MSc THESIS

Congestion Management with feedback queue.

Alexandre Abi Khaled
Congestion Management with feedback queue.

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Alexandre Abi Khaled
born in Beirut, Lebanon

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Congestion Management with feedback queue.

by Alexandre Abi Khaled

Abstract

The phenomenon of HOL (Head of Line) blocking occurs in communication where the data packets are under some conditions stopped waiting for the appropriate time for routing. This will cause communication delay in the network. Considerable research has been initiated by the demand for low-cost network topologies with no internal blocking (non-blocking topologies), capable to switch in parallel any possible combination of one-to-one, input and output connections. In order to be able to dynamically share the resources of the network, packet switched networks depend on statistical multiplexing. When several packets demand access to the same link, some of them maybe delayed waiting for other packets to move first. The short term or longer term delay causes degradation in the throughput of a certain network. When a link, route or destination is overloaded, while the demand on it is still increasing, congestion appears and latency increases. This will simply increase the demand of the link, route or destination beyond its capacity and show degradation in throughput and performance. This thesis investigates the handling of a classical and hard problem congestion management, reaching the suggestion of an alternative solution. In this work, a solution for congestion caused by head of line HOL blocking is proposed. An extensive taxonomy on available solutions is offered to the reader. We are able to present a solution with a feedback queue added to the input queue on a given switch in the network. A series of multiplexers control the routing of packets within the input, output, the main and feedback queue. We present an algorithm that controls the function of the multiplexers in order to eliminate the Head of Line blocking. The proposed algorithm proves to be a promising method as with the low cost of one additional queue, results close to the best case scenario have been achieved, performance improved and packet latency lowered to a margin close to the best case scenario.

Laboratory: Computer Engineering
Codenumber: CE-MS-2010-01

Committee Members:

Advisor: Dr. Ioannis Sourdis
Advisor: Dr. Ir. Georgi Gaydadjev
Chairperson: Dr. Koen Bertels, CE, TU Delft
Member: Dr. Ir. Wouter Serdijn
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Figures</strong></td>
<td>vi</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>vii</td>
</tr>
<tr>
<td><strong>Acknowledgements</strong></td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem Statement</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Motivation of the thesis</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 The phenomenon of congestion</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Lossless vs. Lossy networks</td>
<td>3</td>
</tr>
<tr>
<td>1.2.3 HOL blocking (Head of line): a definition.</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Thesis Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Outline of the thesis</td>
<td>5</td>
</tr>
<tr>
<td><strong>2 The SARC architecture and network interfaces</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Interprocessor Communication</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Progressive address translation</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Congestion Management</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Conclusion</td>
<td>10</td>
</tr>
<tr>
<td><strong>3 Congestion management taxonomy and related work</strong></td>
<td>11</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Overview of Interconnection Networks</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Complex dynamics of congestion trees</td>
<td>13</td>
</tr>
<tr>
<td>3.4 Approaches to congestion management</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1 Approach 1: Avoidance based</td>
<td>18</td>
</tr>
<tr>
<td>3.4.2 Approach 2: Prevention based</td>
<td>19</td>
</tr>
<tr>
<td>3.4.3 Approach 3: Detection based</td>
<td>20</td>
</tr>
<tr>
<td>3.5 Congestion Elimination</td>
<td>22</td>
</tr>
<tr>
<td>3.5.1 Avoidance based techniques</td>
<td>22</td>
</tr>
<tr>
<td>3.5.2 Prevention based techniques</td>
<td>25</td>
</tr>
<tr>
<td>3.5.3 Reactive techniques</td>
<td>29</td>
</tr>
<tr>
<td>3.6 HOL blocking Elimination Strategies</td>
<td>32</td>
</tr>
<tr>
<td>3.6.1 HOL blocking elimination at Network Level</td>
<td>32</td>
</tr>
<tr>
<td>3.6.2 HOL blocking elimination at Switch Level</td>
<td>35</td>
</tr>
<tr>
<td>3.7 Conclusion</td>
<td>39</td>
</tr>
</tbody>
</table>
4 Congestion handling with feedback queue

4.1 Introduction .................................................. 41
4.2 Algorithm and pseudo code of the solution .................. 41
  4.2.1 Description of the RCT table ............................. 42
  4.2.2 Algorithm and multiplexers control ..................... 43
4.3 The simulator ................................................. 51
  4.3.1 Sicosys Simulator ...................................... 51
  4.3.2 SIM Simulator .......................................... 52
4.4 Simulation results ........................................... 54
  4.4.1 Explanation of the scenarios ............................ 54
  4.4.2 The worse case scenarios ............................... 57
  4.4.3 The best case scenarios ................................ 57
  4.4.4 Results of the proposed solution ....................... 58

5 Conclusions and future work ................................ 65
  5.1 Summary .................................................... 65
  5.2 Thesis Contributions ........................................ 66

Bibliography ....................................................... 67

6 Appendix A: The configuration files for the used tests .... 71
List of Figures

1.1 congestion in the lowest queue. [10] ........................................... 1
1.3 Input queued crossbar. [11] ....................................................... 4

2.1 Traditional systems: local communication only via cache coherence. [10] 8
2.2 Preferred architecture: merge the cache controller and the network interface. [10] .......................................................... 9

3.1 Network Types. [19] ................................................................. 12
3.2 Congestion tree. [1] .................................................................. 14
3.3 (a) Queue status at the detection moment (b) after detection. [1] ...... 15
3.4 Basic RECN Treatments. [1] ....................................................... 16
3.5 Congestion tree detection with different RECN versions. [1] ......... 17
3.6 Methods for storing elements: (a) row-major order; (b) column-major order; and (c) block-column data allocation. [9] .................. 18
3.7 A software combining tree. [9] ................................................. 19
3.8 The ultracomputer switch incorporating combining capability. [9] ... 20
3.9 Taxonomy for congestion control algorithms. [27] ....................... 21
3.10 Lower bounds on network bandwidth for various hot-spot rate. [28] 23
3.11 Average network delay versus bandwidth for a network of size 256. [28] 24
3.12 Estimated bandwidth improvement. [20] ...................................... 27
3.13 A shared memory multiprocessing system with feedback. [20] ...... 27
3.14 Percentage of useful free output virtual channels versus traffic for different message destination distributions. [3] ............................... 31
3.15 Operation of the congestion control mechanism INC. [4] .............. 32
3.17 ATLAS I chip overview [13] .................................................... 34
3.18 QFC-like ATM credit protocol used in simulations [13] ............... 34
3.20 (A) Conventional nodes organize their buffers into FIFO queues restricting routing. (B) A network using virtual-channel flow control organizes its buffers into several independent lanes [7] ..................... 36
3.21 Packets Congestion [15] .......................................................... 38

4.1 solution of the SARC [10] ......................................................... 41
4.2 Solution .................................................................................. 42
4.3 scenario 1 . case A in the algorithm ........................................... 47
4.4 scenario 2 . case B in the algorithm ........................................... 48
4.5 scenario 3 . case C in the algorithm ........................................... 48
4.6 scenario 4 . case D in the algorithm ........................................... 49
4.7 scenario 5 . case E in the algorithm ........................................... 49
4.8 scenario 6. case F in the algorithm ........................................ 50
4.9 scenario 7. case G in the algorithm ........................................ 50
4.10 scenario 8. case H in the algorithm ....................................... 51
4.11 scenario of N deques and could correspond to any of the cases A to H in
the algorithm controlling the MUX/DEMUX operation ..................... 51
4.14 Algorithm Flow chart ......................................................... 54
4.15 Head Of Line ........................................................................ 55
4.16 HOL node ............................................................................ 56
4.17 worse case scenario queue design ............................................ 57
4.18 best case scenario queue design .............................................. 57
4.19 4*4 2D-mesh comparison, worse case and solution are measured with 2
hotspots ................................................................................. 59
4.20 8*8 2D-mesh comparison, worse case and solution are measured with 2
hotspots ................................................................................. 60
4.21 16*16 2D-mesh comparison, worse case and solution are measured with 2
hotspots ................................................................................. 60
4.22 32*32 2D-mesh comparison, worse case and solution are measured with 2
hotspots ................................................................................. 61
4.23 comparison 4*4 2D-mesh with random traffic ............................. 61
4.24 comparison 8*8 2D-mesh with random traffic ............................. 62
4.25 comparison 16*16 2D-mesh with random traffic .......................... 62
4.26 comparison 32*32 2D-mesh with random traffic .......................... 63
6.1 scenario 1 .............................................................................. 71
6.2 scenario 2 .............................................................................. 72
6.3 scenario 3 .............................................................................. 72
6.4 scenario 4 .............................................................................. 73
6.5 scenario 5 .............................................................................. 74
6.6 scenario 6 .............................................................................. 75
List of Tables

4.1 Example of the RCT table in the proposed solution . . . . . . . . . . . . 43
4.2 Table of multiplexers control possibilities . . . . . . . . . . . . . 44
4.3 Pseudo code of the solution algorithm . . . . . . . . . . . . . . . . . . 45
I present this work to all those who have encouraged me to pursue a Master in Computer Engineering. Special thanks to those who have helped and assisted me brainstorming on this topic especially my dear friend Frederico Pratas MSc. Thanks to my advisor Dr. Ioannis Sourdis for his support and contribution in this Thesis work, additionally to Dr. Georgi Gaydadjev for his support. Last but not least, I thank my family and my brother Ir. Andre Abi Khaled MSc and my partner Arancha Garcia Pesquera for her patience. Finally, this Master and the full study would not have been possible without the support of my managers at IBM respectively: Ir. Rien Schot, Otto Klaren and Geert-Jan de Koning, without forgetting the investment that IBM has set in order to assist me financially on finishing this program.

Alexandre Abi Khaled  
Delft, The Netherlands  
November 13, 2010
Introduction

Congestion in interconnection networks forms when different packets request the same resource. In lossless networks which do allow dropping packets, congestion rapidly propagates to other switches. The phenomenon of packets trapped behind a waiting one is known as Head of line (HOL) blocking. HOL blocking causes network throughput to degrade and as a result, packet latency can increase dramatically. Furthermore in this chapter, we present the problem statement and a detailed explanation of the HOL in addition to the objective of the thesis.

1.1 Problem Statement

Scalable computer architectures SARC are evolving from a traditional system where the off-chip NI communication is tight to remote networks. The new alternative is to have the network interface contain hardware paths and functions for performing data transports over the off-chip network. The L1 NIs are unable to differentiate between the L2 queues. This is known to cause the HOL, Head of Line blocking, where packets ready to go, get caught behind stopped ones in the L1 NI causing an unneeded delay. Figure 1.1 illustrates the congestion in the lowest queue on the L1 NI number 3. As we notice the green packet is trapped behind the red ones.

![Figure 1.1: congestion in the lowest queue.][10]

Multi-level NIs also help reduce the number of distinct queues that are needed for flow or congestion control purposes. Conversely, when multiple network destinations appear
as a single route entry in the RCT (Routing and Congestion Control Table), it makes no sense to have more than one corresponding queue for flow/congestion control purposes. Network interfaces, like network switches, use separate per-destination outgoing queues (virtual output queues. Congestion control stops the head packet of a queue because it is destined to a congested area, packets behind it in the same queue also get stopped; if these latter packets are in fact destined to non-congested area, performance is needlessly degraded. To avoid such negative effects, separate, per-destination queues are used; Figure 1.1 shows 4 such queues in the L1 NI, and 4 others in the L2 NI. On one hand, we want to economize on the number of such queues, especially in the L1 NI where resources are limited. On the other hand, at the L1 NI level, the number of visibly distinct destinations does not exceed the number of entries in the L1 RCT: since the RCT specifies network routing at least up to the next NI any two destination addresses (in global virtual address space) that belong to the same address region in the RCT have the same network routing; hence it makes no sense to map these into different queues, because the L1 NI has no criterion based on which it could decide which queue to use.

As a result, the L1 NI uses a single queue for all packets destined to the address region 1xxx; then, within the L2 NI, these packets get distributed into 4 different queues (notice that these L2 queues can be advantageously shared among packets arriving from all L1 NIs). In the figure, destination 7 (red) is assumed to be congested, while destination 5 (green) is lightly loaded. The L2 NI uses separate corresponding (red and green) queues, so that green packets (destined to 101x) can move fast, bypassing the slow red packets. Notice that we want the red queue the one dedicated to 111x to be quite long: if that queue filled up, we would need to stop L1 NIs from further filling it. However, L1 NIs are incapable of differentiating among the 4 L2 queues: if instructed to stop traffic towards any of them, their only choice is to stop all traffic towards the L2 NI. This, however, would cause HOL blocking effects: green packets trapped behind red ones in the L1 queues get needlessly delayed. How to best handle such situations is the topic that this thesis is approaching.

1.2 Motivation of the thesis

The scope of the thesis will be an investigation into handling an old and difficult problem congestion management, reaching the suggestion of an alternative idea. The on chip communication is defined by level-1 interface. We define the off-chip Network interface as being a level-2 interface. In order to increase the on chip RTT (Round Trip Time), the level-1 interfaces are tightly coupled to the processor, while the level-2 off chip interface is responsible for off-chip communication which has a longer RTT (Round Trip Time). The departure of packets is delayed in the case of congestion.

1.2.1 The phenomenon of congestion

Considerable research has been initiated by the demand for low-cost network topologies with no internal blocking (non-blocking topologies), capable to switch in parallel any possible combination of one-to-one, input and output connections. In present, the packet switching has supervened upon circuit switching, the blocking problem, which in its turn
1.2. MOTIVATION OF THE THESIS

leads to congestion problems, has assumed new flavors and continues to constitute a hot area of research. In order to be able to dynamically share the resources of the network, packet-switched networks depend on statistical multiplexing which introduces a conflict between packets. When several packets demand access to the same link, some of them may be delayed waiting for other packets to move first. The short term or longer term delay caused degrades the throughput of a certain network, since packets are delayed behind a congested area or behind packets that are waiting to have a free route. When a link, route or destination is overloaded, while the demand on it is still increasing, a congested flow appears. The phenomenon is seen with the impact that the congested flow has on the non-congested one. This will simply increase the demand of the link, route or destination beyond its capacity and show degradation in throughput and performance.

