DEPARTMENT OF COMPUTER SCIENCE

Universal Routing in Processor Networks

James Hanlon

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

James Hanlon, May 2009.

Abstract

Traditional serial approaches to improving the computational capacity of computers have become bounded by physical constraints such as the speed of light and quantum affects. Parallelism is now the dominant paradigm in computer architecture, and increasing parallelism is now the primary method of improving the computational performance of computer systems. *Interconnection networks* are key to the performance of these systems as they mediate communication between the components. Interconnection networks are emerging more generally though, as a universal solution to system-level communication in digital systems. Consequently, optimisation of their performance is becoming increasingly critical.

The design of many interconnection networks are based around regular network topologies, but it is now becoming more important to consider networks where the structure has an *irregular* or *non-uniform* construction. Irregular structures may be produced by hardware faults or dynamic changes. There are also many networks that are inherently irregular, such as the Internet. In such cases, the routing algorithm, which is the way messages are directed from a source to a destination, must be able to perform effectively despite the irregularity. In particular, *universal* routing algorithms are designed to perform well, independent of the network topology.

This project aims to give a comprehensive overview of the current approaches to routing in irregular networks. It will do this by presenting a detailed theoretical and empirical analysis of the current best performing universal routing algorithms, specifically those representing the most recent development in the use of *virtual channels* and *turn prohibition*. The empirical analysis is based on experimentation with a software simulation tool developed for this project. It will evaluate the performance of algorithms on a range of topologies from regular meshes to random irregular classes with properties such as clustering.

The following points outline the main achievements of this project.

1. Implementing a complex network simulation tool to evaluate the performance of different routing algorithms on different topologies under various network parameters.
2. Implementing two of the best-performing universal routing algorithms and identifying errors in their descriptions and several weaknesses in their design.
3. Application of the dual graph representation in the Segment-Based routing algorithm to calculate shortest paths with prohibited turns.
4. Proposals of an enhancements to Segment-Based routing's algorithm for the discovery of segments in irregular graphs, and to LASH-TOR's method of balancing load between virtual layers.
5. Conducting a detailed empirical investigation into the behaviour and performance of the Segment-Based and LASH-TOR routing algorithms. Novel aspects of this were comparison on regular topologies with the topology-specific Dimension-Order routing algorithm and comparison with random graph classes possessing particular properties.

Contents

List of Figures iii

List of Tables v

1. Motivation and Background 1
- 1.1. Convergence Towards Parallel Architectures . . . 1
- 1.2. Importance of Interconnection Networks . . . 2
- 1.3. Parallel Architectures . . . 4
- 1.4. Irregular Interconnection Topologies . . . 5
- 1.5. Design and Nomenclature . . . 6
- 1.6. Aims & Objectives . . . 13

2. Topologies and Routing 15
- 2.1. Topologies . . . 15
- 2.2. Summary of Routing Approaches . . . 18
- 2.3. Routing Algorithms Chosen for Analysis . . . 20

3. Network Simulation 29
- 3.1. Simulation Approaches . . . 29
- 3.2. Simulator Architecture . . . 30
- 3.3. Performance of Interconnection Networks . . . 33
- 3.4. Simulation Measurements . . . 36
- 3.5. Testing & Verification of Implementation . . . 38

4. Implementation & Results 41
- 4.1. Implementation of Algorithms . . . 41
- 4.2. Simulation Experiments . . . 43

5. Evaluation 57
- 5.1. Successes . . . 57
- 5.2. Further Work . . . 58

Bibliography 60

A. Proofs 65
- A.1. Theorem 1 . . . 65
- A.2. Theorem 2 . . . 66
- A.3. Theorem 3 . . . 66

B. Simulator Manual 67
- B.1. Compilation and Running . . . 67
- B.2. Output . . . 67
- B.3. Configuration Parameters . . . 67
- B.4. Extensibility . . . 71

List of Figures

1.1. Example torus and butterfly topologies 8
1.2. Network deadlock and avoidance 11
1.3. Structure of a messages and packets 12

2.1. The structure of example mesh and torus topologies 16
2.2. Example irregular topologies 16
2.3. Channel dependency graph 21
2.4. Example Up*/Down* spanning trees 23
2.5. SR segments and turn restrictions 26

3.1. Input and output port structure 31
3.2. Router block diagram 32
3.3. Example throughput plot 38
3.4. Example latency plot 38

4.1. Results for 8-ary 2-mesh, 1 virtual channel 45
4.2. Results for 8-ary 2-mesh, 3 virtual channels 46
4.3. Results for 8-ary 2-mesh increasing virtual channels 47
4.4. Results for 8-ary 2-mesh with faults 48
4.5. Results for 8-ary 2-cube with faults 48
4.6. Results for a 32 node Erdős-Rényi graph 50
4.7. Results for a 64 node Erdős-Rényi graph 50
4.8. Results for a 32 node expander graph 51
4.9. Results for a 64 node expander graph 51
4.10. Results for a 32 node Barabási-Albert graph 52
4.11. Results for a 64 node Barabási-Albert graph 52

B.1. Example output for a simulation run. 68
B.2. Example simulator configuration file 72
B.3. Example topology construction using a `Construction` 73
B.4. Example explicit topology construction 73

List of Tables

1.1. Comparative delay and power costs in computation and communication 3

3.1. Traffic patterns 34

4.1. Random graph parameters 49
4.2. Comparison of SR segmentation algorithm 53

Chapter 1.

Motivation and Background

Routing is fundamental to the performance of many digital systems we rely on today. Its importance is motivated by two key areas: the use of parallel architectures in computing and of interconnection networks in digital systems, both of which are rapidly developing and becoming widespread in their applications. Interconnection networks have long been used in electronic systems such as telephone networks and more recently the Internet. With the rapid evolution of parallel computing architectures from desktops to supercomputers, their integral importance and wide ranging applications are becoming increasingly apparent.

The purpose of this initial chapter is to establish the significance of parallel systems in modern digital systems, and the consequent importance of interconnection networks as the basis for these systems. It will then introduce the fundamental aspects of interconnection networks: *topology*, *routing* and *flow control*, which are key to this project.

1.1. Convergence Towards Parallel Architectures

Computers are an essential part of our society. They exist in a multitude of forms, completing a wide range of tasks; from embedded applications such as GPS, to supercomputers used to predict future weather patterns. Their most pervasive use in our lives today is through the communication infrastructure that can be created with them, most notably the Internet. The speed of their development has been dramatic; the first machines resembling what today would be considered a computer emerged in the mid-20^{th} century (1940-1945). These occupied whole rooms, consuming huge amounts of power and provided only a fraction of the computational performance of a single modern desktop computer. We now find computers affecting most aspects of our lives.

The development of digital electronics, particularly the fabrication of microprocessors, has allowed more and more processing functionality to be included on a single integrated circuit (IC); where electronic components are manufactured on a single substrate of semiconductor material. Eventually in the 1970s, due to improvements in very large scale integration (VLSI) techniques, the entire computer's central processing unit (CPU) could be integrated into a single chip, which greatly reduced the costs involved in building computer systems. Since the 1970s, the increasing performance of processors has been known generally to follow Moore's Law [29] which states that the number of transistors that can be placed inexpensively on an IC increases exponentially, doubling every two years. This law has been seen to hold (though arguably through self-fulfilling prophecy) since the 1970s, until the present day where VLSI processes can fabricate microchips consisting of billions of transistors, achieving huge clock frequencies. This speed-up has been primarily driven by *bit-level parallelism*, where the amount of data used for computation each cycle is increased. The improvement in fabrication technology has lead recently to a new paradigm of *system-on-chip* (SoC) design where all components of a computer system are integrated into a single IC.

Processor design has developed around the execution of sequential code. It has been driven by increasing the instruction execution frequency as this relates directly to the run time of computationally bounded programs. The view at this time was that power was free and that

transistors were expensive. In order to maintain the trend of Moore's law predictions and the commercial success of processors, they have had to change radically. For example, achieving such high clock rates has only been possible by exploiting parallelism within a serial instruction stream, known as *instruction-level parallelism*. This has been realised with the idea of *super-scalar execution*, where multiple instructions are issued and executed in parallel.

Such innovations in processor design and fabrication techniques are limited and have not continued indefinitely. Only so much parallelism can exploited in an instruction stream, reducing VLSI scales mean that quantum effects start to come into play, clock frequencies start to become limited by the speed at which light can travel from one side of a chip to the other [16] and power consumption has become an important issue. The old thinking has been turned around, with transistors now being effectively free, but power being expensive. Around 2004, due to the combination of these issues, chip manufacturers began to feel it was necessary to change direction; away from the traditional ideas of serial execution and move towards more explicit ideas of parallelism as this was the only way to continue improving performance. Parallelism is now the dominant computer architecture paradigm and increasing parallelism is now the primary method of improving processor performance [2]. This change has been seen in the new generation Intel and AMD multi-core chips and in the development of graphics processing architectures.

Parallelism is not a new idea, and it has been employed in many different areas of computing, most commonly at bit and instruction levels. In particular though, it has been necessary since the 1960s to achieve high computational performance in supercomputer systems by running many processors concurrently to gain *data* and *task-level parallelism*. This approach has continued since then and has been classified more broadly by *high performance computing* (HPC) which includes distributed systems like *clusters* and *grids*, which for example support the massive developing web-infrastructure. Currently the top HPC systems in the world today have in excess of 100,000 cores [41]. The paradigm shift towards parallel architectures is significant, as there is a growing convergence in the ideas between personal computing and supercomputing.

The development of parallel computer systems raises a whole new set of new issues not found in single core systems. Many of these problems include important issues that apply to the whole spectrum of parallel architectures from multi-core desktop computers to massively parallel HPC systems, such as the ways in which communication and synchronisation between different cores and sub-tasks can be achieved. This project is motivated by the increasing importance of the use of parallel architectures in computing and digital systems.

1.2. Importance of Interconnection Networks

Communication is the process of moving data from one location to another and is achieved by the use of an *interconnection network*. Interconnection networks are used in a wide range of digital systems, with the most common uses in processor-memory interconnects, input/output device connections and in packet switching fabrics. Packet switching fabrics for instance, are used by modern telephone networks and have been widely studied for decades. More recently, rapid developments have been made by research focused on interconnection networks for new generations of multi-computer systems. This work has started to find applications in other areas such as local area networks (LANs) and as an alternative for back-plane buses, creating the concept of system area networks (SANs) [22]. The use of interconnection networks are emerging as a universal solution to system-level communication in digital systems [7]. They are now even becoming viable alternatives in IC design to routing dedicated wires, as routing packets on a system interconnect is more economical [10]. An example of this is with Intel's new QuickPath

Operation	Delay	Energy
32 byte register read	125ps	0.6pJ
23 byte ALU operation	650ps	0.3pJ
32 byte read from 8 Kbyte RAM	780ps	3pJ
32 byte transfer across 10mm chip	2300ps	17pJ
32 byte transfer off-chip	-	400pJ

Table 1.1.: Delay and power figures from Dally [9] contrasting the difference in cost between computation and communication.

Interconnect [21], a point to point processor link which also replaces the traditional Front Side Bus architecture. The key point is that interconnection networks are efficient solutions to these problems because the communication resources are shared between the components of the system, instead of existing as dedicated routes.

Independent of their application, interconnection networks have a set of common parameters that must be tailored to the requirement of the application. These include the number of communication nodes and their *average* and *peak bandwidth*, and the required *latency*. With the Internet, often high latencies can be tolerated; a HTTP request across the world may take several seconds, but this makes little difference for a human browsing the web, whereas high latency in a processor-memory interconnect massively effects the performance of the whole system. Interconnection networks must also be designed based on the information that is communicated; whether it can be packetised or must be streamed, what type of traffic behaviour is expected, the required *quality of service* (QoS) regarding the allocation of resources between packets. Important factors are also the *reliability*: a measure of how well the network correctly delivers messages, and the *availability*: the fraction of time the network is operating correctly.

Due to the increasingly widespread use of interconnection networks, the performance of most digital systems today has become limited by their capabilities. In computer systems, the bottle-neck factor has traditionally been due to memory accesses. This was because the rate of microprocessor improvement exceeded the rate of improvement in RAM technology. However, with parallel computer systems the limiting factor in terms of delay and power is due to the physical interconnections. Table 1.2 presents delay and power figures from William Dally [9] for 0.05μm fabrication showing the disparity between computational operations such as register reads and ALU operations and on and off-chip communications. In terms of delay, there is a 2:1 ratio for communication to operation delay, and for power it translates to a 56:1 communication to operation cost on chip and 1300:1 off-chip.

These ratios will continue to widen as the physical performance of interconnects doesn't scale with technology; the cost of sending signals down wires cannot be improved much, but the performance of memory and processors continues to develop rapidly. With the increasing number of cores in parallel systems the interconnect will increasingly dominate delay and energy. It is necessary therefore to architect efficient interconnection networks so to trade-off between factors like energy consumption and throughput. This requires thinking about factors such as *VLSI implementation costs*, *topology*, *routing* and *flow control*.

1.3. Parallel Architectures

The driving force behind much of the research into interconnection networks has been the design of parallel computer systems. These range widely in their forms and uses, but all share similar interconnection network designs, specifically in the use of *packet switching fabrics*. A packet switching fabric is a combination of hardware and software that connects nodes together in a network to allow communication between combinations of nodes. This is usually achieved with *crossbar switches* which act as a hub to mediate communications between computers, but also sometimes with *point-to-point links* (direct connections between computers).

Parallel systems can broadly be divided into the following classes:

- **Massively parallel.** A massively parallel processor (MPP) is a single computer consisting of many networked processors, they are often referred to as supercomputers. These are typically bespoke systems and designed for particular types of fine-grained highly computationally intensive tasks such as simulation of mathematical models for predicting weather or molecular modelling. They are usually built using special purpose processors and interconnects, an example of one is IBM's BlueGene/L supercomputer[1].

- **Clusters.** A cluster is a set of networked computers, commonly by a local area network (LAN). Cluster systems can be built from *off-the-shelf* commodity equipment, giving a high performance-to-cost ratio. Clusters are often used for less-tightly coupled computations, where the communication-to-computation ratio is low n nodes is lower, compared to traditional single-system supercomputers. Increasingly though, supercomputers are being built as clusters, due to developments of high performance interconnects such as Myrinet[2] and Infiniband[3]. Currently, of the world's top 100 supercomputers, 40 are clusters [41].

- **Grids.** Grid computing is the most distributed form of parallel computing and is optimised for loosely coupled computations, usually for many independent jobs. It is often a implemented as a form of cluster such as a data centre, but grids can also be realised over the Internet. They can be used to provide computing as a utility, or to compute so-called *embarrassingly parallel* problems where it is trivial to parallelise as little or no inter-process communication is required. For example the SETI@home[4] project has been successful as it can be run on idle desktop computers requiring communication only of data to work on and the result of the computation.

- **Multi-core.** A multi-core processor is one which contains multiple processing cores, allowing multiple instructions to be issued from multiple streams each cycle. The performance of such systems is limited by the extent to which the computation can be parallelised. Modern multi-processor systems are increasingly being designed as *network-on-chip* (NoC) architectures where processing cores are often arranged uniformly and are connected together by some packet switched network in some uniformly arranged topology such as a *mesh* or *hypercube*[5].

- **Special purpose.** Parallel architectures are also utilised in a range of specialised applications. One common example is *graphics processing units* (GPUs) which use *systolic*

[1] www.research.ibm.com/bluegene/
[2] http://www.myri.com/
[3] http://www.infinibandta.org/
[4] http://setiathome.ssl.berkeley.edu/
[5] These will be described in more detail in Chapter 2

arrays, where a single instruction acts on multiple items of data, known more generally as a *single instruction, multiple data* (SIMD) architecture. Recent GPUs have allowed general purpose computations to be performed with them, but this allows only for loosely coupled parallelism for example with linear algebra calculations. Another is *application specific integrated circuits* (ASICs), which are ICs customised for a specific use and are usually designed as a NoC.

The key feature of processor, or more generally HPC networks, as opposed to regular computer networks is the tight-coupling between processes. This means that messages must be delivered in high volume, while at low latency.

1.4. Irregular Interconnection Topologies

Many of the interconnection networks found in parallel systems, particularly those related to HPC where the coupling between processes is tight, are based on regularly arranged topologies such as *arrays*, meshes or hypercubes. Regular topologies offer a good trade-off between performance such as high throughput and wide path diversity against the constraints of the available fabrication technology. Low dimensional mesh and hypercube networks have been popular choices in many systems due to their low packaging constraints; the physical constraints in fabrication of the wires between logic blocks. It is becoming more important now though to consider topologies where the structure has an irregular or non-uniform construction. This is motivated by developments in the following areas.

1. **Fault-tolerance** is becoming an increasingly important issue. As VLSI processes can fabricate huge numbers of transistors in NoC architectures, the probability of faults within them increases; some small area of the chip maybe defective while the rest is fully functional. Similarly, with systems of multiple chips arranged in uniform topologies, such as in supercomputers, large numbers of processing elements increases the likelihood of failure. Hardware faults can result in the original network to degenerate into some non-uniform topology, causing the original routing algorithms to become ineffective. With a routing strategy resistant to faults the functional elements of the chip or functioning processing elements may be fully utilised.

2. **Application-specific NoCs** commonly consist of heterogeneous components of varying size, making uniform interconnections difficult or impossible when the system is arranged on a chip die. It is therefore necessary to employ a routing strategy able to route around irregularities whilst preserving desirable NoC routing properties such as low power consumption, low latency and small VLSI implementation costs. Also, emerging applications such as networked audio, where audio signals are carried over a network instead of expensive audio signal cables would require non-uniform topologies but with low latency high bandwidth communication.

3. **Dynamic changes.** Irregular topologies may also arise from *dynamic* changes to the connectivity of the network. The ReNoC (Reconfigurable NoC) architecture [39] allows changes to the topology including long range links and direct links between intellectual property (IP) blocks so the system can be customised for the application running on the chip. Topology changes may also be caused by the use of bypassing techniques, where a particular node's routing function is avoided by a wire in a single direction in order to reduce the latency and energy costs associated with the routing circuit.

4. **Nano-scale NoC assemblies.** Traditional NoCs are built using a top-down approach, where IP blocks are connected using a regular or customised topology. Self-assembled nano-scale electronics is a rapidly developing new technology, which is a potentially exciting new approach to NoC fabrication. However, such networks would be inherently irregular and would require appropriate means for reliable and efficient communications [40].

5. **Application to existing networks.** Innovations in routing fabrics for HPC systems may also be applied to existing irregular systems. For example, the Internet, where the current routing infrastructure will not be able to meet the future demands on its size due to the poor scalability of router design [42].

1.5. Interconnection Network Design and Nomenclature

Every application using an interconnection network to communicate between system components will have a specific set of requirements of the network, some with greater importance that others. For example, a network may require very low latency, but not absolute reliability. To architect a network to meet these requirements it is necessary to consider the three core aspects of interconnection networks: *topology*, *routing* and *flow control* within the technology available. This section will introduce these concepts.

1.5.1. Topology

In an interconnection network, the physical topology is the arrangement of communication channels between a set of nodes, i.e. which nodes are physically connected to which other nodes. A network can also have a *logical* or *virtual* topology, where communications follow a mapping of paths on the physical connections. There are a great number of different topologies, each with a different set of properties. Selecting a suitable topology is vital to the design of a network as the routing and flow control mechanisms will rely heavily on its properties and the physical implementation must be within the means of the fabrication process in terms of cost and technology.

Ideally, an interconnection network would be fully connected to allow simultaneous direct communication between all pairs of nodes, achieving optimal bandwidth and latency. This approach can be applied to systems with few terminal nodes, but it does not scale to larger networks as the degree of each node would be equal to the number of nodes. The network capacity must scale with the number of processors, properties of the combination of a good choice of topology and routing strategy.

Topology Model

It is useful to represent the topology of a interconnection network abstractly as a directed graph. This allows a mathematical formulation of the routing problem to analyse the behaviour.

Definition 1. (Interconnection network). *The vertices N represent the set of processing nodes and the edges C represent the set of communication channels between then nodes. Each channel $c = (n_s, n_t) \in C$ connects a source node n_s to a destination node n_t where $n_x, n_y \in N$. The source node of a channel is denoted s_c and the destination d_c.*

The above definition describes a *direct* network, where each node is *terminal*, in that it acts as a source and sink for messages and also as a switch to route incoming messages. In contrast, an

indirect network contains non-terminal nodes that are used just for switching. When referring to the number of nodes in a network N will often be used instead of $|N|$, the context will always make this clear. Also, a message refers to a variable-sized piece of data generated by a processing node to be sent to a different destination node. This message is broken down into a set of smaller possibly variably-sized packets by the network layer to be transported over the network for the allocation of control state.

Definition 2. (Channel). *A channel $c \in C$ has associated latency and bandwidth values given by functions $l : C \to \mathbb{R}^+$ and $b : C \to \mathbb{R}^+$ respectively. An undirected graph can be thought of as a generalisation of this where each edge has a channel in each direction. Messages can be generated and consumed at any nodes $n \in N$.*

Definition 3. (Channel set). *Each node $n \in N$ has a channel set $C_n = C_{I_n} \cup C_{O_n}$. Where $C_{I_n} = \{c \in C | d_c = n\}$ is the input channel set, and $C_{O_n} = \{c \in C | s_c = n\}$ is the output channel set. The degree of n is $\delta_n = |C_n|$ which is the sum of in degree $\delta_{I_n} = |C_{I_n}|$ and the out degree $\delta_{O_n} = |C_{O_n}|$. Let the set of virtual channels be denoted C', and the virtual channels belonging to a channel $c \in C$ denoted c'.*

Definition 4. (Path). *A path is a sequence of contiguous channels $P = \{c_1, c_2, ..., c_n\}$ where $d_{c_i} = s_{c_i 1}$ for $i = 1, ..., (n-1)$. The source of a path is $s_P = s_{c_i}$ and the destination of a path is $d_P = d_{c_n}$. The length or hop count of a path is $|P|$. I is said to be* connected *if at least one path exists between all source and destination pairs.*

Definition 5. (Minimal path). *A* minimal *path from n_x to n_y is one that connects n_x and n_y and has the smallest hop count. The set of all minimal paths from n_x to n_y is denoted $R_{x,y}$. $H(x, y)$ is the minimal path between n_x and n_y.*

Properties

For a network topology to have good communication performance, it is important that the diameter is small as it is a lower bound on for the worst-case time necessary on the maximal path distance between a pair of nodes.

