CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Traffic isolation techniques for Networks-on-Chip

Master's thesis in Computer science and engineering

HANNES ERIKSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Traffic isolation techniques for Networks-on-Chip

HANNES ERIKSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Traffic Isolation Techniques for Networks-on-Chip

HANNES ERIKSON

© HANNES ERIKSON, 2022.

Supervisor: Ahsen Ejaz, Department of Computer Science and Engineering
Examiner: Ioannis Sourdis, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Traffic Isolation Techniques for Networks-on-Chip

HANNES ERIKSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As the number of cores available on modern multiprocessor Systems-on-chip increases, the traditional bus interconnection fails to provide enough scalability to handle the increased network load. To handle these shortcomings, an interconnection network, called Network-on-chip, can be used to provide better performance and scalability to the number of cores, supporting simultaneous transmission of multiple messages from different cores. However, there are some security vulnerabilities in this type of network. The network can be overloaded, potentially preventing critical applications to communicate properly, which can by achieved by an attacker performing a denial-of-service attack. Attackers can also potentially deduce the contents of network traffic based on fluctuations in response latencies, known as timing side-channel attacks. By isolating traffic flows, the potential impact of these problems can be reduced. This thesis presents a network-on-chip featuring three techniques that provide the user with tools to isolate traffic flows. The three techniques are (1) source throttling, (2) fixed virtual channel allocation per traffic flow, and (3) fixed timeslots for the switch allocator. Source throttling can be used to limit the traffic injection rate of problematic nodes. By statically allocating virtual channels to high-priority flows, packets belonging to these flows can be given contention-free access to resources of the NoC. Finally, schedulable switch allocator timeslots prevent malicious nodes from using timing information to find out when and what a node is transmitting. Through simulation, the different techniques' effectiveness in protecting against attacks is evaluated. The results show that source throttling can provide protection against denial-of-service attacks with few aggressor nodes but cannot protect against timing side-channel attacks. Fixed allocation of virtual channels effectively protects against denial of service attacks, even with many aggressor nodes, but does not provide protection against timing side-channel attacks. Separate switch allocator timeslots are not effective on their own, but by combining fixed virtual channel allocation with separate switch allocator timeslots, protection against timing side-channel attacks is shown to be possible.

Keywords: Computer, computer architecture, computer science, engineering, project, thesis, Network-on-Chip, System-on-Chip, Interconnection networks.

Acknowledgements

First, I would like to express my gratitude towards my supervisor Ahsen Ejaz for his invaluable help and support during the course of this project. I would also like to thank my examiner Ioannis Sourdis for all his useful input during the project. To my friends and family, thank you for your support. Completing the project would have been infinitely more difficult without all your help.

Hannes Erikson, Gothenburg, September 2022

Contents

List of Figures xi

List of Tables xv

1 Introduction 1
- 1.1 Problem . . . 2
- 1.2 Goals . . . 2
- 1.3 Approach . . . 3
  - 1.3.1 Source Throttling . . . 3
  - 1.3.2 Fixed Virtual Channel Allocation . . . 4
  - 1.3.3 Separate Switch Allocator Timeslots . . . 4
  - 1.3.4 Evaluation . . . 5
- 1.4 Related Work . . . 6
  - 1.4.1 Source Throttling . . . 6
  - 1.4.2 Separation of Flows . . . 6
  - 1.4.3 Timing Side-Channel Attacks . . . 6
- 1.5 Outline . . . 8

2 Background 9
- 2.1 Networks-on-Chip . . . 9
  - 2.1.1 Packets and Flits . . . 10
  - 2.1.2 Flow Control . . . 11
  - 2.1.3 Topology . . . 11
  - 2.1.4 Routing . . . 12
  - 2.1.5 Performance Metrics . . . 13
- 2.2 Router Architecture . . . 14
  - 2.2.1 Router Pipeline . . . 15
  - 2.2.2 Virtual Channel Allocator . . . 16
  - 2.2.3 Switch Allocator . . . 17
  - 2.2.4 Combined VC and Switch Allocation . . . 18
- 2.3 NoC Security Vulnerabilities . . . 18
  - 2.3.1 Denial-of-Service Attacks . . . 18
  - 2.3.2 Timing Side-Channel Attacks . . . 19
- 2.4 Countermeasures Against NoC Security Vulnerabilities . . . 20
  - 2.4.1 Throttling . . . 20
  - 2.4.2 Network Partitioning . . . 21

2.4.3 Traffic Management 22

3 Design **25**
3.1 Source Throttling 26
3.1.1 Stage One: Flit Counting 28
3.1.2 Stage Two: Budget Check 29
3.1.3 Throttling Implementation 29
3.1.4 Drawback of Proposed Approach 30
3.2 Fixed Virtual Channel Allocation to Traffic Flows 31
3.2.1 Approach 31
3.2.2 Cost of Implementation 33
3.3 Separate Switch Allocator Timeslots 34
3.3.1 Input Arbitration Masking 35
3.3.2 Output Arbitration Masking 36
3.3.3 Performance Implications 37
3.4 Design Summary 38

4 Evaluation **39**
4.1 Experimental Setup 39
4.1.1 Simulation Types 40
4.1.2 Isolation Configurations 40
4.2 Protection Against Denial-of-Service Attacks 41
4.2.1 Source Throttling 43
4.2.2 Fixed Virtual Channel Allocation 46
4.2.3 Separate Switch Allocator Timeslots 48
4.2.4 Denial-of-Service Protection Summary 49
4.3 Protection Against Timing Side-Channel Attacks 49
4.3.1 Source Throttling 50
4.3.2 Fixed Virtual Channel Allocation 54
4.3.3 Separate Switch Allocator Timeslots 54
4.3.4 Timing Side-Channel Protection Summary 57
4.4 Performance Impact of the Proposed Techniques 58
4.5 Summary 59

5 Discussion and Conclusion **61**
5.1 Source Throttling 61
5.2 Fixed Virtual Channel Allocation 63
5.3 Separate Switch Allocator Timeslots 64
5.4 Comparison with Related Works 66
5.5 Conclusion 67

A Appendix 1 **I**
A.1 Simulation Results Denial-of-Service Attack I
A.2 Simulation Results Timing Side-Channel Attack II

List of Figures

2.1 An interconnection network connecting several nodes. 9
2.2 Organisation of flits. Header flits carry routing information and therefore have a larger portion of the flit reserved for this information. This is called the header field of the flit. The body and tail flits only have flit type and VC ID in their header field. 10
2.3 A 4x4 mesh network of nodes with (x,y) naming. Each node can be connected to another node in up to four directions: North, East, South, West. 12
2.4 Example of allowed (green) and disallowed (red) turns using XY routing. Node (0,0) sends a packet to node (2,2). 13
2.5 The organisation of a NoC router. The main components are the input and output blocks, the crossbar (switch), the credit management units, virtual channel allocator (VA), switch allocator (SA), and Next Route Computation (NRC). The figure is adapted from [1]. 14
2.6 A router with the five possible flit directions in a mesh network marked out. The directions are North, East, South, West, and Resource/Local. Flits are injected to the network through the resource port. 15
2.7 The standard router pipeline, consisting of five stages. The link traversal stage is carried out between routers. 15
2.8 The Switch Allocation stage of the router pipeline. 16
2.9 A block diagram describing VA and SA stages of the router pipeline without any of the isolation techniques implemented. 17
2.10 The combined pipeline, where the NRC, VA, and SA stages are performed in parallel. 18

3.1 The block diagram of the input block, switch allocator, and virtual channel allocator of the router with the isolation techniques implemented. The added functionality for source throttling is shown in orange, fixed VC allocation in red, and the separate switch allocator timeslots in yellow. 27
3.2 The budget check extension to the switch allocator. This check is only added for the local port, meaning that all other ports are unaffected by this change. 31

3.3 The organisation of flits with the separate virtual channels extension, shown in red, increasing the size of the header field of head flits. Only the H and HT flits are changed. 31
3.4 Layout of the updated SA stage, allowing for separate VCs. 32
3.5 The Allowed VC computation stage. This operation filters out all VCs that are not allowed, not free, or do not have enough credits available. 33
3.6 The extended SA stage, allowing for separate switch allocator timeslots. 34
3.7 The Input Arbitration masking stage, performed after credit check and before input arbitration. 36
3.8 The Output Arbitration masking stage, performed after input arbitration. 37
3.9 The updated SA stage. Tables are read one cycle in advance, and results are placed in registers that are read during execution. 38

4.1 Block diagram of the simulation setup. The queue size signal is used for closed loop simulations. 40
4.2 Different configurations of the aggressor-victim traffic pattern. Node {0,1} (green) sends packets to node {2,2} (orange), but experiences congestion in the network due to interfering traffic from the red nodes, sending a large volume of requests to node {2,2}. 42
4.3 The latency increase of the victim node incurred by the denial-of-service attack. The average packet latency increases by 740% compared to the baseline. 43
4.4 The effect of source throttling using different budget (B) values in a closed-loop simulation with an unrestricted injection rate of 2.5 flits/node/ns using six aggressor nodes and one victim node (see Figure 4.2d). In this simulation E=2, T=32. 44
4.5 Effect of source throttling with one aggressor node and varying budget. As the budget decreases, so does the victim latency. 45
4.6 Comparison of the packet with latency using different configurations of fixed VC allocation using the aggressor-victim traffic pattern. . . . 46
4.7 Victim latencies during a denial-of-service attack using different amount of aggressor nodes. In these simulations, one VC is allocated to the victim and the rest of the network shares the other VCs. 47
4.8 Latency results for separate switch allocator timeslots along with fixed VC allocation with an aggressor injection rate of 0.375 flits/node/ns. The benchmark latency when isolation is disabled and the results for only using fixed VC allocation are included for reference. 49
4.9 The traffic pattern used for simulating a timing side-channel attack. The aggressor relies on the response time latency to deduce whether the victim is transmitting. 50
4.10 Victim and aggressor latency of the NoC when a timing side-channel attack is performed. The aggressor response time latency difference when the victim is transmitting clearly shows that victim traffic is present. 51

4.11 Comparison of aggressor response time latency under different budget values under a timing side-channel attack using an injection rate of 1.63 flits/node/ns. Victim injections included for visualisation purposes. The Extra budget was set to 2, and the epoch to 32 cycles. . . 51
4.12 The difference in average packet response latency and injected packets between B=26 and B=27, showing that this is the point when the network eventually saturates. The injection rate is 1.63 flits/node/ns, E=2, T=32. 52
4.13 The difference in average packet response time latency when the injection rate is reduced from 1.63 to 1.36 flits/node/ns. There is no victim traffic and the configuration is: B=26, E=2, and T=32. 53
4.14 Simulation results with an injection rate of 1.36 flits/node/ns, B=26, E=2, T=32. The difference between victim traffic and no victim traffic can be seen in the aggressor latency differences pictured in orange and grey. 53
4.15 The latency difference of the aggressors when allocated 1, 2, and 3 VCs using the fixed VC allocation technique. The victim has one VC by itself. The latency difference is noticed in all three cases when the victim transmits. 54
4.16 Aggressor packet latencies for the fixed VC allocation and separate switch allocator timeslots at different aggressor injection rates. 55
4.17 The difference in Response Time Latency (RTL) for the aggressor between allowing reuse of unused timeslots and not allowing it. A lower average packet latency is noted, but also a differing pattern when the victim is transmitting. 56
4.18 The difference in victim latency for different reusability schedules. By allowing only the victim to reuse timeslots a lower victim latency can be achieved. 57
4.19 The performance achieved by the standard and isolation-enabled NoCs with no isolation enabled. A throughput difference of under 0.5% and a latency difference of 3.47% is noticed for the NoC with isolation features at the saturation point of 1.3 flits/node/ns. 58

A.1 Denial-of-service attack simulation results with four aggressors and one victim. The budget varies while E=2, T=32 I
A.2 Denial-of-service attack simulation results with two aggressors and one victim. The budget varies while E=2, T=32. I
A.3 Simulation results for different injection rates using fixed VC allocation and separate allocator time slots, using the schedule described in section 4.3.3. The network saturates at 0.168 flits/node/ns, leading to a large increase in latency when that point is passed. II

List of Tables

2.1 Two packets of flits sent from source node (0,0) to two different destinations. The two packets will not use the same network resources. . 21

3.1 The parameters and their possible values for the source throttling technique. 27
3.2 Example of VC allocation using the separate virtual channel technique in a 4x4 NoC. Node (0,2) is the only node that can use VC number 2 in all routers, providing isolation of flows. 32
3.3 The input port table parameters for the separate switch allocator timeslots technique. 35
3.4 The output port table parameters for the separate switch allocator timeslots technique. Each entry is in the form of a vector of size N. . 35
3.5 The fields and their size for both the input and output port tables. . 35

4.1 The configurations used during the simulation showed in Figure 4.4. . 44
4.2 Comparison of average victim latency increase compared to the baseline using source throttling with a differing amount of aggressor nodes. 45
4.3 The latency decrease achieved when running the aggressor-victim traffic pattern in a system with 4 VCs per port with different amount of VCs allocated to the victim node. 46
4.4 The input allowed VCs (upper) and output allowed input port directions (lower) port tables used to provide isolation using fixed VC allocation and allocator timeslots. All timeslots are set to be reusable. The input port directions are **N**orth, **E**ast, **S**outh, **W**est, **R**esource, and **U** means Unreserved. 48
4.5 The schedule configured for testing the separate switch allocator timeslots technique, showing the allowed input ports for the south output port. The victim packets come from the west port, and the aggressor from the north port. N = North, W = West, E = East, R = Local Resource. 55
4.6 Response time latency differences for different injection latencies compared to the zero load latency. 56
4.7 The throughput and latency measurements when the injection rate is set to 1.3 flits/node/ns. 59

A.1 The aggressor latency with one aggressor and different throttling budgets. For all simulations, E=2, T=32. II

Glossary

Denial-of-Service	A denial of service, DoS, attack aims to flood specific parts of a network with traffic, causing it to become slower or stop functioning correctly..
Flit	A flow control unit, flit, is the data-carrying entity of the network. Transmission of data in a Network-on-Chip is done on a flit level..
Network-on-Chip	Network-on-chip, or NoC, is a term that refers to the interconnection network connecting resources in a System-on-chip, SoC.
System-on-Chip	System-on-chip, SoC, refers to an integrated circuit that contains components of a computer system such as CPU and memory on one chip.
Virtual Channel	Virtual Channels is a technique which is used to improve performance of NoCs by allowing several virtual channels for each physical channel in the network. The virtual channels take turns accessing the physical channel.

1
Introduction

In a modern multiprocessor System-on-Chip (SoC), the many different nodes such as processor cores, memories, and caches, can be connected by a shared network called Network-on-Chip (NoC) [2]. Since the NoC is a shared resource, packets in the network can encounter contention. This contention can expose weaknesses in the system, both in terms of performance degradation and security vulnerabilities.

Regarding the performance implications, increased congestion might lead to lower throughput and increased latency for packets in the network as they have to wait for resources to become available. This can both be due to several nodes sending data to the same node as well as a single node sending excessive amounts of data, flooding the network.

The increased congestion can also lead to problems regarding quality of service and fairness within the network [3]. Congestion might lead to packets not arriving at the destination within the required time, which can lead to problems in systems that have real-time requirements. To combat this, techniques for ensuring quality of service can be used. Nodes further away from the destination could also experience higher latency due to the sent packets having to contest for shared network resources at multiple points along their path.

From the security point of view, denial-of-service (DoS) attacks can cause performance degradation and potentially cause system to stop functioning properly [4]. An attacker also could use compromised or malicious code to perform timing side-channel attacks, possibly inferring sensitive information about the communicating nodes by observing patterns in their communication [5]. This can for instance be done by using the aforementioned congestion at shared links as a timing side-channel, inferring data by analysing the throughput or latency of its response through the network.

1.1 Problem

The challenges described above can become problematic for systems in which security is important or in cases where there are hard real-time constraints for critical operations. To this end, keeping the traffic of certain nodes separated in the network may provide the desired properties in order to alleviate these problems in terms of security and performance predictability.

A possible way to combat the aforementioned issues is through isolation of traffic flows within the network. By isolating flows, they would not compete for resources at the same time, providing protection against denial-of-service and timing side-channel attacks, and increasing performance predictability. Isolation of flows can be implemented using a Weighted Fair Queuing (WFQ) scheduling scheme [6], but this approach suffers from high requirements in terms of buffer size and complexity of scheduling computations [7].

Another approach for traffic flow isolation is through replication of resources such as links and routers. This would remove any interference between messages from different nodes, but having multiple copies of hardware would be costly in terms of both area and power consumption. There is, therefore, an opportunity to create a high-performance NoC which can provide the strengths of physically isolated links together with high throughput and comparatively low area overhead.

With regards to this, combining replication of some network components with several techniques for security and performance isolation can deliver a high-performance NoC with quality of service and dependability features. This thesis aims to explore several possible designs of such a NoC, and compare them through simulation-based evaluation.

1.2 Goals

The goal of the project is to explore possible designs of a NoC which ensure isolation of flows, with focus on security and performance predictability aspects. The challenges lie in designing a network which is optimised for both of the aforementioned parameters. The main scientific goals of the project are therefore:

- Implement techniques for isolation of flows in a Network-on-Chip.
- Evaluate the efficacy of the implemented traffic isolation techniques when used to counteract denial-of-service and timing side-channel attacks.

To this end, three different isolation techniques were designed and implemented. Each of the techniques were evaluated and compared using relevant performance metrics. A baseline implementation was also used, which the subsequent designs build upon. The three techniques are:

1. **Source Throttling**. By throttling traffic sources that overload the network the level of congestion can be decreased. This is done by counting the number of flits that are injected into the network and disallowing further injection once a quota is met. This increases the complexity of the router, which can negatively impact area and power consumption, but provides a less congested

network and prevents nodes from overloading the network with excessive traffic. This approach aims to decrease the effectiveness of denial of service attacks, providing security and increasing the performance predictability of the attacked node.

2. **Fixed Virtual Channel Allocation**. Input virtual channels in each NoC router can be statically allocated to specific source nodes for isolation of flows in the network during virtual channel allocation. This approach allows for separation of flows as the virtual channels are divided among the sources, but can decrease performance of the network as less network resources are available for each specific source node.
3. **Separate Switch Allocator Timeslots**. Temporal separation of access to the shared crossbar allows for isolation between flows during switch allocation. This is achieved through a configurable access schedule. Through this technique, flits originating from nodes that should not interfere do not compete for switch arbitration in the same cycle. This design requires fixed virtual channel allocation and a schedule for the switch allocation stage to be effective.

1.3 Approach

The three techniques, source throttling, fixed virtual channel allocation, and separate switch allocator timeslots are implemented independently of each other and can be used either independently or in combination. This way, the system designer can decide what isolation properties are desired for the specific use case.

1.3.1 Source Throttling

Source throttling decreases network congestion and can therefore prevent nodes from flooding the network with excessive requests, for example during a denial-of-service attack. This is especially important to counteract in the case of a low priority or malicious node taking up crucial network resources, preventing critical flows from accessing the resources. Furthermore, as the latency variations caused by sudden bursts of requests in the network is decreased the reliability of the system increases. Two problems that need to be solved in this approach are knowing when to throttle and how much to throttle.

Since congestion means that network resources are taken up by one or several sources, counting the available resources and making sure that they are below a certain threshold, before allowing a source to send more packets, allows throttling of a specific source. Baydal, et al. presents a solution that counts the number of "free useful virtual channels", and requires a sufficient amount of available virtual channels before allowing injection into the network [4]. A similar solution with a counter that keeps track of how many packets or flits the source has injected into the network in a given time period is used in this project.

The second problem is how much to throttle. Being too aggressive with the throttling can lead to degraded performance, since blocking too many requests can lead to under-utilisation of the network. We tackle this problem by using a static throttling

quota, and empirically test different configurations. This reduces the complexity of the design while still providing high configurability. Using adaptive network-load-aware throttling [8] could provide a better performance for the throttled system as the network can adapt to the current traffic situation, but is also more complex to implement as opposed to a static technique.

Our approach to implementing source throttling in our system can therefore be described in the following way:

- Measure traffic intensity from every source node. Count the number of injected flits per given time-period and throttle once a threshold is reached.
- A static throttling quota is used to reduce the complexity of the design.

The potential drawbacks of this approach are the performance degradation due to exceedingly aggressive or loose throttling combined with the added overhead of the flit-counting mechanism. This approach also impacts area and power consumption of the system as extra logic is added to the routers.

1.3.2 Fixed Virtual Channel Allocation

By having separate allocated virtual channels for different flows, isolation between flows in the virtual channel allocation stage is ensured through separation in space. As VCs are allocated to separate sets of source nodes, determining how many VCs can be allocated for each priority level is a challenge. Completely separating traffic flows by statically allocating VCs reduces the variations in performance caused by network congestion, thus providing security guarantees in terms of protection against denial-of-service and timing side-channel attacks. However, the number of available virtual channels is limited, meaning that compromises in the allocation of the virtual channels have to be made.

To decide which packets should be allowed to use which virtual channel, a table that defines which source nodes should be able to use what virtual channels is implemented. The information regarding the allowed virtual channels is stored at the head of each transmitted packet, informing the virtual channel allocator what virtual channels are allowed for this specific source node.

The following approach is therefore used to enable fixed virtual channel allocation:

- Assign certain virtual channels to critical sources that allow them to utilise the VCs without interference from other sources. The information regarding allowed VCs should be included in the head flit of each packet.
- Mask virtual channel allocation requests based on the allowed VC information at the head of each packet.

1.3.3 Separate Switch Allocator Timeslots

Separate switch allocator timeslots can be used to ensure that each VC in a router is only allowed to forward a flit through a crossbar during its pre-allocated timeslot. This decreases the effectiveness of timing channel attacks, due to an inability to measure the latency of the requests. This functionality can be implemented through a schedule where timeslots can be reserved for certain source nodes' requests, sepa-

rating access of the shared resources in time.

One of the problems with this approach is regarding how to synchronise the timeslots between the different routers. In order to not impact performance too much, the timeslot schedule needs to consider how flits pass through the network routers in sequence. This pattern can be likened to a traffic light system, whereby scheduling the on and off times, good traffic flow can be ensured. To support this, a global counter is used.