![Figure 1.2: Congestion caused by hotspot A.][1]

According to Figure 1.2 we notice that the sources 1 through 4 are injecting packets for which the destination is the same hotspot A. This leads to the formation of a congestion tree. Some other packets that are generating from source 4, simply need to cross the congestion tree area towards other destinations. Let’s assume a packet needs to travel from source 4 towards destination B, then, this packet will suffer from the congestion and will possibly be delayed as if it was suffering from congestion itself. We describe this to be a poor service due to the congested area.

1.2.2 Lossless vs. Lossy networks

We define the lossless network to be strict on dropping packets, while a lossy network can afford the loss of packets. Throughout this thesis, the related work, and the implementation, a lossless network model is adapted. Thus; dropping packets is not an option. Whether we take a cluster disk network, an interconnected network, a data center or an inter-processors communication network, they all share the common feature of being lossless networks. It is more complex and of a high level of difficulty to handle congestion in lossless networks compared to handling it in lossy networks. We cannot drop packets
when we discover that they are congested. Instead, the congested packets need to be held for example in inside network buffers or network interface buffers. It is evident now that buffers may and will overflow from congested packets causing more congestion along the network. Preventing the congestion and the overflow of buffers is referred to as congestion management or congestion control.

1.2.3 HOL blocking (Head of line): a definition.

To drive our discussion into a more concrete stage, lets consider the following simple input queued crossbar switch. It uses a single first in first out (FIFO) queue at the input which buffers packets that are not able to be directly routed towards their output destination.

\[ \text{Figure 1.3: Input queued crossbar.}[11] \]

According to Figure 1.3, the packets enter the one of the queues regardless of the output destination. The red packets are destined to output 1 which is receiving a packet from input 2. This means that the packet on input 1 will have to wait until it has clearance to move towards output 1. To make it simpler, red packets have their destination busy, thus; they halt. The blue packet behind the red one on input 1, needs to move towards the blue output 2 but it cannot even though the route towards output 2 is not congested. This is a clear and simple example of how congestion on a given area can cause congestion on other parts of the network as described earlier in 3.1. Lets focus again on input 1, the fact that the red packet is blocking the blue packet from moving, is defining the head of line blocking HOL. The behavior is not different in general lossless multistage networks. As mentioned previously in 3.2, any type of lossless network can suffer from HOL blocking. Congested packets trapped in the buffer, spread congestion out and trim down the throughput. The absence of an efficient control that deals with the HOL blocking problem implies poor performance and possibly higher power consumption.

1.3 Thesis Objectives

The main objective of this work is to propose:
• Present a solution that eliminates blocking in order to manage congestion.
• Run measurements on the worse case scenario: lowest performance and lowest cost.
• Run measurements on the best case scenario: highest performance and highest cost.
• Run measurements on our solution: close to the high performance achieved with the best case scenario, and near the low cost of the worse case scenario.
• Compare all the measurement results and define the contributions of this thesis work.

1.4 Outline of the thesis

After stating the problem and a brief introduction, the layout of the thesis should take the reader through an overview on the SARC architecture as an introductory explanation to the NI (network interface) in general and the congestion problem, this will be presented in chapter 2. Chapter 3 offers the taxonomy of congestion management techniques. Chapter 4 presents the idea of the solution and describes the implementation, the simulation and the test results. Chapter 5 represents a summary of the work and the contributions of the thesis. An appendix is added to describe the function of the simulator and serve as a user manual.
2.1 Introduction

This chapter offers an overview of the progressive address translation and the Network interface (NI). The overview of these two components is needed in order to understand the factors behind the congestion problem that is being treated in this thesis. Once the components have been described, we can move towards the root causes and the description of the congestion problem in the SARC architecture. If the reader desires to go into the details of the SARC architecture then he should refer to [10]. Section 2.2 discusses about the Interprocessor Communication (IPC) that consists of moving information among producers, consumers, and storage and informing threads of computation about the time when information or resources become available. Section 2.3 illustrates the principle of the progressive address translation, the packet does not carry the address of the final destination. Section 2.4 talks about the congestion management.

2.2 Interprocessor Communication

Interprocessor Communication (IPC) consists of moving information among producers, consumers, and storage and informing threads of computation about the time when information or resources become available. Data and notification (control) transport occurs through the interconnection network, and network interfaces are where IPC mechanisms are implemented. Data locality is of capital importance for the performance of scalable systems, so the judicious choice of time and place of data transport becomes the main concern of IPC.

There are two styles of data placement and IPC data transport:

1. Implicit Data Placement and Communication: data are located and then moved not under the direct control of the application software (or its libraries or runtime system); instead, data copies are placed and moved as a result of cache misses or cache coherence events, which are indirect only results of application software actions. The benefit is simplicity: the application (or runtime, or compiler) programmer need not worry about where data should reside and how and when they should be moved just use anything, anytime you need it! The disadvantage is inability to optimize for the specific data transport patterns that occur in specific applications; such optimization becomes especially important in scalable systems, because the mechanisms of cache coherence being ignorant of what the application is trying to do fail to deliver acceptable performance in large-scale multiprocessors. Figure 2.1 explains the local communication only via cache coherence [10].
Figure 2.1: Traditional systems: local communication only via cache coherence.[10]

2. Explicit Data Placement and Communication: data is allocated and moved under software control; this can be application software introduced either by the programmer or by the compiler or library / runtime system software. The burden is how to best control data placement and transport. The benefit is scalable performance. Thus, explicit communication mechanisms become a necessity for large-scale multiprocessors to deliver their promise: the hardware must provide explicit data transport primitives, and the software has to little by little figure out ways to beneficially use them [10]. The SARC architecture aspires to reconcile these two worlds, and to hopefully merge the hardware primitives that they both use.

Figure 2.2 illustrates a potentially better architecture. Both a cache controller and a network interface contain hardware paths and functions for performing data transports over the network; it is estimated that, by merging the two of them together, we can economize on circuits by sharing these common parts. Additionally, the NI functions of detecting events and responding to them can potentially be used for implementing software-controlled cache protocols, while still handling the most time consuming parts of such protocols in hardware. This may prove to be the best way, in terms of performance-verification/complexity tradeoff, to implement directory-based coherence protocols; additionally, this allows the runtime system to introduce monitoring and adaptivity characteristics into these protocols [10].

### 2.3 Progressive address translation

The principle of the progressive address translation is that the packet will not carry the address of the final destination. The packet carries the virtual address and this is being translated by the physical address in the tables. The tables job is to provide the physical route to the packet. I refer to those tables as translation tables. In the domain
2.3. PROGRESSIVE ADDRESS TRANSLATION

Figure 2.2: Preferred architecture: merge the cache controller and the network interface.[10]

of network routing, the translation table is known to be the routing table. Figure 3.2 depicts the progressive address routing [10]. Every translation table gives the packet the next (or following) destination directing it through 1 or few hops. The translation table contains the exact information of its adjacent hops or the devices that are close to it [10].

Figure 2.3: Progressive address translation / routing.[10]

Each packet (imagine that it starts from processor P) may pass through several translation tables on its way to its destination. The packet always carries its virtual destination address. Each translation (routing) table directs the packet for a few more steps (through a few more switches the round devices in the figure) towards its final destination. Each table contains accurate (definitive) location (routing) information for the data items (e.g. pages) that are close to it, but only approximate information for those that are further away: go to that table and ask there, it says.

Figure 2.3 also illustrates a couple of practical aspects of progressive translation. The function of the tables can be interpreted to be to provide run-time configurable network
routing, rather than fixed geographic routing: each destination address is allowed to change its location in the net. However, it would be too expensive, in general, to provide this capability inside each network switch especially so inside the switches of networks-on-chip (NoC). In a large multiprocessor system, with the many millions of objects in its global address space, translation/routing tables would be hopelessly large, if we could not aggregate together entries referring to objects that are nearby in both logical and physical space. Internet routing tables handle such situations using longest-prefix matchings: when the table contains both an entry for 1xxx and 100x, addresses 1000 and 1001 are translated using the latter (longest) entry that they match, 100x, while addresses 1010, 1011, 1100, ..., 1111 are translated using the former entry [10].

2.4 Congestion Management

The network that moves data from node to node realizes the inter-processor communication and has a decisive influence to the overall performance of multiprocessor systems. Assuming that the NI primitives minimize the latency of message initiation and completion at source and end node parties, respectively, amounting to very fast messaging under zero or small network load. However, as the volume of exchanged data increases, the contention among data for shared resources (e.g. network channels or end node processing units) underpins communication latency; when this contention becomes unbearable, Congestion can appear. In congestion (or hotspot) situations, data (and thus processing) may experience unacceptably long delays. The cost-performance merits of a congestion control scheme depend on the topology, the utilization, and the efficacy of the underlying network, while both the NIs and the network itself can offer mechanisms to manage congestion [10].

2.5 Conclusion

This chapter concludes that the Interprocessor Communication (IPC) consists of moving information among producers, consumers, and storage and informing threads of computation about the time when information or resources become available. The two styles of data placement and IPC data transport are the Implicit Data Placement and Communication and the Explicit Data Placement and Communication. The benefit of the Implicit Data Placement and Communication is that the application programmer need not worry about where data should reside and how and when they should be moved just use anything, anytime you need it. The downfall is in ability to optimize for the specific data transport patterns that occur in specific applications. The benefit of the Explicit Data Placement and Communication is scalable performance. This means that the hardware must provide explicit data transport primitives, and the software has to little by little figure out ways to beneficially use them. The principle of the progressive address translation is that the packet will not carry the address of the final destination. The packet carries the virtual address and this is being translated by the physical address in the tables.
3 Congestion management
taxonomy and related work

3.1 Introduction

This chapter offers an overview about the different techniques of congestion management. Section 3.2 discusses the general overview of the Interconnection Network. Section 3.3 depicts the dynamic complexity of congestion trees where the HOL (Head of Line) blocking is one of the main concerns that take place in networks based on switches with queues at their input ports. Section 3.4 discusses the different approaches to congestion management. The avoidance-based strategies and prevention based strategies are discussed. Section 3.5 talks about the techniques of congestion elimination, e.g. the avoidance based techniques, prevention based techniques, and the reactive techniques. Section 3.6 illustrates the HOL (Head Of Line) blocking elimination strategies, e.g. the HOL blocking elimination at Network Level and the HOL blocking elimination at Switch Level.

3.2 Overview of Interconnection Networks

The case of connecting two nodes together is straightforward. One link is only required to transfer packets between them. If a higher number of nodes is needed, the number of links required to totally connect the nodes grows geometrically as the following equation illustrates [19]:

\[ l(N) = \frac{(N-1) \cdot N}{2} \]

Where \( L \) is the number of links and \( N \) is the number of nodes. This equation shows that if an extra node is added to the network then the network complexity increases. Therefore, it is crucial to decrease the network complexity. A router discovers the paths of the packets through the network. Parts of the information can be conveyed through the network between two nodes not necessarily directly connected. The router can read the destination of the packets and send them to the nearest node to the destination if necessary. The switched networks are type of networks, which the links are shared; this means that each link is not only carrying data between the nodes at its ends, but also the data forwarded by these coming from nodes further away. Men can have different network types or topologies depending on which subset of links from the total interconnection network is chosen. The network is regular if the connection scheme does respond to a certain rule otherwise it is irregular. The network is said to be asymmetric, in the case of regular networks, if the view of the network from every node is not the same, otherwise the network is symmetric. The networks are said to be n-dimensional depending on the number of neighbors a node can have. For example, a network where each router has two neighbors so it is 1-dimensional. The following figure depicts an example of different network topologies discussed above [19].

In order to solve a big issue in network on chip, the set of computing elements that cooperates together is called parallel computer or parallel execution of the pack-
ets. These elements are connected together with an interconnection network as those explained above so that these nodes can cooperate together. However, to structure the communication between the different processing elements, there must be a higher level mechanism. Parallel computers are distinguished in two categories. Firstly, the Message Passing Computers are the machines which are known as multicomputers. The structure of the communication between the processing nodes is based on send and receive. A famous example of that is the Parallel Virtual Machine. On the other hand, the Shared Memory Computers which are also known as multiprocessors have one huge memory shared by all the processors of the machine [19].

Routing is an algorithm that allows information to reach its destination. There is a router unit on every node that enables it to forward packets of data to other nodes in the network. This means that the router just decides to which of its neighbors it will forward the packets to. The router does not decide itself the whole trajectory of the packets of information in the network. This is done by the routing algorithm which is distributed among all the routers. The routing algorithm contains the knowledge of the network topology and the source and destination nodes of the information. Men distinguish between deterministic and adaptive routing algorithm. In the deterministic routing algorithm the information traveling between two given nodes gets assigned always the same path, depending on which route is assigned to a pair of source and destination nodes. In the adaptive routing algorithm, the information finds its way through the network depending on traffic conditions [19].

The number of equally sized packets is divided into arbitrary sized messages; each of these packets is composed of a sequence of flits. A flit is composed by the amount of information the physical infrastructure handles at a time, for example the width of
3.3. COMPLEX DYNAMICS OF CONGESTION TREES

the link. The flits of a packet must be received properly ordered and not mixed with those form other packets while the packets in a message can arrive in any order. Store-and-forward is one of the flow control functions, in which a router waits until it receives the end of the packet before sending it to the next router. On the other side we have worm-hole flow control. With this function, a router does not wait for the end of the packet to send it over to the next router. The benefit of this approach is that the time a packet needs to get to its destination is only proportional to the number of hops and the buffering space needed is much smaller. However there is a drawback, when the head of the packet is blocked, the packet body stops advancing and blocking some channels from other routers [19].

3.3 Complex dynamics of congestion trees

Nowadays, the main concern of the designers of on-chip networks and large parallel computers is how to reduce the cost and power consumption of the interconnection network. More attention is spent on how to reduce the power consumption. For this reason, researchers have proposed techniques using frequency/voltage scaling to reduce the system power consumption. Due to the slow response in the presence of traffic variations and the suboptimal frequency/voltage settings during transitions of using frequency/voltage scaling methods. Therefore these techniques are quite inefficient. Traditionally, the interconnection network has been expanded. Reducing the number of components in a network like switches and links and increasing their utilization is considered to be a simple way to reduce cost and power consumption. On the other hand, this method will lead to increase in network contention. Usually, as the traffic is bursty, this solution may lead to congestion solutions. Generally speaking, due to flow control, congestion spreads quickly through the network, forming congestion trees. The worst case scenario of the congestion effect is that the packets that are blocked due to congestion prevent the advance of other packets stored in the same queue and dramatic network speed degradation takes place. This is the so-called Head of Line (HOL) effect [1].