Definition 6. (Diameter). *The* diameter *of a network H_{max} is the largest minimal hop count over all pairs of nodes in the network.*

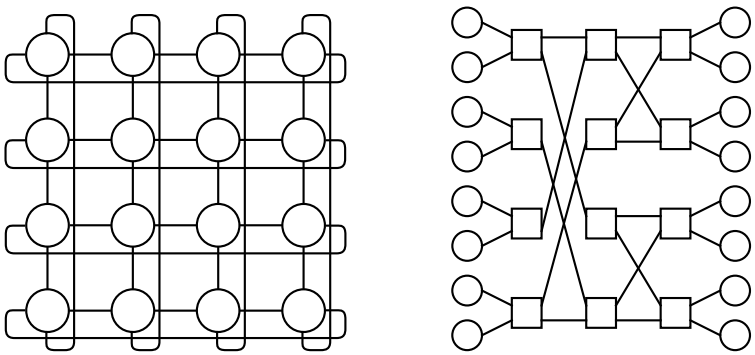
$$H_{max} = \max_{x,y \in N} H(x, y)$$

Theorem 1. *Every network of size n and maximum degree δ must have a diameter bounded by*

$$H_{\max} = \Omega\left(\frac{log|N|}{log(\delta - 1)}\right).$$

(For a proof see Appendix A.1)

It is also important that the network does not have a *bottleneck* where many paths share a small set of nodes. For example, the complete binary tree topology forms a bad communication network since for any node, half of the possible destination nodes can only be reached by traversing the root node. This factor is measured by the *bisection width* of a network.



(a) Torus (4-ary 2-cube) (b) Butterfly (2-ary 3-fly)

Figure 1.1.: Example torus and butterfly topologies, important classes of regular topologies.

Definition 7. (Network cut). *A cut of a network $C(N_1, N_2)$ is a set of channels that partitions the set of nodes N into two disjoint sets N_1 and N_2. Each element of the cut $C(N_1, N_2)$ is a channel with one end in N_1 and the other end in N_2. The number of channels in the cut is $|C(N_1, N_2)|$ and the total bandwidth of the cut is*

$$B(N_1, N_2) = \sum_{c \in C(N_1, N_2)} b_c.$$

Definition 8. (Channel bisection and bisection bandwidth). *A bisection of a network I is a cut that partitions the nodes into two disjoint sets N_1 and N_2 where $N_1 = N \backslash N_2$, such that they have size $\leq \frac{|N|}{2}$. The bisection width or channel bisection B_C is defined as*

$$B_C = \min_{|N_1| = \lfloor |N|/2 \rfloor} C(N_1, N_2).$$

The bisection bandwidth B_B is the minimum bandwidth over all bisections in the network.

$$B_B = \min_{|N_1| = \lfloor |N|/2 \rfloor} B(N_1, N_2)$$

A more general form of bisection width is the *expansion* of a graph. It describes for a subset of vertices S, how quickly they *expand*, in the sense that it is connected to many vertices of the set $\bar{S}$, the complementary set of vertices.

Definition 9. (Expansion). *The expansion α of a network I is defined as*

$$\alpha(I) = \min_{U \subset N} \frac{|C(U, \bar{U})|}{\min(|U|, |U'|)}.$$

Theorem 2. *For all networks $I = (N, C)$, the expansion can be at most 1. (For a proof see Appendix A.2)*

Topology Classes

There are many different classes of network topology used in parallel systems, and infinite variations on each one. Many of these classes have been developed from two main families; *butterflies* and *tori*. Butterfly networks are indirect and are attractive they as have optimal diameter,

but have shortcomings with no path diversity, where the bound given by Theorem 1 is tight (only one path connecting each pair of nodes), and they also require long physical wires in their implementation. Tori are a class of direct networks which have good packaging characteristics and good physical communication locality. Figure 1.1 shows examples of these important types of topology.

In the 1970s, high dimensional binary n-cube or *hypercube* networks were popular because of their low diameter but gradually due to realistic packing and fabrication constraints it was shown that low-dimensional networks such as 2-D or 3-D meshes outperformed hypercubes. Low dimensional meshes and tori have uniform length wires and logically minimal paths are almost always physically minimal as well. For these reasons, low dimensional meshes and tori are used predominantly today in interconnection networks, although it is probable that use of high degree indirect networks like butterflies will increase in the future as the bandwidth of router chips increases relative to the message length [7]. The focus, in terms of regular networks, in this project will be on meshes and tori.

1.5.2. Routing

Routing in a network is the process of directing messages generated by a source node through the network to a destination node. It is important that routing algorithms have low communication latency, provide high network throughput and in HPC systems have a simple or compact implementation. The network topology and physical implementation define these characteristics, providing a lower bound on the performance of a routing algorithm. It is down to the design of the algorithm to determine the extent to which the potential of this is realised.

There are a number of features that may be found with good routing algorithms, contributing to low latency and high throughput. The most important factor is freedom from *deadlock*, where a set of packets become blocked waiting for each other. Adaptivity is also important, in order to cope with faults of certain traffic behaviours to balance communication loads. Lastly, routing packets along shortest paths will directly effect latency and throughput. The properties of a routing algorithm will depend on the application of the system. For instance, in embedded systems, power consumption may be important and worth the cost of some latency. In other situations it be important to optimise for one or a combination of factors such as network bisection size, traffic flows and hot-spots, fault tolerance and VLSI implementation costs.

Taxonomy of Routing Algorithms

Uni-cast (single destination) routing algorithms can be divided into two categories *source* and *distributed*. With source routing, the source node computes the full path of a packet to the destination and stores it in the packet header. This means that intermediate nodes can have a simple forwarding function, but each node must have information about the whole network, and packet headers can become quite long; both factors effecting scalability. In contrast, with distributed routing, the path of a packet is determined at each node from a destination address. This method requires more complicated routing functions, but scales very well. Distributed routing algorithms can in turn be divided into the following classes, by the way in which they select from a set of possible paths R_{n_x,n_y} between a source n_x and a destination n_y.

- **Oblivious** routing algorithms make routing decisions *oblivious* to the state of the network. A set of paths are chosen in advance for every source-destination pair $(n_s, n_t) \in N$, and every packet for that pair must travel along one of these paths. The selection of one of these paths may be random, but cannot depend on the state of the network.

- **Deterministic** routing algorithms are a subset of oblivious algorithms. Deterministic algorithms will always supply the same path between a source and destination pair (n_s, n_t), even if $|R_{n_s,n_t}| > 1$, hence ignoring any diversity in the paths of the topology. This deterministic nature provides no adaptivity to network conditions for fault tolerance or load-balancing, but determinism does mean that they are often simple to implement and to make deadlock-free.

- **Adaptive** routing algorithms use information about network traffic and/or channel status to avoid congested or faulty regions. This may include current or historical information on the state of a node or link and buffer and resource utilisation. Adaptive algorithms can be *minimal* where packets are always forwarded closer to their destination or *non-minimal* where they may not always be. The use of non-minimal paths adds complexity as packets can arrive out of order. They can also be *fully adaptive* where packets can be routed adaptively along all available physical paths, or *partially adaptive* where packets are limited to a subset.

Routing Definition

A routing algorithm is represented by a routing function R that given some input values, for example, the current and destination nodes, returns a channel, for deterministic functions or set of channels for adaptive functions. For adaptive algorithms a selection function ρ is used to select a channel. With this representation, issues regarding channel dependencies and deadlock are related to the routing function R. As an example, an adaptive routing function R may have the following form:

$$R : N \times N \rightarrow \mathcal{P}(C)$$

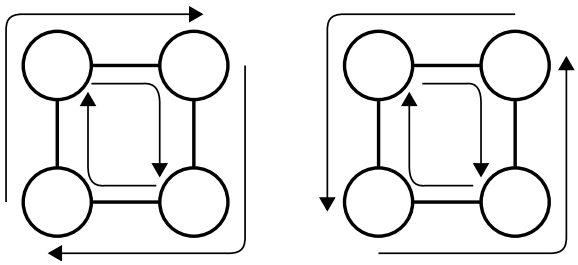
where the function takes a the current and destination node as arguments and supplies a set of alternative output channels to send a packet from the current node n_c to the destination node n_d, denoted by $\mathcal{P}(C)$, the power set of C.

Deadlock, Livelock and Starvation

The nodes of an interconnection network send, receive and forward-on packets. Messages will usually traverse several intermediate nodes before reaching their destination, however even with a fully-functioning network with fault-free paths connecting every source and destination node, packets may become blocked and not able to reach their destination. There are three situations that can cause this to happen.

- **Deadlock:** Buffer capacity in a network is a finite resource and must be allocated to a packet before it can proceed. A deadlock occurs when messages cannot advance towards their destination because the buffer space requested by them cannot be allocated as they are full, in turn holding resources requested by other messages. In this situation the messages are blocked forever, which is a catastrophic situation for the network.

- **Livelock:** A related situation is called *livelock* which occurs when messages are not able to reach their destination, even if they are not permanently blocked. A message may travel around its destination, but not able to reach it because the channels required to do so are occupied by other packets, this can only occur when messages are allowed to follow non-minimal paths.



(a) (b)

Figure 1.2.: Figure (a) shows a simple network in a deadlocked configuration. Figure (b) shows how deadlock can be avoided by restricting the available routing directions.

- **Starvation:** The last situation is known as *starvation*, which can cause packets to be permanently blocked due to heavy traffic not allowing the release of resources requested by the packet to proceed.

Figure 1.2 shows a simple mesh network of four nodes. If each node was to send a packet to the opposite corner (indicated by the arrows in Figure 1.2(a)) at the same time, and the routing algorithm routes packets in a clockwise direction then each link will become blocked and the system deadlocks. Instead of routing routing all packets in a clockwise direction, a deadlock-free algorithm for this network routes two of the messages anti-clockwise (assuming bi-directional links) as indicated by the arrows in Figure 1.2(b) allowing all of the messages to travel without deadlock.

Routing Mechanics

Routing mechanics relates to the way in which a routing algorithm is implemented. The most common way to do this is with a *routing table* either just at the source with *source routing* or at each hop with *distributed routing*. Table look-ups relate to the routing relations where values for each pair of inputs are stored in the table. Another way is with *algorithmic routing* in which the routing relation can be evaluated at run time a function of the input values. Algorithmic routing is usually restricted to simple routing algorithms on regular topologies.

1.5.3. Flow Control

Flow control is the process of managing the available resources between communicating nodes. In interconnection networks there are two types of resources: buffers and channels. Flow control manages the flow of data in a network, preventing nodes from being over or under-loaded by determining the allocation of these resources. Good flow control strategies will allow the network to operate within a close range of its ideal bandwidth with low latency. There are a number of different flow control methods, ranging from simple *bufferless* schemes where blocked messages are just rerouted or dropped, to more efficient *buffered* schemes which *decouple* the allocation of adjacent channels in time. This project will focus on *wormhole* and *virtual channel* flow control together as they have become a standard method of flow control in HPC parallel systems.

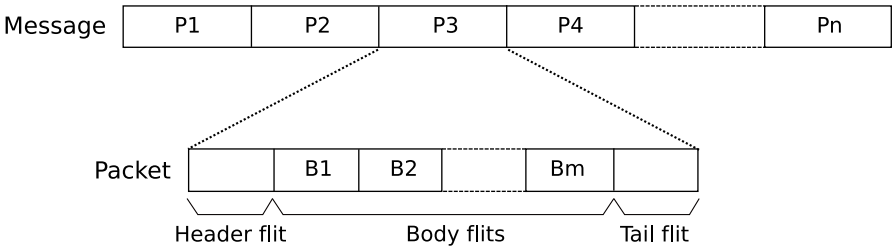


Figure 1.3.: A message is broken down into a sequence of n packets, each packet is transmitted across the network as a sequence of m fixed-sized flits with appended head and tail flits.

Wormhole Flow Control

To achieve low latency across an interconnect it is essential that wormhole flow control[6] is used [8] as it makes efficient use of buffer space as variable-sized packets are broken down into smaller fixed-size units called *flits*. Routing fixed-sized flits greatly reduces the required buffering capacity at each node and the latency of packet delivery, compared to *store-and-forward* or *cut-through* flow control where whole packets must be buffered. The size of a flit is dependent on the implementation of the network. Often a flit is sent in smaller units called physical transfer digits or *phlits* which are defined as the amount of data that can be sent down a channel in one clock cycle. Phlits are not used to allocate resources, the link level protocol must interpret the flits.

Messages generated by a processor are broken down into packets for allocation of control state in the network. Packets are sent across a network as a series of flits which are used to allocate buffer capacity and channel bandwidth. A packet is sent by a sequence of *body* flits with head and tail flits appended to the beginning respectively. The header flit contains routing information and a sequence number and proceeds first, allowing it to allocate a channel for the subsequent body flits. If the resources cannot be allocated, the packet is blocked. As body flits are received they are forwarded along the same path as the header and need only to acquire buffer space at the downstream node to advance. On receiving a tail flit, the channel allocation made to the header flit can be released and made available for a new packet. Wormhole flow control has its name as the flits of the packet travel through the network like a *worm* in that the header flit may arrive at the destination before all remaining flits have been transmitted. Figure 1.3 illustrates how messages are broken down into packets, and packets in turn into flits.

Virtual Channel Flow Control

Virtual channels are another method of flow control that works very well with wormhole flow control. The idea is that a set of virtual channels, each with their own flit buffer and control state share a single physical channel to overcome the problem of blocking. A physical channel becomes blocked when it has been allocated to a packet but a subsequent flit cannot progress due to downstream buffer capacity. With virtual channels, the blocked channel's state can be set accordingly and another virtual channel can use the link. Virtual channels also have important applications in avoiding deadlock, as Dally puts it [7]

[6] It is often referred to as wormhole routing or wormhole switching, but in fact is not related to switching or routing.

> Virtual channels are the Swiss-Army^{TM}knife of interconnection networks. They are a utility tool that can be used to solve a wide variety of problems.

As with wormhole flow control, a header flit arriving at a node must acquire a virtual channel (channel state), a downstream flit buffer and channel bandwidth. The channel bandwidth is allocated between the set of virtual channels via some allocation policy, usually *round-robin*. Subsequent body flits use the same virtual channel allocated to the header but they must acquire space in the downstream buffer and channel bandwidth. Virtual channel flow control decouples the allocation of channel state from channel bandwidth by preventing packets from acquiring channel state then blocking use of the channel, i.e. available bandwidth.

1.6. Aims & Objectives

1.6.1. Project Aims

The main aim of this project is to investigate and explore the current approaches to routing in irregular processor networks. Specifically, it will focus on algorithms that represent the most recent developments in *universal routing*, i.e. the current best-performing algorithms that work independent of the structure of the network topology. Analysis of these algorithms, both theoretically and empirically, will give a detailed picture of their behaviour and performance characteristics.

Routing algorithms are generally divided into those that use virtual channels to break deadlock, and those that don't. The motivation for this project is that no comprehensive universal routing comparison has yet been made. In particular, between algorithms that use either *virtual channels* or *turn prohibition* to break *deadlock*[7], they are usually considered separately. Furthermore, evaluation of universal algorithms is rarely conducted on regular topologies and results for irregular topologies are never specific to the properties of the topology. Therefore, it is in these areas that this project will focus.

1.6.2. Methodology & Objectives

The methodology for achieving these aims is broken down into the following objectives.

1. **Literature review.** Research and investigate the literature on universal routing algorithms to identify the current best-performing algorithms, specifically representing the use of turn prohibition or virtual channels to break deadlock. These algorithms will form the basis of the investigation.

2. **Build a simulator.** Implement a software network simulation tool to enable the performance evaluation of different routing algorithms and network topologies under a range of simulation and network parameters. This will require researching the functionality and architectural details of components in a network system, in particular network routers.

3. **Implement algorithms.** Implement the algorithms identified in the literature review to work with the simulator. Doing this will give an in-depth understanding of their behaviour and properties to be obtained, on which a discussion of their functionality and any issues identified by the implementation to be based. For a full comparison it will also be necessary to implement base-case algorithms as a benchmark.

[7] These terms will be introduced in detail in the following section.

4. **Perform experiments.** Obtain a detailed set of experimental results using the simulation tool to give an in-depth analysis of the performance of the algorithms. The results will primarily include a range of different network topologies; specifically different classes of random irregular topologies.

5. **Evaluate results.** Based on the experimental results, provide an analysis of the behaviour of the algorithms with respect to different network topologies and their characteristics, and to key simulation parameters.

Chapter 2.

Topologies and Routing

The premise of this project is to investigate the performance of universal routing algorithms on irregular topologies. This chapter will present the topologies and routing algorithms used in this analysis and experimentation. It will build upon the routing and topology background presented in the last chapter, presenting full technical details. The third key topic of flow control will be discussed in Chapter 3.

2.1. Topologies

The following classes of regular and random irregular topologies will be used to evaluate the performance of routing algorithms.

2.1.1. Meshes and Tori

Mesh or k-ary n-meshes are class of direct networks composed of k orthogonal dimensions containing $N = k^n$ nodes. An n-dimensional mesh has $k_0 \cdot k_1 \cdot ... \cdot k_{n-1}$ nodes, k_i nodes along each dimension where $k_i \geq 2$ and $0 \leq x_i \leq n-1$. Each node $x \in N$ is identified by an n-digit radix-k address (or coordinate) $(x_{n-1}, x_{n-2}, ..., x_0)$ where $0 \leq x_i \leq k_i - 1$ for $0 \leq i \leq n-1$.

Two nodes x and y are neighbours if and only if their identifiers differ by 1 only in one dimension. Hence, nodes in a mesh have between n and $2n$ connected nodes, whereas nodes in tori have constant degree due to extra links. These extra *wrap-around* links give tori regularity (constant node degree) and symmetry. Graphs with constant node degree d are known generally as *d-regular*, although the use of the terms *regular* and *irregular* in this project will relate to the uniformity of the topology.

Definition 10. (Mesh and Torus). *For $k, n \in \mathbb{N}$ a k-ary n-mesh is a graph with node set $N = k^n$ and channel set*

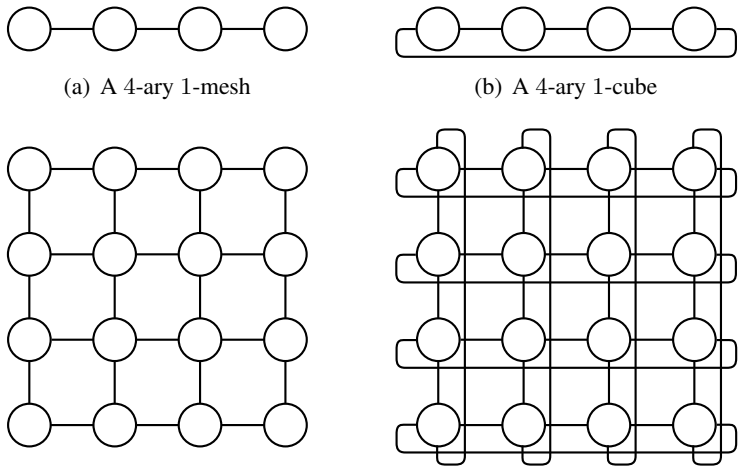
$$C = \left\{ \{(x_{d-1}, ..., x_0), (y_{d-1}, ..., y_0)\} | x_i, y_i \in N, \sum_{i=0}^{d-1} |x_i - y_i| = 1 \right\}.$$

A torus or k-ary n-cube network consists of a k-ary n-mesh with additional channel from $(x_{d-1}, ..., d_{i+1}, 0, d_{i-1}, ..., x_0)$ to $(y_{d-1}, ..., y_{i+1}, 0, y_{i-1}, ..., y_0)$ for $0 < i \leq d$. Hypercubes are a special case of tori and meshes: 2-ary n-cube $\Leftrightarrow$ 2-ary n-mesh.

Figure 2.1 shows some example mesh and torus topologies. Figure 2.1(a) shows a 4-ary 1-mesh, or 3-node linear array, Figure 2.1(b) shows a 4-ary 1-cube, or 3-node ring, Figure 2.1(c) and Figure 2.1(d) are the 2-dimensional versions.

2.1.2. Irregular Topologies

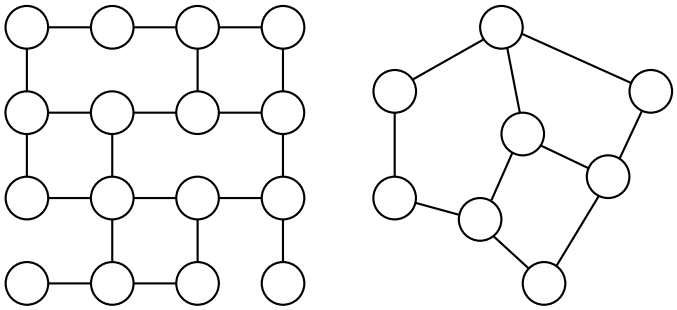
Given the motivations described in Section 1.4, irregular network topologies found in multi-processor systems are often based on a regular structure such as a mesh that has degenerated into



(a) A 4-ary 1-mesh (b) A 4-ary 1-cube

(c) A 4-ary 2-mesh (d) A 4-ary 2-cube

Figure 2.1.: The structure of example mesh and torus topologies



(a) (b)

Figure 2.2.: Example irregular topologies. (a) shows a degenerated 4-ary 2-mesh topology, (b) shows a random irregular topology.

an irregular structure due to faults of dynamic topology changes. There is a distinction to be made though between degenerated *semi-irregular* topologies and *fully-irregular* or *random* topologies. Figure 2.2(a) shows an example degenerated mesh topology from failed links. Figure 2.2(b) shows a simple example of an entirely random topology.

Random Graph Models

Random graphs are important mathematical constructs with varied applications. Recently, a new area of research has emerged in complex networks, which has been motivated by studying the properties of networks such as the Internet, flight-paths and even the biological structure of cells. The key feature of such networks is that for any pairs of nodes, the distance between them is small relative to the size of the network. This is known as the *scale-free* property and implies excellent communication properties.

