Warshall, Bellman–Ford, etc. algorithms.

The simulator conceptualizes the network as a graph where vertices represent distributed FPGA boards and edges - fiber optical channels. The general algorithm of the simulator is the following:

1. The JavaScript part of the code generates routing tables for a specific network size with the restriction of packets being forced to route using minimum hop count. The node in the top left corner is considered to be the "master node", i.e. it receives the data packets from all other nodes and doesn't send packets itself.
2. If needed for certain experiments, the Python part of the simulator may analyze all of the routing tables and select the most interesting ones for further analysis.
3. Lastly, JavaScript and Cytoscape are used again to simulate the routing tables one by one and store the data about packets latency and jitter.

The simulator is capable of changing the clock jitter (i.e. how much FGPAs' clocks differ from one another), number of flits per packet, number of cycles it takes to

transmit a packet, number of cycles it takes an FPGA to process a packet and send it to the next node in a path, number and frequency of link synchronization cycles.

5.2 LOAD BALANCING EXPERIMENTS

One of the first hypotheses of creating a perfect routing table is to achieve the load balance: the routing table would need to spread the packets across the network such that all the links would have to transfer the lowest amount of packets. Obviously, with an all-to-one pattern and forcing minimum hop-count, the most congested area of the network is around the master node. For a 5x5 network, for example, each of the four links connected to the master node would ideally transfer (25-1)/4=6 packets. In other words, for a network with dimensions N x M, the ideal number of packets transferred by the links attached to the master node is $L_{ideal} = \left\lceil \frac{NM-1}{4} \right\rceil$.

5.2.1 LOAD BALANCING EXPERIMENTS

As a baseline, it was decided to randomly generate millions of routing tables for networks of different sizes and analyze how many of them have the property defined above. The experiment was held as follows:

1. 1,000,000 random routing tables were generated for each of the six network sizes. All the routing tables forced packets to follow the paths of minimum hop count.
2. For each routing table, the link with highest utilization was identified, which under the all-to-one traffic pattern is always found to be connected to the master node.
3. The number of packets traversing that link became a part of the distribution. Such numbers for all the routing tables altogether create final distribution.

4. All the simulations described above were carried out for networks of sizes 10x10, 14x7, 16x6, 25x4, 33x3.

With regard to load balance:

- For a 10x10 network, out of 1,000,000 networks, approximately 9,000 (or 0.9%) networks had the ideal load-balance of L_{ideal}=25.
- For a 7x14 network, only 500 networks were balanced.
- As the aspect ratio increases, no balanced tables are found among the one million routing tables.
- For example, for a 3x33 network, the lowest most busy link was found with the load of 35 packets (although L_{ideal}=25).

Table 5.1 shows the percentage of ideal routing tables (routing tables with the maximum link load being L_{ideal}) out of a million of routing tables generated for each network. For networks with aspect ratio greater than 14:7 (2:1), no routing table is found, which illustrates the difficulty of the problem proposed. Also, we can note that the "more rectangular" the network gets, the bigger the larger link load becomes, which means the network becomes increasingly less balanced.

Table 5.1 Load balance experiments.

	10x10	7x14	6x16	4x25	3x33
L_{ideal}	25	25	24	25	25
Maximum link load found	25	25	29	32	35
Percentage of routing tables with L_{ideal}.	0.9%	0.05%	0%	0%	0%

Percentage of routing tables with L_{ideal} from the new methodology	100%	100%	100%	100%	100%

Figure 5.1 shows a sample distribution of maximum link loads for 1,000,000 randomly generated routing tables. The same bell-like curves were found for other network sizes, the only difference is how far to the left the curve is shifted: for 10x10 network, it is located very close to the origin, and the bigger the aspect ratio is, the further to the right the curve is located.