The HOL (Head of Line) blocking is one of the main concerns that take place in networks based on switches with queues at their input ports. The HOL effect is due to the fact that blocked packets destined to congested output switch ports prevent the advance of packets destined to other non congested output ports. The switch architecture in on-chip networks is affected by the congestion in the network. In modern switch designs, men distinguish between two types of switch architecture and they are the Input Queuing switches (IQ switches), combined Input and Output Queuing (CIOQ switches) and Output Queuing (OQ switches). S packets can be removed with a switch with a speedup of S and submitted to S packets to each output within t seconds, where t is a time slot. The time slot is the arrival time between packets at the input ports. The HOL blocking can be diminished by augmenting switch speedup. The speedup of the OQ switches is N while the speedup of IQ switches is 1. CIOQ switches have speedup values between 1 and N. Therefore, the HOL blocking should be controlled by a suitable congestion management technique and the switch speedup increase is limited [1].

According to [1], congestion is very difficult to be solved due to the quick growth of congestion trees. Recently, congestion is a problem because networks are over di-
mensioned. Generally speaking, the congestion trees always grow from the root to the leaves. This is not true for networks with CIOQ switches but it is the case with OQ switches. A deep analysis of congestion trees and their dynamic evolution is presented in [1]. It helps in finding solutions for congestion management in interconnection networks with CIOQ switches. A detailed analysis of the dynamics of congestion trees when using CIOQ switches is presented in [1], showing their complex evolution. The proposed enhanced congestion management method is described in [1] where the complex evolution of congested trees takes place.

Congestion spreads from the root to the leaves, due to the use of flow control. It has been thought that congestion trees exist from the fact that the root of the congestion tree becomes congested. Only in a particular scenario that rarely happens, where the different traffic flows that form the tree join only at the root occurs. When flows meet at the root switch, the rate at which packets arrive is higher than the rate at which packets can be transmitted through the output port. Thus, queues begin to fill at the root switch and thus, congestion starts, due to the available bandwidth at the output port of the root switch is divided among the different flows and is lower than the bandwidth required by each flow [1]. Figure 3.2 illustrates the traditional view of the formation of the congestion tree.

![Figure 3.2: Congestion tree][1]

The main concern in [1] is to develop new techniques able to keep track of the complex dynamics of congestion trees in networks with CIOQ switches. One of the proposed congestion control mechanism is RECN (Regional Explicit Congestion Notification). RECN focuses on eliminating the HOL blocking induced by congestion trees. This is done by
3.3. COMPLEX DYNAMICS OF CONGESTION TREES

dynamically allocating separate buffers for each congestion tree and maximum throughput is still to be achieved. At every input and output port of a switch, a set of additional queues referred to as Set Aside Queues (SAQs) are used. SAQs are used to store packets passing through a congested point (root of a congestion tree). A memory is associated to each set of SAQs to store the packets. The memory line contains the control info required to identify a congested point and to manage the corresponding SAQ. So, the RECN keeps track (with a control bit on each memory line) of the network points that are leaves of a congestion tree [1].

To enhance RECN, congestion at switch ingress ports is first detected. RECN defines SAQs at ingress and egress ports, but it only detects congestion at egress ports. Congestion occurs at ingress ports. In these cases, RECN never detects congestion at the root. Instead, it detects congestion at the immediate upstream switches, but only when the root ingress queues are full. Therefore, RECN is not reacting as quickly as possible to eliminate HOL blocking at an important part of the tree. The congestion point is the output port by itself when the congestion at the egress side is detected. The reason why an ingress normal queue fills over a threshold is because packets requesting a certain output port are being blocked. It is not trivial to decide which one is the congested output port as packets in the ingress queue can head to different output ports. The congestion point is the output port by itself when the congestion at the egress side is detected. It is because packets requesting a certain output port are being blocked when an ingress normal queue over a threshold is filled. It is not trivial to decide which one is the congested output port as packets in the ingress queue are headed to different output ports. It is proposed to replace the normal queue at each ingress port by a set of small buffers (referred to as detection queues). So, at ingress ports, the memory is now shared by detection queues and SAQs [1]. Figure 3.3 describes the Mechanism for detecting and handling congestion at the ingress side.

![Figure 3.3: (a) Queue status at the detection moment (b) after detection.][1]

The second technique in enhancing the mechanism for detecting and handling con-
CHAPTER 3. CONGESTION MANAGEMENT TAXONOMY AND RELATED WORK

gestion is the actions taken upon reception of notifications. This is done in order to ensure that no out of order packet delivery is introduced. RECN does not allocate SAQs for all the notifications received [1]. Figure 3.4.a shows an example where RECN does not allocate a SAQ when receiving a notification.

Figures 3.4.b and 3.5.b depict how the enhanced RECN mechanisms detect congestion trees and how the original (or basic) RECN detect congestion trees. The simulation
results of a congestion tree that is formed by eight sources injecting packets to the same destination (hot-spot) at the full rate of the link are illustrated in Figures 4.a and 4.b. The Switch speedup is set to 1.5 and at the same time, all the sources start sending packets. The thick arrows indicate the paths followed by congestion notifications (RECN messages) from congestion points, thus indicating where SAQs are allocated for a particular congestion point. The points (ingress or egress) that are considered by the mechanism as congestion roots are indicated as dots in both figures [1].

Figure 3.5: Congestion tree detection with different RECN versions.[1]

3.4 Approaches to congestion management

All processors have access to a globally shared memory through an interconnection network in shared-memory multiprocessor systems. N processors and N memory modules are possessed in large, shared-memory systems that use a multistage interconnection network. The access time of each memory module is one access request at a time. Several processors request concurrent access to a critical data structure, they will lead to contention for the memory module where the data structure is located. Even if these requests are to different data items within the same memory module contention occurs. This will lead to the so-called phenomenon the hot-spot contention and such a memory module is called a hot memory module. Hot-spot contention leads to contention for interconnection network buffers and links. In particular, memory contention leads to what is known as tree saturation. Tree saturation is the formation of a congestion tree that extends to all the switches connected to the processors and that has its root at the switch connected to the hot memory module. The consequence of tree saturation is that it causes severe performance degradation. Hotspot contention caused by one application can potentially affect the whole system in multi-user systems if it is not managed properly. Therefore, it is crucial to minimize the negative effects of hot-spot contention. In [9], the objectives of hot-spot management strategies are defined and a taxonomy
that classifies such strategies as avoidance-based, prevention based or detection-based methods are proposed. It then reviews and compares several representative strategies and identifies several issues that need further research [9].

Tree saturation is eliminated by any strategy intended to reduce the effects of hot-spot contention should. The presence of hot spots should not adversely affect normal, non-hot-spot traffic and due to the memory contention, it is reasonable to expect hot-spot requests to suffer delays. In [9], several strategies for handling hot-spot contention are proposed. They are divided into three classes: avoidance-based, prevention-based, and detection-based strategies. The main concern of avoidance-based strategies involves a different way of distributing data so as not to create hot spots in the first place. The main use of prevention-based strategies is that decisions are made to control message traffic at runtime to prevent tree saturation and Detection-based strategies is about allowing the formation of tree saturation, but the system detects such contention and invokes congestion-control mechanisms to control it [9].

![Figure 3.6: Methods for storing elements: (a) row-major order; (b) column-major order; and (c) block-column data allocation.][9]

**3.4.1 Approach 1: Avoidance based**

According to [9], the avoidance-based strategies are discussed. An application can distribute data to avoid hot-spot contention by using the Gaussian elimination technique. In the following example, it has been determined that contention for the pivot row causes severe performance penalty and creates memory contention. In most major languages, except FORTRAN, row-major order for storing matrices is used; row is mapped to a memory module and elements of a matrix are stored on a row-by-row basis. Figure 3.6.a illustrates an example assuming that we have four memory modules; the system could store a 4 * 4 matrix R in the row major order. If the element rows are distributed to different memory modules contention for a row is avoided. Figure 3.6.b shows the system that stores the matrix on a column-by-column basis. Other problems that affect the application performance even more adversely have been introduced, while successfully avoiding memory contention. Due to false sharing, the column-major distribution
3.4. APPROACHES TO CONGESTION MANAGEMENT

introduces 15 times as many cache misses. This leads to the performance degradation. Bianchini and colleagues have introduced a block column data-allocation method to solve this problem. This method divides each row into cache blocks and then does the column-major allocation at the cache-block level rather than at the element level [9]. Figure 3.6.c shows the block-column allocation method. Software combining is proposed by Pen-Chung Yew, Nian-Feng Tzeng, and Duncan Lawrie. A tree of data items is created with software combining, as illustrated in Figure 3.7.

![Figure 3.7: A software combining tree.[9]](image)

The root of the tree acts as the original hot spot. From the tree in Figure 3.7, there exists 73 data items each initialized with a value of 8 since $N = 512$ and the trees branching factor $K$ is 8. These 73 data items are distributed as follows: 64 at level 2, 8 at level 1, and 1 at level 0. With each group sharing one of the data items corresponding to the leaves of the tree, the nodes are partitioned into 64 groups of eight. The last node decrements its parent data item. When this node in each group decrements its data item to zero. This algorithm terminates when the data item corresponding to the root of the tree is finally decremented to zero. In this example, hotspot contention is reduced because there are 73 data items each being accessed by only 8 nodes instead of one data item being accessed by 512 nodes [9].

3.4.2 Approach 2: Prevention based

Another strategy discussed in [9] which is the prevention based strategies. The Hardware combining strategy is a combination of basic techniques of preventing tree saturation like in software combining. The main concern of hardware combining is enhancing the switch by accomplish combining it in hardware. Examples of two computers that use this method, the Ultra computer and IBM RP3 systems. Figure 3.8 illustrates the network switch that performs combining in the Ultra computer. From Figure 3.8, it is depicted that a wait buffer in each switch keeps a record of the messages combined. Every switch combines memory request messages directed at the same memory location.
into a single message. The switch generates multiple replies to all combined messages using the information in its wait buffer, when the reply to a combined message reaches the switch that performed combining. Pair wise combining is used as a scheme by Ultra computer and IBM RP3 systems because it combines two messages into one. The system cost is increased by enhancing the network to support hardware combining [9].

Figure 3.8: The ultracomputer switch incorporating combining capability.[9]

3.4.3 Approach 3: Detection based

Another strategy discussed in [9] which is the detection based strategies. The Threshold-based feedback is a feedback scheme that monitors the size of the queue at each memory module. The memory module is hot and notifies the processors, if the size is at or above threshold Th. By holding back requests to the hot module, the processors respond. The system considers the module normal, and processors can resume directing requests to it, when the queue size falls below threshold. This scheme invokes the limiting mechanism, when the system determines that a memory module is hot. The feedback with limiting damping scheme is the cause of limiting only one hot memory request to enter the network from all N processors. The main difference between the feedback with limiting damping scheme and the feedback scheme is that the feedback with limiting-damping scheme allows the processors to submit at most one hot memory request per cycle. Under the feedback scheme the processors stops submitting requests to the hot module while it is hot. Therefore, the feedback with limiting-damping scheme eliminates the bandwidth restrictions imposed by the conservative basic feedback scheme [9].

Due to ever-growing network bandwidth and intensive network applications, congestion control in packet-switching networks became a high priority in network design and research. In [27], a new taxonomy of congestion control algorithms in packet-switching computer networks is proposed. Congestion control scheme is viewed as a control policy to achieve prescribed goals (e.g., round-trip delay, or throughput) in a distributed network environment based on control-theoretic concepts. Due to the saturation of network resources such as communication links, processor cycles, and memory buffers, congestion in a packet-switching network is a state in which performance degrades. The long delay of message delivery, waste of system resources, and possible network collapse are adverse
efforts resulting from such congestion, when all communication in the entire network ceases [27].

Most congestion control approaches are classified into two categories: approaches for congestion avoidance, and approaches for congestion recovery. The definition of a control system is: collection of objects bonded by some form of interdependence. The system will not remain in a state of equilibrium relative to each other and the external interface. The state of the system is changing with time in a manner which is entirely dependent on the characteristics of the stimuli and the bonds of interaction under the influence of external stimuli. The main concern of a control system is to maintain desired system responses regarding various input stimuli, to achieve dynamic characteristics of a system. The proposed taxonomy in [27] focuses on the decision-making process of individual congestion control algorithms.

The main categories in the taxonomy are: Open loop congestion control algorithms, closed loop congestion control algorithms. Each category is divided into several subcategories. Figure 3.9 illustrates the classification tree of the taxonomy. The algorithms in which the control decisions of algorithms do not depend on any sort of feedback information from the congested spots in the network are the open loop congestion control algorithms. The state of the network is not monitored dynamically. Controller or control actuator purely based on its own knowledge of local node, such as the bandwidth capacity of the local links, and the available buffers in the system are served by the congestion control algorithm. These techniques have an admission handling mechanism and have a continuous activation feature. Another advantage is stabilizing the traffic arrival process. The destination control algorithms intend to control the network traffic either at the destination or some intermediate nodes along the path to a destination. The rate of flow at the sources of traffic is controlled by the open loop congestion control algorithms at the source [27].
On the other hand, the closed loop control algorithms make their control decisions based on some sort of feedback information to the sources. The feedback can be either local or global: local means the feedback information comes only from immediate neighbors, whereas global means the feedback information goes all the way from destination to source. These algorithms are able to monitor the network performance dynamically, with the provision of feedback. Men distinguish two types of feedback involved in these algorithms and they can be explicit or implicit. In the explicit feedback scheme, feedbacks have to be sent explicitly as separate messages. If there is no necessity of sending the feedback explicitly, the scheme is said to be an implicit feedback scheme [27].