Much of the literature on routing in irregular networks considers the performance of algorithms on unspecified random graphs, without any clarification of the properties of the graph, particularly communication. This seems to be a large oversight, as there are many classes of random graphs,

each with different properties that will effect routing behaviour. To look in-depth at the performance of universal routing algorithms, the following random graph models will be used to as they represent three important properties of graphs relating to structure and consequent communication properties: scale and non-scale free, and clustering of subsets of nodes.

Erdős-Rényi Random Graph

One of the simplest, but most important classes of random graph is called an Erdős-Rényi random graph [15]. These graphs will almost certainly not be scale-free and will not contain clusters.

Definition 11. (Erdős-Rényi random graph). *An Erdős-Rényi random graph $G = (n, p)$ is one where an edge exists between two nodes $n_1, n_2 \in n$ with probability p.*

Theorem 3. *For a random graph $G = (n, p)$, the expected degree of a fixed vertex v is $p(n-1)$. (For a proof see Appendix A.3).*

Expander Graph

Definition 12. (Expander). *A graph is called an expander if has constant expansion*

Expander graphs are an important class of graphs with remarkable communication properties [30, 43]. Intuitively, any sub-set of vertices is well connected to the rest of the graph, i.e. they are scale-free. Random d-regular graphs have good expansion with high probability. [38] presents an explicit construction of random d-regular graphs.

Given a random d-regular graph G with n nodes, where each node is connected to d other vertices, chosen at random. Let U be a subset of G of at most $n/2$ vertices. For some $v \in U$, v will be connected to roughly $d \times |\bar{U}|/n$ vertices in $\bar{U}$ and hence we expect

$$|C(U, \bar{U})| \approx d \times \frac{|U||U|}{n}$$

and so

$$\frac{|C(U, \bar{U})|}{|U|} \approx d\frac{|\bar{U}|}{n}.$$

Since $|\bar{U}|$ has its minimum at approximately $n/2$ is follows that $\alpha(G) \approx d/2$, independent of the size of the graph.

Barabási-Albert Random Graph Model

The Barabási-Albert model [1] is an algorithm for generating random scale-free, clustered graphs using a preferential attachment mechanism, where a clustered graph is one that contains sets of highly interconnected nodes. The algorithm works by creating m_0 initial nodes with no edges. New nodes are then repeatedly added, connected with an edge to an existing node i with probability

$$p_i = \frac{\text{degree}(i)}{\sum_j \text{degree}(j')}.$$

After t steps, the network contains $N = t + m_0$ nodes and mt edges. The outcome is a small set of heavily connected *hub* nodes, and many lesser connected nodes. Intuitively, new nodes

preferentially attach themselves to hubs. This results in a power law degree distribution of the form

$$P(k) \sim k^{-3}.$$

The average path length l grows as

$$l \sim \frac{\ln N}{\ln \ln N}.$$

2.2. Summary of Routing Approaches

2.2.1. The Theory of Deadlock-Free Routing

There has been a great deal of research in the area of routing in multi-processor systems. Much of the work in the 1980s dealt with methods of deadlock-free routing algorithms in packet-switched store-and-forward networks. These ideas were based on the concept of *structured buffer pools* which enumerated the network resources and traversed them in order so to eliminate cyclic channel dependencies that caused deadlock. Gopal proposed several fully adaptive minimal routing algorithms, based on structured buffer pools, which are known as hop algorithms [19]. In the positive-hop algorithm, for a network of diameter D, a minimum of $D + 1$ buffers per node were required as a packet in buffer i would be stored in buffer $i + 1$ in the next node.

In 1987, Dally and Seitz [12] introduced an important new theory for developing deadlock-free routing algorithms for wormhole routing based on removing cyclic channel dependencies through the use of virtual channels and restricted routing. This was later extended by Duato in 1991 [13] to allow the design of adaptive algorithms but also conditions necessary to develop algorithms even with the existence of cyclic channel dependencies. Duato also developed a theoretical basis for fault-tolerant routing in wormhole networks [14]. Glass and Ni [18] in 1992 made an important contribution with a method for designing deadlock-free routing algorithms without the use of virtual channels, instead by prohibiting enough routing turns to break cyclic channel dependencies. This early theoretical work has underpinned much of the later developments in routing.

2.2.2. Routing in Regular Networks

Routing in regular networks has been studied extensively. There are several key approaches that are used most commonly, such as dimension-order routing, and also that have formed the basis for work on universal routing, such as the Turn model's idea of turn prohibition.

- **Dimension Order routing** (DOR) is one of the simplest deadlock-free routing algorithms for regular topologies and can be implemented algorithmically. It works on regular topologies that can be decomposed into orthogonal dimensions and routes messages by nullifying the offset in each dimension in-turn.

- **The Turn Model** [18] is a general framework for restricting routing algorithms in mesh networks. It works by prohibiting turns that form cycles. Deadlock can be avoided by prohibiting just enough turns to break all cycles.

- **Valiant's trick:** Valiant [43] proposed a simple but incredibly effective trick to balance load for any traffic pattern on arbitrary topologies. The idea is to route packets to an intermediate destination (usually randomly chosen) before routing to the final destination. Each phase then appears to be uniform traffic. Any routing algorithm can be used for the two phases, for instance with tori DOR can be used.

- **Planar-Adaptive routing** [6] combines virtual channels with prohibited turns by defining adaptive planes in a k-ary n-mesh, consisting of two adjacent dimensions. Within a dimension any minimal, adaptive routing algorithm can be used. By limiting a plane to two dimensions, the number of virtual channels required is independent of the number of dimensions and the size of the network.

- **Duato's Protocol:** [13, 14] Duato's theoretical work for designing deadlock-free adaptive routing algorithms, referred to as Duato's protocol, is most commonly applied to add minimal adaptivity to DOR in mesh and tori-based networks.

2.2.3. Universal Routing in Irregular Networks

Approaches to the development of routing strategies for irregular networks have been based around either the use of virtual channels or turn prohibition to eliminate deadlock. The following points give a brief description of notable algorithms that apply to arbitrary topologies as opposed to tolerating faults in a specific topology.

Using Virtual Channels

- **Multiple virtual networks.** In 1991 Linder and Harden [25] developed an adaptive and fault tolerant wormhole routing strategy for tori based on the use of virtual channels to create multiple virtual networks. This was an important idea, but impractical because the number of virtual channels required grew exponentially with the size of the network.

- **Layered shortest path (LASH) routing.** Linder and Harden's work formed the basis for LASH routing [37, 26], which divides a network into virtual layers where each layer is assigned a set of source-destination pairs such that it is deadlock-free.

- **Transition orientated routing** [32] (TOR) also used virtual channels to create virtual networks but used the Up*/Down* routing algorithm [35] to decide when to make transitions between layers. However, due to the large number of restrictions imposed by Up*/Down*, a high number of virtual networks could be required.

- **Descending Layers (DL) routing** [24] is a similar virtual network scheme developed in 2003. It divides the network into sub-networks with the same topology consisting of layers of virtual channels and establishes a large number of paths across sub-networks in order to reduce path length and congestion.

- **LASH-TOR:** [36] In 2004, a new algorithm called LASH-TOR, based on the ideas of LASH and TOR was developed which reduced the required number of virtual layers and allowed transitions easily between them.

Using Routing Restrictions

The problem with the use of virtual channels is the requirement of more complicated hardware due to the requirement of extra buffers. Also, where virtual channels are available, then the number is often limited and often dedicated to services such as flow control. Algorithms based on routing restrictions, building on the turn prohibition model, do not require virtual channels, instead apply directional restrictions to the use of channels to ensure cyclic channel dependencies are broken.

- **Up*/Down* routing** [35] was proposed in 1991 and is widely used in LANs. It works by constructing a spanning tree of the network, eliminating all cycles, and routing packets up and then down the tree. Up*/Down* is simple and applies to irregular topologies well, but makes poor use of network connectivity and creates congestion around the root node of the tree.

- **Smart routing** [5] was developed in 1996 as an alternative to Up*/Down*, and works by breaking channel dependency cycles by explicitly building a dependency graph and searching it. Cycles are broken that minimise a heuristic cost function.

- **FX routing** [33] looked to improve the performance of Up*/Down* in regular topologies and in balancing traffic. FX still uses a depth-first-search to construct a spanning tree but removes cyclic dependencies in a more flexible way.

- **L-Turn routing** [23] was again based on Up*/Down* but uses a less prohibitive spanning trees which distribute traffic more evenly to provide adaptive deadlock free routing.

- **Segment-based routing** (SR) [27] was developed in 2004 and works by partitioning the topology into segments and placing turn restrictions locally within a segment. This introduces a locality independence property allowing fewer restraints compared to algorithms such as Up*/Down* and FX.

2.2.4. Static and Dynamic Configuration

Universal or fault-tolerant algorithms are adaptive to irregular topologies in the sense that they can be configured (by the use of virtual channels or routing restrictions) for a particular topology. In the case of fabrication faults, the routing logic may be configured statically once before use to compensate. If the network must react to topology changes caused by faults or from bypassing nodes during execution, this may be achieved with a static reconfiguration phase while the system is temporarily halted, or dynamically while it is still running.

Dynamic reconfiguration is an attractive concept particularly for bypassing as it can be performed on the basis of the behaviour of communications, as opposed to static analysis which requires knowledge of the application and topology and can be computationally expensive. Such dynamic reconfiguration functionality is complex and hence current practical approaches only consider static reconfigurations.

2.3. Routing Algorithms Chosen for Analysis

The focus of this project will be an empirical study of the current approaches to topology-agnostic routing in processor networks. To do this, it will look specifically at two algorithms that represent the most recent developments in turn prohibition and use of virtual channels; Layered Shortest Path Transition Orientated Routing (LASH-TOR) and Segment-based Routing (SR). Up*/Down* routing, which is widely used to provided deadlock-free routing in LANs will used as a basis for comparison. The basis for the use of virtual channels or turn prohibition is to prevent deadlock. Before introducing the algorithms, the conditions necessary for deadlock freedom, fundamental to the design of any routing algorithm, will be introduced.

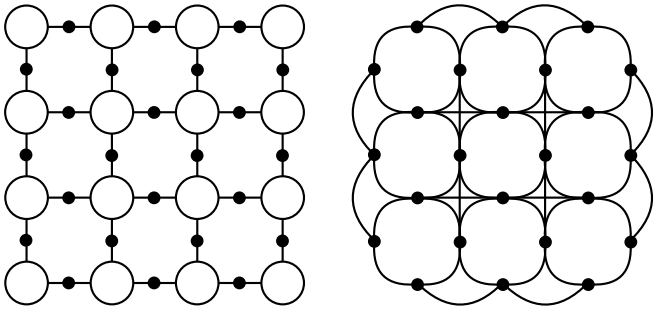


Figure 2.3.: The (undirected) channel dependency graph (right) for a 3-ary 2-mesh (left). Channel nodes are marked with filled circles. It is presented here undirected for simplicity, but for detecting deadlock it must be directed.

2.3.1. Conditions Necessary for Deadlock Freedom

The basis for deadlock freedom in the design of routing functions is to prevent cyclic dependencies between the use of channels. The following theorem by Dally [12] originally introduced this condition. A key assumption to this theorem is that a queue can only contain flits belonging to one packet. After accepting a header flit it must accept the remainder of the message before it can accept another header flit. This is so that when a message is blocked, the header flit will always occupy the head of the queue. The theorem uses the routing function $R : C \times N \rightarrow C$ which takes the current channel and destination node and returns an output channel. It also requires the following definitions.

Definition 13. (Channel dependency graph). *A channel dependency graph (CDG) D for a given interconnection network I and routing function R, is a directed graph $D = (C, E)$. The vertices of D are the channels of I, the edges of D are the pairs of channels connected by R:*

$$E = \{(c_i, c_j) : R(c_i, n) = c_j \text{ for some } n \in N\}$$

There can be no self-loops in D as channels cannot route themselves.

As an example, Figure 2.3.1 shows the channel dependency graph for a 3-ary 2-mesh.

Definition 14. (Configuration). *A* configuration *is an assignment of a set of flits to each queue, all of them belonging to the same message. The number of flits in the queue for channel c_i is denoted* $\text{size}(c_i)$. *If the first flit in the queue for channel c_i is destined for node n_d, then* $\text{head}(c_i) = n_d$. *If the first flit is not a header and the next channel reserved by the header is c_j then* $\text{next}(c_i) = c_j$. *Let $C_h \subseteq C$ be the set of channels containing a header flit at their queue head. Let $C_d \subseteq C$ be the set of channels containing a body or tail flit at their queue head. A configuration is legal if and only if*

$$\forall c \in C \begin{cases} \text{size}(c) \leq \text{capacity}(c) \\ \text{size}(c) > 0 \Rightarrow c \in R(c, \text{head}(c)) \end{cases}$$

Definition 15. (Deadlocked configuration). *A* deadlocked configuration *for a routing function R is a nonempty legal configuration such that the following conditions hold:*

$$1. \quad \forall c \in C_h \quad \begin{cases} \text{head}(c) \neq d \\ \text{size}(c_j) > 0 \; \forall c_j \in R(c, head(c)) \end{cases}$$

$$2. \quad \forall c \in C_d \quad \begin{cases} \text{head}(c) \neq d \\ \text{size}(\text{next}(c)) = \text{capacity}(\text{next}(c)) \end{cases}$$

In a deadlocked configuration, no flit is one hop from its destination. Header flits cannot advance because the queue for the output channel supplied by the routing function is not empty.

Theorem 4. *A routing function R for an interconnection network I is deadlock free if and only if there are no cycles in the CDG D.*

Proof. $\Rightarrow$ Suppose a network has a cycle in D, the cycle must be of length 2 or more as no 1-cycles can exist. Thus, a deadlocked configuration can be created by filling the queues of each channel in the cycle with flits destined for a node two channels away, where the first channel of the route is along the cycle.

$\Leftarrow$ Suppose a network has no cycles in D, since D is acyclic, a total order to the channels of C can be assigned so that if $(c_i, c_j) \in E$ then $c_i > c_j$. Consider the least channel in this order with a full queue c_l. Every channel c_n that c_l feeds is less that c_l and thus does not have a full queue. Thus, no flit in the queue for c_l is blocked, and no deadlocked configuration can exist.

Duato later showed [13, 14] that this condition was sufficient but not necessary to eliminate deadlock and was consequently too restrictive. Arguing that applied to non-adaptive routing it can increase congestion in heavily loaded regions and the use of adaptivity would avoid this. He showed that a more flexible condition existed for adaptive routing algorithms that allowed cycles in the CDG.

2.3.2. Dimension Order Routing

Dimension order routing (DOR) is a type of algorithmic routing for meshes and tori. It exploits the orthogonality of each dimension in these topologies and works by nullifying the offset of a message from its destination, in each dimension in turn. Each node is given a radix-k address, where k is the dimension of the network and digit i is denoted d_i. For a mesh network a direction D is computed for dimension i by

$$D_i = \begin{cases} +1 & \text{if } d_i > s_i \\ -1 & \text{otherwise} \end{cases} .$$

DOR for meshes is deadlock free as no cycles can be created, but in torus networks with extra wrap-around links, deadlock can occur within a dimension.

2.3.3. Up*/Down* Routing

Up*/Down* routing, commonly referred to as the spanning tree protocol, was first proposed in [35]. It works by creating a loop-free logical assignment of directions to the physical links. The assignment uses a spanning tree constructed from an elected root node to do this. Links have either an *up* or *down* direction depending on whether a traversal moves closer to the spanning tree root node. A legal route is then defined as one that never uses a link in the *up* direction after it has

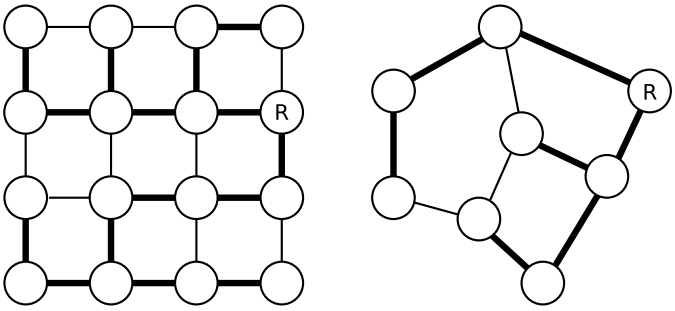


Figure 2.4.: Example regular and irregular topologies with spanning trees marked with bold links. The root node for the tree is marked with an 'R'.

used one in the *down* direction. Figure 2.3.3 shows two example topologies with spanning trees calculated.

Up*/Down* routing has two serious shortcomings, the first is that much of the traffic has to pass through the root node leading to congestion poor load-balancing. Secondly, many of the physical links are left idle, wasting spare network capacity. Many improvements and variations have been proposed to Up*/Down* over the years [5, 33, 23], generally aiming to alleviate these two issues.

2.3.4. Layered Shortest-Path Transition-Orientated Routing

Layered shortest-path transition-orientated routing [36] (LASH-TOR) is the product of two routing approaches. Layered shortest-path [37] (LASH) routing which provides deterministic shortest path routing any network topology. It makes use of virtual channels to divide the network into different virtual networks (layers) to avoid deadlock, each layer has a set of source-destination pairs assigned to it such that no cycles are created in the CDG. Transition-orientated routing [32] (TOR) also uses virtual networks, but uses Up*/Down* as a baseline routing algorithm to decide when to change to a new virtual network (when a forbidden down-up transition appears).

The LASH-TOR algorithm is an extension to LASH to allow transitions between layers using the methodology employed in TOR. The result is full minimal routing with a low virtual channel requirement that outperforms LASH on all network topologies with a bounded number of virtual channels. The remainder of this section, based on the proposal for LASH-TOR [36], will explain how the algorithm works.

Definition 16. (Network layer). *A* network layer L_i *of a network* I *is a subset of virtual channels in* I *such that each link has two channels, one in each direction in* L_i.

Definition 17. (Layering). *A set of network layers* $L = \{L_1, ..., L_n\}$ *is a layering if and only if*

1. $\forall i$, L_i *is a layer of* I.
2. $L_i \cap L_j = \emptyset \; \forall i \neq j$, $0 \leq i, j \leq n$.
3. *All channels in* I *belong to only one* L_i.

Definition 18. (Traffic assignment function). *Given a layering* L *of a network* I, *a* traffic assignment function T *of* L *is a function*

$$T : N \times N \rightarrow \mathcal{P}(L) \times \mathcal{P}(N)$$

that takes a source-destination pair (n_s, n_d) and returns subsets $L' \subseteq L$ and $N' \subseteq N$ where $L' = \{L_1, ..., L_{|L'|}\}$ is an ordered set of layers used in the minimal path and $N' = \{n_1, ..., n_{|N'|}\}$ is the set of layer transition nodes where $n_i \in N'$ represents the node that a transition from layer L_i to L_{i+1} is made. Also, $|N'| = |L'| - 1$.

The traffic assignment function calculates for a minimal path between two nodes n_s and n_d the assignment of the path to layers; each layer containing some subsection of the path, such that the CDG for each layer remains cycle free. If the whole path can be assigned to a single layer then $N' = \emptyset$. The routing function R used for LASH-TOR is defined as

$$R : N \times C' \times N \to C'.$$

That is, given the current node, current virtual channel and destination node, R returns an output virtual channel.

Description of Algorithm

The input is a network $I = (N, C)$ and a bound l on the number of layers such that $|L| \leq l$. Layer $ud \in L$ is used as a fall-back for any paths which cannot be normally assigned to layers $L_1, ..., L_{l-1}$ by using Up*/Down* routing. The CDG of layer L_i is denoted $\mathrm{CDG}(L_i)$.

1. **Initialise T and R.** Let $T(s, d) \leftarrow \mathrm{undefined}$ for all source-destination pairs $n_s, n_d \in N$, $s \neq d$ and $R \leftarrow \mathrm{empty}$.

2. **Compute paths.** For all source-destination pairs (n_s, n_d), calculate the set of minimal paths m between them within the network I, there may be multiple minimal paths.

3. For each pair (n_s, n_d):

   a) **Compute transitions.** For each minimal path $m_i \in m$, evaluate $T(s, d)$ to obtain $\{L', N'\}_{m_i}$. Enrich R with m_i according to the transitional nodes $n_i \in N'$, ensuring this does not create any cycles in $\mathrm{CDG}(L_i)$ for each layer $L_i \in L'$. If there are insufficient layers ($|L'| > |L| - 1$), then $\{L', N'\}_{m_i} \leftarrow \mathrm{undefined}$.

   b) **Choose smallest transition.** For each minimal path $m_i \in m$, select the m having $\{L', N'\}_{m_i}$ defined with the smallest $|L'|$. If one exists then let $T(s, d) \leftarrow \{L', N'\}_{m_i}$ and update R accordingly, otherwise $T(s, d) \leftarrow \mathrm{undefined}$.

4. **Assign Up*/Down* paths.** For all pairs (n_s, n_d) with $T(s, d)$ undefined, determine the path between them using Up*/Down* routing and update $\mathrm{CDG}(up)$; by definition this will contain no cycles.

5. **Recompute transitions.** For all pairs (n_s, n_d) with $T(s, d)$ undefined, evaluate $T(s, d)$ using the same shortest path used in 3a (m_i), this time allowing the use of the layer ud. If there now exists a valid set of transitions $\{L', N'\}_{m_i}$, let $T(s, d) \leftarrow \{L', N'\}_{m_i}$, otherwise $T(s, d) \leftarrow \{\{ud\}, \emptyset\}$ and update R accordingly.

6. **Balance layers.** To distribute paths more evenly among the layers, select a random pair (n_s, n_d) whose path is entirely assigned to L_1, and reassign it to a random layer $L_j \in L$ if it does not create a cycle in $\mathrm{CDG}(L_j)$. Repeat this until the number of sub-paths in each layer is evenly balanced.

2.3.5. Segment-Based Routing

Segment-based routing [27] (SR) is a deterministic routing methodology that uses turn prohibition to break deadlock instead of virtual channels. Its unique feature is a *locality independence* property, which allows bidirectional turn restrictions to be placed within small sub-graphs, or *segments* of the network topology without effecting any other segments, unlike many other turn prohibition-based algorithms such as FX and L-Turn routing. The remainder of this section, based on the description in [27], will describe how the SR algorithms works.