This type of allocation can limit how fast a network can process flits from different sources, since in the worst case flits might have to wait $N - 1$ time-slots for their turn, where N is the number of unique timeslots. The impact of this problem would however be lowered if flits are allowed during several timeslots, as well as by only reserving some of the timeslots for the critical or high-security flows.

By reserving all timeslots, performance predictability is ensured, as the traffic flow is defined before the system starts. This also provides security guarantees in terms of protection against attacks aiming to deduce information based on fluctuations in latency and in terms of protecting against denial-of-service attacks.

The following approach is therefore used to enable time separation of access to the switch allocator:

- Create a schedule which defines what virtual channels and input ports should be allowed to utilise the switch at each cycle.
- Mask switch allocation requests based on the schedule information.

1.3.4 Evaluation

The main performance metrics that are relevant for this project and are listed below. Using these metrics allows for comparing the designs from several points-of-view, showcasing the strengths and weaknesses of each of them. Comparing to a standard benchmark also allows us to draw conclusions about if the suggested designs provide a better system compared to the benchmark.

- Performance-oriented
  - Throughput
  - Latency
- Security
  - Protection against Denial-of-Service attacks
  - Protection against Timing Side-Channel attacks

In order to accurately test the performance of the system and how well it performs from a security point of view, several different traffic models consisting of varying flows should be tested. Two general approaches to this problem are using either application-driven workloads or synthetic workloads [1]. While application-driven workloads would be ideal, a more feasible approach is to use synthetic workloads for testing, as these can be more easily designed and tuned. However, approaches for creating more realistic synthetic flows do exist [9], [10].

We use a combination of different synthetic workloads when testing the system. These workloads are used to simulate denial-of-service attacks, timing side-channel

attacks, as well as uniform-random traffic for measuring the network performance. The experimental setup and workloads are presented in detail in section 4.1.

1.4 Related Work

There exist several different approaches on how to implement some of the techniques presented in this thesis. This section will summarise the most relevant ones to this project.

1.4.1 Source Throttling

Baydal, et al. have proposed and injection limitation mechanism for NoCs [4]. This mechanism implements a throttling of requests when the network approaches a point of saturation. Their proposed mechanism operates in the routing control unit and counts the number of free output virtual channels in the direction that the flit is travelling. This is done because it does not matter if output virtual channels in a direction that is not going to be used are not available. Conversely, it does not matter if most output virtual channels are free, if the ones that are going to be used are full.

To decide whether or not to pause injection of requests, the number of useful output virtual channels in the source router is compared to a fixed threshold value, and when the number of free VCs fall below that value, injection requests are rejected.

1.4.2 Separation of Flows

In the QNoC architecture Bolotin, et al. introduce service levels for different classes of flits [11]. Four flit classes are introduced, with different priorities. The flits of different service levels are separated by different buffers, and higher-priority flits can preempt lower priority flits. The service levels are stored in control wires. The ideas of this kind of separation have inspired the separate virtual channels and separate switch allocator timeslots techniques, where tables containing the allowed VCs and allocated timeslots, which are then either stored in the flits themselves or in registers that are read by the switch allocator.

1.4.3 Timing Side-Channel Attacks

Wang and Suh have proposed two techniques for handling timing channel problems in NoCs, Spatial Network Partitioning (SPN), and Temporal Network Partitioning (TPN) [5]. The SPN approach divides the NoC into different security domains, each given a subset of the available cores. In TPN VCs are instead statically allocated to different security domains.

The security domains can also be configured so that each of the domains only have access to the switch allocation and link traversal router pipeline stages in every other cycle. The TPN approach bears similarities both with our proposed separation of virtual channels, as well as with the separate switch allocator timeslots. However,

wheres TPN divides the network into two security domains, we opt for a configurable approach where each flow can be given complete isolation from other flows if required.

SurfNoC, presented by Wassel, et al. [12], implements separate timeslots for communication between different nodes by dividing flows into classes and implementing a schedule for when the different are allowed to use the network resources. The result is likened to a wave, where flits in a certain class wait for the "wave", allowing them to travel between the nodes when the wave arrives. The schedule is repeating and allows for isolation between flit classes while still sharing network resources. The idea of a repeating schedule defining when certain flows are allowed to use the network resources is implemented in our approach for separate switch allocator timeslots. However, we differentiate between flit directions instead of classes in our approach.

1.5 Outline

The thesis is organised as follows:

Chapter 1 introduces the thesis project, topic, and related work.

Chapter 2 presents the background and theory relevant for the understanding of the thesis topic.

Chapter 3 introduces the three techniques for enabling isolation in the NoC.

Chapter 4 presents the results of the evaluation of the NoC, carried out through simulation.

Chapter 5 concludes the thesis by summarising the key findings, discussing the advantages and drawbacks of the proposed NoC and outlining possible future work.

2

Background

This chapter presents the basic concepts and relevant theory within the subject of Networks-on-Chip. This allows the subsequent chapters to discuss the related theory in greater depth.
In this chapter, an introduction to the network organisation, the NoC router, and related concepts are presented. This includes the organisation of network messages, topology, router organisation and pipeline, as well as techniques for increasing the performance of the router. Basic flow control and routing theory is also presented. The chapter concludes with an overview of the different attack patterns that the presented techniques will be evaluated against.

2.1 Networks-on-Chip

Interconnection networks can be found in most digital systems containing at least two components, where the interconnection network is "a programmable system that transports data between terminals" [1]. These components can be processor cores, memories, I/O devices, and more. Figure 2.1 illustrates an interconnection network connection several components. The interconnection network itself can be organised in a number of ways, ranging from buses to complex multi-dimensional networks.

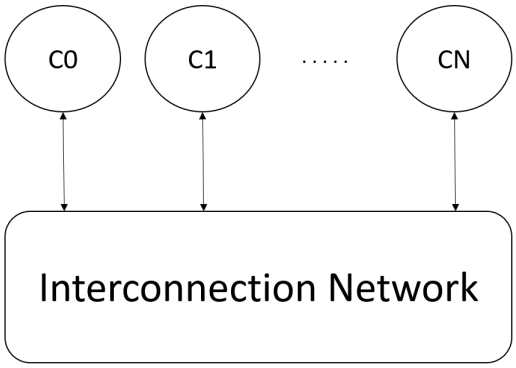


Figure 2.1: An interconnection network connecting several nodes.

As late as the 1980s, buses were the most common type of interconnection networks used [1]. However, as the number of components in the network increased, the drawbacks of the bus became apparent. One of the main drawbacks of the bus is that only one source can transmit on the bus in each clock cycle. Therefore, as the number of sources increase, the bus utilisation increases as well.

Nowadays, interconnection networks often consist of point-to-point networks, which allow concurrency and potentially higher performance than bus-based networks [1]. These networks are used to send messages between the different components, hereafter referred to as nodes, in the network.

2.1.1 Packets and Flits

In the network, messages are sent in the form of *packets*. In turn, each packet can contain one or more flow control units, *flits*. The data is then carried in these flits. There are four different types of flits considered in this work.

- **Head flits (H)**, which carry routing information such as source and destination nodes, virtual channel ID, and next route information, as well as a data payload. The head flit is responsible for allocating necessary router resources, meaning that all packets need to contain a head flit.
- **Body flits (B)**, which always follow a head flit or another body flit, contain the virtual channel ID and a data payload. The amount of body flits in a packet are decided by the amount of data that needs to be transferred. Therefore, not all packets contain a body flit.
- **Tail flits (T)**, which follow a body or head flit, contain virtual channel ID and a data payload. The tail flits deallocate the resources that the head flit has allocated when it passes through the routers. If the packet length is longer than one, the packet must contain a tail flit.
- **Head-Tail flits (HT)**, represents a single flit packet. It is used when only a small amount of data needs to be transferred, meaning that one flit is enough to store all the necessary data. This flit both allocates and deallocates the router resources.

The flits are divided into the organisation seen in Figure 2.2. The *header* part of the flits is larger for Head or Head-Tail flits, as they are responsible for allocating network resources and therefore require routing information such as source and destination. Body and tail flits only need to follow the head flits, and therefore only require their flit type and the VC ID.

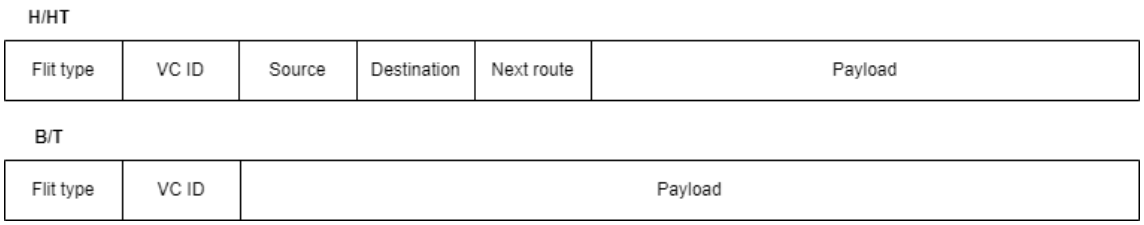


Figure 2.2: Organisation of flits. Header flits carry routing information and therefore have a larger portion of the flit reserved for this information. This is called the header field of the flit. The body and tail flits only have flit type and VC ID in their header field.

2.1.2 Flow Control

The flow control of the network determines the allocation of the resources in the network [1]. A good flow control method should result in the network operating close to the maximum performance possible, while a worse method might result in worse performance. The goal of the flow control method is therefore to facilitate the usage of the network's resources in an efficient manner.

The resources that must be allocated in the network include physical channel bandwidth, buffer space, and control state [1]. Packets are allocated a control state, and the head flit of the packet contains the necessary information for routing the packet to it's destination. The subsequent body and tail flits follow the head flit, using the resources allocated by the head flit. The tail flit then deallocates the network resources.

One common technique for flow control in NoCs is wormhole flow control. With wormhole flow control, the allocation of network resources such as buffers and channel bandwidth is done at flit level [13]. One problem with this mode of control is that if the transmission of a packet blocks for some reason, all other packets following this packet will also block. This means that packets get blocked while queuing in the buffer, which is called Head-of-Line blocking [13]. By using virtual channels, packets that would otherwise be blocked can use the otherwise idle channels.

The virtual channels technique divides the physical channels into several virtual channels [1]. Through this, a packet that would otherwise be blocked by another packet can utilise its virtual channel to still be allowed to pass through the router. In the NoC considered in this thesis, both the input and output ports have several virtual channels per port.

Furthermore, credit-based flow control is used together with the virtual channels in the NoC considered in this thesis. With credit-based flow control each router keeps track of the number of free buffer slots in each of the virtual channels of downstream routers using a *credit count* [1].

Each time a flit is sent the credit count is decremented and if the counter reaches zero no more flits can be transmitted until the credit count has once again increased. Each time a flit has left the virtual channel at the downstream router a credit is sent back to the upstream router, indicating that there is now another empty slot at the virtual channel of the downstream router. The credit count is then incremented at the upstream router. This means that flits are only allowed to be sent from the upstream router if there are free buffer slots available at a virtual channel of the downstream router.

2.1.3 Topology

The interconnection network is comprised of several nodes, as described in figure 2.1, and can be laid out in several different topologies. Some of the more common topologies include *ring*, *mesh*, and *torus*. When deciding what type of topology to use there are several factors to include in the decision, most importantly *performance* and *area cost* [1].

This work will consider only the two-dimensional mesh topology, depicted in Figure

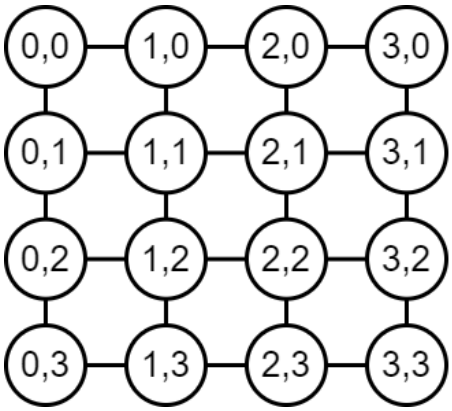


Figure 2.3: A 4x4 mesh network of nodes with (x,y) naming. Each node can be connected to another node in up to four directions: North, East, South, West.

2.3. In a two-dimensional mesh each node is named by their position in the network, in the form of (x,y). For each node, five possible turns are possible for an incoming flit. These are north, east, west, south, and local/resource.

Travelling in the north or south direction means travelling along the y-axis of the mesh, while east and west represents the x-axis. The local port is used to either transmit, inject a flit into the network, or receive, remove a flit from the network. A flit can not skip a node on its route, meaning that to get from node (0,0) to (3,0), all nodes along the x-axis have to be passed through on the way to the destination.

2.1.4 Routing

Routing packets is the act of selecting a path from the source to the destination in the network [1]. Ideally, a routing algorithm should be able to balance the load of the network such that the traffic has no impact on the throughput of the network.

There are two main types of routing algorithms: *Oblivious*, and *Adaptive* [1]. Oblivious routing algorithms do not factor in any outer information when calculating a route through the network. Deterministic routing is an oblivious routing algorithm which always chooses the same path between two nodes, regardless of traffic and other factors that might affect the network. The advantage of these algorithms is that they are easily implemented and avoid deadlock. Due to this they can however lead to sub-optimal load balancing. Adaptive routing algorithms on the other hand use information such as traffic and network state and history when calculating a path. These algorithms can provide better routing decisions, but are also more complex.

A routing algorithm can also be minimal or non-minimal [1]. A minimal route is the shortest path from point A to B. If another path were to be taken, the route is non-minimal. While using minimal routing provides low delay in a network with low contention, non-minimal adaptive routing could reduce overall latency in a highly congested network.

The dimension ordered XY routing algorithm is a deterministic routing algorithm for 2D meshes [14]. In XY routing, the routers in a mesh network are named (x,

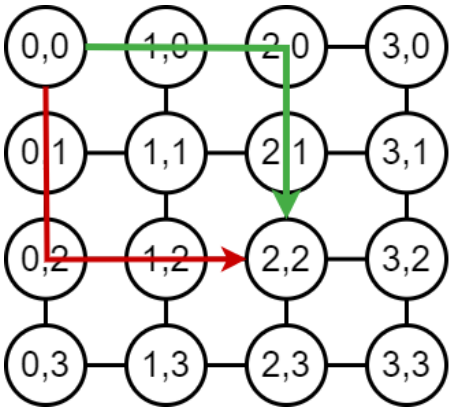


Figure 2.4: Example of allowed (green) and disallowed (red) turns using XY routing. Node (0,0) sends a packet to node (2,2).

y) based on their horizontal (x) and vertical (y) placement within the mesh, as can be seen in Figure 2.3. At each router, the algorithm compares the current location (C_x, C_y) to the destination (D_x, Dy). When $C_x = D_x$ & $C_y = D_y$ we have arrived at the destination. The algorithm first considers the x direction, routing the packet along the x-axis until $C_x = D_x$. Then, the same procedure is repeated along the y-axis until the packet arrives at the destination. Figure 2.4 shows an example of allowed and disallowed turns using this routing algorithm.

An added benefit of the XY routing algorithm is that the route computation can be carried out one network hop in advance. Using the XY dimension ordered routing algorithm, the information needed to determine the direction at the next router is the identity of that router, (x,y), and the destination identity (x', y'). Once the direction of the upcoming hop is known, the identity of the next router can be determined as either (x±1, y), or (x, y±1). Combining this information with the direction of the coming hop allows the algorithm to determine the route one router in advance.

This technique removes the dependency between route computation and switch allocation request, since when the flit arrives at the new router, the router computation stage is already completed. This technique is called *Next Route Computation*, NRC.

2.1.5 Performance Metrics

Throughput and *latency* are two metrics that can be used to measure the performance of a network [1]. Throughput describes the data rate of the network, how much traffic that can travel through the network in a given time period and is measured in bits per second. The throughput of the network is dependent on several factors including the topology, routing, flow control and is expressed in terms of the whole network.

Latency describes the amount of time needed for a packet to pass through the network. The latency of a packet in the network can be divided into three categories, head latency, serialisation latency and contention latency [1]. This gives us a total

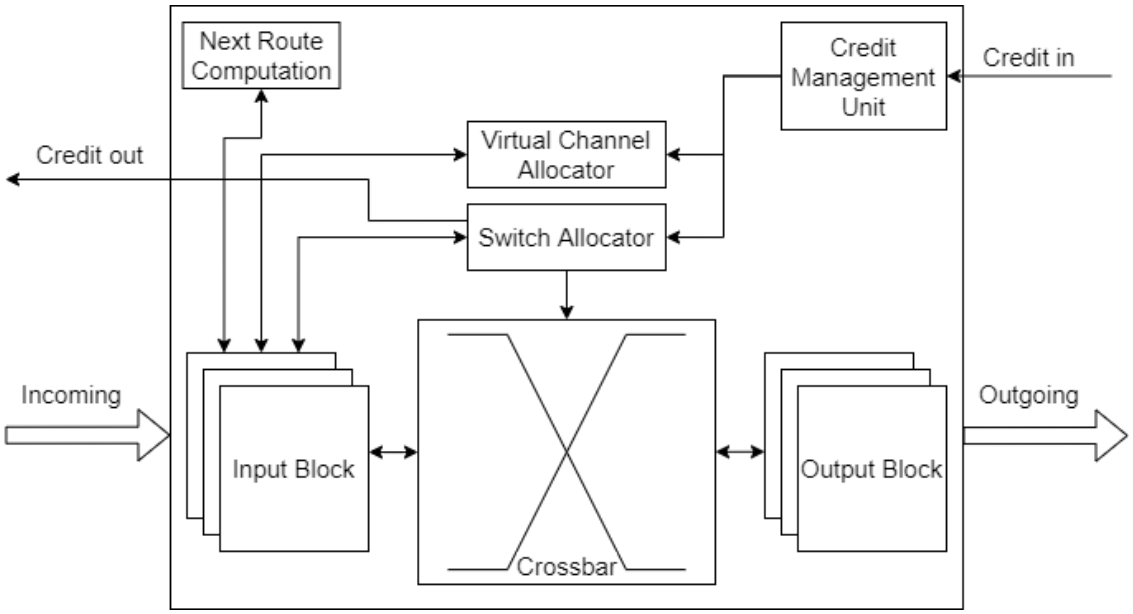


Figure 2.5: The organisation of a NoC router. The main components are the input and output blocks, the crossbar (switch), the credit management units, virtual channel allocator (VA), switch allocator (SA), and Next Route Computation (NRC). The figure is adapted from [1].

latency of

$$T = T_h + T_s + T_c.$$

The head latency is the time required for the head flit to travel through the network, including time spent in routers and time spent on the physical links. This is affected by the choice of topology and router configuration. The serialisation latency is the time required for the rest of the packet to follow the head flit. The contention latency on the other hand depends on the characteristic of the competing traffic.

2.2 Router Architecture

The router in a NoC in general consists of input units, a route computation unit, virtual channel and switch allocators, a switch, and output units. An example figure of a router can be seen in Figure 2.5. The packets arriving at the router first enter into the input block of its input port and a route is computed, which determines the output port of the packet. Then, a Virtual Channel is reserved in the Virtual Channel Allocation (VA) stage. After a VC has been reserved, the Switch Allocation (SA) stage arranges a crossbar connection between input and output blocks, which the individual flits then use to travel through to the selected output block.

In a mesh topology there are in general five possible directions for a flit to take: *North*, *East*, *South*, *West*, and *Local/Resource*, illustrated in Figure 2.6. This means that for an incoming flit from incoming direction D, there are four possible outgoing directions. The direction of an incoming head flit is calculated by the routing unit, based on the destination field in the header of the flit.

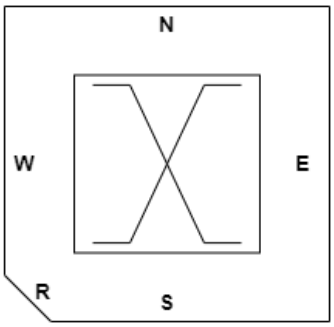


Figure 2.6: A router with the five possible flit directions in a mesh network marked out. The directions are North, East, South, West, and Resource/Local. Flits are injected to the network through the resource port.

2.2.1 Router Pipeline

The process of traversing a router and continuing through the network is done in a pipeline consisting of several steps. The general pipeline has five stages, which can be seen in Figure 2.10. This section will briefly describe the different stages. The virtual channel allocator and switch allocator implementations will be described more thoroughly in Sections 2.2.2 and 2.2.3.

- Next Route Computation, **NRC**
- Virtual Channel Allocation, **VA**
- Switch Allocation, **SA**
- Switch Traversal, **ST**
- Link Traversal, **LT**

Figure 2.7: The standard router pipeline, consisting of five stages. The link traversal stage is carried out between routers.

During Next Route Computation, information in the header flit of a packet is used to select the appropriate output port based on the destination of the packet, gathered from the header-section of the head flit (see Figure 2.2) [1]. The result of the routing computation is placed in the route field of the virtual channel state. As described in Section 2.1.4 this is can be done one hop in advance in our case, meaning that the result is already available once the flit arrives at the router.

Using this information, a VC request is sent to the VA stage, requesting an output virtual channel at the desired output port. The VA stage keeps track of reserved (used) VCs, and reserves new ones to incoming requests. Therefore, incoming VA requests are combined with the current available VCs, creating an arbitration request.

VC arbitration selects one among V virtual channels and is done for each input port. A grant signal is generated to winning requests, and the list of reserved VCs is updated. The granted VC ID is output to the input block.

In the next step, the head flit enters switch allocation. The SA stage consists of several steps, shown in Figure 2.8. In the first credit check stage credits at the output virtual channel are checked. In order to be able to send the flits from the input VC to the output VC there needs to be at least one credit (available buffer slot) at the output VC.

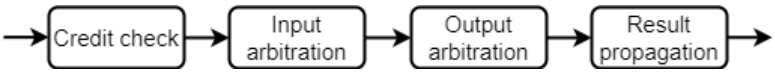


Figure 2.8: The Switch Allocation stage of the router pipeline.

The arbitration is then performed in two steps, input and output arbitration. The input arbitration selects one out of V input virtual channels for each of the P ports. The output arbitration then selects one out of the input port requests. A grant is given to the requests that win both input and output arbitration.

In the final SA stage, grants are propagated to the input and output ports and crossbar selection signals are generated. The flit is now allowed to traverse the switch.