3.5 Congestion Elimination

3.5.1 Avoidance based techniques

The shared-memory multiprocessor system such as the IBM RP3 or Ultra computer may contain hundreds or even thousands of processors and memory modules. Communication between these processors and memory modules are provided by multistage interconnection networks such as the Omega network. In these systems, memory contention is created at some memory modules by any variable shared by these processors. This memory contention can cause severe congestion in the interconnection network, and can create a phenomenon called tree saturation. Due to hot-spot contention, tree saturation can seriously degrade the effective bandwidth of the shared memory system. The basic idea in [28] is to incorporate some hardware in the interconnection networks to trap and combine data accesses when they are fanning in to the particular memory module that contains the shared variable. It is hoped that memory contention at that memory module can be eliminated because data accesses can be combined in the interconnection network. Thus, the cost of the required hardware for such schemes is extremely high. Only for combining networks consisting of 2 x 2 switches, the switch size and/or cost by a factor between 6 and 32 increases due to the extra hardware. The hardware cost will be even greater with k x k switches (k > 2). The extra hardware also tends to add extra network delay. In [28], the shared memory multiprocessor system likes Cedar with a standard, buffered Omega network providing interconnection, and without expensive combining hardware. In addition, hardware facility in the shared memory modules to handle necessary indivisible synchronization operations for the shared variables is used. This hardware bypass regular memory accesses without delay and, hence, will not be penalized. Each memory module will handle memory accesses, including those memory accesses to shared variables, one at a time [28]. Due to the hot-spot variable, a software tree is used to do the combining, to eliminate memory contention. This idea is implemented in software instead of hardware. This scheme can achieve quite satisfactory results as compared to more expensive hardware combining as it is shown in [28].

Hot-spot accessing is classified in two ways: limited or unlimited depending on whether a given processor can have only one or more than one hot-spot request outstanding. Depending on whether the total number of accesses is fixed, or whether the total number varies depending on the number of conflicts or some other factor, the number of accesses will be fixed or variable. When the case is unlimited-fixed, then the total
number of requests generated by all the processors is fixed, but since it is assumed that
the addition is done indivisibly by logic in the memory, then each processor can have
more than one outstanding request to add an element to the shared sum assuming the
addition of vector of numbers to form a sum. When the case is limited-fixed, then the
processors are decrementing a counter to see which the last processor is. When the case
is limited variable, then the number of requests to the hot-spot depends on how soon the
variable gets reset, each processor continually reads the value of a shared variable until
the value changes, for example from zero to one and this is illustrated by busy waiting
where the processors may all be waiting for one processor to complete some task. It is
preferable that all shared variables in a software combining tree (i.e., the nodes of the
tree) reside in separate memory modules and it is important to minimize the possible
memory contention when the combining trees is implemented for hot-spot accesses [28].

The largest combining tree that is constructed for a hot-spot is a tree with minimum
fan-in. The total number of nodes in a combining tree with N leaves is $N/2 + N/4 + \ldots + 2 + 1 = N - 1$. The nodes in a software tree are assumed to be in separate
memory modules. Thus, it is always possible to spread those nodes across N separate
memory modules. More hot-spot traffic is created even though that traffic generates less
memory contention by constructing a software combining tree that creates many shared
variables. Assume that the software combining tree has a fan-in of k for each node, and
the hot-spot rate from a processor is $r \times h$. Because the number of accesses is not fixed to
begin with, the additional accesses caused by the combining tree are difficult to quantify
for variable access patterns. The total number of accesses could even be less than that
required by a single shared variable; the state change can be propagated more quickly
by the combining tree than by N accesses to a single shared variable, since busy-waiting
is often the cause of variable access patterns (with $q = 1$), and the number of accesses
for a busy wait operation depends on how quickly the state change is propagated to the
children in the tree, in practice [28].

![Figure 3.10](image)

Figure 3.10: Lower bounds on network bandwidth for various hot-spot rate.[28]

Because of conflicts in the network it is still not possible to achieve 100 percent
memory bandwidth, in a packet-switching Omega network, with finite buffers in each
switching element and with hot-spot rate $h = 0$. If a crossbar switch is used, then these
conflicts are also possible. R is assumed to be the maximum request rate reaching a memory module when no hot spot exists, then in a steady state, R is also the maximum request rate allowed for a processor. Therefore, R can be considered to be an absolute upper bound on the bandwidth per processor. The value of R depends on the network buffer size, the length of a request, and the network switch size. Figure 3.10 depicts lower bounds for various system sizes with h varying from 0-32 percent. According to Figure 3.10, we can deduce that those curves are in a very narrow range. The top dotted line in Figure 3.10 shows R, the maximum bandwidth we can get when there are no hotspot requests. The degradation factor is 1 + h (k - 1). This degradation factor is independent of the system size and reaches a minimum when k = 2. The optimal software combining tree for maximum memory bandwidth has a minimum fan-in of 2, given unlimited hot-spot requests [28].

Figure 3.11: Average network delay versus bandwidth for a network of size 256.[28]

There cannot be more than N hot-spot requests in the system at any time, if the hot-spot request rate is limited. While a hot-spot request is pending, regular requests still may be issued for systems with instruction look-ahead or with data perfecting capability. When h is very small, it is unlikely that there will be more than one hot-spot request pending at any time. When there is a pending hot-spot request, the case where no additional requests, hot or regular, are issued by the processor, is only considered. the delay of the hot-spot requests affects the bandwidth. The inherent nature of the hot-spot that prohibits further processor requests, if a software combining tree is used to eliminate the memory contention caused by the hot-spot requests. Several simulations for N - 256, with h varying from 0-32 percent are performed to study the effectiveness of a software combining tree. Figure 3.11 depicts the delay and maximum bandwidth when neither a software combining tree nor a hardware combining network is used. From
every curve from left to right we can conclude that each point represents a larger value of $r$. While $r$ increases bandwidth increases while delay stays relatively constant up to a point of saturation. After the saturation point, bandwidth ceases to increase while delay gets worse. The results clearly show low bandwidth and increased average network delay results. The maximum bandwidth of $0.63N$ is achieved when $h=0$ [28].

### 3.5.2 Prevention based techniques

According to [20], it has been proposed the use of feedback control schemes in multiprocessor systems. In a multiprocessor, individual processors do not have complete information about, nor control over, the overall state of the system. The main problem caused by hot spot accesses in multiprocessors using multistage interconnection networks is the tree saturation. The performance of all processors in the system is degraded by tree saturation, including those not participating in the hot spot activity. The feedback schemes are used to control tree saturation, reducing degradation to memory requests that are not to the hot spot, and thereby increasing overall system performance. Simulation studies presented in [20] show that feedback schemes can improve overall system performance significantly in many cases and with relatively little hardware cost. Damping schemes in conjunction with feedback are shown to further improve system performance. The use of feedback in computer systems design by targeting the problem of tree saturation in parallel computer systems that use buffered multistage interconnection networks (MINs) has been illustrated in [20]. Tree saturation is a special form of network congestion that can severely degrade the performance of multistage interconnection networks [20].

A blocking, buffered $O(N \log N)$ multistage interconnection network with distributed routing control is a widely proposed interconnection network for medium to large scale multiprocessors. The Omega network is an example of such a MIN. The Omega network contains $\log N$ stages of switches. One stage at a time, the memory requests enter the network at the inputs to the first stage and proceed to the outputs of the last stage. The routing decisions are made local to each switch. The state of any particular switch is unknown to other entities (processors, memories, other switches) in the multiprocessor, and a particular input request pattern to the network might cause the request arrival rate at some point to exceed the capacity with which that point can service requests, since there is no global control mechanism [20].

The main concern of tree saturation that was first observed by Pfister and Norton in conjunction with requests to a hot spot is the direct result of an arrival rate being too high. According to their analysis, the hot spot was caused by accesses to a shared synchronization variable. The requests will back up in the switch which connects to the hot module when the average request rate to a hot memory module exceeds the rate at which the module services the requests. The queues are backed up in the switches that feed it, when the queue in this switch is full. The tree of saturated queues may extend all the way back to every processor depending upon the number of outstanding requests and the reference patterns of the various processors. Until the saturated queues are drained, any request which must pass through any of the switches in the saturated tree, whether to a hot module or not must wait. Combining can be used when the problem is caused by accesses to synchronization variables or more generally, by accesses to the same memory.
location. Hardware combining uses special hardware in the network switches to combine requests destined to the same memory location. The response for the combined request is broken up (recombined) into responses for the individual requests. The effect of using combining hardware in the network switches increases the hardware cost of a multistage interconnection network by a factor of 6 to 32 [20].

Software combining is only applicable to known hot spot locations such as variables used for locking, barrier synchronization, or pointers to shared queues. Combining can improve overall network bandwidth by reducing the rate at which requests are submitted to the hot module, since the overall bandwidth of the network is determined by the number (or equivalently the rate) of requests that have to be serviced by the hot module. Since the memory requests alleviate tree saturation, then the latency of memory requests that do not access the hot memory module is also improved by software combining. If the hot requests are to different memory locations within same memory module, that is, the entire memory module is hot; unfortunately, combining cannot alleviate the bandwidth degradation or the tree saturation. A larger percentage of shared variables residing in a particular module could arise stride accesses that result in the non uniform access of the memory modules, or temporal swings in which variables stored in a particular module are accessed more heavily. To solve this problem, the researchers have suggested scrambling the memory to distribute memory locations randomly across the memory modules. It is hoped with a scrambled distribution that non uniformities will occur less often though. Tree saturation can still occur if any of the switches in the network have a higher load (in the short term) than other switches at the same stage even though processor requests may be distributed uniformly among the memory modules. The latency of memory requests that do not access the hot module, in the short term, but eventually tree saturation will occur even with alternate queue designs may be improved by alternate queue designs [20].

The requests that compound the problem must be prevented from entering the network until the problem has subsided to alleviate the problem of tree saturation. The requests to a hot module must be made to wait outside the network, at the processor-network interface (PNI), until the hot module is ready (or slightly before it is ready) to service them, and then enter the network at a rate at which they can be serviced by the hot module. From Figure 3.12, it is concluded that the system throughput should increase. If this can be done effectively, then tree saturation will be eliminated.

Figure 3.13 illustrates a multiprocessing system with feedback. Select state information is tapped from the MIN and the memory modules and fed back to the PNIs (Processor Network Interface). When the arrival rate of requests at the memory modules to exceed their desired values, then the PNIs use this information to modify their collective input to the network it is causing state values of the system. When such a performance degrading situation is occurring, then the feedback information must indicate to the processors. The difference between the arrival rate and module service rate is the state information that needs to be fed back to the processors, since tree saturation is caused by the arrival rate at the memory modules exceeding the rate at which the module can service requests. Unfortunately, the arrival rate by itself might be hard to measure. When the length of the queue feeding the module begins to grow, is a good indication that the arrival rate at a module is too high, and that tree saturation might
3.5. CONGESTION ELIMINATION

Figure 3.12: Estimated bandwidth improvement.[20]

develop [20].

Figure 3.13: A shared memory multiprocessing system with feedback.[20]

The queue length is a good candidate for use as state information to be fed back to the processors. The PNIs could respond by reducing the rate at which requests to that module are submitted into the network, as the queue length for a particular memory module grew. If the queues can be kept from overflowing then the Tree saturation will not occur. The length of a memory module queue has the desired range between one and zero if there are no outstanding requests for that memory module and the capacity of the buffer used to contain the queue. While the memory module is idle, If the queue
becomes empty, and there are outstanding requests for that module, then we are making inefficient use of the memory module. If tree saturation begins to develop and can interfere with requests to other memory modules, then the queue overflows its capacity. To keep the queue lengths within the desired range, there is a considerable amount of freedom available in the feedback policy chosen. The scheme used in [20] is quite simple and it defines a memory module to be hot if its queue length exceeds a certain threshold \( T \) and define it to be cold otherwise. While blocking requests to hot modules outside the network, The PNIs respond by allowing requests to cold modules to enter the network freely. Shortly after the module becomes hot, this causes the arrival rate of requests at a hot module to drop sharply [20].

It is hoped that average arrival rate will remain at its desired value and the queue length will remain within its desired range, by compensating for periods of high arrival rate with periods of low (or zero) arrival rate. This will help in avoiding the occurrence of tree saturation. An implementation with very low hardware complexity is the choice of this simple feedback scheme. When a module becomes hot (the size of the queue exceeds a predefined threshold \( T \)) and when a module becomes cold (the size of the queue falls below the threshold \( T \)), all that is needed to be done is to monitor the size of the queue at each memory module and notify the processors. Having N-bit feedback bus from the memory modules to all the processors is the most obvious way to do this. The status of each memory module is determined by the processors by simply looking at the information on the bus. Distribution of a bus can become difficult, as system size increases where a fan-out tree of buses can be used. This leads to an increase in the latency of feedback, but speed is not critical. The onset of tree saturation occurs as quickly as several network traversal times, this it has been estimated [20].

Less than a single network traversal time, the feedback information can be supplied to the processors. The number of modules that experience hot-cold or cold-hot transitions in a given cycle is low, alternate designs can be used for this feedback circuit, when the information content of the information being fed back to the PNIs is low. Large scale parallel programs typically have only one or two hot spots at a time and this is shown by preliminary experience. An N-bit feedback bus is clearly wasteful in such cases. The used feedback circuit is a log, \( N + 2 \) bit bus; arbitration is required, and this may also be implemented as a tree for large systems, since multiple transitions may take place in a single cycle. The hardware required for this feedback, while not trivial, is small in comparison to the MIN itself. The branching factor for a feedback tree would be quite large in particular, due to the lenient speed requirements [20].

A lookup table in a fast memory is maintained in Each PNI. This table is updated automatically from the transition messages on the feedback bus, and stores a single bit per memory module (indicating if the module is hot or cold). If the destination module is cold and when submitting a request into the network, the PNI checks the status of the requests destination module in the table and submits the request only. The processor stops from making additional requests, when a request is being held back at the PNI. as soon as the module becomes hot, the feedback scheme prevents a hot module from causing full tree saturation because, requests to that module are stopped from entering the network [20].
3.5.3 Reactive techniques

When the network is close to or beyond saturation, then Deadlock avoidance and recovery techniques suffer from severe performance degradation. During some intervals, many parallel applications produce burst traffic that may saturate the network, and increase execution time. In [3], a new mechanism to avoid network saturation that overcomes these drawbacks is proposed. The aim of this new technique is that each node estimates network traffic locally by using the percentage of free virtual output channels that can be used for forwarding a message towards its destination. When this number exceeds the threshold value, network congestion is assumed to exist and message injection is forbidden [3].