Definition 19. (Segment). *A* segment *of a network I is a set $S = \{N', C'\}$ of nodes $N' \subseteq N$ and channels $C' \subseteq C$. Segments are disjoint; $S_i \cap S_j = \emptyset \; \forall i \neq j, 0 \leq i, j, \leq |N|$. Segments can be of the following types:*

1. **Starting segment**. *A starting segment is a set of nodes starting and ending on the same node, forming a cycle. The cycle can be broken by placing a bidirectional turn restriction on any node, except the starting one (as a cycle could be introduced between two subnets).*
2. **Regular segment**. *A regular segment starts and finishes on a link and will contain at least one node. Any cycles created by regular segments can be broken with a single bidirectional turn restriction on any node belonging to the segment.*
3. **Unitary segment**. *A unitary segment contains only one link, no traffic can be allowed to cross the link as this could introduce cycles through connected segments. To do this, a bidirectional turn restriction can be placed at the node at one end of the segment, between it and every connected node.*

Definition 20. (Subnet). *A* subnet *is a set of nodes and channels contained in one or more segments, that is connected to the rest of the network through only one link.*

Description of Algorithm

The pseudo-code of two main methods are given in the paper, but for simplicity, a higher-level description is given here. The input to the algorithm is a network $I = (N, C)$.

1. **Initialise.** Pick a random start node $n \in N$ and mark it as *starting*. Let the current subnet $\alpha \leftarrow 0$.
2. **Repeat.**
   a) **Find a new segment.** Try to find a new segment S starting from node n of nodes and links not already belonging to a segment. If n is marked as *starting* the segment must also end on n, otherwise it must end on a node belonging to another segment. Essentially, a random walk is performed on the graph.
   b) **On success.** Add the segment to a list of segments $s = s \cup \{S\}$ and go to 2a.
   c) **On failure.** Mark this node as *terminal* and pick a node $n' \in N$ belonging to a segment in the current subnet with at least one channel not belonging to any segment and set $n \leftarrow n'$.
   d) **Start new subnet.** If such a node does not exist, pick a node n' not belonging to any segment that is not marked as *terminal*, but is attached to a *terminal* node. Set n' as *starting* and start a new subnet; $\alpha \leftarrow \alpha + 1$.
   e) **Terminate?** Again, if such a node does not exist, then terminate as no new segments can be found.

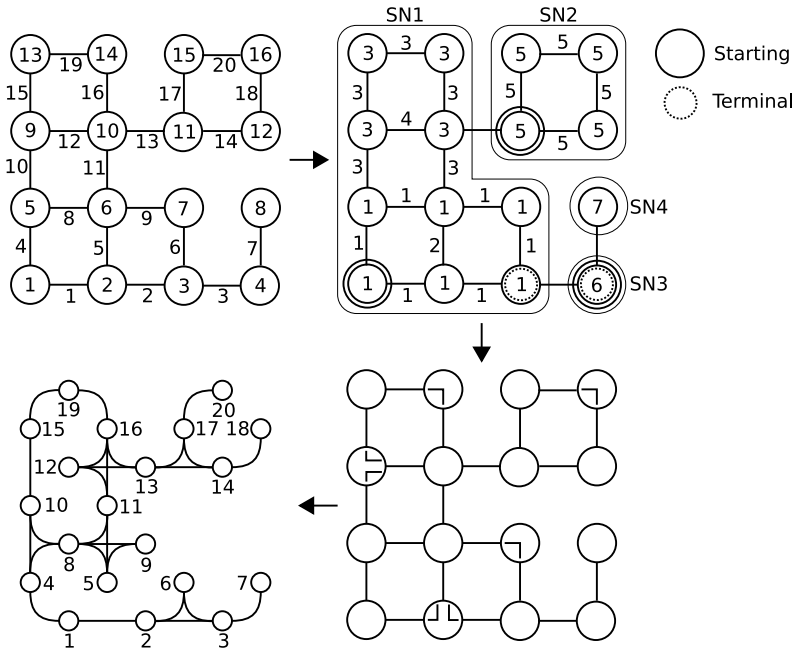


Figure 2.5.: Example stages of the segment routing algorithm operating on a degenerated mesh topology. The top left shows the topology with labelled nodes and links. The top right shows the computed segments labelled with numbers and subnets are outlined and marked with SN*. Bottom right shows possible placement of the bidirectional turn restrictions, where each angled line in a node indicates the direction of a prohibited turn. Finally, bottom left shows the dual representation used to calculate minimal paths.

Placement of Bidirectional Turn Restrictions

There is no way to place bidirectional turn restrictions on the network graph I in order to calculate shortest paths, and no alternative method of doing so is mentioned by the paper. A neat way of doing it though, is to construct a *dual graph representation* [3], which is similar to a CDG, defined as follows:

Definition 21. (Dual graph). *A* dual graph W *of a network* I *is one in which the nodes represent the channels and the edges represent the turns that can be made between the channels:*

$$E = \{(w_i, w_j) : c_i \to c_j \text{ is a valid turn for some } c_i, c_j \in C\}$$

For each node $n \in W$, $\mathrm{src}(n)$ denotes the source node in I and $\mathrm{dest}(n)$ denotes the destination node. Turn restrictions for I can made by removing the appropriate edges from W of I. The calculation of shortest paths is slightly more complicated as paths between two nodes $n_s, n_d \in I$ exist between all pairs of nodes $w_s, w_d \in W$ such that $\mathrm{src}(w_s) = n_s$ and $\mathrm{dest}(w_d) = n_d$.

2.3.6. Summary of Presented Algorithms

The fundamental difference between the LASH-TOR and SR algorithms is their use of virtual channels. That said, SR is not incompatible with them; if virtual channels are available in a network, they can provide underlying flow control as described in Section 1.5.3. Although break

deadlock in different ways, they are most similar in that their routing functionality is based on oblivious minimal routing, where messages are routed along a set path independent of the state of the network.

Another key point that differentiates them from light-weight routing algorithms such as DOR and Up*/Down*, is that they both employ centralised and computationally intensive configuration phases. In any real implementation this means that a single node in the network would need access to a full map of the topology and significant computational power to initialise the routing tables, which is feasible, but limits the applications. In many situations, such as LANs, this is difficult if not impossible. One of the reasons Up*/Down* routing has been so successful is that it can be implemented in a distributed way, so that each node contributes to determining the spanning tree of the network. Essentially, this is the trade-off made in leveraging good routing performance independent of the topology.

Chapter 3.

Network Simulation

Analytical techniques are useful to explore the behaviour of routing algorithms but it becomes more difficult to do so with complex protocols acting on arbitrary topologies. Network simulation is a powerful tool which can be used for various aspects of research and development of routing including analysis of protocols, understanding of behaviours and evaluating new ideas. In this project, simulation is used as the main tool to evaluate the performance of routing algorithms. This chapter will start by looking at different simulation approaches, then give a detailed description of the architecture of the network components and finally outline how performance is quantified and measured from the simulations.

3.1. Simulation Approaches

There are two main approaches to software simulation: *discrete-event* and *cycle-based*.

3.1.1. Discrete Event

In discrete event simulation, the state of the simulator is represented only at discrete points in time, under the assumption that the time in between can be safely ignored. Depending on the level of simulation, whether at hardware or application level, the choice of events provides a suitable level of abstraction. For example, the process of a network node transmitting a packet of data over a point-to-point link to another node is in entirety a very complicated procedure and might include encoding of the data into electrical waveforms and performing error detection. However, for a flit-level simulator these details can be ignored and simply abstracted as a delay, as they make no difference to the higher level routing and flow control protocols. The process would be represented as a transmitted packet event at time T, then after some delay $\triangle T$ the packet is received at the destination at time $T + \triangle T$ as a second event. Each discrete event changes the state of the simulation. Simulation proceeds by taking events in order from a list of pending future events, sorted by time stamp.

3.1.2. Cycle-Based

In cycle-based simulation, all components are are tied to a global clock and are synchronously updated each time step. The updates are generally performed in two phases. The first phase reads some input state and computes some output values as a function of these. The second phase copies each components output value into the inputs of the connected components. As an example, consider a router where in the first time phase, packets would be read from inputs of incoming links, and placed in output queues depending on the result of the routing function. In the second phase, packets ready to be sent would be copied to the inputs of the outgoing links. The critical invariants in cycle-based simulation are the order of evaluation of phases and the order of evaluation of functions within a phase.

3.1.3. Trade-offs

Discrete-event simulation has several key advantages; firstly that it can be used to model either synchronous or asynchronous systems as events can be arbitrarily timestamped. Secondly, only events occurring in the future are evaluated, hence idle components add no computational overhead as they do in cycle-based simulation. In larger systems, this can give a significant performance benefit. However, the effects are marginal for smaller systems (4 - 64 nodes) and although network communication is often asynchronous, cycle-based simulation simplifies the complexity as it allows some assumptions to be made about timing. Based on these reasons, the simulator implementation for this project will use a cycle-based approach.

3.2. Simulator Architecture

To evaluate the performance of routing algorithms with wormhole flow control on arbitrary topologies it is necessary to accurately model the communication behaviour at a flit-level. This section will describe the architecture of the simulator and its implementation in software. The design space for wormhole flow controlled interconnection networks is broad but the design of this simulator aims to generalise where it can, but where particular design decisions or assumptions have made, they are stated in the following sections. The following sections describe the architecture in a software-implementation sense, in that some of the details may not apply to a hardware implementation.

A cycle-based design lends itself to object-orientated approach as network components such as routers and links can be represented by objects with specified inputs, outputs and functionality. A network is represented by node and link objects, the arrangement of link connections between nodes determines the topology. Each node contains a processor which acts as a source and sink for packets, and a router which forwards or queues incoming flits to an output port or towards the processor if it is the destination.

3.2.1. Router

The router is at the heart of the simulator. It implements the routing and flow control functionality required to make successful communications across the network. The two main design decisions for the simulator, as part of the router architecture, were to include virtual channels for flow control and as a resource for some protocols, and also to use credit based flow control to provide *buffer management* and *back-pressure*.

Input and Output Ports with Virtual Channels

Each router has a set of input and output channels which will be referred to as ports. Each input and output port is divided into a set of v virtual channels. Physically this is realised as a set of v buffers each with some control state. Virtual channels associated with an input port contain a flit buffer commonly with capacity for around 4 flits, the control state for this maintains the following fields. A state vector (G,R,O,C) records the global state of the router (G), which can be either idle (I), active (A), waiting for credits (WC) or waiting for a virtual channel (WV). The output port route (R) and output virtual channel (V) values reference the corresponding output port and channel. And lastly, a credit count (C) is maintained which relates to the number of available slots in the downstream flit buffer corresponding to port R and virtual channel V.

An output virtual channel differs slightly with a state vector (G,I,C); it contains space to buffer a single flit as buffering needs only to be performed on input ports. It also maintains the same

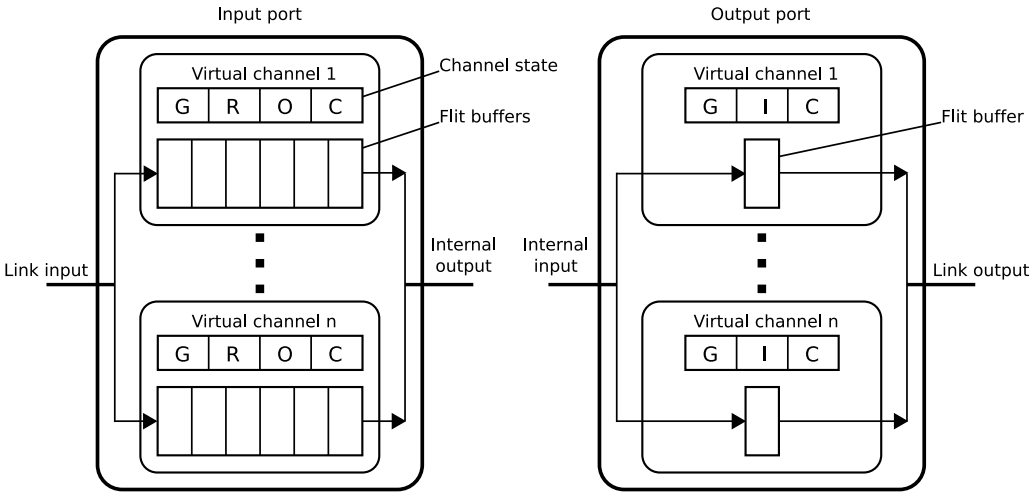


Figure 3.1.: A diagrammatic representation of a routers input and output ports with contained virtual channels

global state (G) with the exception of waiting for a virtual channel, and also the same credit count (C) as the corresponding input channel. Lastly, it maintains a reference to the downstream input virtual channel (I) that it is forwarding flits to. Figure 3.2.1 shows a diagrammatic representation of input and output ports with their contained virtual channels.

Credit-Based Flow Control

Credit-based flow control is used to manage the use of buffers to ensure that flits are not sent unless there is space downstream to accept and buffer them. It works by an output virtual channel in node n_a maintaining the number of free entries in the downstream input virtual channel buffer (C). When this is greater than zero, flits can be transmitted and the credit count decremented. At the downstream node n_b, the credit is returned back upstream when the flit leaves that buffer. This credit contains the number of the virtual channel returning the credit. This process can be thought of as n_a allocating buffer space to the transmitted flit with the credit. This space can only be reallocated when the credit is returned. The size of input buffers must match the latency (in cycles) of the time taken for a credit to be allocated and successfully returned. If there are fewer spaces than this, the supply of buffers will be exhausted before the first credit is returned.

Credit-based flow control creates a concept of back-pressure. This is if a node has to block traffic for a particular virtual channel, when the physical channel is used by another virtual channel, it will cause the buffer at the previous node in the path to start filling up. This will gradually happen for all preceding nodes in the path until all the flits along the path become blocked, unless of course the blockage is resolved.

Router Components and Data Paths

The router is comprised of a set of input and output ports connected with a crossbar switch. Decisions specifying the path flits take through the router are made by a routing function and the virtual channel allocator. Figure 3.2.1 illustrates the arrangement of these components and the connections between them.

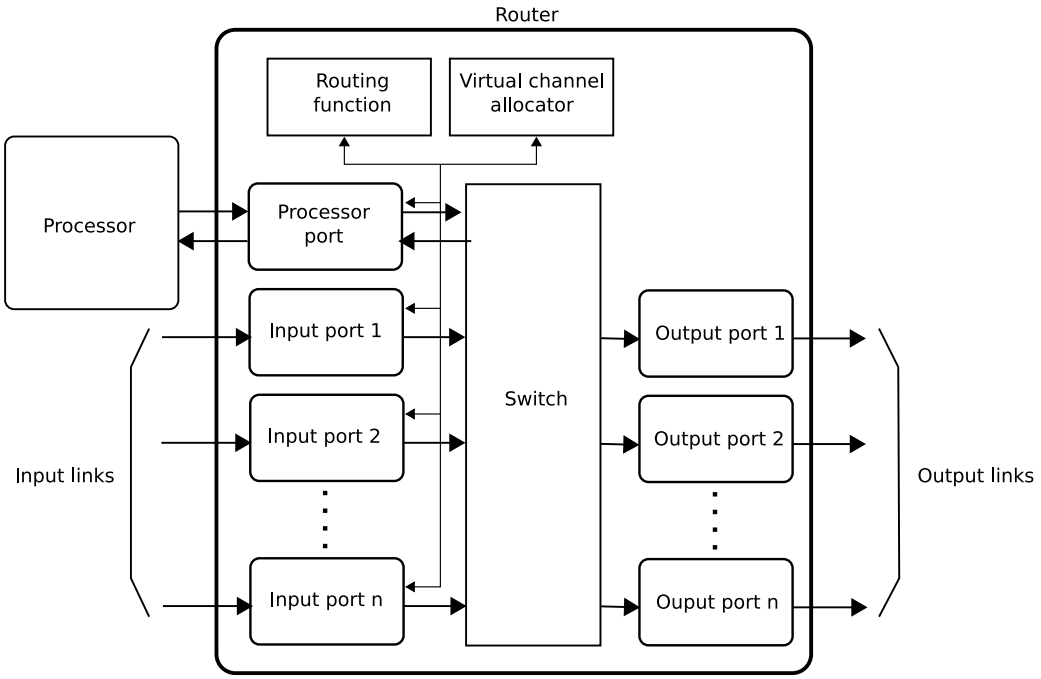


Figure 3.2.: A diagrammatic representation of a router showing the main components and the connections between them

When a flit arrives at an input port of a router it is placed into the buffer of the virtual channel specified as a parameter in the flit, this applies to head, body and tail flit types. The implementation of the router has a non-blocking crossbar connecting inputs with outputs, so flits from each input can be simultaneously transmitted across to their corresponding output ports. For each input and output port, virtual channels are serviced on a round-robin basis. When a virtual channel is serviced, the flit is read and if it is either a body or tail flit the output port (R) and virtual channel (O) are read and if there is space, the flit is written to the specified output buffer. If the flit is a header flit then the output port and virtual channel need to be determined before it can proceed. As a header flit passes through a router, it can be thought intuitively as setting up connections between input and output virtual channels, in doing so acquiring channel resources for subsequent flits.

The output channel will always be decided by the routing function as it relates to the direction it travels in the network. The virtual channel may either by dictated by the routing function if it is necessary to prohibit deadlock, or automatically allocated by the router. To allocate a virtual channel, one must be found that is both in the idle state (I) and with a full set of credits. The last condition is not entirely necessary, but it prevents a dependency from being formed between two flits sharing the same virtual channel in succession. The first condition ensures the channel contains no flits from other messages as this is a necessary condition for ensuring deadlock-freedom, which is proved in Section 2.3.1. If no virtual channel can be allocated the global state of the virtual channel is set to WV. When flits are successfully removed from the buffer, and the number of credits reaches zero, then the global state is set to WC.

3.2.2. Processor

The processor acts to generate messages according to the traffic pattern and injection process, and to consume messages sent by other nodes. It is connected to the router by its own input and output ports (illustrated in figure Figure 3.2.1) which each have only a single virtual channel. The processor transmits flits according to the same credit-based flow control scheme used on inter-router links as the flits buffered in the processor input port may become blocked by the switch if it is unable to allocate an output virtual channel on the router.

3.2.3. Links

Links are used both to connect routers together and the processor the router. Links are unidirectional but are modelled to contain a second signal travelling in the opposite direction as the data to transmit the credit messages back upstream. Links have an associated propagation delay in cycles, modelling the physical propagation of electrical or optical signals along a cable or fibre. If the delay is d cycles then the link can simultaneously carry d flits and d credits in either direction at different stages along the journey.

3.3. Performance of Interconnection Networks

The combination of topology, routing algorithm and flow control determine the behaviour and properties of an interconnection network. In order to assess the performance, two key metrics are used: *latency* and *throughput*. To measure the performance using these metrics, a network must be subjected to traffic workloads. Workloads can be driven by an application, as they would do normally from recorded traces of real execution, or synthetically from statistical models.

3.3.1. Synthetic Workloads

Synthetic workloads are a simplification of that of real execution of applications, but aim to capture the *spatial* and *temporal* elements of them. These are described by a spatial distribution of traffic over nodes in a network, and a temporal distribution by an injection process. The traffic workload used to evaluate routing algorithms in this project will be based on permutation traffic and a simple bursty injection process as these are most flexible for a range of topologies. More complicated statistical traffic models do exist [44], but are not as widely used and are less applicable to arbitrary topologies.

Traffic Patterns

The spatial distribution of traffic in a network is described by a matrix Λ where each element $\lambda_{s,d}$ describes the probability of node s sending a message to node d. Table 3.1 shows some common traffic patterns. The most simple of these is uniform traffic, where each node has an equal chance of sending a message to a random other node, represented by entries in the traffic matrix as $\lambda_{s,t} = 1/N$. The other patterns are permutations where each node s transmits to a single destination d, represented by a permutation function π, such that $d = \pi(s)$. Bit permutations calculate each bit of the destination address d_i as a function of one bit of the source address s_i; $d_i = s_{f(i)} \oplus g(i)$. In digit permutations, each radix-k digit of the destination address d_x is a function of a radix-k digit of the source address s_y; $d_x = f(s_{g(x)})$[1].

[1]Digit permutations only apply to networks where nodes addresses can be expressed as n-digit, radix-k numbers

Type	Name	Pattern
Uniform	Random	$\lambda_{s,d} = 1/N$
Bit permutation	Complement	$d_i = \neg s_i$
Bit permutation	Reverse	$d_i = s_{b-i-1}$
Bit permutation	Rotation	$d_i = s_{i+1} \mod b$
Bit permutation	Shuffle	$d_i = s_{i-1 \mod b}$
Bit permutation	Transpose	$d_i = s_{i+b/2 \mod b}$
Digit permutation	Tornado	$d_x = s_y + (\lceil k/2 \rceil - 1) \mod k$
Digit permutation	Neighbour	$d_x = s_y + 1 \mod k$

Table 3.1.: Traffic patterns

Random traffic patterns are most often used to test network performance, but have limited use as they distribute load evenly and can give misleading results. Permutation patterns are much better to stress topologies and routing algorithms, for example the tornado is designed to work against the locality properties of tori whereas the neighbour pattern exploits it. Several of the patterns are motivated by commonly occurring workloads, such as the transpose pattern which is based on the communication involved in a matrix transpose operation, and the shuffle permutation which is based on the communication behaviour of the Fast Fourier Transform and sorting algorithms.

Injection Processes

An injection process determines the average number of packets injected per time period. The most common injection process is called the *Bernoulli process*. It models injection with a binary random variable X such that the probability of injection is equal to a process rate parameter r; $P(X = 1) = r$. Simply, it flips a weighted coin each time step and with probability r, injects a packet.

Network traffic is often time-varying or correlated; characteristics not captured by the Bernoulli process. It can be extended simply to capture time-variations with a Markov modulated process (MMP). A MMP is used to modulate the rate of a Bernoulli process with the current state of a Markov chain. A simple example is with two states: on and off, where the injection rate is r_1 or 0 respectively. At each time step transitions can be made between states. Each transition has associated probability that it is made. With two states the probability of moving to the on state is α and to the off state is β. This gives $1/\alpha$ as the average time between busts and $1/\beta$ as the average burst length. For larger α, the transitions occur more frequently and the injection process is often then referred to as *bursty*.