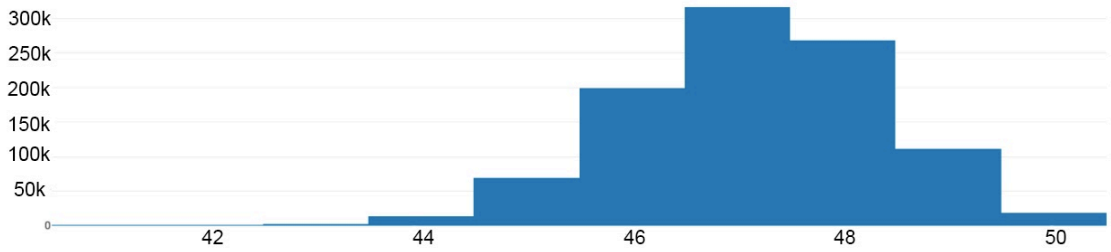


Figure. 5.1. Maximum load per link distribution for a 50x2 2D mesh torus network for random routing tables.

5.2.2 LOAD BALANCING EXPERIMENTS WITH NODE FAILURE

Similar experiments were carried out (Figure 5.2) for networks with one node being inactive - it was considered not being able to receive or send any data or, to put simply, removed (or dead). It is clear that the further away the node is from the master node, the fewer packets need to be rerouted and less impact it brings to the network routing in general. Thus, it was decided to remove the nodes adjacent to the master. It was also noted that for rectangular networks, removing the node along the shorter

dimension brings more impact. That is why the experiments were carried out for two possible positions of the dead node: to the east (E) and to the south (S) directions from the master node. The ideal number of packets for the three links attached to the master node is no $L_{ideal} = \left\lceil \frac{NM-2}{3} \right\rceil$.

Table 5.2 shows the experiment results. The dead node location is displayed in the second row, L_{ideal}, L_{real} are displayed in the third and fourth rows respectively. If $L_{ideal} < L_{real}$, then no proper routing tables were found and the last row shows 0%. Simulating randomly generated routing tables brings us to the following conclusions:

1. Deleting a node from a squared network brings almost no impact on load balance.
2. Unless the network is squared, deleting a node from a longer dimension is more advantageous for load balancing. The reason for that is the network has to route more data packets across the links which are less popular when there are no failures. Moreover, deleting a node from a shorter dimension makes it more difficult to find load balanced routing tables during random generation.
3. In general, for all network sizes, there are more load balanced routing tables found for the node failure simulations compared to simulations for non-faulty networks.
4. The best result was shown by 10x10 matrix - only 0.9% of all the routing tables had perfect balanced load which makes it a difficult problem to solve especially on an embedded processor and impossible to use on an US Navy military ship.
5. As well as with the previous experiments, the new approach involving ILA code always produces a routing table with perfect load balance.

Table 5.2 Routing tables with node failure generation comparison.

	10x10		7x14		6x16		4x25	
Dead node location	E	S	E	S	E	S	E	S
L_{ideal}	33	33	33	33	32	32	33	33
L_{real} from random routing tables	33	33	33	33	32	32	33	33
Percentage of routing tables with L_{ideal}	0.9%	0.9%	2.5%	0.003%	6.6%	0.0008%	0.1%	0.0001%
Percentage of routing tables with L_{ideal} from the new methodology	100%	100%	100%	100%	100%	100%	100%	100%

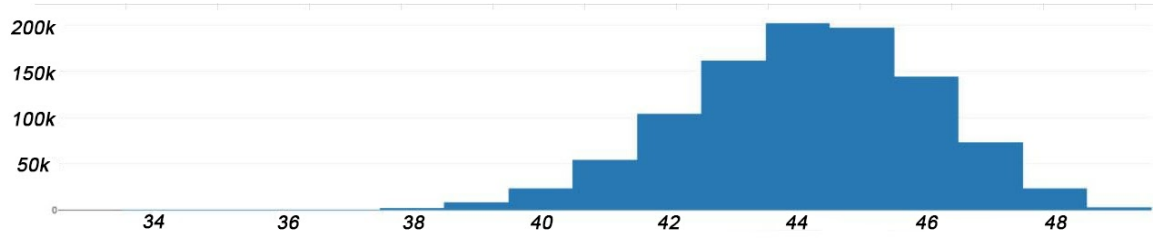


Figure 5.2 Maximum load per link distribution for a 3x33 2D mesh torus network.