Several techniques to prevent congestion in multiprocessor interconnection networks have been proposed but, they either suffer from a lack of robustness or detect congestion relying on global information that wastes a lot of transmission resources. A new mechanism that uses only local information to avoid network saturation in wormhole networks is presented in [4]. It is robust and works properly in different conditions depending on the estimated traffic level; this new mechanism first applies preventive measures of different intensity and if necessary, it uses message throttling during predefined time intervals that are extended if congestion is repeatedly detected. The evaluation results of different network loads and topologies show that the proposed technique avoids network performance degradation, but without introducing any penalty for low and medium network loads [4].

The new adaptive routing framework which provides a unified framework for efficient deadlock-free adaptive routing and fault-tolerance is the Compressionless Routing (CR). CR illustrates the tight-coupling between wormhole routers for flow control to detect and recover from potential deadlock situations. CR is extended to Fault-tolerant Compressionless Routing (FCR) to support end-to-end fault-tolerant delivery. CR and FCR networks can achieve superior performance to alternatives such as dimension-order routing. Deadlock-free adaptive routing in networks with no virtual channels, simple router designs, order-preserving message transmission, applicability to a wide variety of network topologies, and elimination of the need for buffer allocation messages are the key advantages of Compressionless Routing. Data integrity in the presence of transient faults (nonstop fault-tolerance), permanent faults tolerance, and elimination of the need for software buffering and retry for reliability are the additional advantages of Fault-tolerant Compressionless Routing [14].

A new mechanism to avoid network saturation is presented in [3]. The main concern of this technique is to estimate locally the traffic in the network. This is done by using the percentage of useful free virtual output channels and allowing injection if it is greater than a threshold. To achieve good results in all situations, this method is with different message destination patterns and message sizes. The implementation of this mechanism requires only adding some hardware to the node, but it does not require new signals nor sending extra information. An efficient congestion control mechanism consists of three relevant properties: robustness and no penalty for the behavior of the network for low and medium loads. For different message destination distributions and message sizes, different topologies and network sizes, the mechanism should work properly for these
situations. The mechanism should not penalize the network when it is not saturated. It is desirable that the mechanism do not delay injection messages into the network, when network traffic is not intense. New problems should not be generated in the network by using the new mechanism. Messages are killed and transmitted again later, if they are stopped for more than a threshold. This will lead to a decrease in channel efficiency, increase message latency and generate more traffic in the network. Message throttling is a technique that stops or slows down the injection rate of messages into the network. Some nodes may begin to apply strict restrictions before others do, thus reducing traffic in the network, if they rely only in local information. The other nodes may not trigger the congestion control mechanism as a consequence [3].

The main concern of the new approach that is presented to perform congestion control in [3] is to control network congestion when the network is close to its saturation point. Message throttling is used in every node to keep the network out of saturation, when network load is considered to be high. When the traffic in the network goes down again, then the message injection into the network is absolutely stopped. A count of the number of free virtual output channels is performed, before injecting a newly generated message. This is only the case of the channels that are useful for forwarding the message towards its destination is considered in the count. The goal of this technique is that although some network areas are congested, it does not matter if they are not going to be used by the message. This technique should prevent injection regardless if the other channels are free in case the channels that may forward the message towards its destination are congested. To estimate the absence of network traffic, the number of useful free virtual output channels is used. The number of useful free virtual output channels relative to the total number of virtual channels that may be used for forwarding the message towards its destination is used to estimate network load, to make the mechanism independent of message destination distribution. The main concern is to use a fixed threshold that is expected to work fine with all messages destination distributions. The threshold value has to be higher than the value where performance starts falling and has to be fine-tuned empirically. Figure 3.14 illustrates the deadlock recovery fully adaptive routing algorithm and 3 virtual channels per physical channel, for the uniform and perfect shuffle distributions of message destinations [3].

Many packets block, when network traffic is near to the saturation point. The message advance speed decreases as physical channels are multiplexed into more virtual channels. In order to detect network congestion, the new mechanism uses message advance speed, where it measures the number of transmitted flits during a fixed time interval. A counter that increases each time one flit is transmitted is associated by every virtual channel. The channel is considered congested if it is busy and the number of transmitted flits is lower than a threshold value called the flit counter (fc) at the end of each interval. The header of a message either will block or advance at a low speed, when it reaches a congested area. This will lead to the consequence that the remaining flits of the message will also block or decrease their advance speed. Congestion is detected in network areas remote to the current node by tracking the speed in the network. Each node manages two different flags: the injection and the network flags. The network flag is set when congestion is detected in any virtual channel that is being used by messages injected by other nodes; on the contrary, the injection flag is set when some of the virtual chan-
Figure 3.14: Percentage of useful free output virtual channels versus traffic for different message destination distributions.\[3\]

nels that the node is using to inject messages into the network are congested. Injection and Network Congestion (INC) detection is the referred mechanism. Message injection restrictions are applied, when either the injection or network flags are set. When only network flag is set, then these measures are more restrictive when the injection flag is set. As congestion propagates quickly through the network, some preventive measures should be applied. The current node is not directly generating the problem, when only the network flag is set. The measures get more restrictive, if congestion is repeatedly detected in later intervals. The number of enabled injection channels is divided (integer division) by a factor r1, when injection congestion is detected. Injection is completely stopped but only during some interval called forbidden injection interval, if this quotient reaches zero. Regardless of the detected network traffic, and the first pending message, if any, is injected then one injection channel is enabled, after some cycles. Injection restrictions get harder or, on the contrary, are reduced, as network traffic is estimated periodically. This technique uses limited forbidden injection intervals for two main reasons. The first one avoids starvation and, the second one allows that other nodes also detect congestion. When congestion is no longer detected, injection bandwidth is recovered. Injection restrictions are also reduced gradually. Injection limitation is smoothed, after an interval without detecting any congestion. Injection bandwidth is increased by reducing the forbidden interval and later when it reaches the minimum value, by increasing the number of injection channels by one channel at a time. Figure 3.15 illustrates the Operation of the congestion control mechanism [4].

Dedicating hardware is inefficient because it is allocating permanently resources to handle very infrequent events, in case the potential deadlock situations are rare. Deadlock recovery may be more attractive than deadlock prevention, depending on the frequency of potential deadlock situations (PDS). A deadlock normally means the end of any network simulation and estimating the number of deadlocks that occur is difficult. The entire network is eventually blocked. By simulating a deadlock-free routing algorithm which uses two virtual networks, an adaptive one and a deadlock-free deterministic one, we are able to conservatively estimate the number of PDS. Because removing the need
CHAPTER 3. CONGESTION MANAGEMENT TAXONOMY AND RELATED WORK

Figure 3.15: Operation of the congestion control mechanism INC.[4]

for deterministic channels is sufficient to make the network deadlock free, the counted number of times messages needed to use the dimension-order routed virtual channels is used as a conservative (upper bound) on the opportunities for deadlock. Figure 3.16 illustrates the number of potential deadlock situations, normalized with respect to the adaptive channel utilization which is the solid line. The load-latency curve is also depicted in the figure which is the dotted line. The adaptive channel paths that are blocked for longer than 200 cycles for every message is forced to move to dimension-order routing and is counted as a PDS. By counting the number of times messages move through adaptive virtual channels, the adaptive channel utilization is calculated. Because messages in the deadlock set cannot be distinguished from those merely waiting on the deadlock set, the PDS counts overestimate the actual deadlock frequency. Figure 3.16 illustrates the potential deadlock situations [14].

3.6 HOL blocking Elimination Strategies

3.6.1 HOL blocking elimination at Network Level

In high-performance computing architecture, the Multiprocessing (MP) on networks of workstations (NOW) is of crucial importance. The routing interconnection networks in traditional MPs, use fixed-size flits and backpressure. One of the major contending interconnection technologies in NOWs is the ATM. It uses fixed-size cells, while backpressure can be added to it. ATM with backpressure that has interesting similarities with wormhole routing is discussed in [13]. ATLAS I, a single-chip gigabit ATM switch is implemented in [13]. According to a protocol resembling Quantum Flow Control (QFC),
3.6. HOL BLOCKING ELIMINATION STRATEGIES

Figure 3.16: Potential Deadlock Situations (PDS).[14]

the ATLAS I includes credit flow control (backpressure). The simulation done in [13] proves that this protocol performs better than the traditional multi-lane wormhole protocol: high throughput and low latency are provided with less buffer space. ATLAS I is fair in terms of latency in hot-spot configurations unlike wormhole, and Little sensitivity to bursty traffic is also demonstrated by ATLAS I [13].

ATLAS I is a general-purpose, single-chip gigabit ATM switch. It is developed within the project of the European Union ACTS. The main concern of the ATLAS I is in high-throughput and low-latency systems ranging from wide area (WAN) to metropolitan (MAN), and local (LAN) area networking. ATLAS I is suitable for supporting a mixture of services ranging from real-time, guaranteed-quality-of-service to best-effort, bursty and flooding traffic. The architecture of the ATLAS I consists of a 1616 ATM switch, with serial links of 622 Mbits/s each. ATLAS I can also be configured as 88 at 1.25 Gb/s, or 44 at 2.5 Gb/s, using link bundling. Using BULLs STRINGS GBaud serial-link transceiver, the ATLAS I links run ATM on top of IEEE Std. 1355 HIC/HS physical layer. Because of simpler circuitry, lower latency, and the capability to encode (unbundled) credits, HIC/HS was preferred over SDH/SONET. The chip contains internally a 1717 switch. The ATLAS I chip operates as a crossbar with a 256-cell shared buffer; 51 logical output queues and 3 multicast queues are maintained in this shared buffer. In ATLAS I men distinguish between three levels of priority, each level having its own queues; the output scheduler empties higher-priority queues before considering lower-priority ones. The control section of ATLAS I is shared among all incoming and outgoing links, and consists of two 5-stage pipelines. Figure 3.17 illustrates an overview the architecture transistor chip [13].
The network traffic is never dropped under credit-based (backpressure) flow control, because it is only transmitted when buffer space is known to exist at the receiver. The product of the peak throughput times the round-trip time (RTT) of data and credits defines the buffer space that is necessary and sufficient. According to buffer space availability, the single-lane backpressure indiscriminately starts or stops the transmission of any flit or cell. Due to head-of-line blocking effects, this causes poor performance. Multi-lane backpressure solves this performance problem by selectively allowing unblocked traffic to proceed, while preventing congested connections from occupying all available buffer space thus delaying all other traffic as well. A number $L$ of lanes exist on a link, in multi-lane wormhole routing. A new packet waits until a free lane exists, and then it acquires such a lane, when it arrives at the link entrance. Figure 3.18 illustrates a credit based flow control protocol [13].

Figure 3.17: ATLAS I chip overview [13]

Figure 3.18: QFC-like ATM credit protocol used in simulations [13]

Flits of packets that have acquired a lane can be transmitted, in any interleaving fashion, on the link; each flit carries the lane identifier of the packet to which it belongs. A flit is only transmitted if it acquires a credit for its lane. The receiver buffer is
partitioned into separate space for each lane; each credit is for one particular lane. The lane of the last flit of a packet is freed, after it is transmitted. Dedicating separate buffer space to each connection of virtual circuits and using a separate credit counter for each virtual circuit is the simplest way to introduce credit flow control for virtual circuits in ATM. The buffer space per virtual circuits must be equal to the virtual circuits peak throughput times the round-trip-time. The allocation of the static buffer ensures complete independence of every virtual circuit from all others, but it is also quite wasteful in terms of memory space. By dynamically sharing the receivers buffer among connections, the required buffer space can be significantly reduced [13].

The performance of the two protocols ATM and wormhole is compared in [13] using simulation, with realistic and detailed models that are based on ATLAS I. According to the results illustrated in [13], it is deduced that for a same buffer size and number of lanes, ATM performs consistently better than wormhole. The saturation throughput for ATM approaches the link capacity with buffer sizes as small as 4 or 8 cells per link (for 6-stage networks with short links), and stays high regardless of the number of lanes. When bursty or hot-spot traffic is in the network, multi-lane ATM switches give much lower delay than single-lane ones. Non-hot-spot traffic remains virtually unaffected by the presence of hot spots, when the number of lanes is higher than the number of hot-spot destinations. Wormhole performance is negatively affected to a much larger degree than ATM because of bursty and hot-spot traffic [13].

### 3.6.2 HOL Blocking elimination at Switch Level

By dividing the buffer storage associated with every network channel into several virtual channels, the throughput of the network is increased. Every physical channel is accompanied with several small queues, virtual channels, rather than a single deep queue. The virtual channels attached with one physical channel are allocated independently; on the other hand they compete with each other for physical bandwidth. The reason of Virtual channels is to decouple buffer resources from transmission resources. By using bandwidth of the network, the decoupling of buffers allows active messages to pass blocked messages. The simulation results in [7] show that the throughput is increased by a factor of 3.5, approaching the capacity of the network, given a fixed amount of buffer storage per link. Each node of in the interconnection network possesses a set of buffers and a switch. According to [7], the buffers are partitioned into sets associated with each input channel, an input-buffered node, as depicted in Figure 3.19 [7].

By associating the buffers on the output of each stage with the inputs of the next stage, an output-buffered switch can be considered to be an input buffered switch with a non-blocking first stage. The organization of a conventional network is shown in Figure 3.20 (a), where the flit buffers associated with each channel into a first-in, first-out (FIFO) queue. The restriction of this organization is the allocation of each flit buffer to contain only flits from a single packet. The physical channel is idled, if this packet becomes blocked and this happens because no other packet is able to acquire the buffer resources needed to access the channel. The flit buffers associated with each channel are organized by A network using virtual channel flow control into several lanes as it is illustrated in Figure 3.20 (b). The buffers in each lane can be allocated independently of the buffers
in any other lane. This added allocation flexibility increases channel utilization and thus throughput [7].