The injection rate of the two-state MMP can be calculated by considering the steady state distribution between on and off states. Let x_0 be probability of being in off state and x_1 be the probability of being in the on state. In a steady state we have $\alpha x_0 = \beta x_1$ and since $x_0 + x_1 = 1$, the steady state probability of being in the on state is

$$x_1 = \frac{\alpha}{\alpha + \beta}$$

then the injection rate r of the two state MMP is

$$r = r_1 x_1 = \frac{\alpha r_1}{\alpha + \beta}.$$

The last element of this synthetic workload model is the lengths of the messages sent between nodes. The simplest approach is to fix the length as a constant, or more realistically, the length can be chosen probabilistically from a measured distribution of packet lengths.

3.3.2. Throughput

The throughput of a network is the amount of data accepted per unit time at each input port. A given network topology with links of bandwidth b will have a maximum theoretical throughput Θ which can be achieved under perfect routing and flow control. This provides an upper bound on the performance achievable with a given topology. The maximum theoretical or *ideal* throughput is related closely to the channel load. The load γ_c on a channel c is is defined as the ratio of bandwidth demanded from channel c to the bandwidth of the input ports. Maximum throughput occurs when one or more channels in the network become saturated. The *maximum channel load* $\gamma_{\max}$ relates to the channel carrying the highest proportion of traffic in the network:

$$\gamma_{\max} = \max_{c \in C} \gamma_c.$$

Network saturation occurs when $\gamma_{\max} = b$ for some bottleneck channel c. The ideal throughput of a network is then defined as

$$\Theta = \frac{b}{\gamma_{\max}}.$$

Ideal throughput for regular topologies can be estimated by considering the load over the channels of a bisection. For arbitrary topologies, and arbitrary traffic patterns, computing $\gamma_{\max}$ requires finding the optimal distribution of packets across a network that minimises channel load. This can be done by solving a multi-commodity flow problem.

A simple lower bound for $\gamma_{\max}$ is calculated by the ratio between minimum channel demand, the number of channel traversals required to deliver a set of packets for a given traffic pattern, divided by the number of channels in the network, assuming the best case when all channels are loaded equally.

$$\gamma_{\max} \geq \frac{H_{\min}|N|}{|C|}$$

An upper bound for an arbitrary traffic pattern Λ can be calculated by considering an ideal routing function $R_{x,y}$ which can balance load over all minimal paths $P \in R_{x,y}$ evenly, i.e. $1/|P|$ is carried by each channel of each path. The sum over paths is weighted by $\lambda_{x,y}$. $\gamma_{\max}$ is then the maximum load over all channels, bounded by

$$\gamma_{\max}(\Lambda) \leq \max_{c \in C} \left(\frac{1}{N} \sum_{x \in N} \sum_{y \in N} \lambda_{x,y} \sum_{P \in R_{x,y}} \begin{cases} 1/|R_{x,y}| & \text{if } c \in P \\ 0 & \text{otherwise} \end{cases} \right)$$

3.3.3. Latency

The latency of a packet is the time taken to reach its destination, including the time the first header flit takes to travel across the network T_h, and L subsequent flits to be transmitted and received at the destination;

$$T = T_h + \frac{L}{b}.$$

Latency is also effected by queueing delays from the flow control mechanism caused by multiple packets arriving at a router travelling in the same direction, and from processing delays due to router decisions.

3.4. Simulation Measurements

To evaluate the performance of algorithms in terms of the latency and throughput metrics, it is necessary to measure the run-time state of simulation process. The behaviour of the simulator is non-deterministic as the traffic workload is randomly generated based on a seed value each run. Because of this accurate statistical measurements must be made which measure the underlying process to best estimate the values for the metrics. The accuracy of the measurement can be assessed to ensure a rigorous approach.

3.4.1. Warm-up Period

Measurements are based on measuring a *steady state* process, that is one in which the statistics are stationary and do not change with time. This ensures that *systematic errors* introduced by some bias in the measurements or simulation can be kept at a minimum. The simulator starts with empty buffers, so the first few packets traversing the network will experience little contention, as more traffic is injected the contention will increase. The simulator reaches a steady state after the level of contention remains at a constant level. The initial phase is called the *warm-up period* and in this time, all events should be ignored. One way in which the length of this period can be approximated is by collecting a set of samples, choosing an initial warm-up period either by a guess or inspecting a graph, and performing a linear fit on the samples not included of the warm-up period. If the line from the linear fit is approximately horizontal (gradient 0) then the simulator is considered in a steady state, if not the warm-up period can be adjusted and the linear fit recalculated.

Warm-up periods have been extensively researched within the statistical field of Monte Carlo simulation. Possible more rigorous approaches are Klomogrov-Smirnov tests, comparison of multi-chain quanitle distributions or Riemann Sums. The problem of estimating the length of a warm-up period can be an art rather than a science, however for the purposes of the investigated problem, the linear method proposed above is sufficient.

3.4.2. Steady State Sampling with Batch-Means

To measure the underlying process of the steady state simulation, the *batch-means method* can be used to obtain a sequence of independent samples (batch-means) by aggregating a set of successive measurements of the simulation into batches. This allows an accurate estimate of the standard deviation of the underlying process which is useful to asses the quality of the sample mean and the *confidence intervals*. Given a set of observations $\{X_0, X_1, ..., X_{n-1}\}$ from a single simulation run, k individual batches B_i, of size s are created. If the number of samples $n = sk$, then the batch means are defined as

$$\bar{B}_i = \frac{1}{s} \sum_{i=0}^{k-1} X_{si+j}$$

for $0 \leq i < k$. The sample mean is then calculated then as the mean value of the batches:

$$\bar{B} = \frac{1}{k} \sum_{i=1}^{k-1} \bar{B}_i.$$

Using each batch average and the sample average, the standard deviation σ^2 is estimated by

$$\hat{\sigma}^2 = \frac{1}{k-1}\sum_{i=0}^{k-1}(\bar{B} - \bar{B}_i)^2.$$

Since each sample is the average over many different measurements, the variance between batches is greatly reduced, which in turn reduces the standard deviation of the measurements. Ideally, each batch should be independent from all others so the sample mean is not biased in any way. In reality though, there may be some correlation between batches due to the expected high auto-correlation of the underlying process due to the queued flits between batches. To reduce this effect the size of the batches should be large. The sufficient number of batches is generally recognised to be around 20-30, a trade-off between accuracy of estimation of our summary statistics, and computational cost. For the experiments performed for this project, each batch will consist of 1000 simulation cycles.

3.4.3. Confidence Intervals

Confidence intervals are used to assess the reliability of statistical measurement. For a particular set of samples, it gives a range over which the true mean is contained with a given level of confidence. Confidence intervals are calculated with two main assumptions: that the underlying process is stationary, which we ensure with the warm-up period, and normally distributed, which we can't assume directly but the central limit theorem says that with a large number of independent random variables, their sum will be approximately distributed. A $100(1-\delta)$ percent confidence interval bounds the range in which the actual mean $\tilde{B}$ falls in in terms of the sample mean:

$$\bar{B} - \frac{\hat{\sigma}t_{n-1,\delta/2}}{\sqrt{k}} \leq \tilde{B} \leq \bar{B} + \frac{\hat{\sigma}t_{n-1,\delta/2}}{\sqrt{k}}.$$

The parameter $t_{n-1,\delta/2}$ of the error term is *Student's t-distribution*.

3.4.4. Throughput

Throughput is measured by recording the rate at which flits are delivered in the network. For a particular batch interval B_i, samples are taken for flits delivered in the interval. The samples are collected by each cycle, each input port adds a sample by incrementing a sample count by one if a flit is received and by zero if not. The throughput is then the count divided by the number of samples. As the offered traffic $\alpha\Lambda$ is increased the throughput should equal the demand, giving a linear relationship until the network becomes *saturated*, where the rate at which flits are delivered cannot be increased. Figure 3.4.5 shows an example throughput plot with confidence intervals.

3.4.5. Latency

Latency is measured by recording the number of cycles taken for each complete packet to be delivered the moment it is generated by a processor. After a test interval has completed, any measurement flits still in transit are recorded so not to bias the sample by excluding long-latency packets. To profile the latency behaviour for a particular algorithm the offered traffic $\alpha\Lambda$ slowly increased from 0.

As the network becomes saturated ($\alpha \geq \Theta$) and the number of undelivered packets slowly increases, the latency of each subsequent batch increases, indicating infinite latency. This means

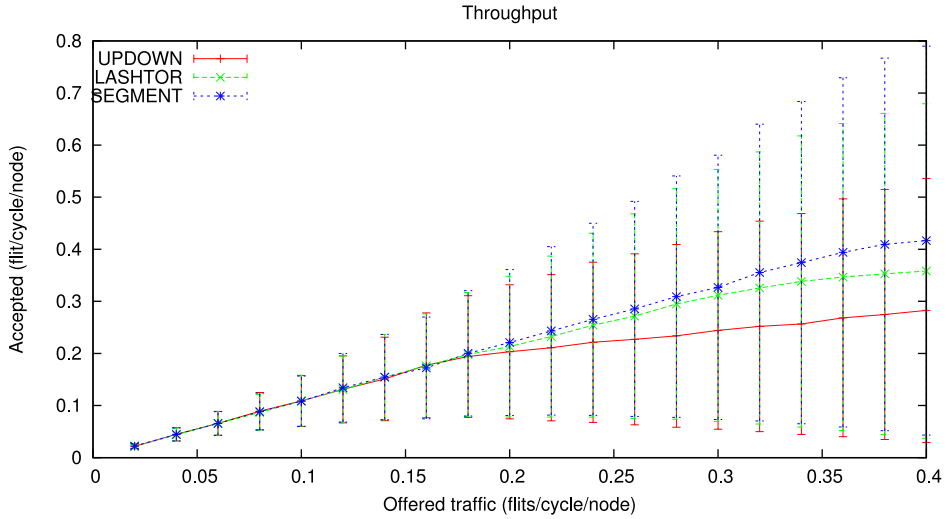


Figure 3.3.: An example throughput plot with marked confidence intervals, showing average accepted traffic as a function of offered traffic.

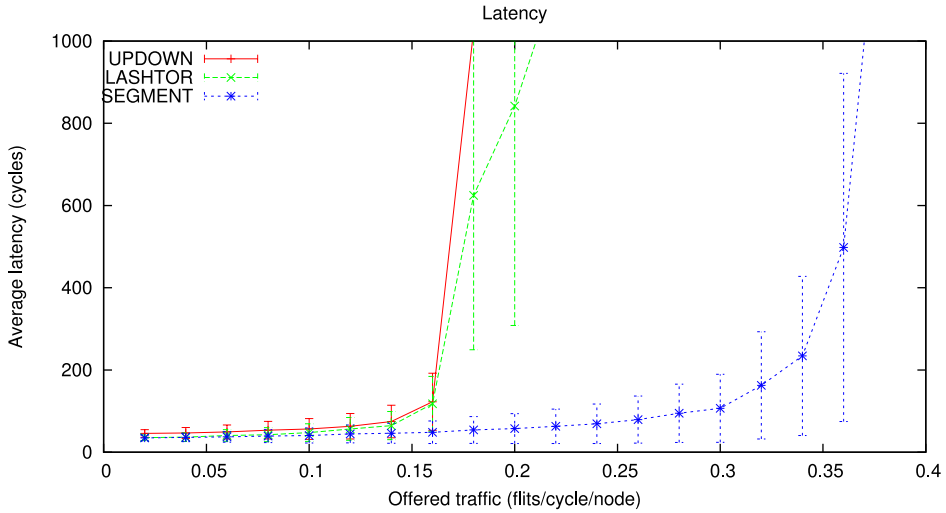


Figure 3.4.: An example latency plot with marked confidence intervals, showing average latency as a function of offered traffic.

the simulation state is no longer steady and the confidence interval for the estimated average grows proportionately. When analysing latency plots, we are interested only when the simulation state is steady and the latency is constant, after this point the results mean very little. Figure 3.4.5 shows an example latency plot with confidence intervals.

3.5. Testing & Verification of Implementation

To ensure the implementation of the simulator architecture described in this chapter was functionally correct, different methods of testing were used throughout development. The architecture of the simulator is modular and scalable which translated easily into an object-orientated style. This meant that the structure could be expressed with generic classes such as **Node**, **Router**, **Processor**

and **Link**, enabling a network to be constructed by connecting **Node**s together with **Link**s. In terms of testing, it meant that the functionality of these simulator components could be scrutinised independently of the others.

In the initial stages of development, a graphical debugging tool was created, which enabled the simulation to be stepped through one cycle at a time and displayed for each node, the state of all input and output ports with their virtual channels for the router and processor. This was invaluable in developing tricky functionality such as virtual channel allocation and message arbitration. Using this tool it was reasonably straight forward to verify the correct functionality of the simulators components.

The second stage of development was to implement the topology generation and routing algorithms. Both of these aspects could be considered independently of the simulator architecture, by defining strict interfaces to the simulator for their functionality. In developing the topology generation, GraphViz[2], a graph visualisation tool made verification straight forward. Verifying the correct behaviour of the routing algorithms was more difficult. The best approaches to doing this were to ensure all packets are delivered and that no deadlock occurred, validation against simple cases worked out with pen and paper, and analysis of the latency and throughput plots against their expected behaviour.

[2] http://www.graphviz.org/

Chapter 4.

Implementation & Results

This chapter forms the analysis of the algorithms described in Section 2.3. It includes all of the *new* work involved with this investigation. It is divided into two sections: the first is a discussion of the implementation details of the two algorithms, and the second presents the results of the empirical study of their behaviour.

4.1. Implementation of Algorithms

Implementing the LASH-TOR and SR algorithms as described by their respective papers, provided an opportunity to gain a detailed understanding of their functionality and to scrutinise their details. The following section outlines the issues highlighted by this.

4.1.1. LASH-TOR

Computational Cost

The most striking feature of LASH-TOR is how computationally intensive it is. The paper [36] claims the full algorithm has a complexity of $O(N^2)$, but even just considering step 2 of the algorithm, this cannot be true. The best known result for all-pairs shortest path is with the Floyd-Warshall algorithm [17] which runs in $O(N^3)$. Furthermore, the Floyd-Warshall algorithm would not be sufficient as LASH-TOR requires all minimal paths between source-destination pairs to be found; calculating all-pairs all-shortest paths will be significantly more than $O(N^3)$.

The LASH-TOR implementation for this project uses the JGraphT[1] library to calculate k shortest paths between a pair of nodes, for which the complexity is $O(kN^2)$. It takes several hours to compute 5 shortest paths (from which minimal ones are picked) for all pairs of nodes for a 128 node network[2]. This is not surprising as the complexity is approximately $O(kn^4)$.

Routing Function

The update to the routing function in step 3b is not at all simple, and there seems no obvious way to implement this with distributed routing tables. As an example to illustrate this, take a source-destination pair (n_s, n_d) having $\{L', S\}_{m_i}$ defined where $m_i = \{n_s, ..., n_d\}$. m_i will also contain other source-destination pairs $(n_{s'}, n_{d'})$ such that $\{n_{s'}, ..., n_{d'}\} \subset m_i$. Step 3b may choose a shortest path $m_{i'}$ between $n_{s'}$ and $n_{d'}$ not contained in m_i. This means that the port and virtual channel references for nodes in the path m_i for destination n_d will differ from those on the path $m_{i'}$ to $n_{d'}$.

As a solution in the implementation, routing decisions are made from a global list of all source-destination pairs, where each pair has $\{L', S\}_m$ defined, instead of routing tables at each node. This means that the routing function takes an additional *source node* argument.

[1] http://www.jgrapht.org/

[2] In which there are $128^2 - 128 = 16256$ source-destination pairs

Layer Balancing

With regards to step 6 of the algorithm (Section 2.3.4), balancing layer load by picking pairs assigned only to the first layer and assigning them randomly to another layer seem a bad heuristic to use because in practise the second layer also often contains many complete paths. A better approach would be for each source-destination pair assigned to a single layer, assign it to another layer with the least load, repeating this until no more pairs are subject to move. This approach was included in the implementation of LASH-TOR. Section 4.2.8 shows the result of this alteration by a comparison with and without this balancing phase on some example topologies.

4.1.2. Segment-Based Routing

Problems with Description

Implementing SR as described by the paper revealed the following problems[3].

1. The node randomly chosen to begin with must be part of a cycle, otherwise no starting segment can be found from it. To fix this, if no initial starting segment can be found, another node is randomly chosen.

2. The termination condition in the `compute_segments` method needs to immediately follow the call to `next_not_visited`.

3. In the method `find`, the node given as an argument should only be set as *tvisited* and added to the segment if it is not *visited*, as regular segments start and end on nodes already contained in segments.

Optimising Segment Discovery for Irregular Topologies

A follow-up paper, proposing the application of SR to Ethernet [28], describes how SR can be applied to random irregular networks. The aim is to reduce the number of unitary segments by making segments as short as possible, which is achieved by computing segments by using the shortest distance to any already visited switch as a premise. No further description of this seemingly simple process is given, but there is no obvious simple solution.

As a solution for the implementation, two situations are considered: firstly, when a starting segment must be found and secondly, when a regular or unitary segment must be found. For the discovery of a starting segment, any link can be chosen initially from the starting node. After this, links are chosen on the basis of the link's target node's distance from the starting node. This is calculated using a shortest path algorithm. Essentially, this produces the shortest path from a random neighbour of the start node, to the start node, using links and nodes not belonging to any segments. To stop paths tracing backwards along the path of the segment, shortest paths are calculated on a copy I' of the network I, and links are removed from I' as they are added to the segment. When a segment is successfully or unsuccessfully found, edges removed in the procedure are added back into I'.

For regular and unitary segments, links are chosen on the basis of the minimum distance to any node belonging to a segment within the current subnet and are removed from I' for the same reason when they are added to a segment. This involves computing the shortest path between all destination nodes of the unvisited links for the current node, and all of the nodes in all segments in

[3]These problems relate directly to the pseudo-code presented in the paper

the current subnet. This extra processing incurs a significant penalty as there can be many shortest paths calculations for each segment, for regular and unitary segments, this grows proportionally with the size of the subnet. Section 4.2.7 shows the effect this extra processing has on the number and type of segments found for a mesh topology.

4.2. Simulation Experiments

This section presents the experiments performed with the network simulator to compare the performance of LASH-TOR and SR. DOR and Up*/Down* routing are used as benchmark base-cases for regular and irregular topologies respectively.

4.2.1. Hypothesis

The premise of performing these experiments is to build a detailed picture of the two algorithms by evaluating them under a range of parameters. The main hypothesis is that random topologies are not homogeneous, and that in fact irregular networks may posses particular properties that effect, positively or negatively, the behaviour of routing algorithms.

4.2.2. Methodology

In order to test this hypothesis, the performance of LASH-TOR and SR will be analysed on different classes of topologies from regular to random, as described in Chapter 2. For each class of topology, network parameters such as the number of available virtual channels are also key variables to the experiments.

Plots

The analysis of routing performance will be based on plots of average latency as a function of offered traffic. It is not useful to look also at throughput as it is a unit-less quantity, expressed in terms of flits and gives little more information that a latency plot. When the network is operating at an unsaturated level ($\alpha < \Theta$), latency will be constant and throughput will equal offered traffic. When $\alpha \geq \Theta$, latency will become infinite which is represented by the vertical asymptote, and throughput will reach a threshold value which is represented by a horizontal asymptote.

As a note, both of the papers for SR and LASH-TOR [27, 36] analyse the performance of the algorithms using combined latency and throughput plots, where latency is given as a function of throughput (accepted traffic). This approach is wrong for two important reasons [7]. Firstly, it does not show that a traffic source queue is considered by the measurement. If it did, latency would be infinite at all offered traffic levels above saturation. In several of the plots the curve confusingly wraps back around because the network becomes unstable above saturation due to the throughput decreasing. Secondly, this instability means that the plot no longer represents the steady state performance of the network because source queues will start growing without bound after saturation. In a steady state, source queues are uneffected by changes in offered traffic and are of infinite size.

Variable Parameters

The two key variables in the experiments are the routing algorithm and the topology. The following results are divided into four sections regarding each class of topology: regular, degenerated regular and irregular. Each of these sections will look in detail at the performance of the algorithms

when run with the class of topology. For each class, results are presented for different network parameters such as size, traffic patterns and number of virtual channels. Each of these parameters are important to the resulting behaviour, but each additional variable increases the number of possible results, which cannot all be presented in this project. The following results aim to present a best selection of these, in order to give a detailed *overview* of performance.

The following parameters remain fixed throughout all experiments. Link delay, the number of cycles a flit takes to travel along a link is 5, all virtual channel input buffers have a capacity of 5 flits (as this equals the round-trip time for buffer allocation) and all packets are a constant size of 20 flits, which makes little difference as smaller packets would just offset the saturation point.

4.2.3. Implementation Issues

Unfortunately, there was an issue with the implementation of SR, causing it to deadlock, that could not be resolved within the time-scales of this project. The deadlock occurs only for larger irregular topologies, making it very difficult to identify the cause of the problem. Specifically, this affected all three types of random graph networks, but only the larger (64 node) d-regular and Barabási-Albert classes. Where results for SR could not be obtained, ones for Up*/Down* are substituted instead. This does not give as comprehensive comparison as aimed, but it allows for a full presentation of topologies where in some cases just the performance of LASH-TOR analysed.