The last stage of the router pipeline is the switch traversal, which includes the physical transmission of signals, placing the information from the input VC into the output VC associated with the requested output port.

2.2.2 Virtual Channel Allocator

The virtual channel allocator performs the virtual channel allocation pipeline stage, VA, which is responsible for selecting an available virtual channel to the head flits that win switch allocation. This process is done in parallel to next route computation and switch allocation. As previously described, the head flit is assigned the specific virtual channel ID, which is then passed on to body and tail flits in the packet. A block diagram of the VA and SA stages can be seen in Figure 2.9.

To keep track of the available VCs, a tracker containing information about the status of each of the VCs is used. This tracker gets information regarding the credit status for each of the VCs along with information about which VCs are reserved. The VC status is set to available for VCs that are both unreserved and have available credits.

The VA stage receives VA-requests from the input port and informs the SA stage which output ports have available VCs. SA only has to consider ports with available output VCs, meaning that output ports without available VCs are discarded by the SA stage.

If the network is not congested, more than one VC may be available at the output port. Therefore, V:1 arbitration is performed among the V available virtual channels using a fixed priority arbiter [15]. The winning VC is forwarded to the SA stage. Here, the SA stage reserves the winning VC for the winning output port, sending a reserve signal back to the VC tracker. The selected VC is now not available for new incoming flits until it has been released.

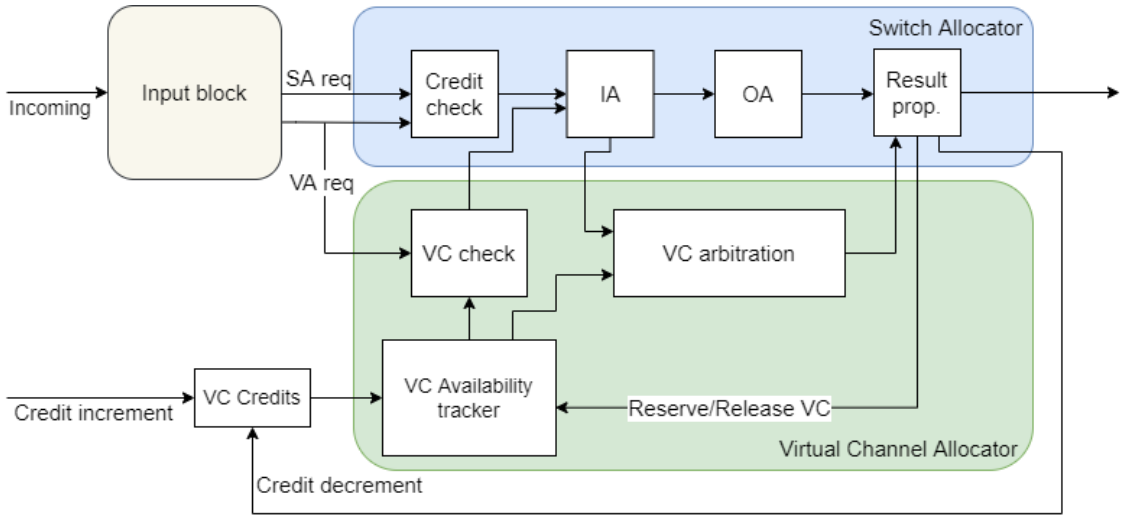


Figure 2.9: A block diagram describing VA and SA stages of the router pipeline without any of the isolation techniques implemented.

2.2.3 Switch Allocator

The switch allocation pipeline stage is done in the switch allocator, which grants requesting flits access to the crossbar connecting input and output ports. After passing through the router, the winning flits also get access to the link connecting the current router to the next router, allowing the packet of the winning head flit to progress through the pipeline and go into the next router.

The switch allocator receives its requests from the input block, where the flits will be considered for switch allocation. During credit check the head flits with available output port VCs (information sent from the virtual channel allocator), are considered for arbitration. Head flits without available VCs are discarded. The credit check also filters out body and tail flits in the case of their allocated VC not having enough credits to allow transmission.

The requests that are allowed through go through two separate stages of arbitration using input-first separable allocation, as described by Becker and Dally [16]. In the first arbitration step, input arbitration (IA), a V:1 round-robin priority arbitration among the V virtual channels is performed. Up to one VC per input port is granted, allowing that input port to contest for output arbitration. Following the input arbitration, output arbitration (OA), is performed. The output arbitration is a round-robin priority arbitration of (P-1):1, where P is the number of ports of the router. Since a router cannot transmit messages to itself, only the P-1 remaining ports have to be considered for OA.

After IA and OA are completed, the winning requests are propagated to necessary parts of the router, alongside control signals. These control signals allocation information to the VC tracker, VC reservation or release, credit management, and crossbar signals corresponding to the winning VC and input output port pair. Subsequently, switch traversal can be performed in the next cycle.

2.2.4 Combined VC and Switch Allocation

The traditional router pipeline consists of five stages. Since the latency of the network is related to the number of stages in the pipeline, reducing the number of stages can decrease latency, given that the latency of each stage does not increase [1]. By combining the NRC, VA, and SA stages, these can be performed in parallel. The next route computation determines the output port of the flit for the next router, which can be calculated at the same time as simultaneously requesting both an output virtual channel and access to the corresponding output port.

If successful, this reduces the pipeline by two steps, lowering latency. The combined VA and SA stages grants a free output VC to head flits that win switch allocation in the same cycle [17]. Doing these two operations in one cycle reduces the depth of the pipeline by one cycle, saving one cycle per hop in the network. By combining VC allocation and switch allocation using this method, the header flit that wins SA is granted a free output VC immediately [17]. This saves a clock cycle per network hop by combining the VA and SA pipeline stages.

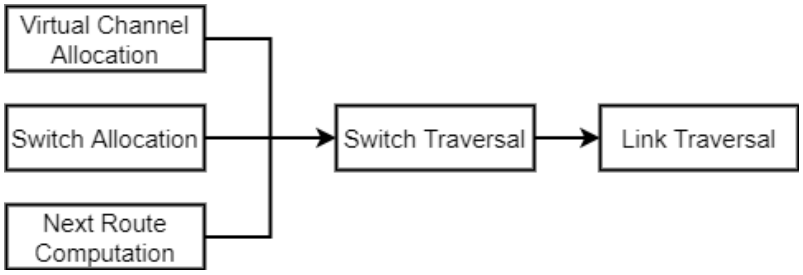


Figure 2.10: The combined pipeline, where the NRC, VA, and SA stages are performed in parallel.

2.3 NoC Security Vulnerabilities

As the number of nodes such as processor cores, memories, and caches on a SoC continues to increase, so does the potential traffic patterns and attacks that can disrupt the functionality of the NoC. This section will present two of these potential disruptions and available remedies against them.

2.3.1 Denial-of-Service Attacks

When the traffic in a network is approaching the saturation point, severe performance degradation can be observed [4]. As the number of requests in a network increase, the contention of the shared network resources such as links and switches also increase. Additionally, in cases where traffic is non-uniform the load on certain parts of the network can increase, causing further congestion [18]. Strategies to avoid congestion include using network status to design routing algorithms [18], limiting the rate of injection into the network once a saturation point is reached [4], and adaptive routing strategies, avoiding congested parts of the network [1].

A type of attack which attempts to reduce the capacity of a network is denial-of-service (DoS) attacks [19]. The attack is performed by sending large volumes of requests to a destination node such as a website, with the purpose of overloading the network, causing performance degradation. In terms of Networks-on-Chip, this type of attack can be performed by sending large volumes of packets to a node, which can decrease throughput and increase latency at the point of congestion. In a robustness analysis for mesh networks under DoS attacks performed by Fang, et al., [20], several guidelines to reducing the effectiveness of DoS-attacks are presented. These include a defence mechanism that detects unusual behaviour, the usage of appropriate routing algorithms, starvation avoidance, and output selection based on neighbouring nodes' information. Furthermore, the deterministic XY routing algorithm is shown to perform better than adaptive routing algorithms for moderate attack rates, while the adaptive algorithms perform better under higher attack rates.

2.3.2 Timing Side-Channel Attacks

Shared resources can lead to different nodes affecting the timing of applications running on other nodes due to interference the shared network resources [5]. Examples of such resources are links, buffers, and switches. Through this interference, malicious nodes can infer information about other applications. This type of attack is called *timing channel attacks*.

In these types of attacks the fact that multiple flows are contending for shared resources is exploited. By using the variations in latency and throughput for requests, information about the data sent can be inferred.

This can be achieved by an aggressor node that intentionally floods a key part of the network used by the victim node, and measures latency as a way to determine when and how much traffic is flowing through that part of the network. This can for instance be used to estimate the fraction of '1' bits in an RSA key [5]. These types of attacks are called *side-channel attacks*.

Another way to exploit the latency is through *covert channel attacks* [5]. In these types of attacks, two nodes that should not be allowed to communicate can exchange information by exploiting shared resources. For example, by letting a high injection rate represent a '1' bit, and a low injection rate represent a '0' bit, two nodes can exchange info without directly communicating.

2.4 Countermeasures Against NoC Security Vulnerabilities

The vulnerabilities described in the previous section have been previously researched and different countermeasures have been proposed. This section will summarise some of these countermeasures.

2.4.1 Throttling

Sources that flood the network with requests can potentially occupy network resources such as switches and links, potentially blocking other flows, resulting in degraded performance of the network. This is the goal of denial-of-service attacks. An approach to solving this problem is *source throttling* [4].

The goal of source throttling is to minimise the harmful impact that sources can cause by flooding the network with requests. This problem is more prevalent when certain sources are sending packets in bursts comprised of large chunks, instead of sending evenly spread out requests [1]. Reducing the level of burstiness reduces delay and jitter in the network, something that might otherwise affect other flows [1]. The rate and burstiness of flows can be represented using ρ for the average rate of the flow and σ for burstiness. With this model, during a specific epoch (period of time), the rate of injection into the network should not exceed $\sigma + \rho T$, where T is the epoch.

A general approach to source throttling is to keep track of how many flits have been injected into the network within a certain time-frame T and compare this number to a threshold value, here called budget, $B = \sigma + \rho T$. This approach would be light-weight, as it only requires a counter for the local input port of each router, and a corresponding signal indicating whether or not the source should be throttled.

The source throttling signal can be represented as a one-bit output signal that is initially set to one, indicating that flits should be allowed to be injected into the network. As a flit enters the network through the input port the counter corresponding to the port is incremented. Once the budget is reached, the output signal is set to zero, indicating that flits should not be allowed to enter the network. The signal is then reset once a certain time has passed, again allowing flits to be injected into the network.

When the signal is set to zero, all requests to switch allocation from the local port are discarded, stalling the insertion of requests from the local port. When the signal is set to one again, the requests are allowed to pass through the router.

The general approach effectively implements source throttling, but contains several drawbacks. Consider Table 2.1 where source node (0,0) sends a packet to each of the two destination nodes (0,2) and (2,1). Using XY-routing, in this example the first packet will travel only in the Y-direction, while the second packet will first travel along the X-direction to (2,0), before going one step along the Y-direction to (2,1). This means that the two packets will not use any of the same resources, except for the router at the source node. Therefore, rejecting the second packet due to the traffic introduced by the first packet does not make sense from a performance point

of view, as these two packets do not affect the same parts of the network.

Table 2.1: Two packets of flits sent from source node (0,0) to two different destinations. The two packets will not use the same network resources.

Cycle	**0**	**1**	**2**	**3**	**4**	**5**
Flit type	H	B	T	H	B	T
Destination	(0,2)	(0,2)	(0,2)	(2,1)	(2,1)	(2,1)

Another potential drawback is if the budget gets met immediately after the head flit has passed through the throttling stage. Since the head flit contains the destination data and allocates resources for the following body and tail flits, network resources could potentially be locked up while not being used. Locking up and not using network resources can cause the same problems as if the network was flooded, essentially removing the positive effects of the source throttling.

Therefore, allowing E extra flits to pass through the network if the head flit has gotten through reduces the performance loss due to the "only head flit gets through" problem. The new budget can therefore be described as $B+E$, where B is the original budget, and E is the number of extra flits that are allowed through if the head flit gets accepted.

This approach adds some extra complexity, as another budget-check is needed. Another approach would be to let the whole packet through if the head flit is within budget, by only comparing head flits to the budget. This approach could work for shorter packet lengths, but for longer packets containing a large number of body flits this could instead significantly reduce the effect of the source throttling.

Consider a case where $B = 64, T = 90$ cycles, and the packet length is 30. In this case, two whole packets would pass through the source throttling mechanism, and the third one would start sending. In this case, letting the whole packet continue through the network would mean that we eliminate any effect the source throttling has. On the other hand, by only letting the first four flits pass through router resources can be deemed wasted, as they will be idle for a large chunk of the epoch T. Note that the idle resource cannot be used by flits from any other packet, as they are allocated to the first packet.

With these examples in mind, we consider the middle road of allowing a certain number of extra flits to be a suitable trade-off. As we can see from the example, the optimal number of extra flits that should be allowed through depends on the situation, which means that the system designer should be allowed to configure this number.

2.4.2 Network Partitioning

Timing side-channel attacks utilise the fact that the delay introduced when multiple nodes are using shared resources can be read by an adversary [5]. By measuring the latency and noticing latency increases when a target node is using the network, information about the transmitted information can possibly be inferred.

Two possible approaches to remove the possibility of timing side-channel attacks by

separating high and low-security flows are spatial and temporal network partitioning [5]. In spatial partitioning, the network is divided into several clusters, where routing is restricted such that routers within the same cluster may only be used to route a packet belonging to a specific cluster. In this way, the flows belonging to different clusters are spatially separated, removing the possibility for a flow from another cluster to perform a timing channel attack.

In temporal network partitioning, the network resources are instead time-multiplexed, where flows belonging to a certain security class are allocated specific timeslots in the shared router resources such as switch allocation and link traversal. This approach can lower the performance of the network as it reduces the number of resources that can be used by the different flows.

Both of the above-mentioned techniques require classification of flows. This can be done similarly to the classes presented in the QNoC architecture [11]. Both the spatial and temporal partitioning techniques provide static allocation of resources, meaning that they cannot be changed to accommodate changing network requirements. This can incur high costs in terms of performance, as certain resources might be underutilised.

An efficient countermeasure against timing-channel attacks is presented by Wang and Suh [5] and uses a priority-based arbitration scheme with a static limiting mechanism. This technique provides one-way information leak protection. Flows are assigned security levels where low priority is assigned to high-security flows, and high priority is assigned to low-security flows. This means that when two flows from these security classes request arbitration at the same time, the low-security flow wins arbitration. The flows are further isolated by assigning a subset of the available virtual channels to each of the security classes. This is done to remove head-of-line blocking between the different security classes.

Since the low-security flits always win arbitration, a denial-of-service attack could potentially be performed by a low-priority node. To combat this, the authors use a throttling mechanism of low-security flows similar to that described Section 2.4.1.

The technique described provides a one-way protection against information-leak, protecting high-security resources from leaking information to low-security resources. However, the fact that low-security flows are allowed to always win arbitration versus the high-security flows can cause problems. The throttling mechanism introduced has several drawbacks that might cause the performance of the network to be degraded. These drawbacks are described and discussed in Section 2.4.1.

2.4.3 Traffic Management

Another proposed remedy against timing side-channel attacks is presented by Reinbrecht, et al. [21], the “Gossip NoC”. The Gossip NoC analyses the traffic in the network to detect potential timing side-channel or distributed timing side-channel attacks. The NoC uses two stages, detection and protection.

In the detection stage the network is monitored and in case of suspected attacks generates gossip messages, which alert neighbouring nodes of potential attacks. When the number of Gossip messages reach a confidence level, the routing algorithm of

the following packets is changed from XY routing to YX routing. This changes the route that packets take through the network.

This approach grants the network dynamic protection against timing side-channel attacks, but suffers from the fact that attackers might become aware of the routing algorithm change and adapt the attack accordingly. The solution to this problem offered by the authors is to limit the number of nodes that can run external code, limiting the potential number of attackers.

3
Design

This chapter describes the design of the three proposed isolation techniques, their advantages and their drawbacks. Before presenting the techniques, an overview of the identified vulnerabilities and suitable countermeasures is presented below.

Denial-of-service attacks operate by overloading the network resources by sending large amounts of requests to a part of the network, with the goal of disrupting normal operation of the targeted victim node. This type of attack relies on being able to overload shared resources such as routers, buffers, and links. This is made possible through a large number of requests, either by using many attacking nodes or through a high injection rate from a few nodes.

The characteristics of this type of attack indicates that a possible way of reducing the effectiveness can be to reduce the injection rate of untrusted nodes. If the number of requests that can be injected into the network during a given time is limited, a single adversary is blocked from disrupting the service of other nodes using this tactic. However, if more nodes participate in the attack, this approach must be tuned such that the combined injection rate is lower than the threshold.

Another way of tackling this problem can be through separation of router resources. If untrusted nodes can only access a subset of all the available resources, the trusted nodes can still operate without being affected by the overload in the resources allocated to the untrusted nodes. With this in mind, we identify that both partitioning of the virtual channels and the switch allocator can be used to separate trusted and untrusted flows.

By allocating a portion of the available resources to either trusted or untrusted sources, we partition the network. This can lead to performance degradation, as less resources will be available for the normal operation of the network. Therefore, it is important that these techniques are highly customisable to suit many different scenarios.

The timing side-channel attack exploits the fact that contention over shared resources can show latency differences in traffic when a targeted traffic flow is transmitting over a route in the network that the attacking node is also using for transmission. This type of attack relies on the network not being able to handle the incoming requests without delay if there are multiple flows utilising the shared resources at the same time.

This means that, like denial-of-service attacks, continuous message transmission from an aggressor is used to carry out the attack. Thus, limiting the injection rate of an aggressor might reduce the effectiveness of this attack. However, the flows still share the resources, and the attack may therefore still be able to be carried out if

the aggressor adjusts its injection rate, since the important metric to the attacker is the latency fluctuations.

A technique that eliminates the root of the problem is partitioning of the network. If an aggressor and a targeted victim node are not allowed to use the same resources at the same time, the aggressor cannot deduce any information about whether or not the victim flow is active. Here, the access to both virtual channels and the switch allocator affects the latency of a flow, meaning that by separating the flows' access to these, the potential for carrying out timing side-channel attacks should be reduced.

In summary, we see that for both of these attacks, source throttling and network partitioning can serve as effective countermeasures. We therefore identify three areas through which to provide isolation. These include *source throttling*, limiting the injection rate of aggressor nodes, and two static network partition techniques, *fixed virtual channel allocation*, and *separate switch allocator timeslots*.

This remainder of this chapter will describe the proposed design of a NoC with capabilities to isolate flows. The design consists of the three approaches to isolation of flows described above and build upon a basic NoC as described in Chapter 2.

3.1 Source Throttling

To combat the problems related to the congestion of the network that occurs during denial-of-service attacks we propose an extension to the general source throttling approach presented in Section 2.4.1. This technique tracks the number of flits that are injected into the network via the local resource port, throttling sources that exceed the predetermined static threshold (here referred to as budget). The over-budget flits are disqualified from switch allocation and instead wait in the input VC buffers.

However, since a source can send requests to several different destinations, utilising different network resources, simply throttling all incoming requests from a source can harm performance more than necessary. Therefore, the throttling mechanism should be aware of the destination of the packets and not only the number of flits injected within a given time frame.

Our approach to source throttling extends each of the routers with a source throttling controller, which reads data from the input port belonging to the local resource, counting the flits that are injected into the network. The design divides the source throttling into two separate stages for the flit counting and the budget check. A second budget check is also added to the switch allocator to handle over-budget head flits that can otherwise get through the source throttling controller due to the one cycle latency in updating the flit counter. The additions in the router between the input block and switch allocator can be seen in orange in Figure 3.1.

The source throttling mechanism has three parameters that affect the rate of throttling:

- **Budget (B)** is the number of flits that are allowed to be injected to the network from a source node, per destination node. A flit originating from node (x,y) with the destination (x',y') can only be injected as long as the

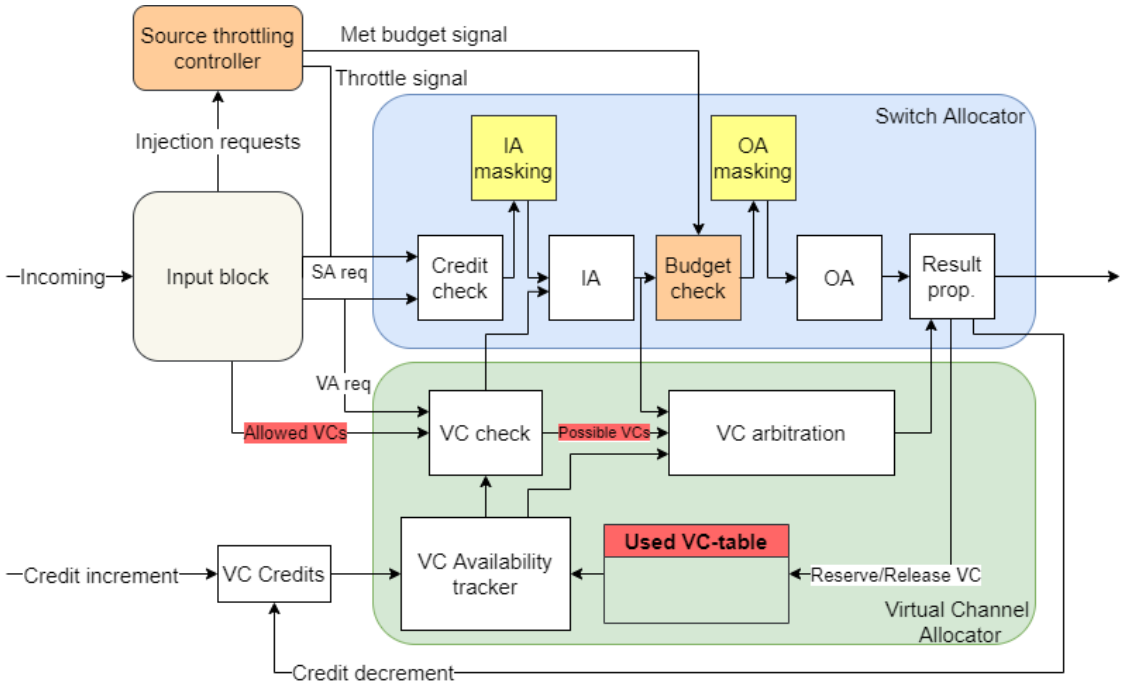


Figure 3.1: The block diagram of the input block, switch allocator, and virtual channel allocator of the router with the isolation techniques implemented. The added functionality for source throttling is shown in orange, fixed VC allocation in red, and the separate switch allocator timeslots in yellow.

corresponding flit counter is less than B.