Figure 5.2 shows one of the examples of distributions which is similar to previous simulations - bell curve shifted more to the right for rectangular networks. The new

approach still shows perfect results: due to the constraints in ILA code, all routing tables generated have the perfect load balance (last row of Table 5.2).

5.3 PACKET LATENCY JITTER EXPERIMENTS

The other problem this dissertation seeks to solve is jitter minimization for the latency of data packets arrival to the master node. Throughout carrying out numerous experiments, we found that a randomly generated routing table with balanced load with respect to the master node doesn't have any advantage in terms of jitter compared to any other randomly generated routing table.

Figure 5.3 separates such routing tables by color: the balanced routing tables are colored in shades of blue and others are colored in red. Such distribution clearly shows that creating a balanced routing table doesn't help with the jitter. We have received the same distributions for other hop counts as well. That brought us to the conclusion that we should not separate the routing tables into two different groups and ought to continue the research giving the same priority to all of them.

The next simulations were run with the following parameters:

1. All routing tables are randomly generated for a 6x12 network.
2. 200 randomly generated routing tables were simulated.
3. Each routing table was simulated for 70 periods and three clock jitter values: 1%, 8% and 15%.
4. The period was set to 900 clock cycles. All routing tables were simulated for 70 periods
5. Each packet is split into 16 flits. It takes 35 cycles to transfer one flit through a link, only one flit is transmitted every cycle. Also, links support pipelining: if the

first flit leaves a node on cycle #1, then the second flit can do the same on cycle #2, and 16th flit will be set for transmission on cycle #16, and the first node will appear on the "other side of the link" on cycle #17, second - #18, sixteen - #52. Since the network only works with "store-and-forward" routing (as opposed to wormhole routing), it doesn't allow a packet to be retransmitted to the next node until it is fully received by the current node.

6. There is no packet interleaving, i.e. flits from different packets can not be mixed up while being sent to the next node. For example, if packet #1 is received by some node at the ith cycle, and packet #2 is received by the same node at the (i+1)th cycle, and they both need to be forwarded to the North, then packet #1 sends the first flit on the (i+1)th cycle and the other 15 cycles right after that. Packet #2 gets an opportunity to send its first flit only on the (i+17)th cycle because the link will be busy sending flits from packet #1 all the cycles before that.
7. The first flit of a packet may be forwarded to the next node only on the next cycle after the entire packet has been fully received by the current node.
8. The jitter is calculated for every packet individually, as described in Chapter 4.6.

The results of the simulation are shown at the three different figures below (5.3, 5.4, 5.5). The first one had a clock jitter set to 1%, second - 8%, third - 15%. After that, we generated three routing tables via the proposed methodology using ILA solver. Routing table #1 forces packets to follow minimum hop count paths, routing table #2 also forces packets to follow minimum hop count paths, but allows packets to be held in the source nodes for a few ticks. Routing table #3 forbids packets to be held in any nodes, but

allows to follow non minimum hop count paths. These three tables were simulated in exactly the same conditions as random routing tables. The results are presented in Table 5.3.