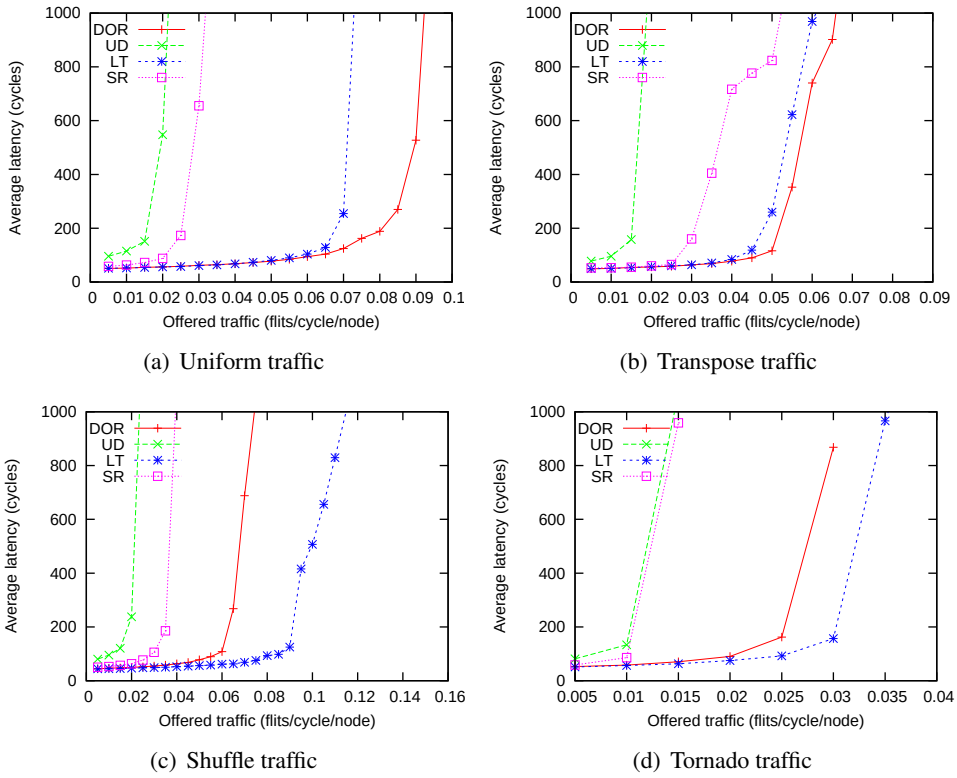
Although the implementation of SR contains this unresolved issue, the following results that include SR are still valid for the following reasons. Firstly, the segmentation of a network can easily be verified by hand on small networks, and for larger networks it is possible to check that each segment is a valid set of nodes and links and that the set of segments covers the network correctly. Secondly, it can again be verified by hand that the bidirectional turn restrictions are placed correctly and that their effect is correct. Thirdly (and most confusingly) the CDG for the paths between all source-destination pairs found using the dual graph is always acyclic, which should indicate the absence of any dependencies that could cause deadlock. For these reasons it is clear that the deadlock is caused by a very subtle issue, that has little effect on the performance characteristics of the algorithm for the topologies that do not cause deadlock.

4.2.4. Results for Regular Topologies

Very few results in the literature surrounding universal routing compare the algorithms against topology-specific algorithms like DOR. Also, when regular or degenerated regular networks are used, they are often tested only with a uniform traffic pattern, which does little to test the overall properties of the algorithm. This first section of results aims to compare the performance of Up*/Down*, SR and LASH-TOR against DOR, which acts as the base-case, on mesh networks under different traffic patterns to provide a firm basis for the overall analysis.

Affect of Traffic Patterns in Meshes

The first set of results, shown in Figure 4.1, show the effect of different traffic patterns on performance for a 8-ary 2-mesh topology of 64 nodes. The traffic patterns were chosen as they posses unique behaviours. Uniform traffic distributes load evenly over the network, transpose is a permutation reflecting the communication behaviour when performing a matrix transpose operation, shuffle is a permutation reflecting the behaviour of sort operations and the tornado works against the locality properties of the mesh. Uniform can be viewed as a base-case, transpose and shuffle as average-case and tornado as worst-case.



(a) Uniform traffic

(b) Transpose traffic

(c) Shuffle traffic

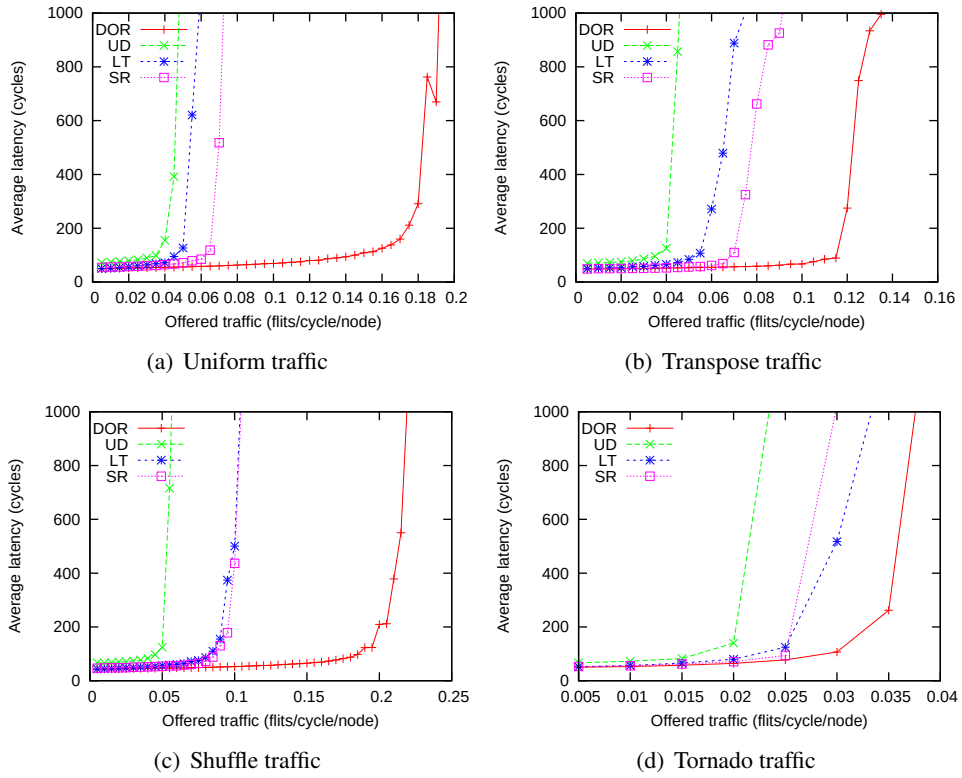
(d) Tornado traffic

Figure 4.1.: Results for routing algorithms on a 8-ary 2-mesh with different traffic patterns. LASH-TOR uses 3 layers to assign all minimal paths, DOR, Up*/Down* and SR use no virtual channels. DOR routing is used here as a basis for comparison.

A mesh topology is used throughout as DOR is deadlock free[4]. Its size is chosen for three reasons: the number of nodes must be a positive power of two for the permutation traffic patterns, a network of 256 nodes takes prohibitively long to simulate and smaller networks offer little extra insight into performance. For these experiments, 3 virtual channels were made available to LASH-TOR so all 4032 source-destination pairs could be assigned normally, for the other three algorithms no virtual channels were available for flow control. For this topology, SR routing removed 105 turns, based on 1 starting segment, 40 regular and 8 unitary segments.

The results show that Up*/Down* routing consistently performs the worst in all experiments, which is expected based on its utilisation of the network. Although its performance is similar to SR with the tornado traffic pattern which saturates the network at much lower traffic levels than the other patterns. Apart from uniform traffic, LASH-TOR out-performs DOR all cases. This is most likely due to the way uniform traffic distributes load evenly over the network, extenuating the penalties incurred with SR turn restrictions, as opposed to permutation patterns where each node has a fixed destination. Although both LASH-TOR and DOR route along minimal paths, LASH-TOR has the benefit of reduced traffic congestion as the 3 virtual layers each provide some level of flow control by separating the different flows.

[4]DOR is not deadlock free for tori as the wrap-around links mean deadlock can occur within a dimension



(a) Uniform traffic

(b) Transpose traffic

(c) Shuffle traffic

(d) Tornado traffic

Figure 4.2.: Results for the same experiments presented in Figure 4.1 but with 3 virtual channels available to DOR, Up*/Down* and SR.

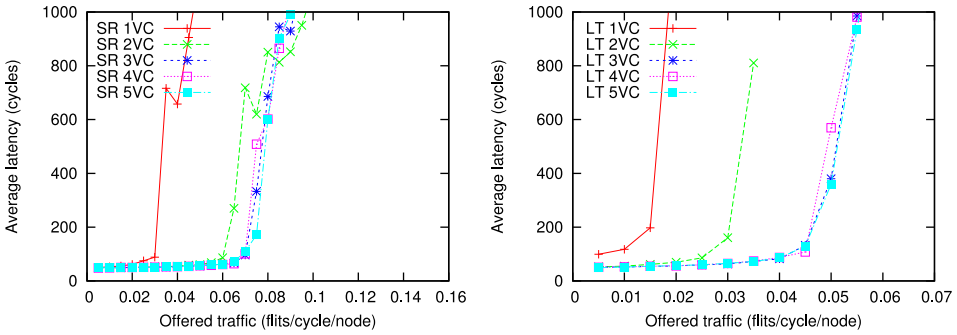
Impact of Virtual Channels

The second set of experiments, shown in Figure 4.2, are the same as the first, but this time 3 virtual channels are available to all of the algorithms. This provides DOR, Up*/Down* and SR with extra flow control to help prevent flows of traffic from blocking others when passing through switches. It gives LASH-TOR no extra advantage. The results clearly show that DOR performs best, and by a significant margin in all cases. The performance of SR is also improved substantially, either matching or out-performing LASH-TOR in all cases.

To look more closely at the effect of virtual channels on the behaviour of SR and LASH-TOR, Figure 4.3 shows the performance of the algorithms on an 8-ary 2-mesh with 1 to 5 virtual channels available. With just one virtual channel, LASH-TOR assigns minimal paths fitting in a single layer to the Up*/Down* layer and any requiring more than one, to Up*/Down* paths in this layer. With two virtual channels, many more minimal paths can be assigned, then for 3, 4 and 5 layers all paths can be assigned within these, offering marginal extra performance benefit. The behaviour of SR is similar, in that a large difference is made with the addition of one channel, then with 3 or more, little or no benefit is given.

4.2.5. Results for Degenerated Regular Topologies

The second set of experiments are performed on degenerated regular topologies. These are created by removing some percentage of links, ensuring the network remains connected, i.e. that there is a single connected component. In the performance evaluation of both SR and LASH-TOR,



(a) SR (b) LASH-TOR

Figure 4.3.: Performance of SR and LASH-TOR on an 8-ary 2 mesh with increasing numbers of virtual channels.

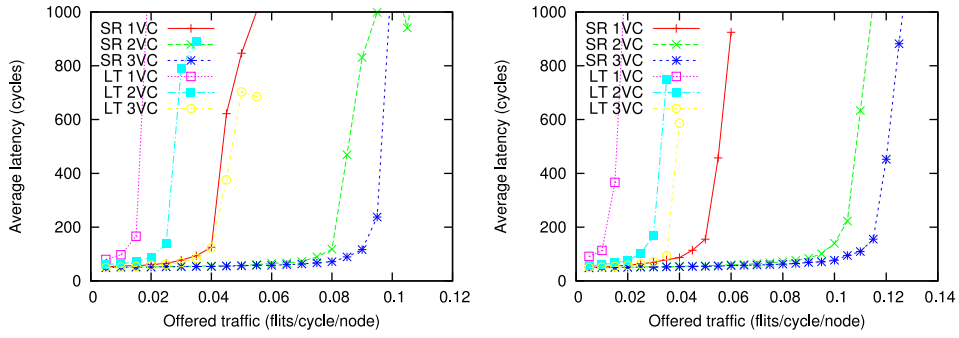
the results for degenerated topologies only consider up to 5% link failures. The results given in this section aim to build a more general picture by considering between 5% and 50% link failures. Topologies with higher failure percentages are not presented as none of the original regular properties will be retained. The resulting degenerated topologies are used to evaluate the performance of SR and LASH-TOR. Additionally, results are given with 1, 2 and 3 virtual channels available for each algorithm. The transpose traffic pattern is used throughout as it gives a good average-case.

The first set of results, presented in Figure 4.4, show the performance of the algorithms on a degenerated 8-ary 2-mesh. The second set, presented in Figure 4.5, are based on a degenerated 8-ary 2-cube (64-node torus network). There is little difference between the results of the two topologies; the algorithm's relative performances remain very similar. This is not unexpected as mesh and torus networks are very similar, we can see though that the extra wrap-around links in the torus improve the saturation level by a small amount in each case. Consistent with the results for the regular mesh topology, with one virtual channel LASH-TOR out-performs SR, but with more than two virtual channels available, SR out-performs LASH-TOR in nearly all cases, often by a large margin.

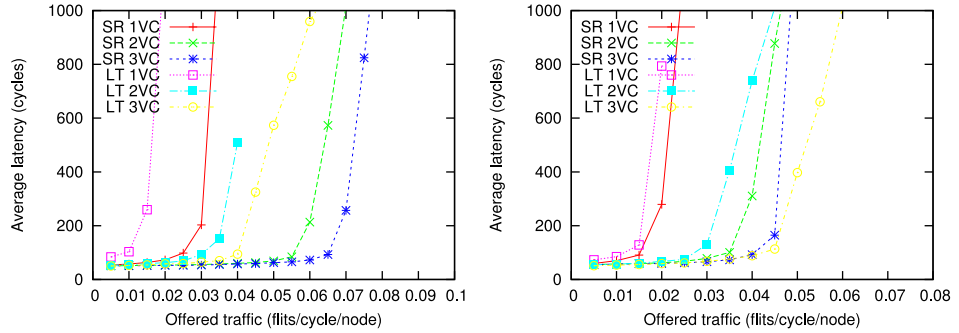
4.2.6. Results for Irregular Topologies

This final section presents results for random irregular topologies; specifically for the three classes of random graph, introduced in Chapter 2, each with unique properties. The aim is to test the main hypothesis and analyse the effect these properties have on the performance of the routing algorithms. Apart from different algorithms, results are presented for three variations of parameters: network size, traffic patterns and number of available virtual channels.

The properties of the random graph models used are more pronounced as the number of nodes increases, but as the size of the network is bounded by the feasible run time of the experiments, a maximum size of 64 nodes is presented. 32 node networks are also presented as a comparison, but it is expected for this size that the behaviour of the algorithms will be much more similar across the classes than for 64 nodes. Table 4.1 outlines the parameters used for the topologies and the resulting number of links. Two traffic patterns are used: uniform again as a base case, and random permutation as it is equivalent to other permutations as there is no regularity or locality in random topologies.

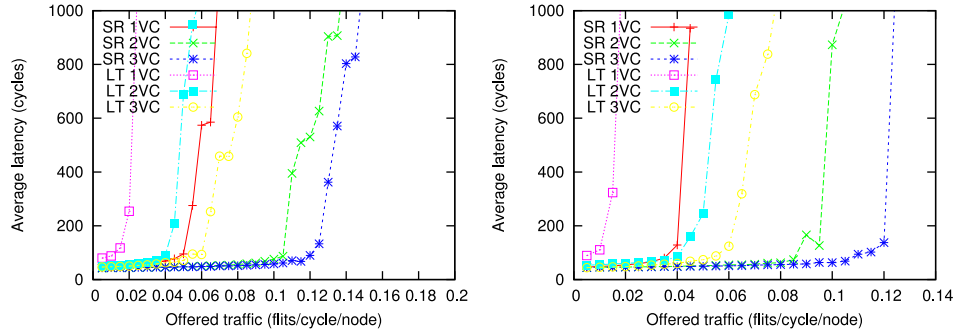


(a) 5% faults (b) 10% faults

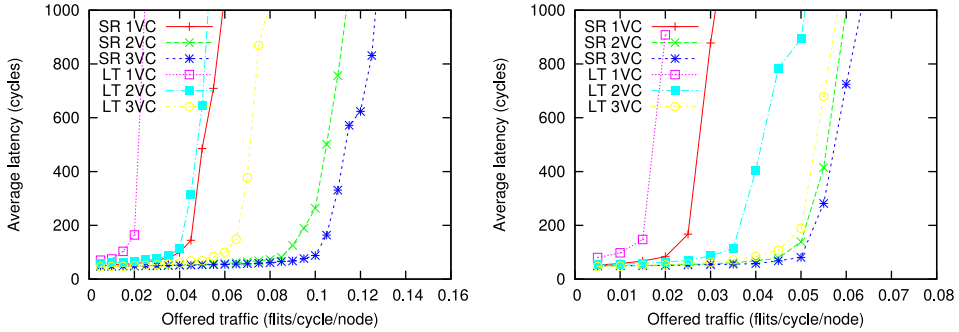


(c) 25% faults (d) 50% faults

Figure 4.4.: Results showing the performance of SR and LASH-TOR with 1 to 3 available virtual channels for an 8-ary 2-mesh with increasing levels of faults.



(a) 5% faults (b) 10% faults



(c) 25% faults (d) 50% faults

Figure 4.5.: Results showing the performance of SR and LASH-TOR with 1 to 3 available virtual channels for an 8-ary 2-cube with increasing levels of faults.

Erdős-Rényi			d-regular expander			Barabási-Albert		
Nodes	p	Links	Nodes	d	Links	Nodes	m	Edges
32	0.1	118	32	3	48	32	2	60
32	0.15	162	32	6	96	32	8	192
64	0.1	430	64	3	96	64	4	240
64	0.15	628	64	6	192	64	16	768

Table 4.1.: The parameters used to construct the random graphs with the resulting number of edges.

Erdős-Rényi Graph

Due to SR deadlocking on both 32 and 64 node Erdős-Rényi topologies, the results shown in Figure 4.6 and Figure 4.7 present performance of Up*/Down* and LASH-TOR instead. The behaviour of both algorithms is very consistent throughout. For a single virtual channel, there is little difference in saturation level, but for 2 and 3 virtual channels, the performance of LASH-TOR improves significantly. For both 32 and 64 nodes, the benefit is larger for the greater edge probability p of 0.15 instead of 0.1. This is because the network becomes more connected, which cannot be exploited by Up*/Down, but allows LASH-TOR greater freedom in choosing its shortest paths that minimise the number of layers required.

With the 64 node topology, it has the effect of saturating Up*/Down* up to 3 virtual channels at offered traffic of less than 0.05 for both traffic patterns, compared with the 32 node case where it saturates at just less than 0.1. This is due to Up*/Down*'s poorer utilisation of the network for 64 nodes. Interestingly, the performance of both algorithms is very similar for both uniform and permutation traffic. The only odd result is for the 32 node topology with $p = 0.15$ and permutation traffic, where LASH-TOR with 2 virtual channels saturates at a higher traffic level than with 3, but this is likely to be an anomaly.

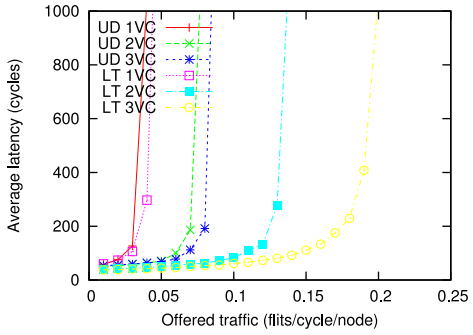
d-regular Expander Graph

The key property of an expander graph is that nodes are well connected to the rest of the graph, i.e. any other nodes are always only a short distance away. Figure 4.8 shows the results for SR and LASH-TOR on a 32 node, 3 and 6-regular random graph with uniform and permutation traffic. The behaviour shown is again consistent with that of the degenerated regular topologies, in that SR easily out-performs LASH-TOR with 2 and 3 virtual channels. Figure 4.8 (c) shows some different behaviour, where saturation is very similar for both algorithms with 2 and 3 virtual channels. In both topologies, permutation traffic gives best performance.

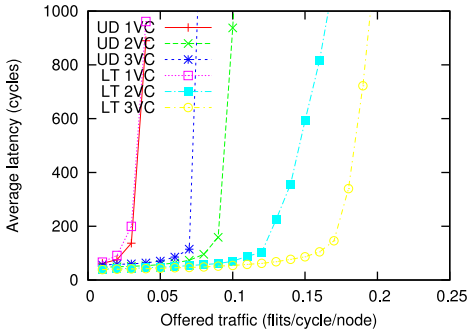
Figure 4.9 shows results for 64 nodes, but this time with Up*/Down* substituted with SR. The main difference between these results, and those for the Erdős-Rényi graph results in Figure 4.7, is that the a third virtual channel is much more beneficial for LASH-TOR. This is because the 64 node d-regular graph is less connected than the 64 node Erdős-Rényi graph, so less paths are available to LASH-TOR, increasing required number of the layers.