- **Extra Budget (E)** is an extra budget that allows up to E extra flits to be injected from a source node, per destination node. The extra budget prevents shorter packets from being throttled in the middle of the packet, potentially reducing head-of-line blocking at the source node.
- **Epoch (T)** represents the number of clock cycles before the flit counters are reset and flit injections are allowed at throttled source nodes.

The throttling quota ranges from 0 flits per T cycles, up to T flits per T cycles, and can therefore be described as

$$\text{Quota} = \min\left\{\frac{B+E}{T}, 1\right\}. \tag{3.1}$$

The parameters and their possible values are shown are show in Table 3.1. The epoch and extra budget are set globally, while the budget is set for each of the source nodes. This allows the system designer to choose which sources to throttle. For example, we can throttle all processor cores that run third-party software.

Table 3.1: The parameters and their possible values for the source throttling technique.

Parameter	**Budget, B [flits]**	**Extra Budget, E [flits]**	**Epoch, T [cycles]**
Possible values	$0 \leq B \leq T$	$0 \leq E \leq$ size(Packet)-1	$T > 0$

3.1.1 Stage One: Flit Counting

The first stage of the throttling process, flit counting, counts the injected flits based on their destination and increments their respective counters. As seen in Figure 2.2, the header field of the head flits contain the destinations and VC IDs of the packet. Since the body and tail flits to do not contain destination information, these values need to be stored and reused when the rest of the flits in the packet go through the controller's counting mechanism. The key observations to be considered are:

- Head flits allocate the output port VC, which is then used by all flits in the packet.
- Body and tail flits always follow the head flit in the same VC, but they can be interleaved with flits from other packets when traversing the links between routers.

The two observations mean that we know that if the head flit of packet P gets allocated VC0, the next flit in that VC will be a body or tail flit from the same packet. This information can be used to increment the correct destination counter for all flits of the same packet. This is done by storing the VCID and destination for each of the VCs when a head flit arrives. This mapping is then used for all subsequent body flits that have the same VCID. After the tail flit has arrived, the mapping between destination and VC is reset.

The process performed in stage one is presented in Algorithm 1. For head flits, their destination is gathered and the counter corresponding to the destination of the flit is incremented. For body and tail flits the mapped VC-destination pair is used to determine the destination and increment the counter. Since the head flit allocates the virtual channel, which is then released by the tail flit, the body flits with the same VC-ID as the last recorded head flit will always have the same destination.

Algorithm 1 Source throttling controller stage 1

Precondition: F is an incoming flit from the local resource, C_x is the counter for destination x, M is a tuple containing the most recent (VC-ID: destination) pair.

```
1:  function FLIT-COUNT(f)
2:      if F.type ∈ {H, HT} then
3:          x ← F.vcid
4:          d ← F.dst
5:          M ← (x, d)
6:          C_d ← C_d + 1
7:      else if F.type ∈ {B, T} then
8:          if F.vcid == M.vcid then
9:              d ← M.dst
10:             C_d ← C_d + 1
11:         end if
12:     end if
13: end function
```

3.1.2 Stage Two: Budget Check

In the second stage, budget check, flits that have entered the output port VCs are checked against the budget and the flow is throttled if it is over-budget. Since the output port VC does not necessarily have the same VCID as the input port VC, the mapping between VCID and destination has to be performed in this stage as well. Furthermore, due to the previously described trade-off in how to handle the problems when only the head flit gets through, the body and tail flits are compared against the extended budget $B + E$.

Algorithm 2 shows the steps taken to find the requests that should be filtered out. Incoming flits are filtered out if they are over-budget. For header-flits, this is done by collecting the destination from the flit and then comparing the value of the counter to the budget, B. If the flit is over-budget, the flit will not be allowed through. A signal notifying the source throttling controller that additional body and tail flits will not be allowed is also activated. If the counter is under budget, the flit is allowed through and additional body and tail flits are allowed.

For body and tail flits the destination is gathered from the mapping performed when the head flit passes through, which is then used to compare the destination counter to the extended budget, E. The flits are allowed through if they are under the extended budget and allowed (due to an in-budget head flit previously passing through).

3.1.3 Throttling Implementation

The throttling itself is carried out through signals sent to the switch allocator. When a flit causes the counter to go over-budget, as seen in Algorithm 2, a *met-budget* signal is activated. The signal stays active until the start of the next epoch, upon which the destination counter is decreased by a value of B. This allows the throttled source to send at most $B + E$ flits in each epoch, where the over-budget flits will be kept in the FIFO buffer.

The throttle signal from the source throttling controller acts as a mask that filters out SA requests from the local resource port of the input block when the signal is active. The budget check stage is added to SA between input and output arbitration, see Figure 3.2. The throttle signal is activated when the budget is met and more than B clock cycles has passed in the epoch. Even though we allow between B and $B + E$ flits to pass through at the start of the epoch, the destination counters are still incremented and the met-budget signal activated as usual. The throttle signal therefore disregards the met-budget signal for B cycles at the start of the epoch.

However, since one cycle is required to update the destination counter registers, one over-budget flit could pass through the throttling check even though it should not. To combat this problem, a met-budget signal for specific VCs is sent from the source throttling controller to the switch allocator, along with a signal that is activated when the current flit is a head flit. When both of these signals are active, requests from head flits that erroneously got through input arbitration due to the above-mentioned “first flit through” problem will be removed before output arbitration.

Algorithm 2 Source throttling controller stage 2

Precondition: F is an incoming flit from the local resource, C_x is the counter for destination d, M is a tuple containing the most recent (VC-ID: destination) pair. N is the extra budget.

```
1:  function BUDGET-CHECK(f)
2:      for each output port VC do
3:          if F.type ∈ {H, HT} then
4:              x ← F.vcid
5:              d ← F.dst
6:              M ← (x, d)
7:              if C_d > B then
8:                  Throttle
9:                  Disallow B/T flits
10:             end if
11:         else if F.type ∈ {B, T} then
12:             if F.vcid = M.vcid then
13:                 d ← M.dst
14:                 if C_d > B + N ∨ B/T-flits not allowed then
15:                     Throttle
16:                 end if
17:             end if
18:         end if
19:     end for
20: end function
```

3.1.4 Drawback of Proposed Approach

Even though the proposed approach removes the problem of flits to different destinations blocking each other, head of line blocking can still occur. Head of line blocking occurs when one or more flits are blocked, stalling the transmission. Flits that reside behind a throttled packet in the virtual channel FIFO buffer then get blocked even if would be allowed to pass given that they were not blocked.

This problem can occur if a destination counter is over-budget, disallowing future injections with that destination during the epoch. All flits that are in the same input block VC then get blocked until the start of the next epoch. The feature of allowing extra flits helps mitigate this problem by allowing smaller packets to finish transmitting, which can allow another packet to a different destination to get to the head of the queue.

Another drawback of this approach relates to the area requirements of the implementation. For each of the destinations, a separate counter is required. This means that if there are $N \times M$ sources in the network, each router needs $N \cdot M - 1$ counters to count the flits to all possible destinations.

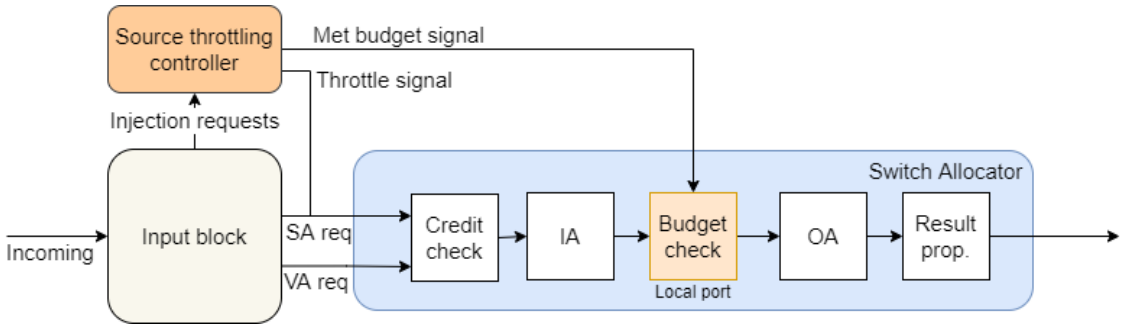


Figure 3.2: The budget check extension to the switch allocator. This check is only added for the local port, meaning that all other ports are unaffected by this change.

3.2 Fixed Virtual Channel Allocation to Traffic Flows

By allocating each source a subset of the available virtual channels, different flows inside the network can be separated inside the router. To support this, we propose an approach where each of the local sources is given a set of allowed virtual channels that the flits originating from that source are allowed to use.

3.2.1 Approach

Our approach to enabling separate virtual channels for different flows utilises a configuration table with information about the allowed virtual channels for each of the nodes, and is shown in red in Figure 3.1. This information is stored in one-hot encoding in the header field of the head flit of each packet, increasing the size of the header field by the number of VCs at each port the NoC, see Figure 3.3. Furthermore, in our implementation the original structure of the VA stage is replaced and instead VC allocation is performed in parallel to the SA stage.

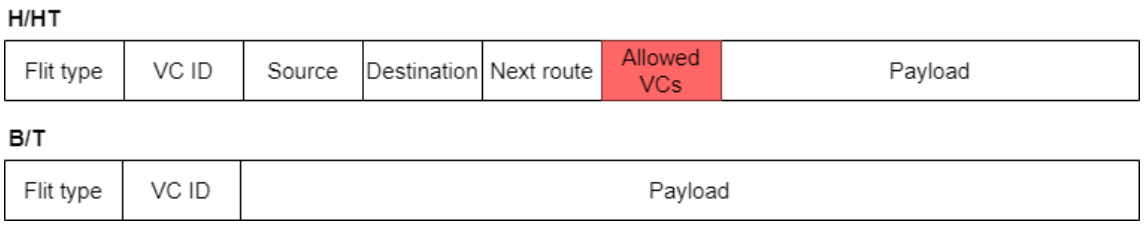


Figure 3.3: The organisation of flits with the separate virtual channels extension, shown in red, increasing the size of the header field of head flits. Only the H and HT flits are changed.

The information regarding the allowed VCs for packets originating from a specific source is configured using a table that specifies the allowed virtual channels for each of the source nodes. In a $N \times M$ mesh network this is represented by a $N \times M$ array of vectors of size V, where V is the number of virtual channels per port.

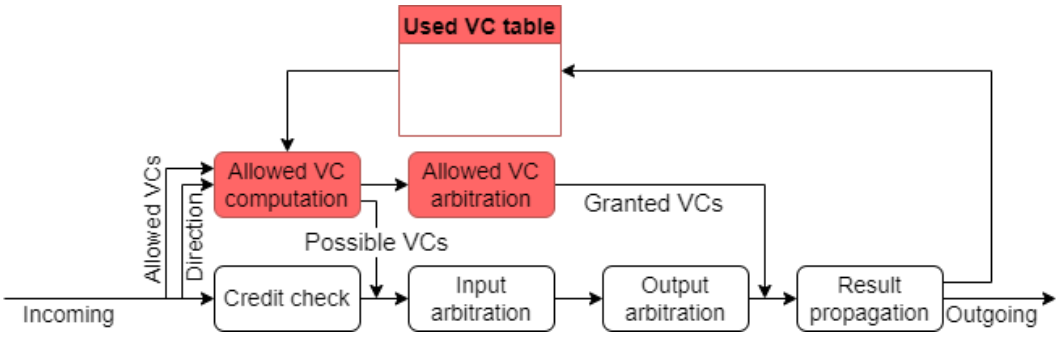


Figure 3.4: Layout of the updated SA stage, allowing for separate VCs.

Table 3.2 shows one such possible configuration. In this example, the virtual channels with ID 0, 1, and 3 are shared among several different sources, while VC2 is reserved for node (0,2). This means that even though VC2 might not be used in a cycle, it cannot be used by a packet from any other source. This provides isolation between the flows as they cannot contend for the same virtual channels.

Table 3.2: Example of VC allocation using the separate virtual channel technique in a 4x4 NoC. Node (0,2) is the only node that can use VC number 2 in all routers, providing isolation of flows.

x/y	**0**	**1**	**2**	**3**
0	{0,0,1,1}	{0,0,1,1}	{0,0,1,1}	{0,0,1,1}
1	{0,0,1,1}	{0,0,1,1}	{0,0,1,1}	{0,0,1,1}
2	{0,1,0,0}	{0,0,1,1}	{0,0,1,1}	{0,0,1,1}
3	{1,0,1,1}	{1,0,1,1}	{1,0,1,1}	{1,0,1,1}

In order to support the fixed allocation of virtual channels, the virtual channel allocator was replaced. Since all possible VCs are pre-configured, the traditional VC allocation stage is replaced by a functionally equivalent arbitration between the allowed VCs, performed in parallel to the switch allocation. The two new steps perform the same functionality as the traditional VA stage, and can be seen in Figure 3.4, *Allowed VC Computation* and *Allowed VC Arbitration*.

The first stage, Allowed VC Computation, is responsible for masking incoming requests with the allowed VCs. This is done in a three step process:

1. Upon incoming SA request for header flits, extract the direction of the packet and the allowed VCs from the head flit.
2. Ensure that at least one allowed VC is available at the chosen direction, by using the *Used VC*-table and performing a credit check at the allowed output VCs.
3. Mask the request with the allowed and available VCs and return the result.

The operations carried out in the allowed VC computation stage are show in Figure 3.5. The allowed VCs and next route computation result (direction) are part of the header field of the head flits and are sent from the input block to the switch allocator. The requested direction is paired with the number of allowed VCs for that specific output port, and a credit check is carried out. The result of this computation is

matched with the available VCs at the requested direction. The resulting allowed requests are then sent to allowed VC arbitration and input arbitration for SA.

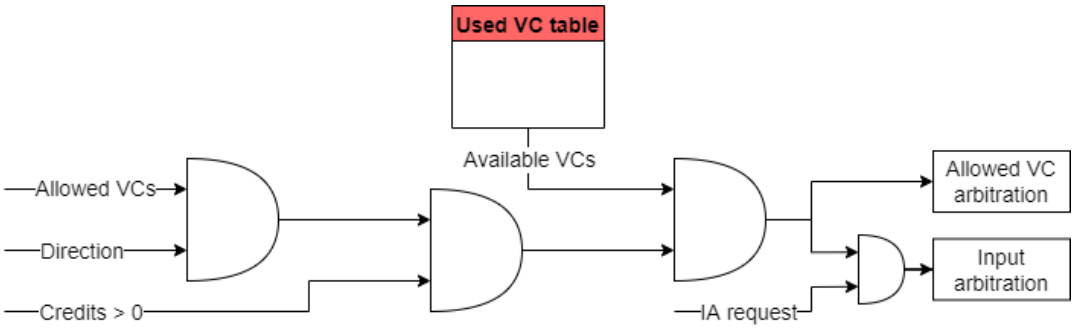


Figure 3.5: The Allowed VC computation stage. This operation filters out all VCs that are not allowed, not free, or do not have enough credits available.

The second stage, Allowed VC Arbitration, is comprised of arbitration of the allowed VCs using a round-robin fixed priority arbiter. One of the possible output virtual channels is selected, and the result is sent to the result propagation stage. In the result propagation stage, the granted VC is matched with the grant of the output arbitration and propagated to the following entities in the router pipeline. The used VC table is also updated, marking the granted VC as unavailable until the tail flit of the packet has passed.

In order to ensure that only VCs that are granted both by the allowed VC arbitration and output arbitration are marked as used, the signal to the Used VC table is only updated once a matching grant has been given in both of these steps in the same cycle.

3.2.2 Cost of Implementation

This proposed approach modifies the VC allocator, and carries out all VC arbitration calculations in the switch allocator. This means that the switch allocator gets more complex as more logic and registers are added. To reduce the potential negative performance effect of this change, the new steps have been designed to be done in parallel with switch allocation, keeping the same combined VA and SA stages approach as in the standard version of the NoC.

This change might however increase the critical path latency of the network, as the complexity of the virtual channel allocation increases. In the standard version of the NoC, any free VC with available credits is chosen by the incoming head flit, while in this design there is a further constraint that the VC must also be allowed. This extra step adds extra complexity to the virtual channel allocation as flits from input VCs have to be matched with a free and allowed output VC.

This approach might also decrease the performance of the network depending on the traffic pattern, as less resources in the form of virtual channels are available for each individual flow. Even though this might not cause problems with low network load, higher loads might increase the number of congestion-related problems in the network.

3.3 Separate Switch Allocator Timeslots

The third proposed isolation technique consists of assigning separate switch allocator timeslots to different flows. This technique can reduce the possibilities of timing side-channel attacks by only allowing flits to use the switch during their pre-allocated and reserved timeslot, by preventing other flits from using that timeslot for switch traversal. Since there are two arbitration stages in the SA stage of the router pipeline, requests going into both input and output arbitration need to be handled.

Flits entering a router have four key pieces of information that need to be consider for switch allocation: input port, input and output virtual channel IDs, and output port (direction). An incoming flit comes through input port P_{in} in VCX and goes to output port P_{out}. By assigning a table of allowed VCs for the input ports and a table allowed input ports for the output ports, the tables can be iterated through in set intervals to create separate switch allocator timeslots. The extended switch allocation (SA) stage of the router pipeline is shown in Figure 3.6, and in yellow in Figure 3.1.

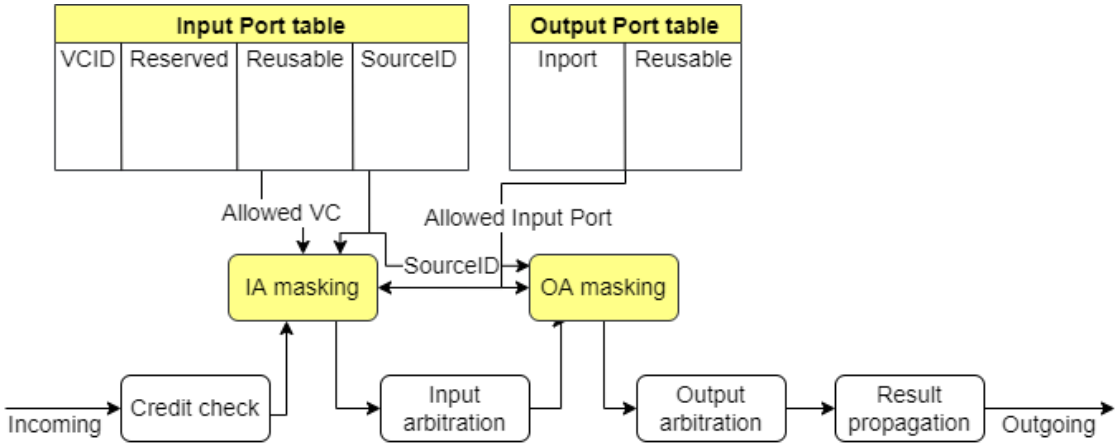


Figure 3.6: The extended SA stage, allowing for separate switch allocator timeslots.

In the implementation of this technique, there are two options considering how to handle unused reserved timeslots:

1. Never allow other flits to use unused reserved timeslots.
2. Allow other flits to use a reserved timeslot if there is no allowed flit present.

The first alternative is to restrict usage of reserved timeslots strictly to allowed flits. This ensures a higher level of isolation and higher predictability of performance, but might lead to lower network performance due to a potentially large amount of unused timeslots. The second option allows for higher utilisation of network resources, but would decrease the level of isolation that is provided.

If all idle timeslots were allowed to be reused by any other source node, an attacker could perform a timing side-channel attack, sending requests and examine the latency pattern in the responses. This could potentially allow the attacker to infer information by examining the activity of other sources. This defeats the isolation

purpose, but allows for higher performance as unused timeslots are not left idle. To handle this problem, we allow each timeslot to be configured to be reusable or non-reusable. Additionally, which source should be allowed to reuse the timeslot can also be configured.

The separate switch allocator timeslots technique is configured by two tables, the input port and output port tables. Here, there are several different parameters that decides the functionality of the technique. The number of timeslots, N, decides how many independent timeslots to split the allocator schedule into.

The input port table is shown in Table 3.3 and contains fields for allowed VC, whether the timeslot is reserved, reusable, and which source is allowed to reuse the timeslot. The output port table is shown in Table 3.4. In the output port table the reserved field is integrated in the field which describes the allowed input port. If the allowed input port is set to the number of ports (five in the standard case), the timeslot will be unreserved. Otherwise it is reserved.

Table 3.3: The input port table parameters for the separate switch allocator timeslots technique.

Parameter	**VCID**	**Reserved**	**Reusable**	**Source ID**
Possible values	$0 \leq$ VC ID $\leq$ NUM_VCS-1	$0 \vee 1$	$0 \vee 1$	$2 \cdot 3$

Table 3.4: The output port table parameters for the separate switch allocator timeslots technique. Each entry is in the form of a vector of size N.

Parameter	**Input port ID**	**Reusable**
Possible values	$0 \leq$ IP ID $\leq$ NUM_PORTS	$0 \vee 1$

The required amount of bits for each of the table entries are described in Table 3.5. This means that the total table size per router can be described as

$$S = N((B_{VCID} + 1 + 1 + 2 \cdot 3) + (B_{Port-ID} + 1)) = N(9 + B_{VCID} + B_{Port-ID}) \quad (3.2)$$

where N is the number of ports, B_{VCID} is the number of bits in the VCID, the reserved and reusable bits are one bit per entry each, the source ID is six bits, and $B_{Port-ID}$ is the number of bits in the port IDs.

Table 3.5: The fields and their size for both the input and output port tables.

	Input port table				Output port table	
Content	VCID	Reserved	Reusable	Source ID	Direction	Reusable
Bits per entry	$\log_2(VCs)$	1	1	2	$\log_2(Ports)$	1

3.3.1 Input Arbitration Masking

After the credit check, the resulting input arbitration request vector is forwarded to IA masking. From the input and output port tables the allowed VCs, the reserved bit, and allowed input ports are extracted, as shown in Figure 3.7.