The flow control is performed at two levels, in a network using virtual channel flow control. The packets are assigned to virtual channels or lanes at the packet level. Channel bandwidth, switch bandwidth, and individual buffers are allocated to flits at the flit level. The lane assignment is performed by the node at the transmitting end of the physical channel. This node keeps track of the state of each lane buffer at the opposite end of the channel for each of its output channels. The state information includes for each lane,
3.6. **HOL BLOCKING ELIMINATION STRATEGIES**

whether the lane is assigned, and if it is the case, how many empty buffers it contains. Based on the destination and the routing algorithm in use for the packet on node A, in an input buffer, it selects a particular output channel. This packet is then assigned by the flow-control logic to any free lane of the selected channel. The packet is blocked, if all lanes are in use. The flit-level flow control is used to advance the packet across the switch and physical channel, once a lane is assigned to a packet. A flit must gain access to a path through the switch to reach the output of node A, and the physical channel to reach the input of node B, to advance from the input buffer on the transmitting node (node A) to the input buffer on the receiving node (node B) [7].

The main concern of Advanced Switching (AS) is more than merely an extension of PCI Express where this last is primarily as a chip-interconnect. On the other hand, the AS functions are vastly different and indeed more sophisticated than that of PCI Express. Congestion management in such a setup is important for optimal utilization of the fabric bandwidth. To control congestion in network fabrics depends on end-to-end and link-by-link schemes for controlling packet injection. Transient congestion occurs in fabrics operating well below their saturation limit. An end-to-end scheme results in under-utilization of the fabric bandwidth while a link-by-link scheme does not prevent congestion from spreading. The paper of [15] explains that a congestion control mechanism which is called Status Based Flow Control (SBFC), it has been incorporated into AS and is specifically targeted for solving the transition congestion problem. The SBFC mechanism enables upstream switch nodes to modify the transmission of packets based on the congestion status of links in a downstream switch. The simulation results in [15] show that the SBFC mechanism indeed permits optimal usage of the fabric bandwidth during periods of transient congestion [15].

Figure 3.21 depicts a typical scenario for showing congestion in both persistent as well as transient. In that scenario three switches A, B and C that is part of a larger AS fabric are shown. The link 2 of switch node C is the common link exit of the packets that are belonging to flows g0, g1 and g2, before they reach their final destination. The packets that belong to f0 and g0 share a common output link which is the link 1 of switch B. The transient congestion is caused if an intermittent flow g3 targets link 2 such that the link bandwidth is momentarily exceeded. It affects the bandwidth of g0, g1 and g2 sharing the link. The flows that are not heading out to the congested link are also affected by this undesirable side effect. Due to the transient congestion, it results in a temporary drop in the input link credit return from C, since packets belonging to g0 back up [15].

According to Figure 3.21, it can be concluded that if switch B is made aware of the transient congestion in link 2 of switch C, it can modify its link scheduling policy such that it gives priority to packets belonging to f0 over that of g0. A switch can incorporate that information and accordingly change its scheduling policy, if the next turn output buffer status were available. This is the main localized congestion control, the so-called status based flow control (SBFC) in AS. The scheduler is allowed to prioritize packets heading out to non-congested output ports by the backpressure from a congested output buffer. Since, the scheduling policy is based on the next turn output buffer status information of a switch located downstream, the SBFC mechanism takes it a step further [15].
Only because AS uses a source-based routing, it must be mentioned that next turn (port) status information can be utilized by an AS switch scheduler. The fabrics like Ethernet, ATM use a destination based approach (lookup-table). A switch in these fabrics, is not able to use the next turn (or port) information to influence the scheduling policy. The incorporation of information at the same time for the next output ports for every entry can increase the table size exponentially, in the order of ports. The downstream switch can report congestion on a per-flow basis, because it can be argued that rather than reporting status on a next turn (port) basis. The number of individual-flows is non-deterministic, unlike next turn output ports that are finite. The downstream switch must inspect individual flows and return status for each flow, to allow flow based status notification. Nowadays, most of the fabrics rely only on traditional approaches like end-to-end and link-based flow control for congestion management. In [15], it describes in detail the two components that are needed for enabling the SBFC mechanism. The first component is the generation of the output buffer status notification by the downstream switch node. The second component is the response taken by the upstream switch node on receiving such a notification [15].

In [17], a general approach has been developed to global reactive congestion control in multicomputer networks. The approach exploits control lines such as those used for handshaking in the flit-level flow control of wormhole routers to distribute information about congestion and uses timeout mechanism to detect congestion. It is also based on a mechanism that limits the demands placed by the network interface and the processing element. The proposed congestion control can provide network stability and predictable network performance is shown in [17]. Providing bounds on average delay and worst-case delay can be done, by choosing the right timeout. The network can be kept furthermore out of saturation with appropriate timeouts [17].
3.7 Conclusion

This chapter concludes that The HOL (Head of Line) blocking is one of the main concerns that take place in networks based on switches with queues at their input ports. The HOL effect is due to the fact that blocked packets destined to congested output switch ports prevent the advance of packets destined to other non congested output ports. The switch architecture in on-chip networks is affected by the congestion in the network. Congestion spreads from the root to the leaves, due to the use of flow control. It has been thought that congestion trees exist from the fact that the root of the congestion tree becomes congested. Only in a particular scenario that rarely happens, where the different traffic flows that form the tree join only at the root occurs. When flows meet at the root switch, the rate at which packets arrive is higher than the rate at which packets can be transmitted through the output port. Thus, queues begin to fill at the root switch and thus, congestion starts, due to the available bandwidth at the output port of the root switch is divided among the different flows and is lower than the bandwidth required by each flow. One of the main points discussed in this chapter is that tree saturation is eliminated by any strategy intended to reduce the effects of hot-spot contention should. The presence of hot spots should not adversely affect normal, non-hot-spot traffic and due to the memory contention, it is reasonable to expect hot-spot requests to suffer delays. Several strategies for handling hot-spot contention are proposed. They are divided into three classes: avoidance-based, prevention-based, and detection-based strategies. The main concern of avoidance-based strategies involves a different way of distributing data so as not to create hot spots in the first place. The main use of prevention-based strategies is that decisions are made to control message traffic at runtime to prevent tree saturation and Detection-based strategies is about allowing the formation of tree saturation, but the system detects such contention and invokes congestion-control mechanisms to control it. Different techniques of congestion elimination like the Avoidance based techniques, the prevention based techniques, and the reactive techniques. This chapter discusses about the HOL blocking elimination strategies like the elimination at network level and at switch level.
4.1 Introduction

In this chapter, we start by presenting the solution algorithm and explain the pseudo code that is responsible for controlling the work of the multiplexers. The solution consists of a feedback in which the congested packets are queued; this is explained in section 4.2. In sections 4.3, I present the simulation bench with an additional explanation on the changes needed in order to run the simulation and be as close as possible to the real situation. In order to run a comparison of test scenarios, I present and interpret the scenarios without the solution implementation in section 4.4. Section 4.5 will represent the use case scenarios with our solution and the relative interpretation.

4.2 Algorithm and pseudo code of the solution

Reflecting back on the problem presented in Chapter 1, Figure 1.1, I present here the form of the solution in a SARC environment in Figure 4.1.

Figure 4.1: solution of the SARC [10]

In Figure 4.2, the multiplexers will serve as an early stage for the detection of congested packets. Since the measurements of the worse and best case scenarios have been conducted on mesh networks, and then the same type of mesh networks will form the test bench. The modified or additional queue described below will be the modification that will take place on an input queue in the switch.

In order to avoid HOL blocking, the algorithm needs to be able to distinguish congested and not-congested packets. This requires to have available all the information stored in the RCT of the network interface; At best when there is congestion at destination a single additional entry is needed at the RCT. When a destination is congested, the RCT(s) is notified and adds the corresponding destination in its table.
4.2.1 Description of the RCT table

The RCT table and progressive address translation has been extensively described in chapter 2, figure 2.3. The packet does not carry the full address of the destination; it only carries a partial address. The routing table then directs the packets for few more steps until the next hop(s) and so on. In order to simulate this behavior in our solution, we choose to use mesh network as the base for all simulations. The reader is able to get a full overview regarding interconnection networks and mesh networks specifically in Chapter 3, section 3.2. The following points describe the RCT table chosen in the mesh:

- There is a router unit on every node that enables it to forward packets of data to other nodes in the network, Figure 3.1.
- The router will decide to which of the neighbors it will forward the packet to.
- The router does not decide itself the whole trajectory of the packet.
- In our case, the routing is deterministic and the packet is assigned the same route between two different nodes.

The above explanation is not the only function of the RCT, it extends to be a congestion management table too; thus; the term routing and congestion control table.
4.2. ALGORITHM AND PSEUDO CODE OF THE SOLUTION

In Figure 4.2, the RCT table contains information regarding the address ranges in order to route packets through the hops of the network. In addition to that, there is a congestion chart which provides information regarding the congested destinations. In the case of a mesh network the addresses are in the range of $2^n$ to the power of $n$. Let's take for example the 4x4 mesh network, the address range is $2^{**4}$, which gives a total of 16 hops ranging from addresses 0000 to 1111. The role of the congestion table is to provide the congestion information on each of the addresses in that range and denotes a congested or not-congested hop. In order to understand in example the function of the RCT table, we redraw the table in Figure 4.2 in the following format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Congested=1, not congested = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>1=&gt; congested</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>1=&gt; congested</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: Example of the RCT table in the proposed solution

This example table shows that destinations (hops) 0110 and 1010 are congested. They behave as hotspots in the network. A graphical representation of this table is drawn in Figure 4.6, where the hotspots are the two red nodes.

The algorithm controlling the queues described in Figure 4.3 has to have the following criteria:

1. All congested packets, when possible, are sent to the additional queue.

2. Entries in the RCT are updated based on congested destinations. A congested entry is update with 1, while a non-congested entry is updated with 0.

3. When a packet in the feedback queue is in non-congested state, bring it as soon as possible to the main queue, after receiving the update form the routing table. The previous prevents starvation of packets from the feedback queue.

4.2.2 Algorithm and multiplexers control

In Figure 4.2, the goal is to identify the congested packets as early as possible. For this reason a feedback queue has been added where the congested packets will be stored and
four multiplex de-multiplex stages. Let’s take a cell that reaches multiplexer 1, then the RCT is checked in order to determine whether the destination is congested or not. If the destination is congested, then it is forwarded towards the feedback queue (lower queue) before it even gets into the main queue (upper queue). This proactive approach should mainly prevent as much as possible congested packets from being present in the upper queue.

At dequeue time, demultiplexer 3 stage checks if during the time any packet that has been in the main queue has a change in destination status. If the packet is enqueued while the destination was free and then this same destination got blocked, then we don’t want this cell to form a head of line, it is being sent again in the feedback queue. Yet again a penalty is allowed at this level, if the feedback queue is full, then a HOL takes place on the main queue.

Multiplexers 2 and 4 serve as entry point respectively to the main queue and the feedback queue. At stages 1 and 3, the routing table is automatically consulted in order to decide the state of (non)congestion. As explained in Figure 4.5, the all routing tables in the network are updated whenever there is a change in the status of the hotspots. This same update of the routing tables initiates the operation of stage 2.

The control of the multiplexers is done taking into considerations the following three parameters which are read at every iteration or time slot:

1. The destination address of the cell at input.
2. The destination address of the cell at head of main queue.
3. The destination address of the cell at the head of the feedback queue.

We set the flag for congestion to be one and the flag of non congestion to be 0. Since we are using this Boolean representation, then the amount of total control possibilities is 8 ($2^3$). At every iteration, the routing table is checked where hotspots information is being updated.

<table>
<thead>
<tr>
<th>Destination input cell</th>
<th>Destination cell at main queue head</th>
<th>Destination cell at feedback queue head</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Table of multiplexers control possibilities

The following algorithm presents all possibilities in order to control the different multiplexers in the circuit. The 8 possibilities mentioned in the table above will behave as the decision making steps for the control signals on the multiplexers.
One has also to be aware of the following assumption and design criteria: In our
design, the priority of the multiplexers is achieved via Round Robin scheduling. Let’s
take for example the first row in the truth table 4.1, we notice that all packets have no
congested destinations. then multiplexer 2 allows forward of the two packets simultane-
ously.