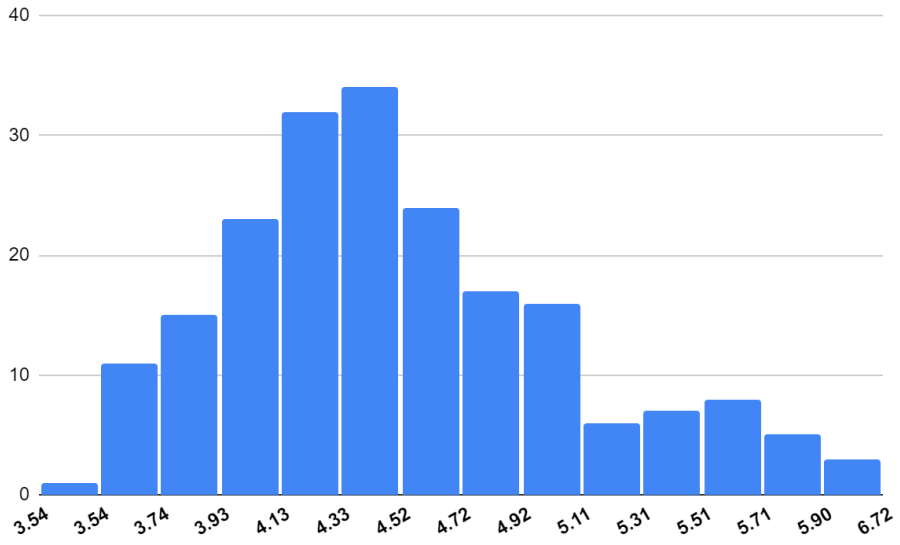


Figure 5.3 Random routing tables jitter distribution. Clock jitter is set to 1%.

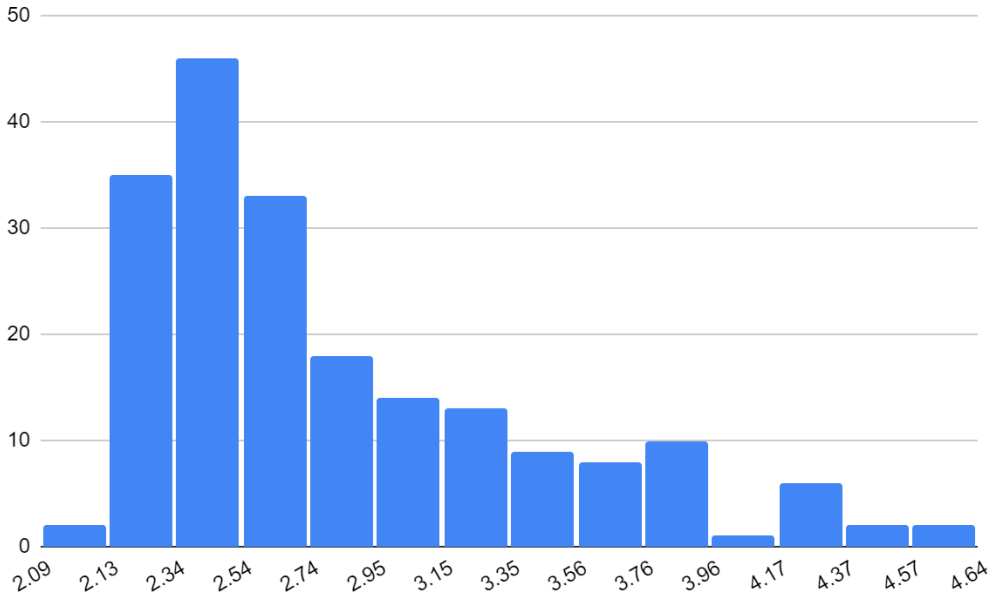


Figure 5.4 Random routing tables jitter distribution. Clock jitter is set to 8%.