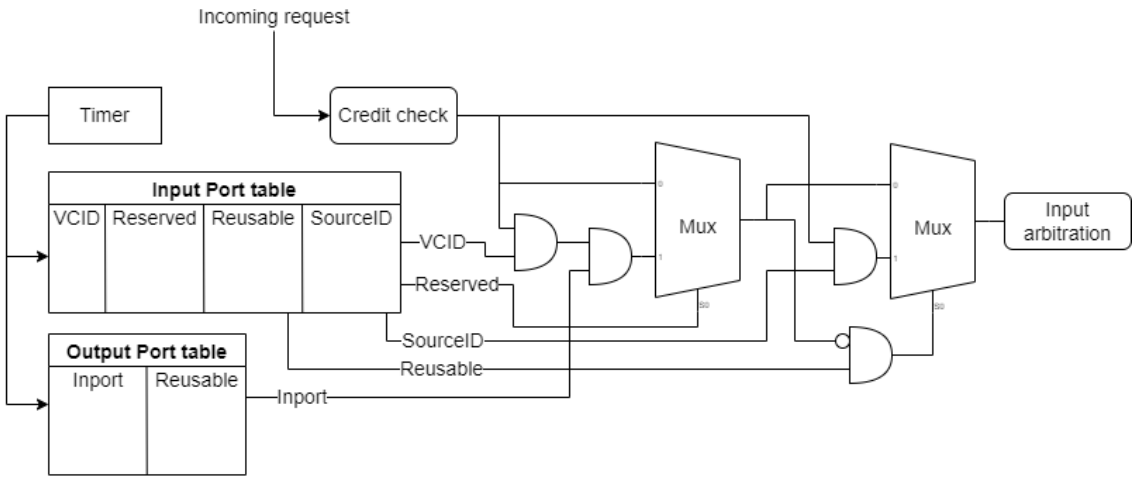


Figure 3.7: The Input Arbitration masking stage, performed after credit check and before input arbitration.

If the timeslot is not reserved, the computations of this stage are disregarded and the input arbitration functions like normal, with the incoming SA request being passed along to input arbitration. If the timeslot is reserved, the input VC ID is masked with the allowed VC from the input port table, leaving the request active only in the allowed VCs.

If the incoming request for the reserved timeslot is zero, meaning that no request is present, another VC is allowed to reuse that timeslot if the reusable bit in Table 3.5 is set, and the source ID matches that in the table. If any source should be able to reuse the timeslot, the source ID value is set to the number of nodes in the network, (N,N) for a $N \times N$ mesh network.

In the next step, if the resulting request is equal to zero (no allowed requests) and the timeslot is reusable, the original SA request is sent to input arbitration. Otherwise, the masked request is sent.

3.3.2 Output Arbitration Masking

After input arbitration, the resulting grant vector is sent through the OA masking stage, where the grant vector is masked with the allowed input port, before entering output arbitration. Once output arbitration is completed, the results are propagated to other parts of the router.

Figure 3.8 describes the layout of the OA masking stage. In this stage, the allowed IP is read from the output port table and compared to the value in the output arbitration vector. For each bit of the request vector, if a match is found, the output arbitration request is passed through to OA. If no match is found, the corresponding bit is set to zero.

This stage features the same reusability functionality as the input arbitration masking stage, allowing a specified source, or any source, to reuse an unused timeslot. The timeslot is only reused if there is no incoming request from the reserved input port.

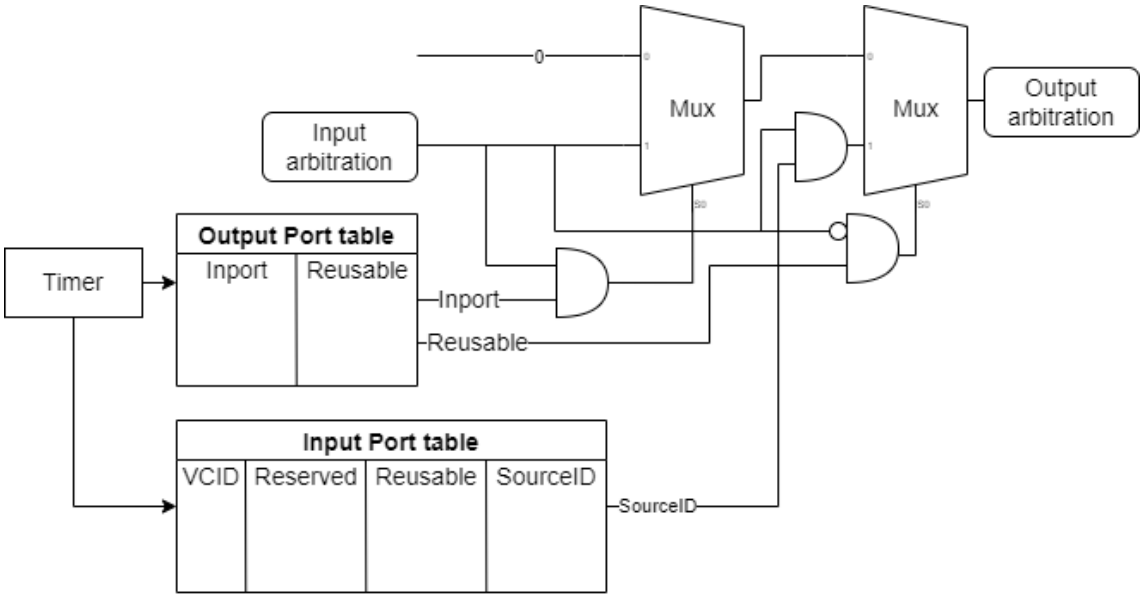


Figure 3.8: The Output Arbitration masking stage, performed after input arbitration.

3.3.3 Performance Implications

Separate switch allocator timeslots extend the functionality of the switch allocator, without significantly changing the overall structure of the router or the pipeline. However, the critical path is affected by the required modifications, as the table lookups for both IA and OA masking are done in the same cycle as their results are used.

To mitigate the effect of these lookups, the tables can be read one cycle in advance, and be done in parallel to the other work in the SA stage. The results of the table lookups are placed in registers, which are then checked by the IA and OA masking stages.

In the register version, only register reads are needed, instead of performing a table lookup. Note that the IA and OA masking stages do not change, only the way the parameters are read. The updated version of the extended SA stage is shown in Figure 3.9.

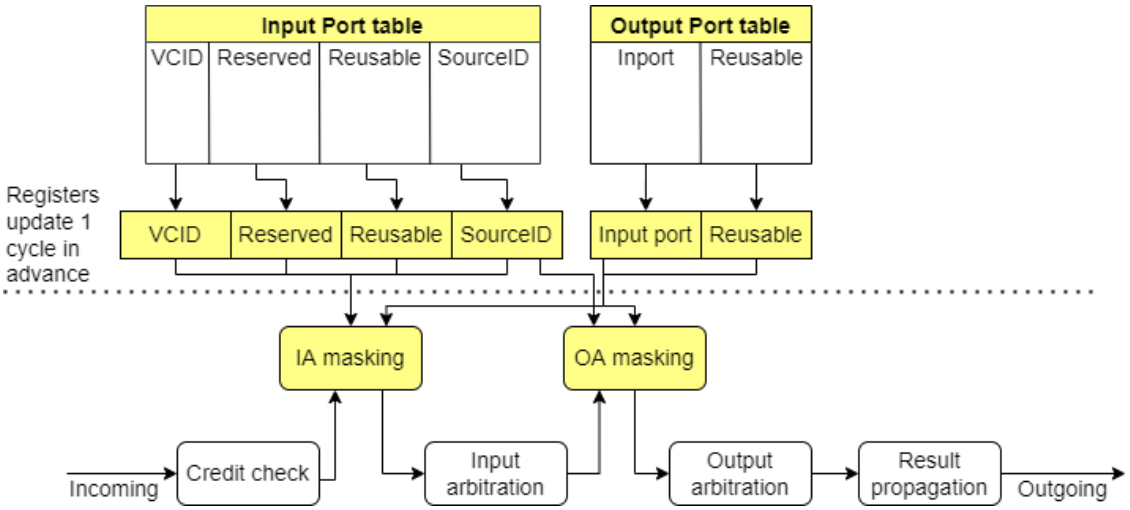


Figure 3.9: The updated SA stage. Tables are read one cycle in advance, and results are placed in registers that are read during execution.

3.4 Design Summary

Three techniques for providing isolation of flows have been presented:

- Source Throttling
- Fixed Virtual Channel Allocation
- Separate Switch Allocator Timeslots

The three techniques all change the router to various degrees. They have all been designed to be able to be enabled or disabled using a mode selector register, allowing for a high degree of versatility in choosing what techniques to use. They can be used independently or in combination, based on the system requirements.

Source Throttling adds a new module to the router, the source throttling controller. This increases the area of each of the routers. Since the operations are done in parallel with the pipeline, the pipeline is not affected in any other way than the wires and logical AND-gates required for masking the SA requests with the throttle signals.

The Fixed Virtual Channel Allocation technique operates in parallel with the SA stage, masking input arbitration requests and output arbitration grants. Since the original VC allocator is replaced with a functionally equivalent system, but with added functionality, some effect on the latency of the system is expected. The switch allocation is affected by the masking of requests, but is otherwise unaffected by the implementation of the technique.

Lastly, the Separate Switch Allocator Timeslots technique inserts two extra stages to the switch allocator, input and output arbitration masking. This means that the critical path of the router is extended. To mitigate this problem, the table lookups are done one cycle in advance, reducing the effect to a register lookup along with masking of requests.

4

Evaluation

This chapter describes the results of the simulations that were carried out in order to verify the performance of the proposed isolation techniques. Short comments on the evaluation will be provided, but the general discussion is presented in Chapter 5.

The rest of this chapter is divided as follows. First, an overview of the experimental setup is presented, including simulation specific information and terminology. Then, the evaluation of the isolation techniques against denial-of-service attacks is presented, followed by the evaluation of the timing side-channel attack prevention and performance impacts of the techniques. The chapter concludes with a summary.

4.1 Experimental Setup

To accurately evaluate the proposed techniques, a model of the NoC with isolation features was implemented using Register Transfer Level abstraction. This implementation builds upon a 3-stage pipeline NoC, here called the Standard NoC. The router architecture and pipeline stages are described in chapter 2. The network was modelled in SystemVerilog.

Both the standard NoC and the NoC with isolation were implemented in a 4×4 mesh network with 5 ports per router and 4 VCs per port and were simulated using QuestaSim. The clock period for the simulation was 500 picoseconds, corresponding to a frequency of 2 GHz.

Synthetic traffic flows were used to simulate the network. The denial-of-service attack and timing side-channel attack traffic were manually configured and are described in their corresponding sections. The traffic flow for determining the performance penalty of using the proposed NoC with isolation features was a uniform random traffic flow which was generated using Matlab.

To simulate sources transmitting and receiving packets, a simulation setup with a packet generator, packet queue, transmitter and receiver was used. A block diagram of the simulation is presented in Figure 4.1.

A stimulus file containing a description of the traffic to be injected into the network is read by a packet generator which generates the packet to be transmitted. These packets are placed in a FIFO packet queue while waiting to get injected into the network. The queue continuously updates the packet generator of the number of packets stored in the queue. Once the transmitter accepts the packet an acceptance message is sent to the queue, indicating that the packet can be removed from the

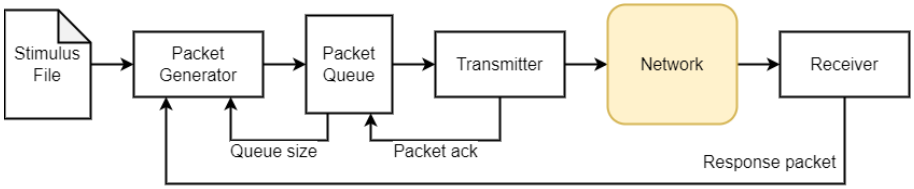


Figure 4.1: Block diagram of the simulation setup. The queue size signal is used for closed loop simulations.

queue. The queue size is then decreased.

The transmitter then injects the packet into the network from the local port of the router specified in the stimulus file. The packet travels from the source to the destination through the routers in the network. After leaving the local output port of the destination, it is received by the receiver.

If the traffic type is set to send responses for received packets, the receiver notifies the packet generator that a response should be sent. The response is then generated by the packet generator and inserted at the front of the packet queue. The response packets are placed at the front of the FIFO queue in order to reduce the time the response packet spends outside the network.

4.1.1 Simulation Types

Two different types of simulation approaches were used, *open loop* and *closed loop* simulations. The denial-of-service and performance evaluations use an open loop simulation, while the timing side-channel attack evaluation use a closed loop simulation. In an open loop simulation the network no feedback on the state of the network is considered during simulation, while in a closed loop simulation, the system gets information about the state of the network, inserting feedback loops in the system [1].

In this simulation the packet generator receives feedback from the packet queue regarding the size of the queue, shown between the packet queue and packet generator in Figure 4.1. This is important for the latency-sensitive simulations carried out when simulating denial-of-service and timing side-channel attacks.

The lifetime of a packet is measured from the time it is generated to the time it is completely received at the destination. It is because of this fact that the response packets are placed at the top of the packet generator queue waiting outside the network. If packets would have to wait in a queue, the latency of the packet would be artificially higher due to to the time spent in the queue. By limiting the queue size and inserting responses at the top of the queue allows us to measure the time that the packets spend in the network.

4.1.2 Isolation Configurations

The three different isolation techniques that were tested are all configurable by the system designer. All techniques have static configurations, meaning that changes to the configurations are done before startup, and can not be changed during runtime.

The source throttling technique has three parameters that affect the rate of throttling, Budget (B), Extra Budget (E), and Epoch (T). The budget specifies how many flits that are allowed to be injected at each source node during an epoch. Once a flit counter at the source node gets larger than B flits, further injection is prevented. The extra budget allows up to E extra flits be injected after the counter reaches B flits. The epoch describes the number of clock cycles before the flit counters get reset. If a source node is throttled, it will be allowed to inject flits again after the epoch of T cycles has passed. These parameters and their allowed values are described in greater detail in Section 3.1 and Table 3.1. During simulation of the Denial-of-Service and timing side-channel attacks, the aggressors were configured to be throttled with varying throttling quotas, while the victim and destination nodes were not throttled.

The fixed virtual channel allocation technique allows the system designer to decide which sources should have access to which VCs. The configuration is done on a per-source basis, meaning that each of the sources can be allocated as many VCs as necessary, and that several sources can be allowed to use the same VCs. The allowed VCs are stored in a table of vectors in one-hot encoding. Table 3.2 shows a possible configuration of VC allocation.

The separate switch allocator timeslots technique is scheduled on a per-port basis. This allows for a high level of granularity in the configuration, and allows the system designer to configure the schedule down to cycle-level. The configurations used for the simulations are shown in the format previously described in Tables 3.3 and 3.4.

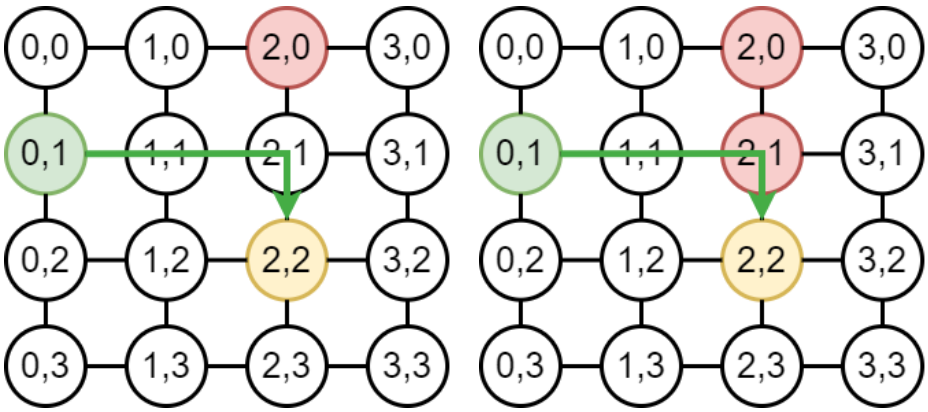
4.2 Protection Against Denial-of-Service Attacks

In order to simulate a Denial-of-Service attack, a traffic pattern designed to overload key links in a mesh network was designed. We call this traffic pattern *Aggressor-Victim*. One node is acting as the victim of the attack and sends out intermittent bursts of 10 packets at a low injection rate, while several other nodes are attempting to interfere by continuously injecting packets at a high injection rate.

Four configurations with differing amount of aggressors were used. The traffic pattern setups used for this simulation are shown in Figure 4.2. This traffic pattern means that the links between nodes {0,1}, {1,1}, {2,1}, {2,2}, where the victim node is attempting to send packets, will be congested to different degrees as a result of the aggressor nodes sending large amounts of requests to node {2,2}.

To showcase the effects of a denial-of-service attack, we simulate the network both with and without the aggressor traffic, shown Figure 4.3. Without any aggressor traffic, the latency for victim traffic injected in bursts of 10 packets with an injection rate of 0.5 flits/node/ns, has an average delay of 8.5 ns. When the aggressor traffic is introduced, but without any isolation features enabled, we see a substantial delay increase for the victim node, as the network fills up with requests faster than they can be handled. The average victim node packet latency is now instead 62.6 ns, a 637% increase of the latency compared to when the victim is not under attack.

Before introducing the isolation techniques, we note that the different techniques are expected to have different impacts both on the victim traffic and the aggressor



(a) One aggressor. **(b)** Two aggressors.

(c) Four aggressors **(d)** Six aggressors.

Figure 4.2: Different configurations of the aggressor-victim traffic pattern. Node {0,1} (green) sends packets to node {2,2} (orange), but experiences congestion in the network due to interfering traffic from the red nodes, sending a large volume of requests to node {2,2}.

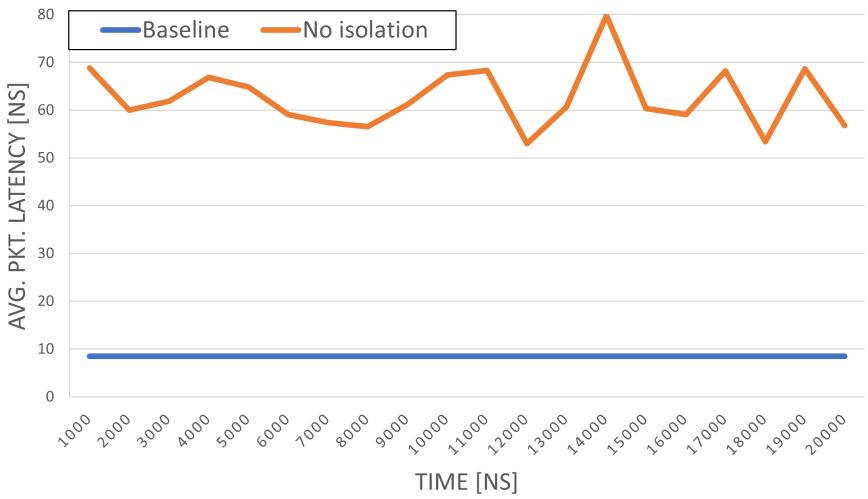


Figure 4.3: The latency increase of the victim node incurred by the denial-of-service attack. The average packet latency increases by 740% compared to the baseline.

traffic. The source throttling is implemented in a way such that it can be configured differently for each of the routers, allowing for throttling of any router that is deemed not to be trusted. The throttling can reduce the injection rate of any router, but cannot affect the traffic after a flit has been injected into the network.

The fixed virtual channel allocation technique allows for isolation of flows during the virtual channel allocation stage, by reserving VCs for the different sources. However, flits must still contend for switch allocation, meaning that they are not unaffected by other traffic flows that are travelling along the same route.

The separate switch allocator timeslots technique on the other hand ensures isolation for the switch allocation, but does not differentiate between the contents of the virtual channels. Therefore, a combination of separate virtual channels and separate switch allocator timeslots is required to isolate both virtual channel allocation and switch allocation.

To determine the usefulness of each of the isolation techniques in protecting against denial-of-service attacks, all three techniques were simulated and their results recorded.

4.2.1 Source Throttling

Source throttling restricts the number of flits that can be injected into the network from the local port during a given period, called epoch. The allowed number of flits depends on the allowed Budget and Extra Budget. In this simulation, different Budget values were tested using the Aggressor-Victim traffic pattern. The aggressor nodes were throttled using different budgets, while the victim node was not throttled. Four different aggressor setups were used, shown in Figure 4.2.

The simulation was carried out using closed loop simulation, meaning that the packet generator waited for the queue to empty before generating new packets. The aggressor traffic was setup such that the unrestricted injection rate of the aggressors

was 2.5 flits/node/ns with an average packet size of 3, representing a high network load. This corresponds to a "next-packet latency" of 2.4 cycles, meaning that packets would be generated faster than they can be injected into the network in an open-loop simulation. This setup was chosen in order to simulate aggressor sources overloading the network.

The tested configurations shown in Figure 4.4 are described in Table 4.1. The latency increase compared to the baseline is of 7.4 times, only varying slightly between the different configurations.

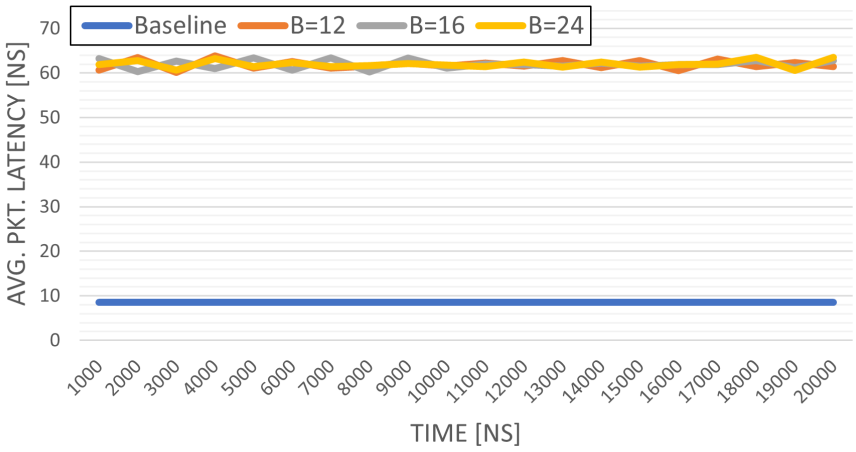


Figure 4.4: The effect of source throttling using different budget (B) values in a closed-loop simulation with an unrestricted injection rate of 2.5 flits/node/ns using six aggressor nodes and one victim node (see Figure 4.2d). In this simulation E=2, T=32.