Function Enqueue
/*Check if there is a packet at input*/
If packet is present at input then

/*Check if any of the Queues is at 50% capacity*/
If queue size <= 50%

/*check the RCT in order to decide on hotspot*/
If destination is congested enqueue in feedback;
Else enqueue in main;

Else halt; /*wait until queue capacity is less that 50%*/

Function dequeue_feedback
/*check if there has been an update on the routing table RCT by */
/*comparing with the previous saved version of RCT*/
If RCT is equal = RCT n-1 of previous iteration
Halt; /*no dequeue is performed from the feedback queue*/
Else dequeue N packets;
/*N= the number of packets at update time */

/*count the number of packets from the head of the queue until*/
/*the non congested one*/
N=0;
Loop Size of queue;
If packet is congested n=n++ /*increment n*/;
Else n remains unchanged;

Dequeue n packets in feedback queue; /*congested packets are back*/
Dequeue packet n+1 into the main queue

Function multiplexer_control
/*the select of the MUX is denoted with a boolean value
1 when a packet needs to be directed to a queue tail and 0 for packets that need to be directed towards another destination*/
/*the other destination could be another MUX/DEMUX or output*/
Boolean: ctrl_M1, ctrl_M2,ctrl_M3, ctrl_M4, ctrl_M5;
Switch (fn(InCell.dest, Qcell.dest, Fcell.dest))
Case A: ctrl_M1=1, ctrl_M2=1,ctrl_M3=1, ctrl_M4=1, ctrl_M5=1
break;
Case B: ctrl_M1=1, ctrl_M2=1,ctrl_M3=1, ctrl_M4=1, ctrl_M5=0
break;
Case C: ctrl_M1=1, ctrl_M2=1,ctrl_M3=0, ctrl_M4=1, ctrl_M5=1
break;
Case D: ctrl_M1=1, ctrl_M2=1,ctrl_M3=0, ctrl_M4=1, ctrl_M5=0
break;
Case E: ctrl_M1=0, ctrl_M2=1,ctrl_M3=1, ctrl_M4=1, ctrl_M5=1
break;
Case F: ctrl_M1=0, ctrl_M2=1,ctrl_M3=0, ctrl_M4=1, ctrl_M5=0
break;
Case G: ctrl_M1=0, ctrl_M2=1,ctrl_M3=0, ctrl_M4=1, ctrl_M5=1
break;
Case H: ctrl_M1=0, ctrl_M2=1,ctrl_M3=0, ctrl_M4=1, ctrl_M5=0
break;

Table 4.3: Pseudo code of the solution algorithm
The following pseudo code reflects the full function of this solution reflecting all 8 possibilities under the condition when there is no change on the RCT:

InCell.dest /*destination address of input cell*/
Qcell.dest /*destination address of cell at head of main queue*/
Fcell.dest /*destination address of cell at head of feedback queue*/

LOOP time slot(i);
/*this is an iteration where the fifo queues are enqueing and dequeuing at the same time and

IF (InCell.dest == 0 AND Qcell.dest == 0 AND Fcell.dest == 0)
Fn(enq) InCell to main queue;
Fn(def) Qcell to out;
Fn (deq) Fcell towards tail of main queue;

ElseIF (InCell.dest == 0 AND Qcell.dest == 0 AND Fcell.dest == 1)
Fn(enq) InCell to main queue;
Fn(def) Qcell to out;
Fn (deq) Fcell towards tail of feedback queue;

ElseIF (InCell.dest == 0 AND Qcell.dest == 1 AND Fcell.dest == 0)
Fn(enq) InCell to main queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of main queue;

ElseIF (InCell.dest == 0 AND Qcell.dest == 1 AND Fcell.dest == 1)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of feedback queue;

ElseIF (InCell.dest == 1 AND Qcell.dest == 0 AND Fcell.dest == 0)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to out;
Fn (deq) Fcell towards tail of main queue;

ElseIF (InCell.dest == 1 AND Qcell.dest == 0 AND Fcell.dest == 1)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of feedback queue;

ElseIF (InCell.dest == 1 AND Qcell.dest == 1 AND Fcell.dest == 0)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of main queue;

ElseIF (InCell.dest == 1 AND Qcell.dest == 1 AND Fcell.dest == 1)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of feedback queue;
Else (InCell.dest == 1 AND Qcell.dest == 1 AND Fcell.dest == 1)
Fn(enq) InCell to tail feedback queue;
Fn(def) Qcell to tail feedback queue;
Fn (deq) Fcell towards tail of feedback queue;

Let’s look closer at the eight scenario steps described in the algorithm above and explain them in detail. The detailed layout of these steps will give a better overview for the reader on how the control of the different multiplexers will behave. The assumption here remains that the multiplexer needs one clock cycle per state.

In the first scenario, the first if statement in the algorithm, the cells at the input, head of the main queue and the head of the feedback queue have all non congested destinations. Multiplexer 1 will forward to multiplexer 2 and the cell is enqueued in the main queue, while multiplexer 3 forwards the dequeued cell from the main queue towards the output. Additionally multiplexer 5 will forward the dequeued cell from the feedback queue into multiplexer 2 towards the main queue. Both cells are fed via Round Robin scheduling.

Figure 4.3: scenario 1 case A in the algorithm

In the second scenario, the second elseif statement, the input cell and the cell at the head of the main queue have non congested destinations. the cell at the head of the feedback queue has a congested destination. In this case 1 will forward to 2 and onwards to the main queue, the cell at the head of the main queue will be forwarded to the output via 2. the dequeue on the feedback queue will go will be enqueued back in the tail of the same queue via 5 to 4.

In the third scenario, the input cell and the cell at the head of the feedback queue have non congested destinations. This part brings us back to the same reasoning mentioned in the first scenario regarding multiplexer 2. While the cell at the head of the main queue has a congested destination, it gets directed to the tail of the feedback queue via 3 and then 4.

In the fourth scenario, the cell at input has a non congested destination while the both cells at the head of both queues have congested destinations. Multiplexers 1 and
CHAPTER 4. CONGESTION HANDLING WITH FEEDBACK QUEUE

2 will enqueue the input cell, 3 will dequeue via 4 in order to enqueue in the tail of the feedback queue and 5 will also send to 4. Again, Round Robin scheduling ensures that both cells get served.

In the fifth scenario The input cell is congested, while the cells at the heads of the queues are free to go. This means that 1 will forward the congested packet to 4 directly in order to enqueue it in the feedback queue destined for congested packets. Multiplexer 5 dequeues the cell that is free to go via multiplexer 2 towards the tail of the main queue and 3 forwards the dequeued cell from the main queue towards the output.

The sixth scenario, the input cell is congested and the output of the feedback queue, while the cell at the head of the main queue is free to go. Multiplexer 1 will forward the congested packet towards 4 and enqueue it in the feedback queue, also 5 will control the path towards 4. This is yet another scenario where multiplexer 4 depends on Round Robin scheduling to serve all incoming cells.

The seventh scenario, the input cell is congested and so is the cell at the output of
the main queue, while the head of the feedback queue is free to go. Multiplexer 5 will direct to 2, while 1 and 3 both direct towards 4.

The eighth scenario, the cells at input, head of main queue and head of feedback queue are destined towards hotspots. All cells are directed towards multiplexer 4 which has to process all.

Now we move to describe the part of the algorithm related to the case where a change occurs on the routing table. When a congested destination becomes accessible again, the RCT gets an update and based on that update all packets in the feedback queue have their .Fcell.dest. bit updated to reflect the new situation. This is described in the algorithm in the following statement:

```
If RCT is equal = = RCT n-1 of previous iteration
Halt; /*no dequeue is performed from the feedback queue*/
Else dequeue from feedback N packets;
```

If there is no update on the RCT table, then the circuit enqueue and dequeue standard
according to the same control mechanism described in the function multiplexer-control. The Halt stops dequeuing from the feedback queue from the head of the queue towards the tail of the same queue. The Else statement signifies that a change has occurred on the already known RCT and the new version of the RCT is similar to the previous version of the RCT. In this situation, the number of congested packets is counted starting by the head of the queue, this number is $N$. $N$ packets are dequeued back to the tail of the feedback queue and this will allow the next following packet $N+1$ to be directed towards the main queue. The design is supposed to keep only congested packets in the feedback queue, if a packet becomes green, then it needs to move as soon as possible and as quickly as possible to the main queue.

Finally, we notice the condition on the queue size set to 50 percent. This has been devised in order to keep half of the queue free. This is needed for in the case where all congested packets in the feedback queue become green and they need to move to the main queue, then we always need to have the right space that accommodates this move.
4.3. The Simulator

4.3.1 Sicosys Simulator

SICOSYS was developed and has been very successful. However, the simulator is nowadays obsolete. Although its not structurally limited, the simulation times it offers are too big to make it a useful tool. SICOSYS is a time driven simulator developed in C++ having in mind modularity, versatility and connectivity with other systems. The models used are intended to resemble the hardware implementations in some aspects while keeping the complexity as low as possible. In this way, the simulator mimics the hardware structure of the routers instead of just implementing their functionality. In
order to benchmark the routers and networks against other alternatives and to enable
the system to keep up with new developments while presenting a homogeneous user in-
terface, the design of the simulator has payed much attention at its extensibility and the
user interface [19].

4.3.2 SIM Simulator

The choice of a simulator is crucial for this thesis work and it was a challenge to research
which simulators describes the Head of Line problem the most. Firstly, the choice of
simulators laid on the ANSI C simulator the so-called sim simulator. SIM is a slotted-
time simulator, instead of using discrete-event simulation such as pulling out events from
the event list. The whole system progresses one slot-time at a time. The main loop of
the simulation looks like it is described in Figure 4.12.

```c
/**... some initialization steps... */
for(now = 0; now < simulationLength; now++){
  /* 1. Check for new arrivals to the switch */
  /* 2. Schedule transfer of cells from input of switch to output of switch */
  /* 3. Schedule departure of cells from switch. */
  
  /* Print Results */
}
```

Figure 4.12: Pseudo code of the main loop of the sim simulator [11]

SIM has separate modules for different components of the switch: the switching
fabric, queuing policies, scheduling algorithms, as well as traffic models. In addition to
that, users can define which among all the possible statistics to be collected. Simulation
times can be increased by turning off some of the unneeded statistics.

1. Traffic: A Traffic source generates the input to the switch. It generates a stream
of cells which satisfies a certain property e.g Bernoulli traffic. At each cell time,
Traffic source checks if it is the right time to feed a cell to the switch creates a cell
accordingly, and the result is passed to the InputAction of the switch, which is then
informed whether it has an arrival at that cell time or not. The advantage here
is the implementation of several traffic types such as bursts and Bernoulli uniform

2. InputAction: A cell arriving at the switch is accepted by the InputAction, which
determines which input queue to place the cell in and, if applicable, whether to send flow control cells back to the previous stage.

3. **InputQueues**: Each input of the switch maintains a separate FIFO for each output port.

4. **Fabric**: The Fabric is the interconnection mechanism between the InputQueues and the OutputQueues. Crossbar switches, fast busses running at multiples of the line rate and buffered Batcher-Banyan networks.

5. **Scheduling Algorithm**: The switch fabric may use a switch-specific scheduling algorithm to decide which cells to transfer in the next cell time. This is particularly important for input-buffered switches. Many scheduling algorithms have been implemented, including FIFO, Parallel Iterative Matching.

6. **OutputQueues**: The cells are transferred from the Fabric to the OutputQueue, which, by default, is a simple single FIFO.

7. **OutputAction**: The OutputAction removes cells from the output queue and delivers them onto the outgoing line. Currently, this actually leads to the destruction of the cell. Since this is not needed in the simulation anymore.

The choice remains on SIM, since it is a lighter, faster and more agile simulator. In other words, it outperforms the Sycosis in the purpose of this thesis.

The simulation model is presented by the flow chart in Figure 4.14. This flowchart represents the modification that need to be made to SIM in addition to the logic explaining the counters used and the checkpoints in order to be able to vary the size of the queues, read the destination address and check if the queues got full. In all cases presented in this section, the queues haven’t filled up except when the size was fixed in order to simulate a full occupancy scenario.
4.4 Simulation results

In this section, a clear presentation of all the simulation scenarios and the suggestion of the head of line solution are presented. After a brief overview has been given over the simulator choice, it is crucial to mention that all the measurements are done on a 2D-mesh. Different configurations have been tested such as 4x4, 8x8, 16x16 and in some scenarios 32x32.

4.4.1 Explanation of the scenarios

In Figure 4.15, an explanation of the head of line in the simulation bench is presented. The mesh network shows 2 red spots. The red spots represent the congested areas or hotspots. Any hotspot in the Network is the cause of a head of line blocking and will
cause latency and delay which will affect the throughput of the network. The following two sections, 4.4.2 and 4.4.3 represent the results of the runs with and without hotspots respectively. We refer to this as being the worse case and best case scenarios.

Figure 4.15: Head Of Line

Figure 4.16 represents a construction of the switch which is a crossbar switch. The packets enter at the incoming queue and based on the destination, it will be directed towards the right output queue. The colors yellow, green and red are simply for the destination explanation. Let's assume destination red is blocked, and then the destinations green and yellow are blocked behind it and in this situation packet red represents a head of line in the queue. In the worse case scenario a destination is being marked as full in the routing table. Once this destination is marked as a hotspot, the de-queue of cells from the input queues will halt, and after every run, the statistics are being collected. In the best case scenario, keeping in mind that we want to fully eliminate the head of line, an additional input is added per destination. The reader can look for the exact details under each of the scenarios in the following sections 4.4.2 and 4.4.3. In section 4.5, the reader can get into the details of the solution and the use case results. All latency measures shown in the figures are in time slots which is one or a multiple clock cycles. Figure 4.1 represents the changes that needed to be implemented to the chosen simulator in order to support this type of reproduction.

The latency measurement is being done per switch on the network, the sum of all the statistics are being collected and then averaged on the number of switches crossed. In every cell, the following time measure is being collected:

2. arrival time: time elapsed between the departure of the cell and the arrival at any given switch. This is initialized to zero at the level of the traffic generator.

3. Input Latency: this is the time that a cell spends in the input queue.

4. Fabric Latency: The time that the cell takes in order to get switched.

5. outputLatency: The time that the cell takes to leave the switch, i.e get dequeued.

In the case where a HOL blocking is introduced, the cells are kept in the buffer and the same time of input and output is applied. The latency of a cell is defined by:

\[ \text{TotalCellLatency} = \sum [\text{arrivaltime} + \text{inputlatency} + \text{fabriclatency} + \text{outputlatency}] \]  

Finally the total latency per cell is being averaged on the total number of crossed routers. This same measurement scenario is implemented through the simulation of all scenarios.

\[ \text{averagelatency} = (\text{totalcelllatency} ÷ \text{numberofcells}) \]  

The description above is applied to all 3 scenarios of the simulation. Every run is 100 times with an injection of up to a 100 cells in each run, in order to get an accurate measurement. As the reader will notice, the measurements are taken with random traffic, then with HOL introduced and finally with HOL introduced and the solution. In reality, the traffic should remain random and HOL will take place randomly, the intention of introducing a permanent HOL in this thesis is to understand the gravity of this type of blocking and the amount of latency that it generates. Two Dimensional mesh wormhole routing is used, When passing between switches in a 2D mesh, a packet follows four possible directions: East, West, North, and South. Eight distinct turns are possible in the path followed by a packet.
4.4. SIMULATION RESULTS

4.4.2 The worse case scenarios

The worse case scenario presents low cost design and is the one with the lowest performance. This is defined with one input queue on each switch of the mesh network. The input queue then has to host all the packets destined towards different destinations. If a destination is congested, then a green packet is trapped behind a red packet destined towards the hotspot. We define the size of the queue in this work to 20 MB. Figure 4.17 explains such a design.

![Figure 4.17: worse case scenario queue design](image)

4.4.3 The best case scenarios

The best case scenario is the design with the most performance on the expense of cost. The design is described in Figure 4.18. We set an input queue per destination.