Barabási-Albert Graph

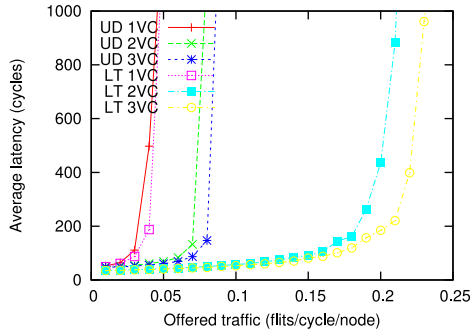
A Barabási-Albert random graph is created with a preferential attachment mechanism which creates highly connected sub groups of nodes, called clusters. A clustered structure may also implies good communication as it can be viewed as a 2-level hierarchy where each node in the



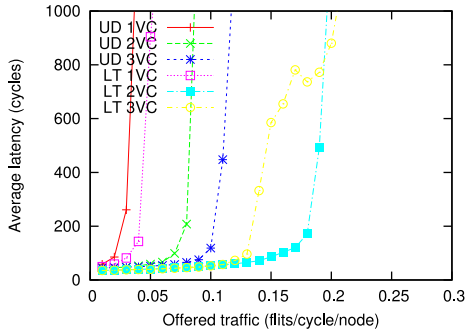
(a) Uniform traffic, $p = 0.1$



(b) Permutation traffic, $p = 0.1$

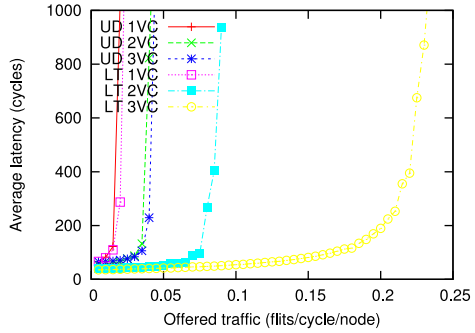


(c) Uniform traffic, $p = 0.15$

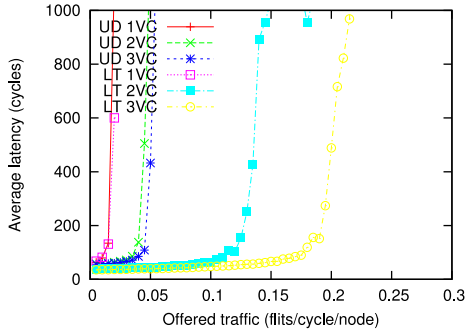


(d) Permutation traffic, $p = 0.15$

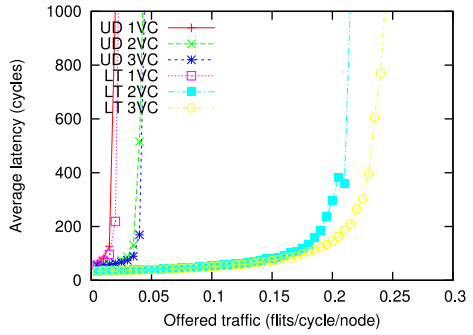
Figure 4.6.: Results for a 32 node Erdős-Rényi graph with uniform and permutation traffic



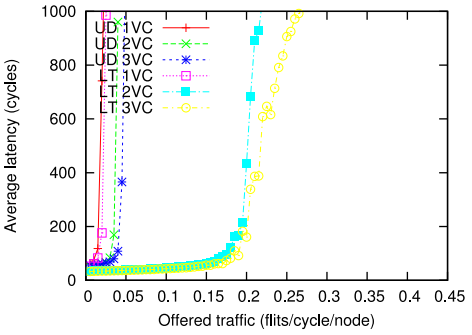
(a) Uniform traffic, $p = 0.1$



(b) Permutation traffic, $p = 0.1$

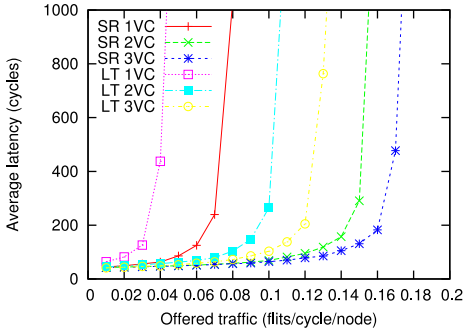


(c) Uniform traffic, $p = 0.15$

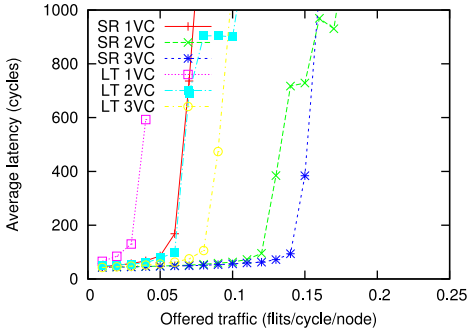


(d) Permutation traffic, $p = 0.15$

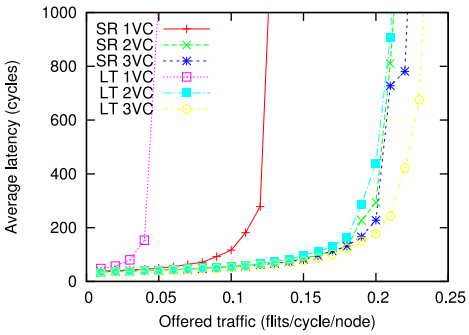
Figure 4.7.: Results for a 64 node Erdős-Rényi graph with uniform and permutation traffic



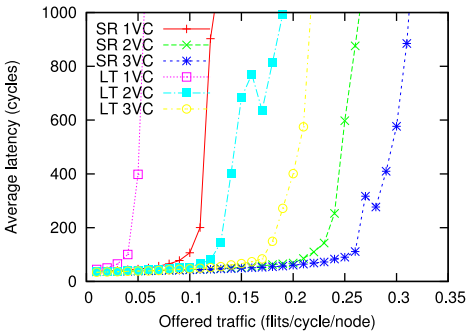
(a) 3-regular with uniform traffic



(b) 3-regular with permutation traffic

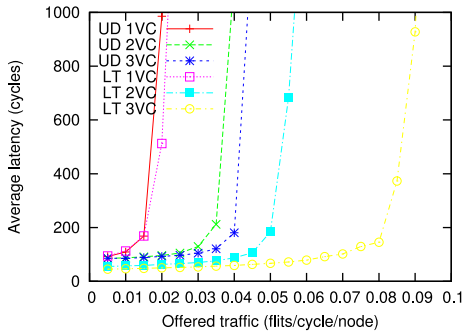


(c) 6-regular with uniform traffic

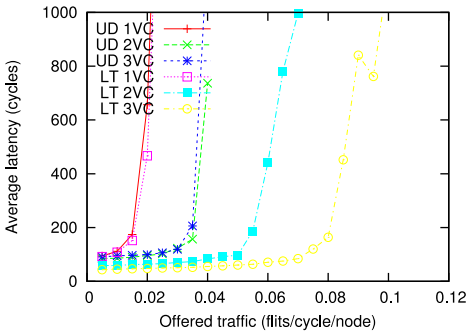


(d) 6-regular with permutation traffic

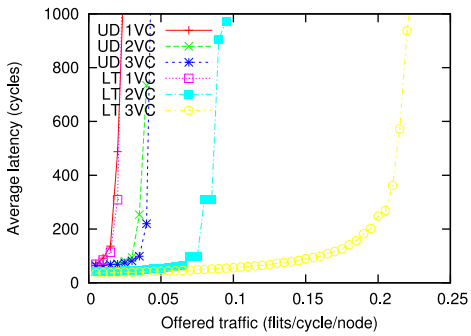
Figure 4.8.: Results for a 32 node expander graph with uniform and permutation traffic



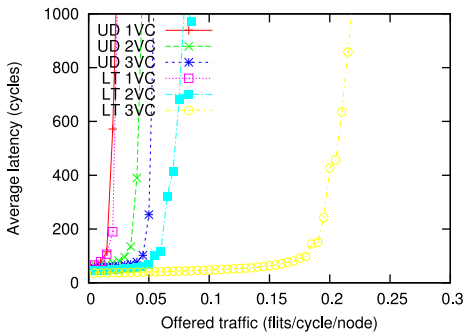
(a) 3-regular with uniform traffic



(b) 3-regular with permutation traffic

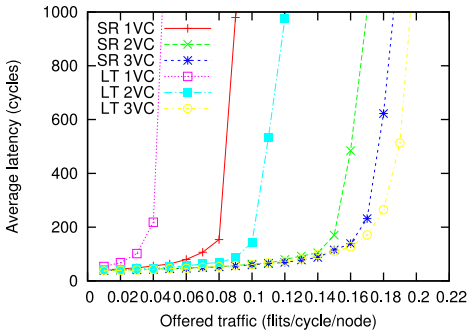


(c) 6-regular with uniform traffic

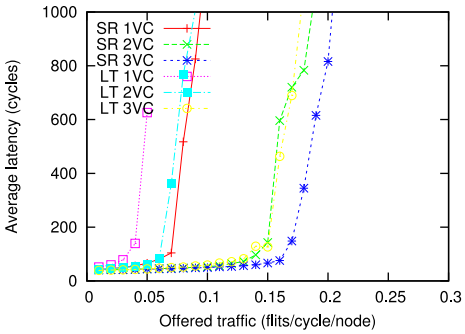


(d) 6-regular with permutation traffic

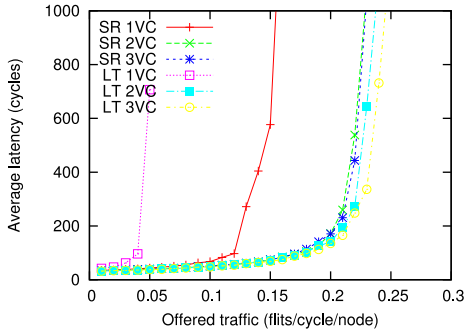
Figure 4.9.: Results for a 64 node expander graph with uniform and permutation traffic



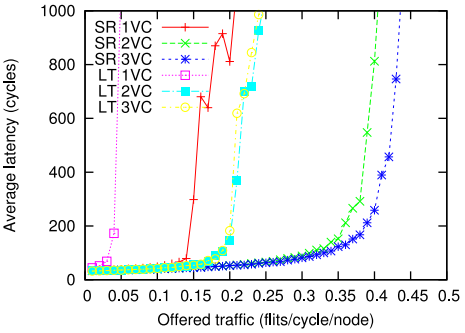
(a) Barabási-Albert ($m = 2$) with uniform traffic



(b) Barabási-Albert ($m = 2$) with permutation traffic

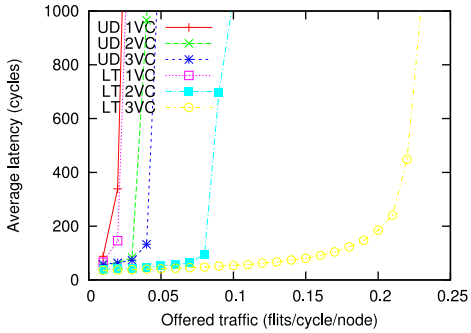


(c) Barabási-Albert ($m = 4$) with uniform traffic

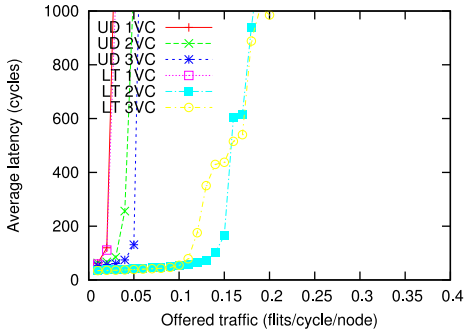


(d) Barabási-Albert ($m = 4$) with permutation traffic

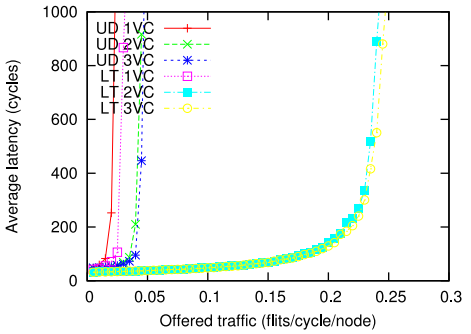
Figure 4.10.: Results for a 32 node Barabási-Albert graph with uniform and permutation traffic



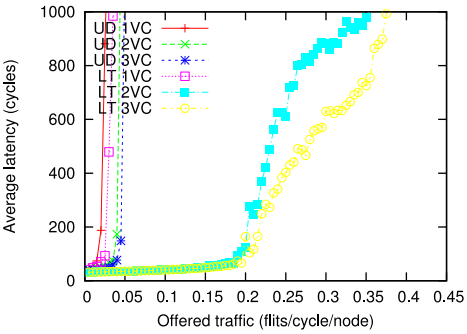
(a) Barabási-Albert ($m = 2$) with uniform traffic



(b) Barabási-Albert ($m = 2$) with permutation traffic



(c) Barabási-Albert ($m = 4$) with uniform traffic



(d) Barabási-Albert ($m = 4$) with permutation traffic

Figure 4.11.: Results for a 64 node Barabási-Albert graph with uniform and permutation traffic

	Random walk				Shortest path			
Topology	Subnets	Start.	Reg.	Uni.	Subnets	Start.	Reg.	Uni.
64 mesh	1	1	8	23	1	1	40	8
64 torus	1	1	11	52	1	1	44	20
64 dmesh (5%)	1	1	7	25	1	1	38	7
64 dmesh (50%)	8	1	5	10	8	1	10	5
64 dtorus (5%)	1	1	3	20	1	1	43	18
64 dtorus (50%)	3	1	14	13	3	1	25	5
32 ER (p=0.1)	1	1	11	15	3	1	18	9
64 ER (p=0.1)	1	1	10	77	1	1	54	97
32 EX (d=6)	1	1	16	39	1	1	54	97
64 EX (d=6)	1	1	1	41	1	1	49	76
32 PR (m=8)	1	1	0	67	1	1	29	94
64 PR (m=8)	1	1	5	110	1	1	60	270

Table 4.2.: A comparison of the basic segmentation algorithm where segments are discovered by performing a random walk on the network, against the shortest path approach where the path is chosen by the shortest distance to existing segment elements. The table shows for both approaches, the different distribution of segments found on a range of topologies. Mesh and tori are square and 2-dimensional. The following shorthand is used *dmesh* and *dtorus* are degenerated mesh and torus networks with some percentage of faults, *ER*=Erdős-Rényi graph, *EX*=expander graph and *PR*=Barabási-Albert preferential attachment graph.

same cluster is a short distance away, and each cluster is only a short distance away. Figure 4.10 shows the results for SR and LASH-TOR on a Barabási-Albert random graph, which are again very similar to those of the degenerated regular (Figure 4.4, Figure 4.5) and d-regular (Figure 4.8) topologies. This is expected as for only 32 nodes, the structure of these graph classes will not be particularly different.

Figure 4.11 shows the results for Up*/Down* and LASH-TOR on 64 node Barabási-Albert networks. They are again very similar to those of the Erdős-Rényi topology (Figure 4.7), except for the performance of LASH-TOR with 2 and 3 virtual channels under permutation traffic, where in the Barabási-Albert network, it is not improved with the third virtual channel.

4.2.7. SR Segment Searching

For SR to be better applied to irregular topologies it is proposed in the paper [28] that segments should be made as short as possible to reduce the number of unitary segments, as these generally incur the most turn restrictions. This is achieved by choosing links in a segment path that are closest to a node already belonging to a segment, as opposed to choosing links randomly and essentially performing a random walk. An outline of the implementation of this is given at the beginning of this chapter. Table 4.2 shows the resulting number and types of segments discovered by SR with and without this change to the algorithm. It clearly shows that the number of unitary segments are reduced as more regular segments are found. In the simple random walk case, between 0 and 16 regular segments are found, whereas between 10 and 60 with extra shortest path calculation.

4.2.8. LASH-TOR Layer Balancing

After shortest paths have been calculated and assigned to layers, a phase of layer load balancing is preformed. This is described in detail at the beginning of this chapter. For an 8-ary 2-mesh with 4 available layers, without balancing, the 4032 source-destination pairs are assigned to only two layers. The load on each layer directly corresponds to the number of edges in each of the associated channel dependency graphs. The CDG for layer 1 has 13322 edges and the CDG for layer 2 has 3039 edges. When layer balancing is used, the two other available layers are used, with the CDG of layer 1 containing 11634, layer 2 7619, layer 3 7619 and layer 4 7619.

This distribution is expected as all paths assigned to two layers will begin in layer 1 and cannot be moved. The other 3 layers are very equally balanced due to the number of length 2 paths that introduce a single edge in a CDG. This kind of result from the balancing phase is common to all of the considered topologies.

4.2.9. Conclusions

Validity of the Hypothesis

The main hypothesis was that the different properties of random graphs will effect the behaviour of the routing algorithms. It it clear from the results presented that this does not hold and little can be concluded about the effects of the different properties of random graphs. There are two likely reasons why this is true. The first is that the size of the random graphs are relatively small. The properties of random graphs such as clustering and high-connectivity apply asymptotically and apply only weakly for small graphs. Repeating the experiments on graphs with hundreds or thousands of nodes could yield far better results, but this would of course require a simulator capable of such large networks. The second reason is that SR and LASH-TOR show little relative differences between graph classes because, fundamentally, they operate in very similar ways by routing along minimal, or near-minimal paths. For more pronounced effects, experiments could be performed with routing algorithms of different natures.

From this we can conclude that small random graphs of up to 64 nodes are essentially equivalent in terms of their effect on their behaviour on routing. It remains though that for much larger networks, such properties may well have an affect, and as parallel architectures continue to develop, we may find they come into play.

Affects of Virtual Channels

The most interesting insight these results have given is the effect that virtual channels have on the performance of SR and LASH-TOR. The point of SR is not to use virtual channels to break deadlock, which means that is more flexible in terms of the systems it can be applied to. When SR operates without any virtual channels, we see the kind of comparison shown in the results in Figure 4.1, where SR is vastly out-performed by LASH-TOR which uses 3. However, when more virtual channels are made available to SR, the improved flow control that these provide means that it's performance is increased beyond that of LASH-TOR's. This behaviour is illustrated by the results in Figure 4.2.

The performance of LASH-TOR may be better than that of SR, but only when SR has no virtual channels available to it. These results indicate that it would always be beneficial to use SR over LASH-TOR as there would be some number of virtual channels available, and obviously never the opposite way. LASH-TOR could compensate for the virtual channel performance boost given to SR, by expanding each layer to v virtual channels to improve the flow control. This would

almost certainly improve the performance of LASH-TOR up-to or beyond that of SR with an equal number of virtual layers, as all paths are routed minimally, unlike SR. It would though, require v times the number of required layers, virtual channels to do this, which would be prohibitive to the use of LASH-TOR in many systems.

Algorithm Enhancements

Lastly, both of the extra features implemented for SR and LASH-TOR gave clear benefits to their performance. For SR's improved segment finding algorithm, it is clear from the results presented in Table 4.2 that this is an effective method. However, because of the overhead incurred in the computation involved with this, it is not in any way scalable and unsuitable for large systems or systems with limited resources. The evaluation in the following section proposes some possible ideas to improve the cost of this.

In contrast, LASH-TOR's layer balancing phase has a minimal computational cost and produces good results. It is very simple though, and only balances paths which are entirely assigned to a single layer. This could be improved by as it is possible to include sub-paths in the balancing.

Chapter 5.

Evaluation

The aim of this project is to investigate and explore current approaches to routing in irregular processor networks. In Chapter 1, six objectives were set out in order to achieve this. The main objectives amongst these were the implementation of the algorithms in a simulation framework and evaluation of their behaviour and performance through a detailed empirical study. This chapter details the current status of the project by evaluating its successes against the original aims. It will also detail possible future directions, based on the work completed.

5.1. Successes

5.1.1. Research & Implementation

The key successes of the implementation aspects of this project were the network simulation tool and the routing algorithms. The simulation tool was critical in gaining a complete understanding of the intricacies of interconnection networks. Designing it in a modular and extensible way, meant that it was straight forward in the early stages of the project to play around with the functionality and implementation of different routing algorithms, topologies, and to see the effect that different network and simulation parameters have on run-time behaviour. For instance, by implementing a simple algorithm that routed along minimal paths, it was significant and very satisfying to see that it deadlocked in some situations in the simulator.

This functionality of the simulator was vital in the implementation of the more complex algorithms LASH-TOR and SR. Again, implementing these meant that every detail of their descriptions had to be scrutinised, which gave an in-depth understanding of their composition. This process also highlighted a number of issues in their design and even problems and errors with their descriptions. The implementation of both the simulator and routing algorithms were not at all trivial as the nature of a network system is inherently complex. Many components operate in parallel with each other, each with its own complex functionality.

One of the most difficult aspects was identifying and fixing implementation bugs. In the early stages of the simulator, these related to issues like the allocation and state transitions of virtual channels. At this stage though, it was possible to step through simulation and inspect the states of components such as that of input and output ports. As the simulator grew more complex and with the implementation of LASH-TOR and SR, it became much harder to track-down and fix problems. This difficulty resulted in a problem with SR, causing it to deadlock only in certain large random topologies. With symptoms of the problem occuring only in networks of 64 nodes and with hundreds of links, there is no straight-forward way to address the problem. Consequently, due to this and the time scales of the project, it could not be resolved. Although the SR implementation was not entirely successful, the issue highlights well the complex nature of deadlock in routing algorithms and why it remains the central focus of research into routing.

In terms of evaluation against the project objectives, both of these initial implementation stages of the project were underpinned with a great deal of research and investigation, before, and continuing throughout. This involved the surrounding literature in academic papers, text books and the Internet, and hence the research and literature review aim was, in that sense, achieved

successfully. The implementation of the simulator was also successful, and implementation of the algorithms as described was partially successful.

5.1.2. Experimentation and Analysis

It was not until mid-way through the experimentation phase of the project that the problems with SR presented themselves. This was a serious problem as it meant that four sets of results comparing LASH-TOR against SR could not be obtained. In order to present the best analysis without SR, Up*/Down* routing was substituted as a comparison with LASH-TOR.

Even with these problems, the set of results obtained from the experiments were comprehensive and informative. They showed that the hypothesis made about the effects of the properties of random graphs did not hold, and the conclusion was made, that for small networks, random topologies are essentially homogeneous. The results gave a separate and important insight into the effect that virtual channels have on the performance of SR. In particular, that given the same number of virtual channels as LASH-TOR, its performance is generally much better. This is significant as algorithms using turn prohibition or virtual channels to break deadlock are not often evaluated against each other. Specifically, no comparison has yet been made between LASH-TOR and SR, which essentially represent the best performing algorithms in these two categories.

The experimentation also included a brief analysis of the enhancements made to each of the algorithms. This did not contribute to the conclusion of the hypothesis, but it helped to illustrate the details and nature of their behaviour, particularly for SR, where a seemingly simple statement made in the original paper, turned out to be highly computationally intensive.

Even with the problems experienced with SR, the obtained results were successful in that they presented a comprehensive analysis of universal routing algorithms. The conclusions made identified key aspects within the results. They were reinforced by the discussion of the details of the algorithms, highlighting primarily the computational and centralised nature of both algorithms.

5.2. Further Work

5.2.1. Experimentation

There are a two immediate changes that could be made to improve the quality of the results. Firstly, to increase the confidence in those obtained on random topologies it would be beneficial to average results over a set of perhaps 10 or more instances of a topology class, each with a different seed value. In doing this, a more general picture would be given of the properties of a topology rather than a single instance, which could by itself produce anomalous results. Secondly, as described in the conclusion of Section 4.2, running the tests for much larger networks of 128, 512 or 1024 nodes would give a much better impression of the characteristics of the topologies. This could be achieved either by rewriting the simulator to optimise its performance, perhaps as event-driven rather than cycle-based, or just to provide much more computing muscle (somewhat ironically) by running it on a HPC system.

Ideally, it would be better to entirely move away from software simulation and implement and test the routing algorithms in real systems, as this would provide a true evaluation of their performance, rather than with a large set of assumptions and abstractions in a simulator. Such an approach may in fact be possible as the Bristol-based company XMOS[1], who design multi-core processors, have recently developed a new board with 16 of their 4-core chips connected together, giving a 64-core system. Each chip runs 8 hardware threads so it would be possible with this

[1] http://www.xmos.com/

system to implement, for example, SR routing on perhaps 4 of the threads, and use the others to generate traffic and monitor the performance of the system in real-time without effecting the routing.

5.2.2. Improvements to Algorithms

The obvious next step in any work continuing from this project is to implement changes to the SR and LASH-TOR algorithms. The following points suggest areas where they might be improved.