Table 4.1: The configurations used during the simulation showed in Figure 4.4.

Configuration	Budget [flits]	Extra budget [flits]	Epoch [cycles]
Orange	12	2	32
Grey	16	2	32
Yellow	24	2	32

The results show that even though as little as 37.5% of all available timeslots were used during an epoch ($B/T = 0.375$), the nature of the attack pattern is such that the network is still too congested for the source throttling to influence the victim's latency. Setting the throttling budget to a value close to zero would offer better protection to the victim, but that would also severely degrade communication originating from the aggressor nodes.

Additionally, simulations with four, two, and one aggressors were carried out. The results of the simulations with two and four aggressors show the same pattern as the simulation with six aggressors, but indicate that when the number of aggressors decreases, the overall latency also decreases. In this case, the simulation with four aggressors resulted in roughly the same latency as the six aggressor case, with an

Table 4.2: Comparison of average victim latency increase compared to the baseline using source throttling with a differing amount of aggressor nodes.

Number of aggressors	Victim latency increase
6	7.4×
4	6.2×
2	1.4×

average victim latency increase of 6.2 times. The two-aggressor simulation resulted in an average victim latency increase of 1.4 times, showing that the network is significantly less congested. The comparison of the three cases can be seen in Table 4.2 and the simulation results can be seen in Appendix 1, Figures A.1, A.2.

In the simulation with only one aggressor, a different pattern can be discerned. When setting $B = T$, thus providing no isolation, a victim latency increase of 26% is noticed. However, as the budget is decreased, the victim latency increase due to the attack is reduced. With $B = 8$, corresponding to a usable timeslot quota of $B/T = 0.25$, the latency increase is 10%. An opposite effect is seen on the aggressor node, which has its latency increased by 172% when 0.25 of the timeslots in an epoch are available. A breakdown of the aggressor latency increase can be seen in Appendix Table A.1

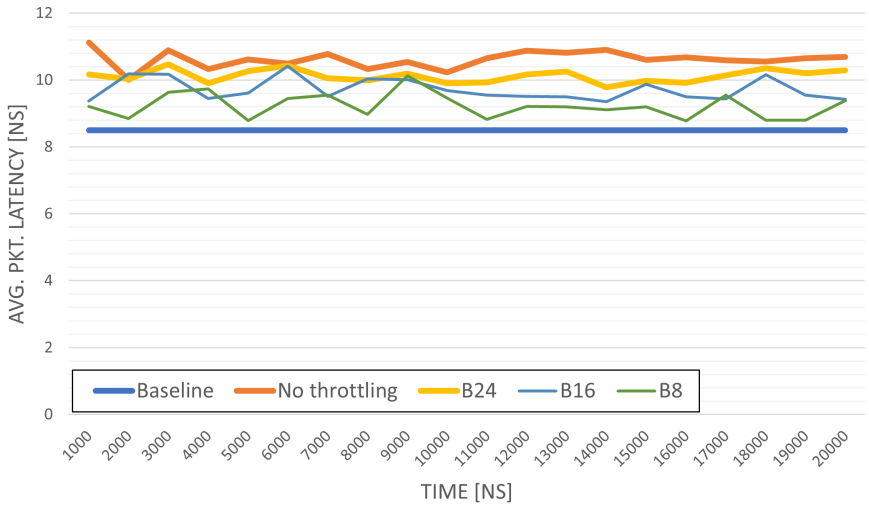


Figure 4.5: Effect of source throttling with one aggressor node and varying budget. As the budget decreases, so does the victim latency.

The results indicate that when the number of aggressors is larger than one, source throttling does not provide satisfactory isolation, as the network is still congested due to the number of nodes injecting malicious requests. Using fewer attackers leads to lower overall latency, but the increased latency because of the attack is still substantial.

When only one aggressor is used the source throttling instead has an effect on the victim latency. The latency increase due to the denial-of-service attack is decreased

Table 4.3: The latency decrease achieved when running the aggressor-victim traffic pattern in a system with 4 VCs per port with different amount of VCs allocated to the victim node.

Victim Configuration	Latency decrease
1 isolated VCs	63.9%
2 isolated VCs	82.3%
3 isolated VCs	84.5%

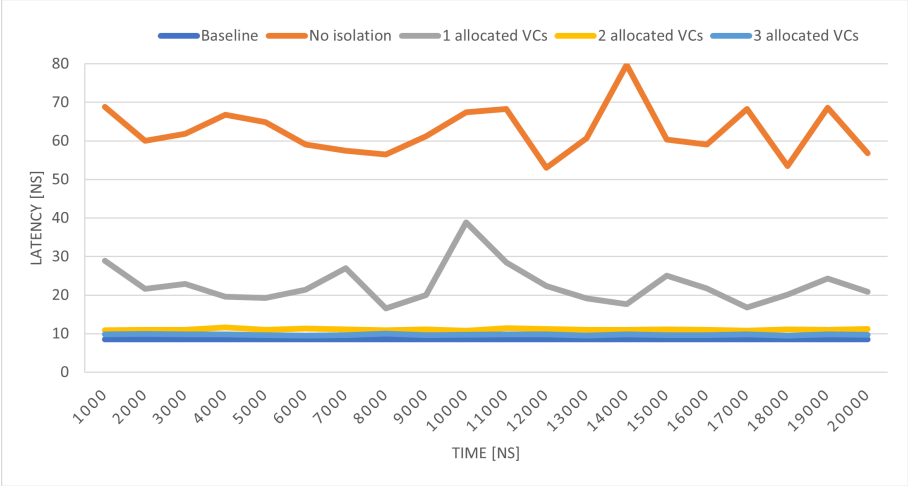


Figure 4.6: Comparison of the packet with latency using different configurations of fixed VC allocation using the aggressor-victim traffic pattern.

as the aggressor budget decreases. This indicates that for one aggressor, source throttling can give protection against denial-of-service attacks. However, as the aggressor is throttled, the aggressor latency instead increases substantially.

4.2.2 Fixed Virtual Channel Allocation

For the fixed virtual channel allocation technique different amount of isolated VCs were tested. When performing the Aggressor-Victim simulation with six aggressors, using different configurations of the number of virtual channels allocated to the victim, the effects of providing more virtual channels to the victim node can be seen.

The results are shown in Figure 4.6 and Table 4.3, indicating that allocating one fourth of the available VCs to the victim results in an average packet latency decrease of 63.9%. As more VCs are allocated this number approaches 85%. The largest difference in victim node performance is when going from zero to one allocated VCs. Allocating a second and third VC to the victim decreases the latency marginally, but diminishing returns can be observed.

Running the simulation with fewer aggressors resulted in the victim latency being decreased further. Using one and two aggressor nodes, one VC allocated to the victim and three VCs to the aggressors led to the victim node performing at the baseline level of latency, thus successfully completely masking out the denial-of-

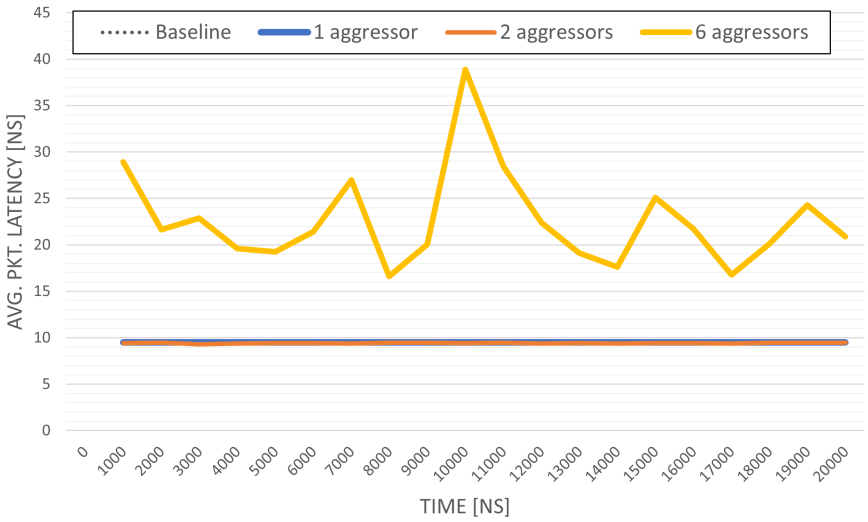


Figure 4.7: Victim latencies during a denial-of-service attack using different amount of aggressor nodes. In these simulations, one VC is allocated to the victim and the rest of the network shares the other VCs.

service attack at the cost of allocating one of four VCs to the victim. A graph of the simulations using different number of aggressors can be found in Figure 4.7.

From the evaluation results we can see that the fixed VC allocation technique provides a significant protection against a denial-of-service attack, effectively eliminating the latency that is introduced when 75% of the available VCs are allocated to the victim node in the scenario with six aggressors. However, allocating this many resources to a single node (or a trusted set of nodes) means that the other nodes receive almost no resources, limiting their throughput and increasing the packet latency.

The aggressive reallocation of resources can be reasonable if you are certain that the other nodes are malicious, but if you instead aim to isolate trusted nodes from all nodes that for example run third-party software, leaving this little resources for the third-party nodes might have performance implications, depending on the use case. Therefore, simply striving to reduce the latency of the victim node might not always be the best tactic, as the whole system has to be taken into consideration.

When only one or two aggressors are present allocating one VC to the victim flow is enough, meaning that the effect on the rest of the network will not be as large. Still, the optimal number of VCs for each flow will have to be empirically tested and fine-tuned for each system.

In summary, the fixed VC allocation technique provides good isolation for protection against this type of DoS attacks among the proposed techniques, by effectively isolating the victim flow from the aggressor flows. By enabling configurable allocation of the virtual channels, isolation from aggressor flows can be achieved, allowing the victim(s) to operate without being affected by the request flooding from the aggressors.

4.2.3 Separate Switch Allocator Timeslots

The performance of the third technique, separate switch allocator timeslots, depends on the chosen schedule. The challenge then lies in creating an appropriate schedule, allocating a suitable number of timeslots to each flow.

In the context of protection against denial-of-service attacks the goal is to remove the flow from the network congestion. To this end, a schedule which allows the victim node to pass through the network using only reserved datapath timeslots can be constructed. The two techniques of fixed VC allocation and separate switch allocator timeslots were combined, and a schedule configuration allocating half of the timeslots of the router ports along the route depicted by the green arrow in Figure 4.2 was constructed.

VC0 was reserved for the victim flow, while the remaining VCs were shared among the aggressors. In the schedule, described in Table 4.4, the number of unique timeslots was set to eight. All ports were set to be reusable if they were not actively used by the allowed VC or port. The table only includes the actively used routers and ports along the victim path, all the other routers and ports were set to be unreserved.

Table 4.4: The input allowed VCs (upper) and output allowed input port directions (lower) port tables used to provide isolation using fixed VC allocation and allocator timeslots. All timeslots are set to be reusable. The input port directions are **N**orth, **E**ast, **S**outh, **W**est, **R**esource, and **U** means Unreserved.

x	y	North	East	South	West	Local
0	1	U	U	U	U	00120023
1	1	U	U	U	00120023	U
2	1	U	U	U	00120023	U
2	2	00120023	U	U	U	U

x	y	North	East	South	West	Resource/Local
0	1	U	RRUURRUU	U	U	U
1	1	U	UUWWUUWW	U	U	U
2	1	U	U	WWUUWWUU	U	U
2	2	U	U	U	U	UUNNUUNN

The results of the simulation show that allocating one VC and half of the allocator timeslots to the victim flow effectively. This is shown in Figure 4.8. Compared to the average benchmark latency of 19.1 ns per packet, the isolation techniques reduced this to 11.6 ns, representing a speedup of 1.64x. However, the average latency of the aggressors was also increased from 106 ns to 164 ns, representing a 54% increase in latency.

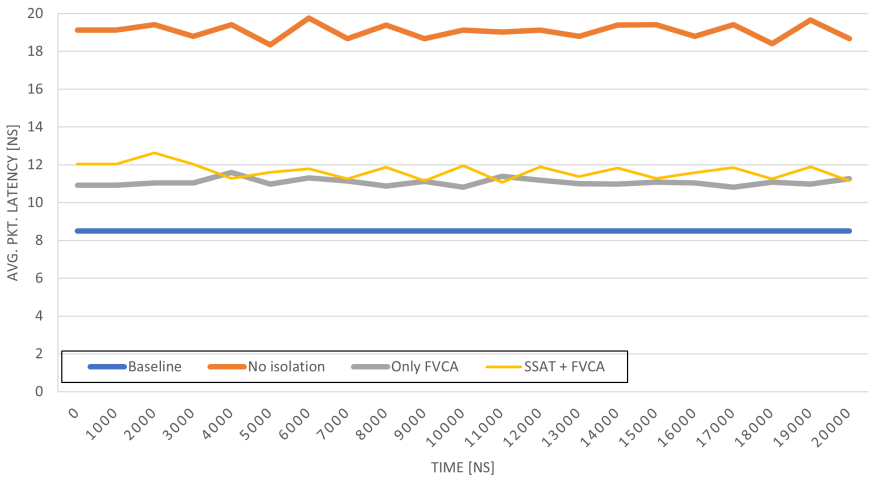


Figure 4.8: Latency results for separate switch allocator timeslots along with fixed VC allocation with an aggressor injection rate of 0.375 flits/node/ns. The benchmark latency when isolation is disabled and the results for only using fixed VC allocation are included for reference.

4.2.4 Denial-of-Service Protection Summary

In summary, the simulations of the denial-of-service attacks show that the performance penalties incurred from these attacks can be alleviated by using one or several of the techniques. Using source throttling proved to be effective when only one attacker was present. In this case, the victim latency was reduced as the aggressor was throttled. This provides better reliability in the performance of the victim node, as even though an attacker is present the victim node can still function normally. When more aggressors were present the amount of source throttling required to keep the network from becoming congested would render the aggressor nodes useless for normal operation.

Fixed VC allocation proved effective against denial-of-service attacks, isolating the victim traffic flows from the aggressors. With only one or two aggressors present, allocating one VC to the victim flows removed all effects of the denial-of-service attack. Further separating the traffic flows based on separate switch allocator timeslots also proved effective, but manually scheduling switch allocation timeslots did not provide any advantage over only using fixed VC allocation in our configuration. Allocating more timeslots to the victim would in turn increase the aggressor latencies, thus reducing the performance of other network activities.

4.3 Protection Against Timing Side-Channel Attacks

In timing side-channel attacks, the malicious node(s) exploit the fact that a victim node that transmits messages to a destination node takes up network resources, meaning that other nodes that transmit messages to that same destination node will

experience an increase in latency. By doing this, the malicious node can determine when, and possibly what, the victim is transmitting.

The goal of the isolation techniques in this scenario is to hide the latency increases caused by the victim node transmitting to the destination node. To that end, a version of the Aggressor-Victim traffic pattern is used, which features only one victim and one attacker. To enable the aggressor to measure the response time latency (RTL) of a message, the destination node sends a response back to the aggressor. A visualisation of the simulated traffic is shown in Figure 4.9. The packet size and response packet size for the simulations was set to 3.

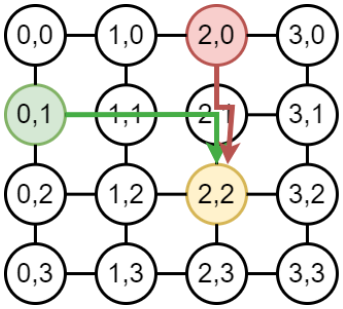


Figure 4.9: The traffic pattern used for simulating a timing side-channel attack. The aggressor relies on the response time latency to deduce whether the victim is transmitting.

To establish a baseline for the proposed NoC, it was simulated using an aggressor injection rate of 1.63 flits/node/ns and a packet size of 3. The NoC was configured to not provide any isolation in this simulation. The results of this simulation is shown in Figure 4.10, and show that the victim node experiences a latency of around 10 ns when transmitting. The aggressor node has a latency of around 12.5 ns when no victim traffic is present. When the victim starts transmitting, the latency rises to around 19 ns, an increase of over 50%.

This increase in latency comes from that fact that the two flows meet at node {2,1}, contending for resources in the shared router. Furthermore, it can be noted that there is also some effect on the latency of the victim flow, as the contention for resources leads to some flits having to wait longer than they would have otherwise had to.

4.3.1 Source Throttling

Source throttling allows for a reduction of the injection rate of a source by changing the budget values of the source throttling mechanism. Figure 4.11 shows the difference in latency of the aggressor node under different budget values.

As can be seen in the results, changing the number of allowed flits has an effect on both the latency for the whole simulation, and the latency difference when the victim node transmits. An injection rate of 1.63 flits/node/ns and 3 flits per packet corresponds to a delay between packets of 3.7 cycles. This means that for every four cycles, one packet will be injected into the network. With this injection rate,

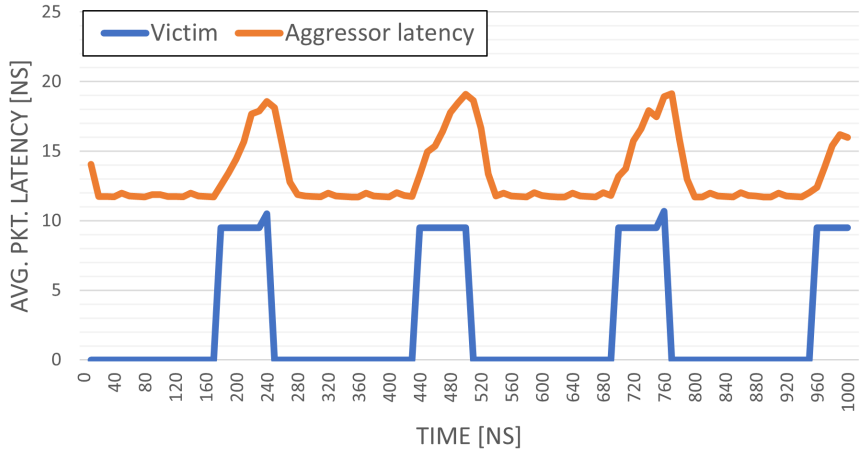


Figure 4.10: Victim and aggressor latency of the NoC when a timing side-channel attack is performed. The aggressor response time latency difference when the victim is transmitting clearly shows that victim traffic is present.

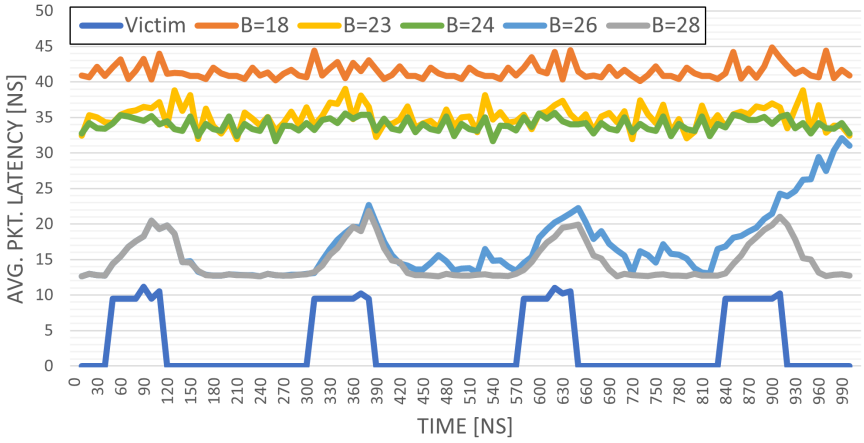


Figure 4.11: Comparison of aggressor response time latency under different budget values under a timing side-channel attack using an injection rate of 1.63 flits/node/ns. Victim injections included for visualisation purposes. The Extra budget was set to 2, and the epoch to 32 cycles.

the network is able to handle the traffic without incurring large latencies when no isolation is provided, shown previously in Figure 4.10.

However, when source throttling is introduced, differences in the latency for the aggressor can be seen. For high budgets, the traffic is not affected compared to the baseline case of the budget being equal to the epoch, $B = T$. The breaking point occurs when the throttle budget is set to 26 flits per epoch of 32 cycles. This is equal to allowing a quota of 0.81 of all cycles to be used, plus up to two extra cycles if the head-flit gets through the throttling mechanism. If the budget is decreased further, the latency will continue to increase as the quota of usable cycles decreases.

We can also observe that for B=26 we notice that after about 820 ns, the network saturates and the latency increases up to the level of the lower budget simulations. Looking further into this, we can examine the difference in throttling budgets. Figure 4.12 shows the difference in average packet latency when we compare the two throttling budgets in a simulation without a victim node. This indicates that for the injection rate of 1.63 flits/node/ns, a quota 0.84 of all cycles need to be usable in order to not saturate the network.

This discovery means that we can also examine what occurs when the injection rate is decreased. Figure 4.13 shows what occurs when the aggressor injection rate is reduced. In this case, we instead see a steady packet latency that is more stable, similar to that of the higher budget example in with the higher injection rate.

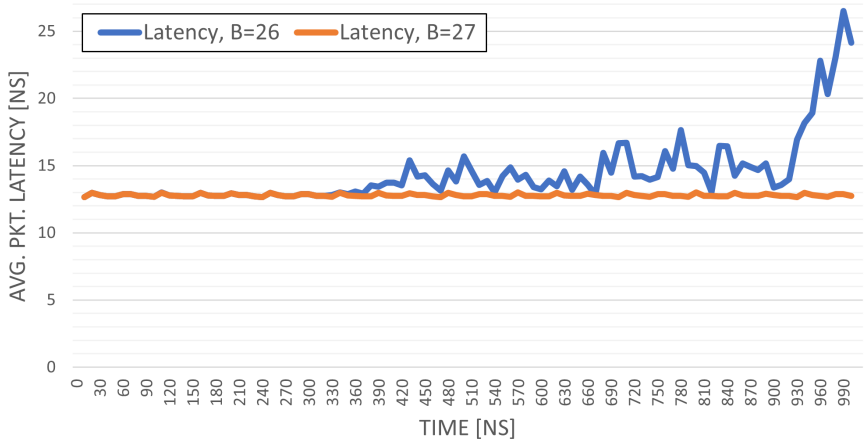


Figure 4.12: The difference in average packet response latency and injected packets between B=26 and B=27, showing that this is the point when the network eventually saturates. The injection rate is 1.63 flits/node/ns, E=2, T=32.