![Figure 4.18: best case scenario queue design](image)

In this scenario, the congested destination in red does not affect the packets destined to green, blue and orange. Thus; the HOL is completely eliminated, the fact that makes this scenario the highest on performance level. The number of additional queues is equal to the number of possible congestion trees in the network. The number of possible connections in a 2D-mesh is NxN. The total number of queues to serve all possible destinations has also to be NxN. If we draw a closer look to the cost, we notice that for example, a 4x4 2D-mesh needs 16 queues per input in order to serve all possible destinations without HOL blocking effect. Assuming again a 20MB size per queue this sums to a total of 320 MB.
4.4.4 Results of the proposed solution

The worse case scenarios are represented with the red graph in all figures 4.19 to 4.26. The measurements are taken with the application of the design scenario described in Section 4.4.2. The reason, for which the worse case scenarios are being measured, is to understand the impact of HOL on performance. In this case the performance of the network is being measured with the average latency of a cell traveling from and input destination towards the output over several hops in the network. Delay in the term of latency is inversely proportional to the throughput of the network. When the overall latency of a network increases, then the throughput decreases accordingly. In order to understand closer the impact of HOL in the worse case, lets have a look respectively at the results.

In Figure 4.19, a 4x4 2D-mesh shows an increase in the latency towards 180 clock cycles at 99 percent injection rate. What is very interesting to observe is that the penalty of HOL shows already at 20 percent of the injection rate. The average latency of the packet is 40 clock cycles when 2 hotspots are introduced in the network. We also notice at around an injection rate of 60 percent that the latency increases linearly. Furthermore; on the worse case scenarios with the 8*8, 16*16 and 32*32 2D-meshes, we notice the same observations mentioned in the case above in addition to the fact that the upper bound of the latency gets respectively less: 100 clock cycles in the case of the 8*8 2D-mesh, 45 clock cycles in the case of the 16*16 2D-mesh and 30 clock cycles in the case of the 32*32 2D-mesh (respectively Figures 4.17 to 4.20). This observation is explained with the fact that in the measurements, 2 hotspots were introduced, the larger the network, the more possibilities for alternative routes that the packets may take. In order to prove this, we run the measurements based on random traffic without inducing hotspots in the network. In this situation, hotspots and HOL could have occurred but their occurrence is left up to the random address generation. In Figures 4.23 to 4.26, the worse case scenario upper bound is ranging between 20 and 30 clock cycles.

The above explanation of the worse case scenario can only be put in context with the best case scenario described in this paragraph. Looking at the same series of figures 4.19 to 4.26, we notice that the latency with the best case scenario is always negligible. All results show that indeed the best case scenario is capable of delivering a negligible delay and the maximum network throughput. The design of this scenario has been described in Section 4.4.3, and it is expected that with such a design, the network achieves maximum throughput. The cost of such a design remains extremely high. Lets take for example the 32*32 2D-mesh, 1024 queues are needed per switch in order to serve all destinations and eliminate HOL. It is clear that this is not an optimal design. In addition, reflecting back on the scalable architecture that was presented in Chapter 2, scalability is at the core. Assuming the network increases dynamically to 70 hops for example, the amount of queues will be immense. Taking the same 20MB queue and multiplying this number by 1024 queues, the result is around 21 GB of memory needed to accommodate such a design. This best case scenario was not measured.

Finally, our solution falls between the worse case and best case scenario. It is close to the worse case scenario in terms of cost. Taking again the same example of 20MB queue, then 40MB are going to be needed to allocate the additional feedback queue. In terms
of latency all figures show an improvement of around 70 to 80 percent on the latency. The results are denoted in Figures 4.19 to 4.26 with the tag solution. The measurements have been taken by applying the design described in Figure 4.2 and the algorithm in Table 4.2. An exception to the improvement mentioned here is Figure 4.21. In the case of 32*32 2D-meshes, the improvement is in the range of only 25 percent. If we go back to the Formula (4.1) in Section 4.4.1, we notice that the switch fabric latency has been added to the total latency over the route of the packet. This fabric latency under the specific case of two permanent hotspots is affected by the amount of updates sent by the RCT. For this reason, we should always keep in mind the computation time of the RCT with regards to propagating updates to all nodes in the network, especially in the case of large networks.

When injecting traffic with random address destination, the same 32*32 2D-mesh shows an improvement in latency close to 70 percent, in this case the hotspots are variable and live short through the cycle of the simulation. To conclude, we clearly measure and prove that the solution of the feedback queue generates high performance in terms of latency while keeping the cost of memory relatively low and close to the usage of one single FIFO queue.

Finally, we need to mention that in this simulation the order of packets was not a priority. The packets in the feedback queue could be sent out-of-order in order to staisfy the function of this algorithm. Since the main objective is to study the effect of the HOL and come up with an algorithm that could be a potential fix of the problem and, at the same time, a compromise between the expensive best case scenario and the worse case scenario. Additionally, in reality the queue in a network on chip environment is in the range of KB, in our simulation that has been enlarged to 20 MB in order to assume infinite queues. In all the examples described and interpreted in this section, we notice that our solution is very close to the best case scenario while remaining lower on cost with only 2 queues per input.

Figure 4.19: 4*4 2D-mesh comparison, worse case and solution are measured with 2 hotspots
CHAPTER 4. CONGESTION HANDLING WITH FEEDBACK QUEUE

8x8 comparison where worse case and solution are measured with 2 hotspots

16x16 comparison where worse case and solution are measured with 2 hotspots

Figure 4.20: 8*8 2D-mesh comparison, worse case and solution are measured with 2 hotspots

Figure 4.21: 16*16 2D-mesh comparison, worse case and solution are measured with 2 hotspots
4.4. SIMULATION RESULTS

Figure 4.22: 32*32 2D-mesh comparison, worse case and solution are measured with 2 hotspots

Figure 4.23: comparison 4*4 2D-mesh with random traffic
CHAPTER 4. CONGESTION HANDLING WITH FEEDBACK QUEUE

Figure 4.24: comparison 8x8 2D-mesh with random traffic

Figure 4.25: comparison 16x16 2D-mesh with random traffic
Figure 4.26: comparison 32x32 2D-mesh with random traffic
Conclusions and future work

5

5.1 Summary

This thesis handles an old and difficult problem congestion management, reaching the suggestion of an alternative idea. In chapter 1, the problem statement has been discussed is that scalable computer architectures SARC are evolving from a traditional system to the new system where the network interface contain hardware paths and functions for performing data transports over the off-chip network. The L1 queues are unable to differentiate between the L2 queues. When instructed to halt traffic, the only choice is stop traffic, they will stop all traffic towards the L2 queues. This phenomenon is called the HOL, Head of Line blocking, where packets ready to go get caught behind stopped ones in the L1 queues causing a non-desired delay.

In chapter 2, the Interprocessor Communication (IPC) was discussed. IPC consists of moving information among producers, consumers, and storage and informing threads of computation about the time when information or resources become available. Two styles of data placement and IPC data transport were discussed. The benefit of the Implicit Data Placement and Communication is that the application programmer does not have to worry about where data should reside. The downfall is in ability to optimize for the specific data transport patterns that occur in specific applications. The benefit of the Explicit Data Placement and Communication is scalable performance. The principle of the progressive address translation is that the packet will not carry the address of the final destination. The packet carries the virtual address and this is being translated by the physical address in the tables.

In chapter 3, the HOL (Head of Line) was discussed. The HOL effect is due to the fact that blocked packets destined to congested output switch ports prevent the advance of packets destined to other non-congested output ports. The switch architecture in on-chip networks is affected by the congestion in the network. Congestion spreads from the root to the leaves, due to the use of flow control. One of the main points discussed in this chapter is that tree saturation is eliminated by any strategy intended to reduce the effects of hot-spot contention should. The presence of hot spots should not adversely affect normal, non-hot-spot traffic and due to the memory contention, it is reasonable to expect hot-spot requests to suffer delays. Several strategies for handling hot-spot contention are proposed. They are divided into three classes: avoidance-based, prevention-based, and detection-based strategies. The main concern of avoidance-based strategies involves a different way of distributing data so as not to create hot spots in the first place. The main use of prevention-based strategies is that decisions are made to control message traffic at runtime to prevent tree saturation and Detection-based strategies is about allowing the formation of tree saturation, but the system detects such contention and invokes congestion-control mechanisms to control it. Different techniques of congestion...
elimination like the Avoidance based techniques, the prevention based techniques, and the reactive techniques. This chapter discusses about the HOL blocking elimination strategies like the elimination at network level and at switch level.

5.2 Thesis Contributions

The contributions of this thesis are:

1. We have designed and suggested an algorithm that manages congestion in scalable architectures: we have been able to suggest this algorithm, measure it and prove it. The same algorithm is viable in terms of cost while still beneficial in terms of performance. We have been able to prove that with an additional feedback queue and the control of 5 multiplexers, we manage to eliminate HOL blocking and improve latency by an average of 75 percent. The proposed algorithm achieves a performance close to the best case scenario while keeping the cost of FIFO queues close to the worse case scenario.

2. This thesis opens the way towards future work, where the expansion of the address range can be studied as an alternative or as an additional term of improvement.


Appendix A: The configuration files for the used tests

Below, the user can find a series of example configuration files that can be used with SIM in order to run the scenarios needed to generate the worse and best case.

```
# ../sim -l1000
Numswitches 1
Switch 0
Numinputs 4
Numoutputs 4
InputAction defaultInputAction
OutputAction defaultOutputAction
Algorithm fifo
  0 bernoulli_iid_uniform -u 0.65
  1 bernoulli_iid_uniform -u 0.65
  2 bernoulli_iid_uniform -u 0.65
  3 bernoulli_iid_uniform -u 0.65
Stats
Arrivals
Departures
Latency (*, 0)
Occupancy
Histograms
Arrivals
Departures
Latency
Occupancy
```

Figure 6.1: scenario 1
# ../sim -l1000
Numswitches 1
Switch 0
Numinputs 4
Numoutputs 4
InputAction defaultInputAction
OutputAction defaultOutputAction
Algorithm fifo
0 bernoulli_iid_uniform -u 0.70
1 bernoulli_iid_uniform -u 0.70
2 bernoulli_iid_uniform -u 0.70
3 bernoulli_iid_uniform -u 0.70
Stats
Arrivals
Departures
Latency (*, 0)
Occupancy
Histograms
Arrivals
Departures
Latency
Occupancy

Figure 6.2: scenario 2

# ../sim -l1000
Numswitches 1
Switch 0
Numinputs 8
Numoutputs 8
InputAction defaultInputAction
OutputAction defaultOutputAction
Algorithm fifo
0 bernoulli_iid_uniform -u 0.99
1 bernoulli_iid_uniform -u 0.99
2 bernoulli_iid_uniform -u 0.99
3 bernoulli_iid_uniform -u 0.99
4 bernoulli_iid_uniform -u 1
5 bernoulli_iid_uniform -u 0.99
6 bernoulli_iid_uniform -u 0.99
7 bernoulli_iid_uniform -u 0.99
Stats
Arrivals
Departures
Latency (*, 0)
Occupancy
Histograms
Arrivals
Departures
Latency
Occupancy

Figure 6.3: scenario 3
Figure 6.4: scenario 4
/usr/bin/sim -l1000, fifo.c has been changed with respect to the old version
Numswitches 1
Switch 0
Numinputs 16
Numoutputs 16
PriorityLevels 1
InputAction defaultInputAction
OutputAction defaultOutputAction
Algorithm fifo
0 bernoulli_iid_uniform -u 0.99
1 bernoulli_iid_uniform -u 0.99
2 bernoulli_iid_uniform -u 0.99
3 bernoulli_iid_uniform -u 0.99
4 bernoulli_iid_uniform -u 0.99
5 bernoulli_iid_uniform -u 0.99
6 bernoulli_iid_uniform -u 0.99
7 bernoulli_iid_uniform -u 0.99
8 bernoulli_iid_uniform -u 0.99
9 bernoulli_iid_uniform -u 0.99
10 bernoulli_iid_uniform -u 0.99
11 bernoulli_iid_uniform -u 0.99
12 bernoulli_iid_uniform -u 0.99
13 bernoulli_iid_uniform -u 0.99
14 bernoulli_iid_uniform -u 0.99
15 bernoulli_iid_uniform -u 0.99
Stats
Arrivals
Departures
Latency
Occupancy
Histograms
Arrivals
Departures
Latency
Occupancy

Figure 6.5: scenario 5
Numswitches 1
Switch 0
Numinputs 32
Numoutputs 32
PriorityLevels 1
InputAction defaultInputAction
OutputAction defaultOutputAction
Algorithm fifo
0 bernoulli iid uniform -u 0.99
1 bernoulli iid uniform -u 0.99
2 bernoulli iid uniform -u 0.99
3 bernoulli iid uniform -u 0.99
4 bernoulli iid uniform -u 0.99
5 bernoulli iid uniform -u 0.99
6 bernoulli iid uniform -u 0.99
7 bernoulli iid uniform -u 0.99
8 bernoulli iid uniform -u 0.99
9 bernoulli iid uniform -u 0.99
10 bernoulli iid uniform -u 0.99
11 bernoulli iid uniform -u 0.99
12 bernoulli iid uniform -u 0.99
13 bernoulli iid uniform -u 0.99
14 bernoulli iid uniform -u 0.99
15 bernoulli iid uniform -u 0.99
16 bernoulli iid uniform -u 0.99
17 bernoulli iid uniform -u 0.99
18 bernoulli iid uniform -u 0.99
19 bernoulli iid uniform -u 0.99
20 bernoulli iid uniform -u 0.99
21 bernoulli iid uniform -u 0.99
22 bernoulli iid uniform -u 0.99
23 bernoulli iid uniform -u 0.99
24 bernoulli iid uniform -u 0.99
25 bernoulli iid uniform -u 0.99
26 bernoulli iid uniform -u 0.99
27 bernoulli iid uniform -u 0.99
28 bernoulli iid uniform -u 0.99
29 bernoulli iid uniform -u 0.99
30 bernoulli iid uniform -u 0.99
31 bernoulli iid uniform -u 0.99
Stats
Arrivals
Departures
Latency
Occupancy
Histograms
Arrivals
Departures
Latency
Occupancy

Figure 6.6: scenario 6