LASH-TOR

- **Adding adaptivity to R.** The implementation of LASH-TOR for this project routed obliviously along minimal paths. This approach to routing does not balance load well or react dynamically to changes in the network. It would be interesting to see the effects that adaptivity in LASH-TOR's routing function R would have on its performance both in preprocessing and operation.
- **Virtual layer balancing.** The description given by the paper of the process of balancing the load over virtual layers considers only paths assigned to a single layer. It could be improved by considering all sub-paths split across layers.
- **Improve base-line routing algorithm.** When it is not possible to assign a path to the set of available layers, it is assigned to a separate layer using paths determined by the Up*/Down* algorithm. A straight-forward performance improvement would be to use a different algorithm, not dependent on virtual channels. For instance, SR could be used.

Segment-Based Routing

- **Improve segmentation.** From the issues outlined by the discussion of SR's segmentation algorithm, it is clear that there is room for a great deal of improvement. The most significant issue is in reducing the computational cost of discovering segments in random topologies.
- **Optimise the placement of restrictions.** Currently, restrictions are placed within segments randomly, but it could be beneficial to optimise against a heuristic based, for example, on latency or throughput. Also, the selection of the start node is also random; maybe this too could be improved.
- **Adding adaptivity to R.** As with LASH-TOR, it would be interesting to see how well adaptivity could be added to SR and the effect it would have on its performance.
- **Path selection.** Whether the algorithm is minimal, or non-minimal the preprocessing phase of SR gives a constrained graph on which any paths can be taken and will not cause deadlock. Path selection could be improved, for example, by optimising link use over all paths.

Bibliography

[1] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, 2002.

[2] Krste et al. Asanovic. The landscape of parallel computing research: A view from berkeley. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf, December 2006. [accessed 12th March 2009].

[3] J. Aez, T. De La Barra, and B. Prez. Dual graph representation of transport networks. *Transportation Research Part B: Methodological*, 30(3):209–216, June 1996.

[4] Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. The effect of faults on network expansion. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 286–293, New York, NY, USA, 2004. ACM.

[5] L. Cherkasova, V. Kotov, and T. Rokicki. Fibre channel fabrics: evaluation and design. volume 1, pages 53–62 vol.1, Jan 1996.

[6] Andrew A. Chien and Jae H. Kim. Planar-adaptive routing: low-cost adaptive networks for multiprocessors. *J. ACM*, 42(1):91–123, 1995.

[7] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[8] William J. Dally. *A VLSI Architecture for Concurrent Data Structures*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[9] William J. Dally. Computer architecture is all about interconnect. 8th International Symposium on High-Performance Computer Architecture, http://www.hpcaconf.org/hpca8/sites/hpca8-panel/DallySlides.pdf, 2002.

[10] William J Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *DAC '01: Proceedings of the 38th Conference on Design Automation*, pages 684–689, 2001.

[11] W.J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):194–205, Mar 1992.

[12] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *Computers, IEEE Transactions on*, C-36(5):547–553, May 1987.

[13] José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, 1993.

[14] José Duato. A theory of fault-tolerant routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 8(8):790–802, 1997.

[15] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae*, 6:290–297, 1959.

[16] S.V. Adve et al. Parallel computing research at illinois: The upcrc agenda. Parallel@Illinois, University of Illinois at Urbana-Champaign http://www.upcrc.illinois.edu/documents/UPCRC_Whitepaper.pdf, November 2008. [accessed 12th March 2009].

[17] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[18] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *In Proceedings of the International Symposium on Computer Architecture*, pages 278–287, 1992.

[19] I. Gopal. Prevention of store-and-forward deadlock in computer networks. *Communications, IEEE Transactions on*, 33(12):1258–1264, Dec 1985.

[20] H. Hofestädt, A. Klein, and E. Reyzl. Performance benefits from locally adaptive interval routing in dynamically switched interconnection networks. In *EDMCC2: Proceedings of the 2nd European conference on Distributed memory computing*, pages 193–202, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[21] Intel. An introduction to the intel quickpath interconnect. http://www.intel.com/technology/quickpath/introduction.pdf, January 2009. [accessed 16th March 2009].

[22] S. Yalamanchili J. Duato and L.M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kauffman Publishers Inc., 2002. second ed.

[23] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano. L-turn routing: an adaptive routing in irregular networks. pages 383–392, Sept. 2001.

[24] M. Koibuchi, A. Jouraku, K. Watanabe, and H. Amano. Descending layers routing: a deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies. pages 527–536, Oct. 2003.

[25] D.H. Linder and J.C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, 1991.

[26] O. Lysne, T. Skeie, S.-A. Reinemo, and I. Theiss. Layered routing in irregular networks. *Parallel and Distributed Systems, IEEE Transactions on*, 17(1):51–65, Jan. 2006.

[27] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie. Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, April 2006.

[28] A. Mejia, J. Flich, J. Duato, Sven-Arne Reinemo, and Tor Skeie. Boosting ethernet performance by segment-based routing. pages 55–62, Feb. 2007.

[29] Gordon E. Moore. Cramming more components onto integrated circuits. Electronics Magazine ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf, 1965. [accessed 12th March 2009].

[30] M. Nielsen. Introduction to expander graphs. www.qinfo.org/people/nielsen/blog/archive/notes/expander_graphs.pdf, 2005. [accessed 30th March 2009].

[31] George F. Riley Richard M. Fujimoto, Kalyan S. Perumalla. *Network Simulation*. Morgan & Claypool Publishers, 2007.

[32] J. C. Sancho, A. Robles, J. Flich, P. Lpez, and J. Duato. Effective methodology for deadlock-free minimal routing in infiniband networks. *Parallel Processing, International Conference on*, 0:409, 2002.

[33] José Carlos Sancho, Antonio Robles, and José Duato. A flexible routing scheme for networks of workstations. In *ISHPC '00: Proceedings of the Third International Symposium on High Performance Computing*, pages 260–267, London, UK, 2000. Springer-Verlag.

[34] Christian Scheideler. Theory of network communication: Network theory, routing (course notes). http://www.cs.jhu.edu/~scheideler/courses/600.348_F04/, 2004. [accessed 24th April 2009].

[35] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *Selected Areas in Communications, IEEE Journal on*, 9(8):1318–1335, Oct 1991.

[36] T. Skeie, O. Lysne, J. Flich, P. Lopez, A. Robles, and J. Duato. Lash-tor: a generic transition-oriented routing algorithm. pages 595–604, July 2004.

[37] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered shortest path (lash) routing in irregular system area networks. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 194, Washington, DC, USA, 2002. IEEE Computer Society.

[38] A. Steger and N. C. Wormald. Generating random regular graphs quickly. *Comb. Probab. Comput.*, 8(4):377–396, 1999.

[39] M.B. Stensgaard and J. Sparso. Renoc: A network-on-chip architecture with reconfigurable topology. pages 55–64, April 2008.

[40] Christof Teuscher. Topology-unaware routing in irregular self-assembled networks-on-chip: an explorative case study. In *Nano-Net '07: Proceedings of the 2nd international conference on Nano-Networks*, pages 1–5, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[41] TOP500.Org. Top 500 supercomputing sites (1-100). TOP500 List http://www.top500.org/list/2008/11/100, November 2008. [accessed 13th March 2009].

[42] Brian Towles. *Distributed Router Fabrics*. PhD thesis, Stanford University, 2005.

[43] L. G. Valiant. General purpose parallel architectures. pages 943–973, 1990.

[44] Hangsheng Wang Vassos Soteriou and Li-Shiuan Peh. A statistical traffic model for on-chip interconnection networks. In *International Conference on Measurement and Simulation of Computer and Telecommunication Systems (MASCOTS '06)*, September 2006.

Appendix A.

Proofs

A.1. Theorem 1

Every network of size n and maximum degree δ must have a diameter bounded by

$$\Omega\left(\frac{log|N|}{log(\delta-1)}\right)$$

.

Proof. This proof is adapted from [34]. Suppose we have a network $I = (N, C)$ of maximum degree δ and size $|N|$. Starting from any node $n \in N$, in one hop we can reach at most δ nodes. In two hops, we can reach at most $\delta \cdot (\delta - 1)$ nodes. In k hops, we can reach at most

$$1 + \sum_{i=0}^{k-1} \delta \cdot (\delta-1)^i = 1 + \delta \cdot \frac{(\delta-1)^k - 1}{(\delta-1)-1} \leq \frac{\delta \cdot (\delta-1)^k}{\delta-2}$$

This has to be at least $|N|$ to ensure n can reach all other nodes in N within k steps. Hence,

$$\begin{aligned}
(\delta-1)^k &\geq \frac{(d-2)\cdot n}{d} \\
klog_{\delta-1}(\delta-1) &\geq log_{\delta-1}(\delta-2) + log_{\delta-1}(n) - log_{\delta-1}(\delta) \\
k &\geq log_{\delta-1}\left(\frac{\delta-2}{\delta}\right) + log_{\delta-1}(n) \\
&\geq -2 + log_{\delta-1}(n) \qquad \text{(A.1)} \\
&\geq \lfloor log_{\delta-1}(n) \rfloor - 1
\end{aligned}$$

In line A.1, we have used that

$$\begin{aligned}
& log_{\delta-1}\left(\frac{\delta-2}{d}\right) > -2 \\
\Leftrightarrow\ & (\delta-1)^{log_{\delta-1}\left(\frac{\delta-2}{\delta}\right)} > (\delta-1)^{-2} \\
\Leftrightarrow\ & \frac{\delta-2}{\delta} > (\delta-1)^{-2} \\
\Leftrightarrow\ & \frac{\delta}{\delta-2} < (\delta-1)^2 \\
\Leftrightarrow\ & \delta < (\delta-1)^2(\delta-2) \\
\Leftrightarrow\ & \delta^3 - 4\delta^2 + 4\delta - 2
\end{aligned}$$

which is true for $\delta > 2$.

A.2. Theorem 2

For all networks $I = (N, C)$, the expansion can be at most 1.

Proof. This proof is adapted from [34]. For every subset $U \in N$, let $E_U = \{(x, y) \in E | x \in U\}$ where an edge appears twice in E_U if $x, y \in U$. Clearly $(U, \bar{U}) \subseteq E_U$. Since $|U| = |E_U|$ it holds that $C(U, \bar{U}) \leq |U|$ and equivalently $C(\bar{U}, U) \leq |\bar{U}|$. Hence, $C(U, \bar{U}) \leq \min(|U|, |\bar{U}|)$ and therefore

$$\alpha(I) = \min_{U \in N} \frac{C(U, \bar{U})}{\min(|U|, |\bar{U}|)} \leq 1$$

A.3. Theorem 3

For a random graph $G = (n, p)$, the expected degree of a fixed vertex v is $p(n-1)$.

Proof. For a graph $G = (n, p)$ and a fixed vertex v, let X_i, an indicator random variable denote each neighbouring vertex such that

$$X_i = \begin{cases} 1 \text{ if } & (v, i) \in E \\ 0 \text{ if } & (v, i) \notin E \end{cases} \quad \text{for } i = 1, ..., n-1$$

The degree of v is then the sum of its neighbouring vertices

$$\begin{aligned} \text{degree}(v) &= X = X_1 + X_2 + ... + X_{n-1} \\ \mathbb{E}[\text{degree}(v)] &= \mathbb{E}[X] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + ... + \mathbb{E}[X_n] \\ &= (n-1)\mathbb{E}[X] \end{aligned}$$

And since

$$\mathbb{E}[X] = 1 \times p + 0 \times (1-p) = p$$

we have

$$\mathbb{E}[\text{degree}(v)] = p(n-1)$$

Appendix B.

Simulator Manual

B.1. Compilation and Running

The simulator is written in Java and can be built and run from source code with the included Makefile, by calling `make`. The simulator can then be run by setting the classpath variables (run `make classpath`) and executing

```
java sim.Main <config.cfg>
```

command `make run` which both sets-up the classpath and executes the program.

B.1.1. Dependencies

The simulator uses the JGraphT[1] library. The library jar must be included in the class path to build and run. The location of this can be specified in the Makefile by setting the variable `JGRAPHT_JAR`. Several command line programs are used to visualise data: Gnuplot[2] is used to generate plots of latency and throughput and GraphViz[3] is used to visualise structures such as network topology and channel dependency graphs.

B.2. Output

The simulator will initially output the configuration parameters. For each simulation run it will give summary statistics for each sample, and when the run completes, a summary of the run will be given. Figure B.1 shows some example output from a simulation run.

B.3. Configuration Parameters

The simulator program takes a single command line argument specifying a configuration file (`*.cfg`) used to specify the run time parameters of a simulation. The configuration is plain text and each line is of the format:

```
<variable> = <value>
```

a # character at the beginning of the line. The parameters `mode`, `topology`, `routing` and `traffic_pattern` are required, but all other parameters can be optionally specified and if absent set to default values.

Randomness is used in various parts of the simulator such as topology and traffic generation, in these cases the random number generator seed can be explicitly specified so that the user can control the randomness. With a particular seed value, the output will be deterministic, which is important to be able to control between experiments. All seed parameters (`*_seed`) can be set to `time`, so that the system time in milliseconds is used.

[1] http://jgrapht.sourceforge.net/
[2] http://www.gnuplot.info/
[3] http://www.graphviz.org/

```
Starting simulation run 1, injection rate = 0.02.

Warmed up after 1000 cycles
Sample      Generated   Received    Flying      Latency     Throughput
99          6406        6400        6           83.31       0.02
Finished sampling, draining packets...
0, 6 left
Done in 124 cycles
[STATS]=========================================
Packets generated    6406
Packets received     6406
Overall latency      81.00640240688723
Overall hops         6.95
Overall accepted     0.021857
Overall min accepted 0.00
Latency std dev      13389.207279223454
Accepted std dev      9.747225191757491E-4
-------------------------------------------------
Starting simulation run 2, injection rate = 0.03.

Warmed up after 1000 cycles
Sample      Generated   Received    Flying      Latency     Throughput
84          8105        7828        277         2198.10     0.03
```

Figure B.1.: Example output for a simulation run.

B.3.1. Simulator Modes

The simulator can be run in two different modes, `debug` or `run`, set with the `mode` parameter.

Debug Mode

The debug mode launches the program with a GUI interface where each node is represented in a tab detailing the state of the input and output virtual channels of both the router and processor. Various status updates are written to separate consoles for the router and processor. The simulation can be run, paused and stopped, or stepped through to inspect the state each cycle. The following two parameters are used to bound the run time behaviour of the simulator when in debug mode:

`max_cycles` Sets the maximum number of cycles the simulation can run for, when reached execution is terminated.

`max_msgs` Sets the maximum number of messages that can be generated in a single simulation run.

Run Mode

Run mode initialises the simulator to perform experiments. Run mode uses the following extra parameters to specify the execution of the experiments.

`sim_runs` Sets the number of complete simulation runs to be collated into the final result.

`sample_period` Sets the size of a sample period in cycles.

`num_samples` Sets the number of samples to be taken, hence `sample_period` * `num_samples` is the number of executed cycles used for measurement.

`latency_thresh` Sets a threshold latency value in cycles. If the average latency in the simulation exceeds this value then the simulation terminates. If this value is set to 0 this it is disabled.

`warmup_cycles` Sets the number of cycles necessary for the simulator to reach a steady state. If this value is set to 0, then this is ignored and the simulator warms up when the percentage change in latency and throughput is less than the parameter `warmup_thresh`.

B.3.2. Topology

The `topology` parameter specifies the network topology and can take the following values. The simulator supports two regular networks and three random irregular constructions. A level of faults can also be specified for the regular topologies.

`mesh` A k-ary n-mesh, where k and n are specified by parameters `k` and `n` respectively.

`tori` A k-ary n-cube (torus), where k and n are specified by parameters `k` and `n` respectively.

`erdosrenyi`	An Erdős-Rényi random graph. The following extra construction parameters can be specified: `num_nodes` sets the number of nodes in the graph and `p` sets the probability of a link between two nodes. A `graph_seed` parameter can be set to control the randomness of construction between experiments.
`dregular`	A d-regular random expander graph. The following extra construction parameters can be set: `num_nodes` sets the number of nodes and `d` sets d, the graph degree. As with the Erdős-Rényi random graph, a `graph_seed` parameter can be set.
`preferential`	A preferential attachment graph. The construction parameters `m` sets the initial number of nodes and `steps` sets the number of extra nodes added to the graph, hence the total number of nodes will be `m` + `steps`. Again the `graph_seed` can be set to control randomness.

B.3.3. Routing

The `routing` parameter specifies the routing algorithm to be used in the network and can take the following values. For each algorithm, some number of virtual channels may be required to provide freedom from deadlock.

`dor`	Dimension order routing, can only be used with mesh or torus topologies. For meshes, only one virtual channel is necessary for deadlock freedom. For tori, two virtual channels are necessary.
`updown`	Up*/down* routing, compatible with any topology. The spanning tree root node is randomly selected, or can be selected by setting `root_node` with a node identifier.
`segment`	Segment-based routing, compatible with any topology. The `seg_seed` parameter can be set as a seed for the random number generator for deterministic calculation of segments.
`lashtor`	LASH-TOR routing, compatible with any topology. The number of virtual channels is dependent on the calculation of paths in the network. If the number of virtual channels (`num_vcs`) is less than the number required by LASH-TOR then the simulation will fail.
`minimal`	Minimal path routing, used for debugging purposes, will deadlock.

B.3.4. Network

The network parameters specify 'physical' parameters of the network components.

`buffer_size`	Sets the size of the buffers in flits on input virtual channels.
`num_vcs`	Sets the number of virtual channels to use for each input channel.
`link_delay`	Sets the number of cycles taken for flits and credits to be transmitted along a link.
`rand_seed`	Sets the random number generator seed for the random elements of network execution such as traffic generation.

B.3.5. Traffic

The spatial distribution of traffic over the network is governed by a traffic pattern, set using the `traffic_pattern` parameter and can take the following values:

`uniform`	Each source sends a uniform amount of traffic to each other node. Destination nodes are selected for each packet uniformly at random.
`bitcomp`	Bit complement. $d_i = \neg s_i$
`bitrev`	Bit reverse. $d_i = s_{b-i-1}$
`transpose`	$d_i = s_{i+b/2 \mod b}$
`shuffle`	$d_i = s_{i-1 \mod b}$
`tornado`	$d_i = s_x + \lceil k/2 \rceil - 1 \mod k$
`neighbour`	$d_x = s_x + 1 \mod k$
`randperm`	Random permutation. A fixed permutation of traffic is chosen uniformly at random from the set of all permutations. The parameter `perm_seed` can be used to control randomness.

The injection process determines the temporal distribution of traffic in the network and is set with the `injection_process` parameter and can take the following values:

`bernoulli`	Bernoulli injection process, the injection rate r parameter `injection_rate` must be set such that $0 < r \leq 1$.
`onoff`	Modulated Markov Bernoulli process with two states 'on' and 'off'. The probabilities of transitions between on and off α and β respectively, can be set by `burst_alpha` and `burst_beta` such that $0 < \alpha, \beta \leq 1$.

Finally, the `flits_per_packet` parameter can be set to specify a constant number of flits per packet.

B.4. Extensibility

The simulator has been designed and written in an object-orientated style as the components of a network can intuitively be thought of as objects, for example `Node` and `Link` objects constitute a `Network`. The simulator has been designed to be an extensible platform that is non-specific to topologies or routing algorithms, consequently the addition of new topologies or routing algorithms is straight forward.

B.4.1. Topologies

A `Network` is a set of interconnected `Nodes`. The static method `Topology.createTopology()` is responsible for creating the set of `Nodes`. This can be done in two ways, the first is to create a `Construction` graph which allows you to specify the topology by adding edges between nodes to it. On completion, the `Construction` has a method `create()` to create the set of `Nodes` for the network. Figure B.4.1 gives a example code-snippet of how to randomly add edges

```
# Simulation mode ============================
mode               = run

# Topology parameters ========================
topology           = mesh
k                  = 4
n                  = 2

# Routing Parameters =========================
routing            = segment
seg_seed           = 1


# Network parameters =========================
buffer_size        = 4
num_vcs            = 4
link_delay         = 1
rand_seed          = time

# Traffic parameters =========================
traffic_pattern    = bitrev
flits_per_packet   = 20
injection_process  = mmp
burst_alpha        = 0.1
burst_beta         = 0.9

# Run mode simulation Parameters =============
sim_runs           = 10
sample_period      = 1000
num_samples        = 100
latency_thresh     = 1000
warmup_thresh      = 0.05
```

Figure B.2.: Example simulator configuration file

```
Construction graph = new Construction(numNodes);

for(int i=0; i<numNodes; i++) {
    for(int j=0; j<numNodes; j++) {
        if(rand.nextDouble < 0.5)
            graph.addEdge(i, j);
    }
}

Node[] nodes = graph.buildTopology();
```

Figure B.3.: Example code for a random topology construction using a `Construction` graph. Input and output `Link` ordering and corresponding port numbers are generated by the `Construction`.

```
Node[] nodes = new Node[numNodes];

for(int i=0; i<nodes.length; i++) {
    nodes[i] = new Node(i, 2, 2);
}

for(int i=0; i<numNodes; i++) {
    nodes[i].connectTo(nodes[(i+1) % numNodes], 1, 2);
    nodes[i].connectTo(nodes[(i-1) % numNodes], 2, 1);
}
```

Figure B.4.: Example code for explicit topology construction. Note that with this method node in and out degree along with port numbers for connections have to be specified.

between nodes to a `Construction` and return the set of nodes. This is the way the random graphs are generated.

The second, slightly more complicated way, is to explicitly construct the `Nodes`. This way is used for `Mesh` and `Torus` constructions as greater control over the port constructions is necessary. To do this, each `Node` must be constructed, then the `connectTo()` method can be used to specify directional link connections. After all links have been connected the `finishedConnecting()` must be called for each `Node`. Figure B.4.1 gives a code snippet for how this type of construction could be used to build a ring.

B.4.2. Routing Algorithms

New routing algorithms can also be simply added to the simulator by implementing the `RoutingFunction` interface. This specifies two methods to return an output port and an output virtual channel based on the current node, the input virtual channel and the destination node. Most of the algorithms implemented in the simulator are based on tables, where lookups are performed to obtain output ports and virtual channels. This requires a static configuration phase, and is implemented with a static method in the routing function class called in the `Network` object's constructor, passing in

the `Network` object as an argument.