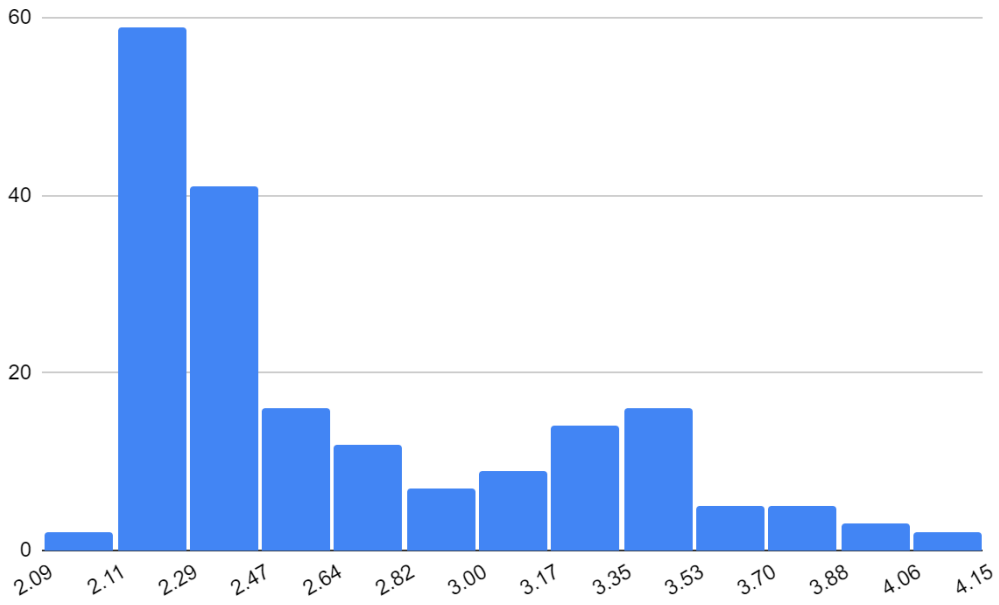


Figure 5.5 Random routing tables jitter distribution. Clock jitter is set to 15%.

Table 5.3 Simulation results of routing tables generated by the new methodology.

	Routing Table #1	Routing Table #2	Routing Table #3
Number of conflicts	52	0	0
1% clock drift	3.52	0.81	1.37
8% clock drift	2.02	0.96	1.74
15% clock drift	2.05	0.97	1.61

After analyzing the simulation results, we came to the following conclusions:

1. All three distributions have a shape similar to a bell curve.
2. Very few routing tables are located on the far left side which makes the problem of finding an optimal routing table difficult to solve.
3. All three routing tables show better packet latency jitter results than randomly generated routing tables.

4. Out of three proposed routing tables, the first one has the worst jitter, because for 6x12 network it is impossible to avoid conflicts if we force packets to follow minimum hop count paths. In fact, ILA Solver showed that the lowest possible amount of conflicts is 52, which it was able to successfully achieve. The other two routing tables don't have conflicts, and, thus, have lower jitter.
5. The second proposed routing table has better jitter than the third one as holding packets in the source nodes entails more predictability than spending that time transferring packets to other nodes that have a different unsynchronized with the other nodes clock. Also, packet transfer time is not a fixed amount of cycles, it has a small variance which accumulates with additional hops.

5.4 TIME EFFICIENCY

One of the objectives for the current research is not only to develop a method to generate an optimal routing table, but also estimate time efficiency for our software pipeline. There are two time consuming steps in our code generating procedure: generating ILA code and solving an ILA model. Table 5.4 shows the amount of time in seconds both of those steps take. All the measurements were made on a Macbook Air (2.2GHz Intel Core i7, 8GB 1600MHz DDR3 RAM, Intel HD Graphics 6000 1536 MB). The experiment was carried out only for the routing table generating algorithm which allows packets to be held in source only.

Table 5.4 Correlation between the size of the network and time to generate routing table.

	No dead node - holdinsource							
	3x3	4x4	5x5	6x6	7x7	8x8	9x9	10x10
Generating LP file	0	0	0	2	11	29	65	1017
Solving lp model	0	0	0	6	19	149	3304	3304

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, we presented a novel way of generating optimal routing tables for 2D Torus Mesh networks with all-to-one static traffic pattern. Such networks are the most natural way for the entire control power system to send all the data to the single centralized (master) node which makes all the decisions about the control system and sends the proper instructions to all other nodes back within the required time period. Since none of the previously developed algorithms simultaneously optimize load balance and jitter, our goal was to predictably generate routing tables with better load balance and packets latency jitter than the best randomly generated routing tables.