Drawing from this, a complete simulation with the lower injection rate was carried out. The results from this simulation are shown in Figure 4.14, showing that with this decreased injection rate, the saturation does not occur at the same quota of available cycles, reducing the latency of the aggressor traffic. However, this also reintroduces the latency difference, meaning that the protection against timing side-channel attacks is removed.

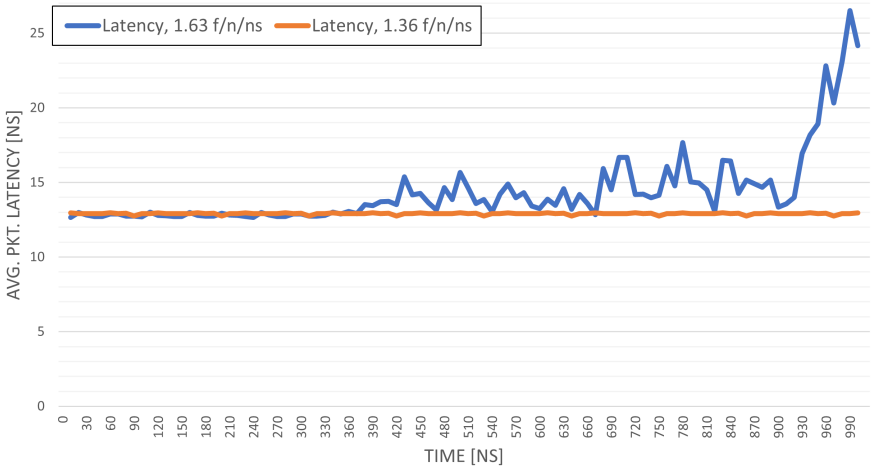


Figure 4.13: The difference in average packet response time latency when the injection rate is reduced from 1.63 to 1.36 flits/node/ns. There is no victim traffic and the configuration is: B=26, E=2, and T=32.

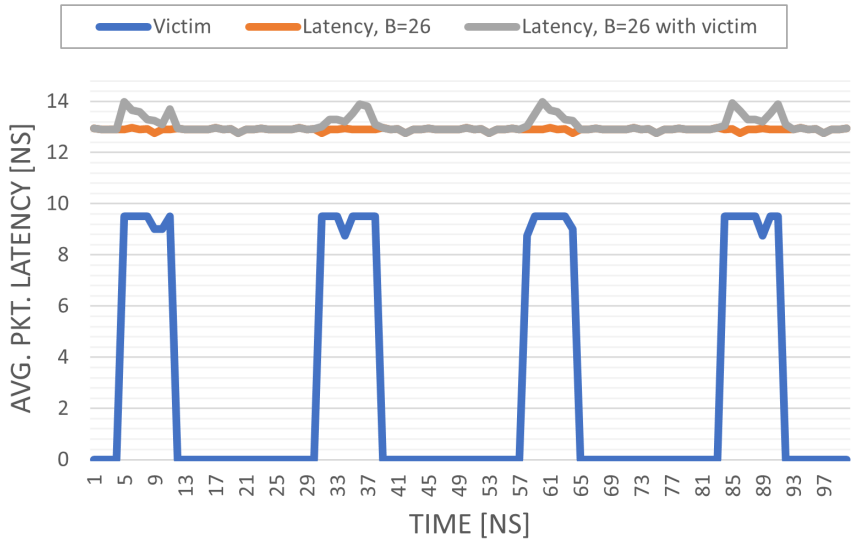


Figure 4.14: Simulation results with an injection rate of 1.36 flits/node/ns, B=26, E=2, T=32. The difference between victim traffic and no victim traffic can be seen in the aggressor latency differences pictured in orange and grey.

From these results, the following observations can be made:

- If a low budget is set, the latency difference seen when victim traffic is injected disappears, but the latency of the aggressor more than doubles.
- If an "optimal" budget for a certain aggressor injection rate is found, the latency difference reappears, potentially allowing timing side-channel attacks.

These observation show that if we opt for a high level of throttling with low quotas of available cycles the latency of the potential aggressor rises significantly. This might hurt the performance of nodes that are not aggressors, but still are throttled. Also,

if a budget level that does not incur this latency increase is found through empirical testing, it instead introduces the same latency differences as when no isolation is used.

The latency difference is not as obvious, but still visible and possible to be exploited by adversaries. Furthermore, this also means that an aggressor that is throttled and is penalised with the higher latency to remove the possibility of timing side-channel attacks can reduce its injection rate to again see the latency differences. This indicates that source throttling might not be an optimal isolation approach to protect against timing side-channel attacks.

4.3.2 Fixed Virtual Channel Allocation

Separating the virtual channels for the victim and aggressor flows has a small effect on the variation in the victim latency, as can be seen in Figure 4.15. Depending on the number of virtual channels allowed for the aggressors the latency differs, but the shape of the latency responses stay roughly the same.

As the results indicate, only isolating the flows through virtual channel allocation is not enough to properly mask out the increased congestion while the victim is transmitting. Since we only isolate the virtual channels the requests still contend for switch allocation, meaning that there is still contention in the network.

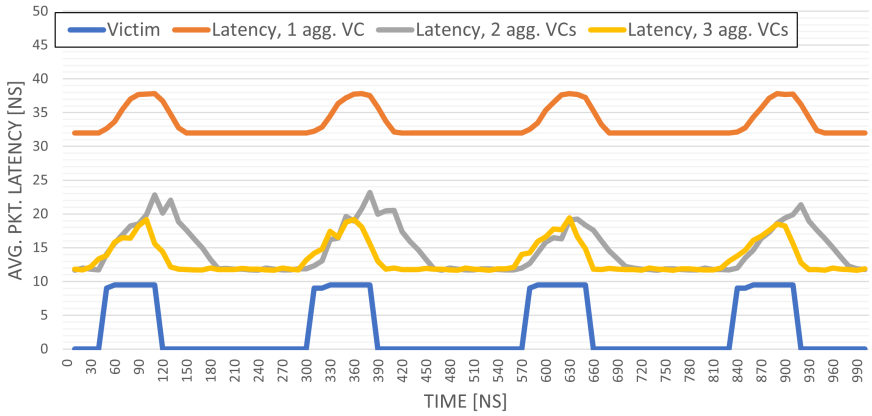


Figure 4.15: The latency difference of the aggressors when allocated 1, 2, and 3 VCs using the fixed VC allocation technique. The victim has one VC by itself. The latency difference is noticed in all three cases when the victim transmits.

4.3.3 Separate Switch Allocator Timeslots

In order to test the effectiveness of separating victim and aggressor flows by scheduling them in different timeslots, a schedule that separated them at their meeting point, node {2,1}, was created. The schedule configurations of the specific node with 8 separate timeslots can be seen in Table 4.5. All input other output timeslots were configured to not be reserved, meaning that any flow can use them. Additionally, VC0 was allocated to the victim flow and VC2 to the aggressor. The destination

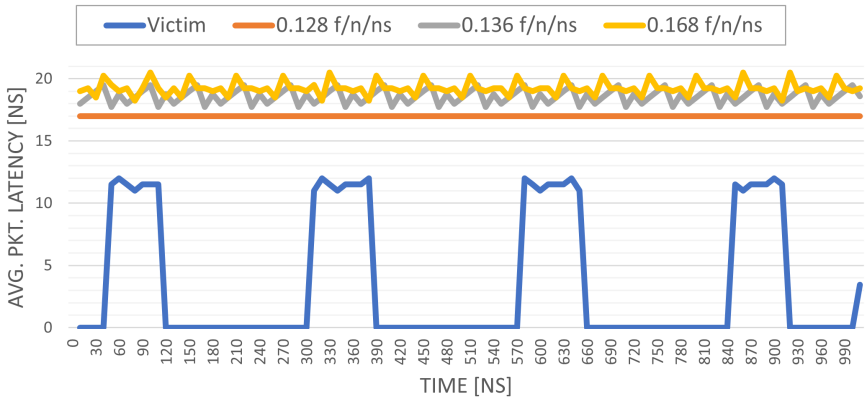


Figure 4.16: Aggressor packet latencies for the fixed VC allocation and separate switch allocator timeslots at different aggressor injection rates.

was configured to be able to use all VCs for the response packets. This schedule setup corresponds to giving each of the two nodes 37.5% of the available timeslots.

Table 4.5: The schedule configured for testing the separate switch allocator timeslots technique, showing the allowed input ports for the south output port. The victim packets come from the west port, and the aggressor from the north port. N = North, W = West, E = East, R = Local Resource.

x	y	Output Port South	Reusable
2	1	NNNWWWER	No

Several different injection rates were tested, with higher injection rates leading to higher latencies as packets had to wait in the network. Figure 4.16 shows the latencies for the injection rates, providing an aggressor latency that is not several times higher than the zero load latency. Appendix Figure A.3 shows all tested injection rates. The results indicate that when the table is configured as above, the isolation techniques are able to effectively remove the latency differences.

Even though isolation can be provided at all the tested injection rates, the latency of the aggressor node still needs to be kept in mind. With an injection rate of 0.375 flits/node/ns, the tested setup was able to provide a latency increase of 26%, corresponding to a speedup of 0.79× for the aggressors. Higher injection rates lead to higher latency increases, and lower injection rates had no difference compared to the value of the 0.375 flits/node/ns injection rate. Therefore, a latency increase is expected to be introduced using this technique with non-reusable timeslots.

Allowing the unused timeslots at node {2,1} to be reused by any source yields different results. Even though the average packet latency of the aggressor decreases somewhat, a differing pattern in the latency graph can be seen when the victim transmits, shown in Figure 4.17. This latency difference, although small, jeopardises the protection against timing side-channel attacks, potentially allowing an adversary to deduce when the victim is transmitting. The average packet latency decrease when allowing the unused timeslots in node {2,1} to be reused amounts to a speedup

Table 4.6: Response time latency differences for different injection latencies compared to the zero load latency.

Injection rate [flits/node/ns]	Latency	Speedup
Zero Load Latency	13500	N/A
0.375 f/n/ns	17000	0.79
0.4 f/n/ns	18590	0.73
0.5 f/n/ns	19245	0.70

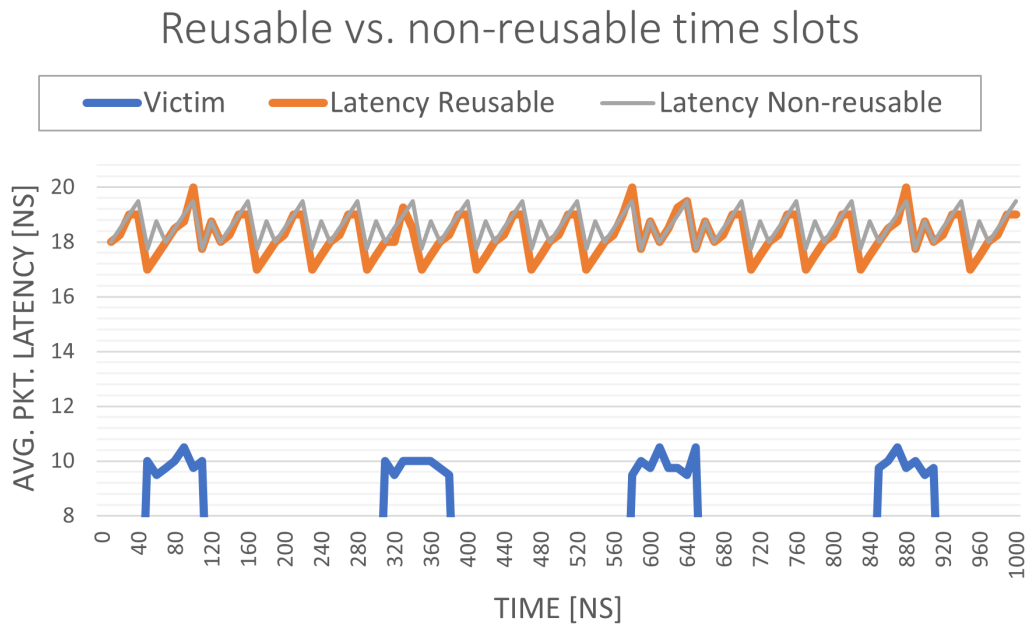


Figure 4.17: The difference in Response Time Latency (RTL) for the aggressor between allowing reuse of unused timeslots and not allowing it. A lower average packet latency is noted, but also a differing pattern when the victim is transmitting.

of 1.02×, or 2%.

By only allowing the victim node to reuse the unused timeslots the victim latency can be decreased somewhat while still keeping the same aggressor latency as in the case of no unused timeslots. Figure 4.18 shows the difference in victim latencies with different reusability configurations. We can see that only allowing the victim to reuse the timeslots brings the latency closer to the baseline case. This however comes at a cost of increased complexity of the scheduling, as the victim has to be scheduled to reuse the timeslots in its path to the destination node.

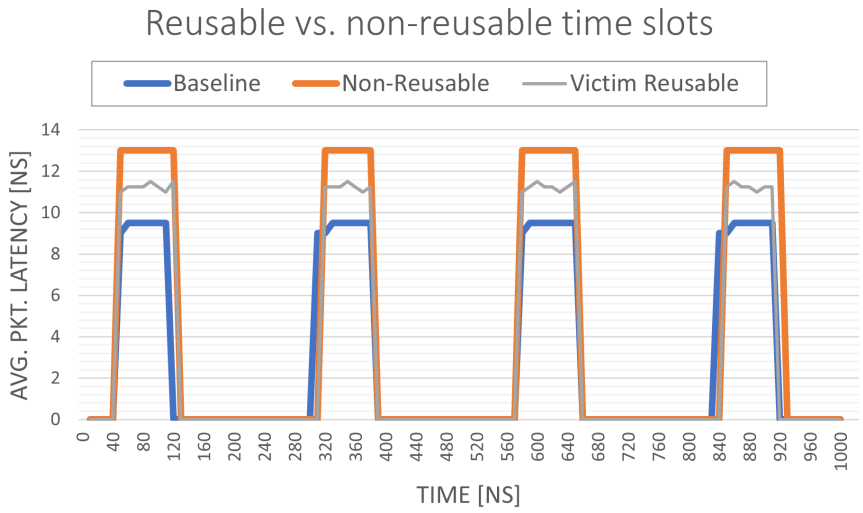


Figure 4.18: The difference in victim latency for different reusability schedules. By allowing only the victim to reuse timeslots a lower victim latency can be achieved.

4.3.4 Timing Side-Channel Protection Summary

In summary, the proposed implementation of source throttling does not provide a good protection against timing side-channel attacks. It is possible to remove the latency effect, but this comes at a cost of significantly higher latency for the aggressor. If all nodes running third-party code gets treated as a potential aggressor, this would mean severe performance implications for these nodes.

Furthermore, if the aggressor notices that it is being throttled, it can simply lower its injection rate to the point where the latency spikes are once again noticeable. These drawbacks show that source throttling is not a suitable technique to protect against timing side-channel attacks.

Fixed VC allocation as an independent technique is not able to protect against timing side-channel attacks. As could be observed in the results, the technique does not provide any meaningful protection against the latency effect, as the victim and aggressor are still fighting for switch allocation in the same timeslots.

Separate switch allocator timeslots together with fixed VC allocation does provide a good protection against timing side-channel attacks. By combining fixed VC allocation with scheduling the shared router timeslots so that no contention occurs between the victim and aggressor, the latency difference effect completely disappeared. This does however require careful planning of the timeslot scheduling, and an injection rate that is low enough to be handled by a subset of all available resources in the network. Allowing unused timeslots to be reused once again introduces the latency difference effect, but also reduces latency of the aggressor node slightly.

4.4 Performance Impact of the Proposed Techniques

Since the techniques add extra functionality to the NoC and modify several router pipeline stages, the potential performance penalty introduced by enabling the techniques can be determined by measuring the latency at the point of saturation. To that end, a simulation of uniform random traffic from each of the sources was performed both with a version of the NoC without any isolation techniques implemented and one with the isolation techniques enabled. During these simulations, the techniques were configured to not provide any isolation when they were enabled, so the network functioned like the techniques were not enabled. To achieve this, the following settings were used:

- Source Throttling: $B = T$ and $E = 0$, essentially letting T flits through during the time period of T flits, eliminating any effect of the throttling.
- Fixed virtual Channel Allocation: Configuring Table 3.2 so that all sources have access to all VCs, which is the same as if the technique would have been disabled.
- Separate Switch Allocator Timeslots: By configuring the input and output port tables to be unreserved for all timeslots.

By measuring the impact of the different techniques at the injection rate where the network saturates, the static overhead that each of the techniques introduce was recorded. Figure 4.19 shows the throughput and latency of the NoC with the isolation features enabled, but without providing any isolation, compared to the values when the standard NoC is used. From the results, we can see that the network saturates at the same injection rate, 1.3 flits/nodes/ns. However, the NoC with the isolation features has slightly higher latency for the measured points.

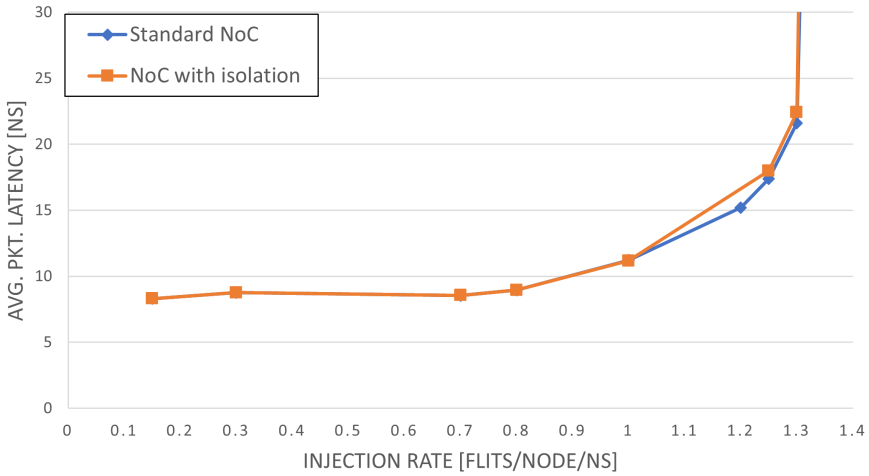


Figure 4.19: The performance achieved by the standard and isolation-enabled NoCs with no isolation enabled. A throughput difference of under 0.5% and a latency difference of 3.47% is noticed for the NoC with isolation features at the saturation point of 1.3 flits/node/ns.

At the saturation point, there are small differences in the performance, as can be seen in Table 4.7. At an injection rate of 1.3 flits/node/ns, the throughput difference is 0.47%, while the latency increases by 3.72%. These differences can be derived from the extended VA-stage in the fixed VC allocation technique, described in Figure 3.1. Latency measurements with each of the techniques used independently show a latency increase when the fixed VC allocation is being used. All in all, the static performance penalty for enabling isolation of flows is relatively small.

Table 4.7: The throughput and latency measurements when the injection rate is set to 1.3 flits/node/ns.

Configuration	Throughput	TP diff.	Latency	Lat. diff.
Standard NoC	0.00043383	100%	21614 ns	100%
NoC with isolation	0.00043179	99.53%	22417 ns	103.72%

4.5 Summary

The three proposed isolation techniques have been tested against two types of attacks that might occur in a NoC. First, a denial-of-service attack was simulated. The evaluation of the different techniques showed that for this specific type of attack, using a fixed VC allocation, where the victim node and aggressor nodes did not share any VCs proved to be the most effective. Using separate switch allocator timeslots in conjunction with the fixed VC allocation also proved useful, as a decrease in victim latency was achieved.

Source throttling did not show any promising results when the number of aggressors was larger than one. In these cases, the number of aggressor nodes were too many for the network to handle even when the throttling budget was decreased. With only one aggressor the source throttling technique showed promising results, as the latency of the victim node was decreased as the available timeslot quota of the aggressor was decreased.

In the timing side-channel attack scenario we were able to show that an aggressor can bypass the source throttling technique by lowering its injection rate until the latency differences become apparent again. Additionally, when the aggressor was throttled to the point that no latency difference was able to be noticed it experienced a severely increased average packet latency. With this information the aggressor could potentially notice that it is being throttled and adjust its injection rate accordingly.

Fixed VC allocation proved to not be useful on its own since the switch allocator is still shared among all sources. Through this, the aggressor could still notice the latency difference.

When using both fixed VC allocation and separate switch allocator timeslots the latency differences were able to be masked out. This came at a cost of increased aggressor latency, but provided protection against the attack. Allowing the victim to reuse timeslots also allowed the victim latency to be decreased while still providing protection from the attack. When any source could reuse idle timeslots, a small

latency difference could be noticed, potentially allowing the aggressor to perform the attack.

Regarding performance, a throughput decrease of 0.47% and a latency increase of 3.72% was noticed at the network saturation point, at an injection rate of 1.3 flits/nodes/ns. This means that the provided isolation only comes at a small performance cost to the system.

In summary, we have shown that fixed VC allocation provides good protection against denial-of-service attacks, and that combining this technique with separate switch allocator timeslots allows for protection against timing side-channel attacks. Source throttling can be used to protect against denial-of-service attacks with one aggressor, but was not able to protect against timing side-channel attacks.

5

Discussion and Conclusion

The design of the system is comprised of three separate approaches to providing isolation of flows, which based on the needs of the user can be used either alone or at the same time. The different approaches each have their own advantages and drawbacks, which will be discussed in this chapter. First, the source throttling approach will be discussed, followed by fixed virtual channel allocation, and separate switch allocator timeslots. The thesis is then concluded by summarising the isolation techniques and their performance for the different use cases presented in this thesis.

5.1 Source Throttling

In the two-stage proposed source throttling approach the user of the network can statically configure the throttling threshold, B, epoch, T, as well as the number of extra flits that should be allowed, E. This provides the user with a high degree of control over what how aggressive the throttling should be, by allowing a quota of $\frac{B+E}{T}$ of flits to be injected in the network.