We base our approach on global optimization using integer linear programming (ILP) software kit, specifically Gurobi. ILP solver optimizes objective function achieving the best network's routing.

Using the software pipeline built in this dissertation this work makes the following contributions:

1. Our software supports routing table generation for any size and any dimensions of a 2D Torus Mesh network, including cases when one of the master node's neighbors has failed and can not send, receive or transfer any packets.
2. The output routing tables are guaranteed to have ideal load balance, meanwhile the random routing table generator (taken as a baseline) failed to create perfect

balance for certain network dimensions even after generating a million of routing tables.

3. Our software can generate four types routing table where packets either (i) rush to the mater node following minimum hop count paths only, or (ii) follow non minimum hop count paths to avoid congested areas, or (iii) are scheduled to be delayed in the source node and then follow minimum hop count paths, or (iv) can be delayed in the source node and then follow minimum hop count paths. That way by the time packets reach congested areas, the links between the node already become available.
4. For the approaches (ii), (iii) and (iv), our software produces routing tables with up to three times less jitter, and with no worse jitter than the best random routing tables for the first approach.

6.1 FUTURE RESEARCH DIRECTIONS WORK

Future work could add more flexibility to our software pipeline as well as account for more network types:

1. Currently, our software works only with 2D Torus Mesh networks as it is mostly commonly used in Power Electronics Building Blocks. Adding support to other topologies, such as hypercube and ring will make our software more versatile.
2. In this dissertation we allowed only one node in a network to fail. It may be useful to consider multiple dead nodes located in different positions and investigate the problem of load balance and jitter in such cases.
3. As of now, our software supports generating and simulating routing tables with store-and-forward routing, as it was required by the practical application

underlined by the US Navy. However, wormhole routing is considered to be a more up-to-date routing technique. Hence, generating routing tables with wormhole routing will be a necessary update for future applications.

4. There are routing algorithms in the literature that may potentially be adapted to generate static route schedules and have potential to control load balance and jitter for periodic traffic patterns. Specifically, Ramanujam and Lin [11], proposed a W2TURN routing technique based on a weighted random selection of paths that contain at most two turns. Olson and Shin [12] created an algorithm which assigns all packets one of two different modes: detour mode for the packets whose shortest paths are blocked by faulty or congested regions and free mode for the ones which should be on their shortest path to the destination. If the packet is in the detour and it can not proceed, it should be transmitted following the link which is immediately counterclockwise of the link by which the message entered the node. Both of those algorithms, if adapted, can help optimize jitter and load balance even better.

REFERENCES

1. I. Panchenko, J. D. Bakos and H. L. Ginn, 2017. *Control system communication architecture for power electronic building blocks*. IEEE Electric Ship Technologies Symposium (ESTS), Arlington, VA: 544-550

2. Wang, Yang & Aksöz, Ahmet & Geury, Thomas & Ozturk, Salih & Kivanc, Omer & Hegazy, Omar. 2020. *A Review of Modular Multilevel Converters for Stationary Applications.* Applied Sciences.

3. H. Ginn, J.D. Bakos. 2018. *Fast Coordination of Power Electronic Converters for Energy Routing in Shipboard Power Systems*.

4. Debnath, J. Qin, B. Bahrani, M. Saeedifard and P. Barbosa. 2015. *Operation, Control, and Applications of the Modular Multilevel Converter: A Review*. IEEE Transactions on Power Electronics, vol. 30, no. 1: 37-53.

5. J. Duato. 1996. *A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks* . IEEE Transactions on Parallel & Distributed Systems, vol. 7, no. 08: 841-854.

6. R. Ramanujam, Bill Lin. 2010. *Destination-based adaptive routing on 2D mesh networks.* 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2010), La Jolla, CA: 1-12.

7. Basu, Anindya & Lin, Alvin & Ramanathan, Sharad. 2003. *Routing Using Potentials: A Dynamic Traffic-Aware Routing Algorithm*: 37-48.