The throttling is destination-aware, only throttling requests to a destination that has already reached the threshold. This approach avoids problems of throttling of requests to neighbouring destinations, which in turn reduces the risk of performance degradation due to erroneous throttling of the wrong requests.

While source throttling does not provide any explicit isolation of flows, the technique does offer a way of reducing the effectiveness of denial-of-service attacks in the network through reducing the congestion. The limitation of the number of requests that are allowed to enter the network during a given epoch can help with keeping the attack rate in the network at a low level. As presented by Fang, et. al., the deterministic XY routing algorithm, which is used in our implementation, then performs better than more complex adaptive routing algorithms [20]. Therefore, utilising the source throttling, a robust system can be achieved.

However, as seen in our evaluation, finding an amount of throttling that also did not affect the average latency of the aggressor nodes was difficult. From the aggressor's point of view, we tested two ways to perform this attack.

1. Injecting a large number of packets from one or a few sources
2. Injecting a slightly lower amount of packets from several sources

In the first case, our implementation of source throttling can be helpful, as the throttling is static and allows only a quota of the available timeslots in each epoch to be used for injection. The effectiveness of this approach can be seen in the

results with fewer aggressor nodes in the denial-of-service simulations. With only one aggressor a good protection against the attack was achieved, as the victim latency increase due to the congestion became less and less noticeable as the budget decreased.

The second case with several aggressor nodes allows the adversary to work around the throttling. Even if only half of the timeslots are allowed to be used, we still see a significant impact on the latency of the victim node. Even if the aggressors were throttled more than this, adding more aggressor nodes would still increase the network congestion. Allowing less than half of the avaialble timeslots to be used also substantially impacts the performance of the aggressor nodes in the cases where it is not being used for malicious intents.

These insights indicate that the technique can be used to protect against denial-of-service attacks, but that the aggressor can work around the protection by utilising more aggressors.

For timing side-channel attacks, the source throttling proved to not be useful, as the attacker can easily find ways around the injection limitations. To understand this result, we can observe that the goal of the timing side-channel attack is to observe latency differences in the response time latency which become apparent when the victim transmits.

From the simulation results we can see that when a source is throttled beneath the threshold, the latency difference is less noticeable. However, since there is still some contention among resources when the victim transmits, we observe a small variance in the latency in these cases. This means that even though we mask out most of the difference, a good detector could still notice this difference.

Another important observation is that the aggressor can work around the throttling by reducing its injection rate such that the number of injected flits per cycle is below the quota of $\frac{B+E}{T}$. If the number of injected flits is still large enough to notice the latency differences, the effects of source throttling can effectively be ignored by the aggressor by adjusting the injection rate.

Implementing source throttling comes at a cost. First, a source throttling controller monitoring the resource port of each of the routers must be added. This controller in turn needs to keep a table of destination counters, which keep track of the number of flits sent to each of the possible destinations. For a $N \times M$ mesh network, there are $N \cdot M - 1$ possible destinations, causing the resources needed for keeping track of flits to scale exponentially with the number of nodes. Counting the number of flits without having to keep large tables might improve the resource efficiency of the design.

A way to reduce the number of tables for each source throttling controller could be to throttle in directions instead of destinations. This would mean that the number of tables would be the same as the number of possible directions, a maximum of four in the case of a quadratic mesh topology.

This approach would however need to be evaluated using different routing algorithms, since the choice of algorithm could influence which ports are used. For example, using dimension ordered XY routing, the east port of node (0,0) of a $N \times N, N > 1$ mesh network would be used more than the south port under uniform

random traffic.

Another potential improvement involves the parameters of the source throttling implementation. In our implementation, the configuration of the throttling parameters is static, which means that the optimal parameters for the specific use case have to be found empirically. A possible improvement is to introduce an adaptive throttling budget by analysing the network usage over time, detecting and throttling unusual or disallowed traffic patterns. A similar approach is presented in [8].

5.2 Fixed Virtual Channel Allocation

The fixed Virtual Channel Allocation technique isolates flows by only allowing a predetermined set of virtual channels for each of the source nodes. This technique has proved useful when handling denial-of-service attacks.

Furthermore, the technique allows the separate switch allocator timeslots technique to function in a predictable way, as the VC of each of the flows can be decided. However, the technique also comes with a few drawbacks, that are mainly related to the static nature of the configuration and the implementation overhead.

The technique replaces the virtual channel allocator, negatively impacting the performance of the system. The additional steps in the modified virtual channel allocation leads to a latency increase of 3.72% and saturation throughput decrease of 0.47%. This latency increase is present even when no isolation is configured, but the technique is still active.

This is however deemed as an acceptable amount of overhead since the aim of the technique is not to increase performance, but rather to provide isolation of flows. Having the technique enabled but not isolating any virtual channels is also not a realistic use case for the technique, since the point is to separate traffic flows.

When used to handle denial-of-service attacks, the latency decreases when allocating VCs to the victim are significant. By allocating one of the available VCs, corresponding to one fourth of the total VC count, we were able to decrease the latency by as much as 63.9% when six attackers were present. This number rises to 82.3% when allocating half of the available VCs to the victim node. As more VCs are allocated to the victim node, the closer the latency gets to the baseline case of no network congestion.

In the case with fewer attackers, the results were also promising. With one and two aggressor nodes, allocating as little as one VC to the victim node allowed us to completely mask out the effect of the attack, resulting in a victim latency that was equal to the baseline case.

This means that fixed VC allocation provides effective protection against DoS attacks, but also affects other parts of the network. Therefore, a compromise between the importance of decreasing the victim flow latency and affecting other parts of the network must be considered.

Opting for an aggressive reallocation of resources can be reasonable if you are certain that the other nodes are malicious. If you instead aim to isolate trusted nodes from all nodes that for example run third-party software, leaving only a small amount of

resources for the third-party nodes might have performance implications, depending on the use case. Therefore, simply striving to reduce the latency of the victim node might not always be the best tactic, as the whole system must be taken into consideration.

For future work, the arbitration and reservation of VCs could be moved back to the VA module, so that the traditional structure of the router is kept intact. As the general idea is still the same, allocating free VCs to an incoming request, this change should be feasible.

Another potential future improvement relates to the static configuration of the allocated virtual channels, which cannot be changed during runtime. This means that if a DoS attack occurs during runtime and the NoC has not been configured to allocate the correct VCs beforehand, nothing can be done about it while the system is still running. A potential future improvement could be to enable adaptive allocation of VCs to high priority flows so that attacks or unforeseen congestion problems that occur during runtime can be handled automatically by the network.

To enable this, a baseline performance could be determined either by measuring it during warm-up or continually during runtime and detecting when the latency deviates from this baseline. This could then be used to allocate more resources to high-priority nodes if the latency increases above a threshold compared to the baseline. To this end, the Fixed Virtual Channel Allocation technique could perhaps be combined with the QNoC service classes presented by Bolotin, et al. [11]. Nodes that send high-priority requests could be allocated more virtual channels if there is a decrease in performance for a crucial part of the network, affecting the latency of the high-priority packets.

This type of improvement would make the system more adaptive and resilient regarding unforeseen problems as well as being easier to configure, as the traffic does not have to be analysed before configuring the system. The challenge lies in implementing the changes without further increasing the performance penalties that are present with this technique.

5.3 Separate Switch Allocator Timeslots

The separate switch allocator timeslots technique enables configuration of a schedule which effectively removes the threats of timing side-channel attacks when combined with fixed VC allocation. Using the two techniques, the network can be configured to only let through traffic from certain VCs and directions at the different timeslots.

These timeslots can be configured in a way such that a high priority node encounters a minimal amount of congestion, while also masking whether it is using its timeslots from potential adversaries. Thus, flits from different sources do not encounter each other in either the VA or SA stages, providing isolation between them.

The advantages of this technique are most noticeable during the timing side-channel attack scenario, since this isolates flows during the switch allocation process, preventing adversaries from deducing information based on this information. When combined with fixed VC allocation, latency differences could be completely masked out while also only incurring a small amount of additional average packet latency

to the aggressor.

The fact that only a minimal amount of average packet latency for the aggressor is incurred means that the techniques could potentially be used for nodes running third-party code that is not trusted. This can then be used as a precaution to protect against timing side-channel attacks while not penalising the nodes too much if they are not actually malicious. Note however that this still requires a properly set up schedule, which can be a complex task.

The technique also proved useful in the context of a denial-of-service attack, given a properly configured allocation schedule. By building an isolated tunnel between the victim and destination nodes, even allocating half of the available timeslots proved to significantly decrease the latency experience by the victim.

The performance of this techniques relies upon the schedule that the user configures, meaning that the performance penalty for a poorly configured schedule can be large. The user must also make sure to not accidentally exclude any possible flows from the schedule, which might cause blocking in the network.

This problem was noticed when fully randomised tables were tested. A certain degree of planning is therefore needed in all cases. This might pose a security problem, if an adversary finds a way to deduce the structure of the schedule by analysing the traffic of the network. This could perhaps be remedied by mixing randomisation with planned scheduling.

Given the high degree of configuration that is available, certain paths in the network can be configured to feature isolation, while others do not. This means that if there are known routes for high-priority traffic between two nodes in a part of the network, the timeslot allocation can be set up such that the isolation is only present in this part of the network. The rest of the network can then operate like normal, without any negative effect on the network except for the increased area needed due to the tables.

Furthermore, the decision to allow unused timeslots to be reused or not also affects performance. Depending on the number of timeslots that go unused, the performance difference between allowing and not allowing these timeslots to be reused can be significant. As could be seen, allowing for unused timeslots to be reused by any source node potentially allows aggressors to perform a timing side-channel attack.

With this in mind, the option for any source to reuse unused timeslots should only be used when performance is more important than protection against these types of attacks. In the special case with the unused timeslots along the victim-destination path being reserved for the victim node, no latency difference could be noticed by the aggressor node, removing the possibility of a timing side-channel attack.

The drawback of allowing only a specific source node to reuse timeslots along a path is that the when this source is not transmitting the network is essentially not allowing these timeslots to be reused. Manual scheduling is also required for configuring this behaviour, further increasing the configuration complexity of this technique.

There will always be a trade-off between security and performance when using these techniques. For instance, a timeslot that is allowed to be reused can allow for timing side-channel attacks, but also reduces the performance penalty by allowing traffic

that would otherwise be blocked to progress through the network. We believe that the choice of how much security and performance should matter differs from case to case, and that the system designer must carefully consider the alternatives and their effects when deciding what configuration to choose.

The main drawback of the proposed technique is the difficulty of configuring the input and output port tables such that they do not impose any unnecessary performance decrease in the network. The option to only reserve some of the timeslots for both input and output ports, as well as allowing unused timeslots to be reused, decreases the effects of this problem. However, the user must still ensure that enough timeslots for all parts of the network are available to avoid blocking and performance decreases.

An improvement that would ease the task of producing a working schedule would be to have a schedule evaluator that scans the schedule for potential blocking problems and locations where a performance decrease due to congestion is likely to occur. This evaluator could also check to see that isolation between two defined sources is provided. Such an evaluator would not have to be part of the system and could instead be built independently and used before loading the schedule onto the system.

A further possible future improvement could be to utilise the fact that when timeslots are reserved, switch allocation is not required. By preemptively granting the scheduled input port access to the crossbar, the switch allocation stage can be completely bypassed. Then, if VC allocation can also be skipped, or performed a cycle earlier, an entire pipeline stage could be removed. This would reduce the pipeline to two stages, Switch Traversal and Link Traversal.

The problem with this approach would be the reusability functionality. If we grant the potential incoming request access to the crossbar before we know if it is there, a look-ahead signal would need to be sent from the upstream router, indicating whether a request is going to occupy its granted timeslot or not. Furthermore, if several sources can potentially use the reusable timeslot, switch allocation would still need to be performed among those potential requests.

In any case, the possibility of further reducing the depth of the pipeline for reserved timeslots is an interesting topic which can be further investigated in future work.

5.4 Comparison with Related Works

The combined system provides a similar but different approach to protection against timing side-channel attacks as compared to the approach described by Wang and Suh [5]. The source throttling mechanism is improved to combat some of the potential drawbacks of a more general approach and the virtual channels are allocated on a per-source basis instead of using classes, allowing for more flexibility in configuring which flows should be allowed to use which VCs.

Furthermore, we implement the strategy of temporal network partitioning of switch allocators in the separate switch allocator timeslots approach instead of allowing lower-priority flits to always win arbitration. This static scheduling approach requires more manual work in creating schedules, but in return allows the flows to split the network usage instead of providing a static priority.

To remedy the performance decreases that temporal network partitioning introduce, the high degree of timeslot reusability possibilities allows the system designer to decide what resources should be allowed to use unused timeslots. By allowing each timeslot to be configured according to the specific needs, the possibility of allowing timeslots belonging to lower security flows to be reused, while high-security flows are not. This provides protection against timing side-channel attacks, while reducing the performance penalty of this approach.

5.5 Conclusion

Three techniques aiming to provide isolation and handle two types of attacks in a Network-on-Chip have been presented. The first technique, source throttling, enables throttling of sources that inject packets into the network at a rate the exceeds the pre-configured threshold. The throttling is carried out in two steps, flit counting and budget check. Finding the correct configuration of throttling parameters to protect against a denial-of-service attack proved difficult when multiple aggressors were present. With a single aggressor, source throttling reduced the effectiveness of the attack, providing a more stable system.

In the case of timing side-channel attacks, we showed that by lowering the injection rate to be lower than the throttling quota, attackers could still perform attacks even if they were throttled. This indicates that source throttling cannot be used as a reliable method of preventing timing side-channel attacks in Networks-on-Chip.

The second technique, fixed virtual channel allocation, allows the user of the NoC to configure which sources should have access to which of the available virtual channels. Although this technique alters the router pipeline by replacing and extending the switch allocation stage, latency decreases of up to 84.5% were observed when isolating a victim flow during a denial-of-service attack. Allocating as little as one fourth of the available VCs to the victim node resulted in a 63.9% average packet latency decrease as compared to running the system without isolated VCs.

The third technique, separate switch allocator timeslots, allows the user to pre-configure a schedule which allocates specific allocator timeslots in the input and output arbitration stages of the switch allocator. Using this technique combined with fixed virtual channel allocation it is possible to completely remove the possibility of noticing a victim transmission using a timing side-channel attack.

The cost of this protection is twofold. The schedule has to be manually configured, and the fact that less timeslots are allocated to each flow increases the packet latency. With this in mind, all effects of a timing side-channel attack were able to be masked out at the cost of a 26% increase in aggressor latency, compared to the zero load latency.

At the saturation point, a latency increase of 3.72% and a throughput decrease of 0.47% was recorded when using the proposed NoC without any isolation enabled, compared to the standard NoC. This increase in latency is related to the changes and extensions of the router pipeline to enable the isolation techniques.

In conclusion, the three techniques presented all take different approaches to providing isolation of flows. Source throttling is aimed towards handling congestion-

related problems, while fixed virtual channel allocation and separation of allocator timeslots are aimed towards providing complete separation of flows. This prevents adversaries from deducting sensitive information based on the usage of the network. The techniques can be used by themselves, or in combination with each other, and are entirely configurable by the user.

The results indicate that source throttling can be bypassed by attackers in both attack cases, and most importantly does not provide any protection from a timing side-channel attack. Fixed virtual channel allocation provided the most efficient protection against denial-of-service attacks, while a combination of this technique and separate switch allocator timeslots effectively masked out attempts to perform timing side-channel attacks.

Bibliography

[1] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[2] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1–es, jun 2006. [Online]. Available: https://doi.org/10.1145/1132952.1132953

[3] G. Kim, M. M.-J. Lee, J. Kim, J. W. Lee, D. Abts, and M. Marty, "Low-overhead network-on-chip support for location-oblivious task placement," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1487–1500, 2014.

[4] E. Baydal, P. Lopez, and J. Duato, "A congestion control mechanism for wormhole networks," in *Proceedings Ninth Euromicro Workshop on Parallel and Distributed Processing*. IEEE, 2001, pp. 19–26.

[5] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 142–151.

[6] C. Wang, K. Long, X. Gong, and S. Cheng, "Swfq: a simple weighted fair queueing scheduling algorithm for high-speed packet switched network," in *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240)*, vol. 8. IEEE, 2001, pp. 2343–2347.

[7] B. Grot, S. W. Keckler, and O. Mutlu, "Preemptive virtual clock: A flexible, efficient, and cost-effective qos scheme for networks-on-chip," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 268–279.

[8] K. K.-W. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu, "Hat: Heterogeneous adaptive throttling for on-chip networks," in *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, 2012, pp. 9–18.

[9] V. Soteriou, H. Wang, and L. Peh, "A statistical traffic model for on-chip interconnection networks," in *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, 2006, pp. 104–116.

[10] R. Prolonge and F. Clermidy, "Network-on-chip traffic modeling for data flow applications," in *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2432516.2432518

[11] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Qnoc: Qos architecture and

design process for network on chip," *Journal of systems architecture*, vol. 50, no. 2-3, pp. 105–128, 2004.

[12] H. M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Networks on chip with provable security properties," *IEEE Micro*, vol. 34, no. 3, pp. 57–68, 2014.

[13] M. Liu *et al.*, "Improving the performance of a wormhole router and wormhole flow control," Ph.D. dissertation, Citeseer, 2005.

[14] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu, "Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip," in *2009 WRI Global Congress on Intelligent Systems*, vol. 3. IEEE, 2009, pp. 329–333.

[15] D. U. Becker, *Efficient microarchitecture for network-on-chip routers*. Stanford University, 2012.

[16] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 2009, pp. 1–12.

[17] Y. Lu, C. Chen, J. McCanny, and S. Sezer, "Design of interlock-free combined allocators for networks-on-chip," in *2012 IEEE International SOC Conference*, 2012, pp. 358–363.

[18] M. Tang, X. Lin, and M. Palesi, "Local congestion avoidance in network-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2062–2073, 2015.

[19] L. Garber, "Denial-of-service attacks rip the internet," *Computer*, vol. 33, no. 04, pp. 12–17, 2000.

[20] D. Fang, H. Li, J. Han, and X. Zeng, "Robustness analysis of mesh-based network-on-chip architecture under flooding-based denial of service attacks," in *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*. IEEE, 2013, pp. 178–186.

[21] C. Reinbrecht, A. Susin, L. Bossuet, and J. Sepúlveda, "Gossip noc – avoiding timing side-channel attacks through traffic management," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 601–606.

A

Appendix 1

A.1 Simulation Results Denial-of-Service Attack

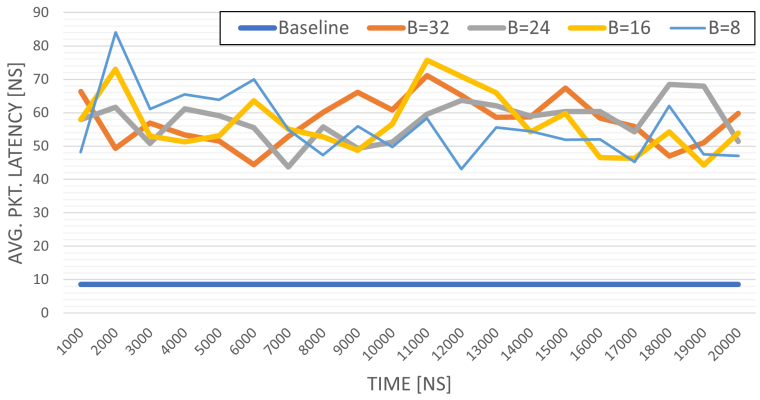


Figure A.1: Denial-of-service attack simulation results with four aggressors and one victim. The budget varies while E=2, T=32

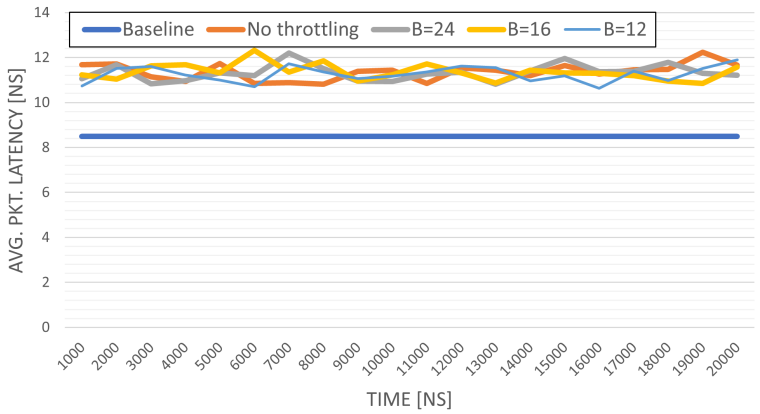


Figure A.2: Denial-of-service attack simulation results with two aggressors and one victim. The budget varies while E=2, T=32.

Table A.1: The aggressor latency with one aggressor and different throttling budgets. For all simulations, E=2, T=32.

Budget	Average aggressor latency	Increase vs. no throttling
32	23596.86408	0%
24	22896.71975	-3%
16	33015.81823	40%
8	64226.05382	172%

A.2 Simulation Results Timing Side-Channel Attack

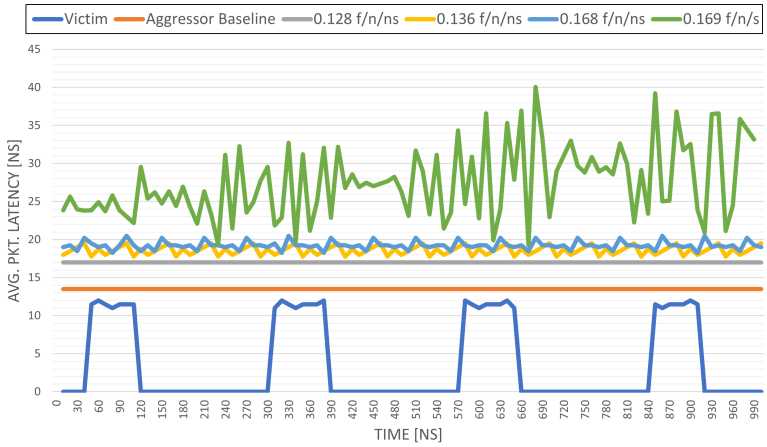


Figure A.3: Simulation results for different injection rates using fixed VC allocation and separate allocator time slots, using the schedule described in section 4.3.3. The network saturates at 0.168 flits/node/ns, leading to a large increase in latency when that point is passed.