8. A. Gupta, A. Singh, B. Towles and W. Dally. 2003. *GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks.* Computer Architecture, International Symposium on, San Diego, California: 194.

9. T. Kimura and M. Muraguchi. 2017. *Buffer management policy based on message rarity for store-carry-forward routing*. 23rd Asia-Pacific Conference on Communications (APCC), Perth, WA: 1-6.

10. W. Binqiang, H. Hongchao, G. Hong and B. Youjun. 2010. *A Multi-next Hop Routing Algorithm Based on Spanning Tree*. International Conference on Multimedia Communications. Hong Kong: 120-122.

11. B. Lin and R. Sunkam Ramanujam. 2009. *Weighted Random Routing on Torus Networks*. IEEE Computer Architecture Letters, vol. 01: 1-4.

12. A. Olson and K. Shin. 1994. *Fault-Tolerant Routing in Mesh Architectures*. IEEE Transactions on Parallel & Distributed Systems, vol. 5, no. 11: 1225-1232.

13. X. Yuan and S. Mahapatra. 2014. *Static load-balanced routing for slimmed fat-trees. Journal of Parallel and Distributed Computing.* Volume 74, Issue 5: 2423-2432.

14. I. Cidon and A. Kolodny, I. Walter and R. Manevich. 2010. *Centralized Adaptive Routing for NoCs.* in IEEE Computer Architecture Letters, vol. , no. 02, pp. 57-60, 2010.

15. J. M. Ferreira et al. 2016. *Efficient traffic balancing for NoC routing latency minimization*. IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC: 2599-2602.

16. Michel A. Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. 2009. *Application-aware deadlock-free oblivious routing*. In Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09). Association for Computing Machinery, New York, NY, USA, 208–219.

17. D. Bovet, J. Brzezinski, G. Gambosi, J. Blazewicz and M. Talamo. 1994. *Optimal Centralized Algorithms for Store-And-Forward Deadlock Avoidance*. IEEE Transactions on Computers, vol. 43, no. 11: 1333-1338.

18. Zhen Zhang, A. Greiner and S. Taktak. 2008. *A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip*. 45th ACM/IEEE Design Automation Conference, Anaheim, CA: 441-446.

19. T. Schonwald, J. Zimmermann, O. Bringmann and W. Rosenstiel. 2007. *Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip Architectures*. 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, Lubeck: 527-534.

20. W. Dally and B. Towles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

21. Wikipedia. “Torus interconnect”. https://www.wikiwand.com/en/Torus_interconnect

22. Freire, Mario & Silva, Henrique. 2001. *Performance Comparison of Wavelength Routing Optical Networks with Chordal Ring and Mesh-Torus Topologies*. 358-367.

23. Wikipedia. “Message switching”. https://en.wikipedia.org/wiki/Message_switching

24. S. Debnath, J. Qin, B. Bahrani, M. Saeedifard and P. Barbosa. 2015. *Operation, Control, and Applications of the Modular Multilevel Converter: A Review*. IEEE Transactions on Power Electronics, vol. 30, no. 1: 37-53.

25. K. Kredo II, P. Djukic and P. Mohapatra. 2009. *STUMP: Exploiting Position Diversity in the Staggered TDMA Underwater MAC Protocol*. IEEE INFOCOM 2009, Rio De Janeiro, Brazil: 2961-2965.

26. Ramesh B., Gururaj H. L.. 2015. "*Network Performance Comparative Analysis of Torus and Modified Mesh Interconnections with Source Routing for Packet Loss*. International Journal of Computer Theory and Engineering vol. 7, no. 4, pp. 273-277, 2015.

27. Gurobi. “Multiple Objectives”. https://www.gurobi.com/documentation/9.1/refman/multiple_objectives.htm

28. Gurobi. “BestObjStop”. https://www.gurobi.com/documentation/9.1/refman/bestobjstop.html