Real-Time Communication Services for Networks on Chip

Zheng Shi

Submitted for the degree of Doctor of Philosophy

Computer Science

The University of York

November 2009

Abstract

Networks-on-Chip (NoCs), have emerged as a new design paradigm to overcome the limitation of current bus-based communication infrastructure, and are increasingly important in today's System-on-Chip (SoC) designs. This architecture offers a general and fixed communication platform which can be reused for a large number of SoC designs. As an integration platform, the NoC should have the capability to provide different level of services for various applications on the same network. There is one kind of communication service, namely real-time communication, in which the correctness relies on not only the logical communication result but also on the completion time bound. In order to meet the requirements of critical functions, system level verification or performance predictability, real-time communication service in NoCs is strongly desired.

Some researchers have introduced priority-based wormhole switching to provide the real-time communication service for on-chip networks. But the schedulability analysis in current results suffers from problems of pessimism or defect. The contributions of this thesis present a group of theoretic analysis works to overcome these problems. In the first part, we present worst-case network latency analysis to validate whether each real-time communication service can meet its timing bound. In order to solve the priority assignment problem, in the second part, we design an efficient branch and bound search algorithm to find the possible priority ordering. To improve the resource utilization, in the third part, a priority share policy is proposed. The relevant schedulability analysis and priority allocation are presented in such a way that the timing requirements are met with a reduced resource overhead. To better validate our analysis technique, a realistic application is simulated on a NoC platform. The simulation results provides evidence that communication latency calculated using the real-time analysis approach is safe, closely matching

the figures obtained from simulation. Utilizing these new approaches, we could flexibly evaluate at design time the schedulability of each real-time service in an on-chip network platform.

Contents

1 Introduction **15**

1.1 Real-Time Communication Service 18

1.1.1 Hard Real-Time Communication 19

1.1.2 Soft Real-Time Communication 20

1.1.3 Non Real-Time Communication 21

1.2 Wormhole Switching 21

1.3 Motivation, Goal and Contributions 23

1.4 Thesis Structure 28

2 On-Chip Networks and Real-Time Service **29**

2.1 On-Chip Communication Network 30

2.1.1 Topology 30

2.1.2 Switching 32

2.1.3 Routing 36

2.1.4 Flow control 37

2.2 Implementation Schemes for Real-Time Services 39

2.2.1 Resource Reservation Discipline 40

2.2.2 Run-Time Arbitration Discipline 46

2.2.3 Summary 53

2.3 Static Analysis of Wormhole Switching 53

2.3.1 Network Model and Constraints 54
2.3.2 Traffic-Flow Model 56
2.3.3 Inter-Relationships between Traffic-Flows 58
2.3.4 Mutka's Contribution 62
2.3.5 Balakrishnan's Contribution 63
2.3.6 Kim's Contribution 64
2.3.7 Lu's Contribution 67
2.4 Summary 69

3 Schedulability Analysis for Priority Based Wormhole Switching 70
3.1 Basic Network Latency Analysis 71
3.1.1 Direct Interference From Higher Priority 73
3.1.2 Indirect Interference From Higher Priority 74
3.1.3 Numeric Examples 81
3.2 Extending the Analysis with Self-Blocking 84
3.2.1 Analysis with Self Blocking 85
3.2.2 Another Example 90
3.3 Parallel Communication and Tightness Analysis 92
3.3.1 Parallel Communication 92
3.3.2 Tightness Analysis of Our Model 95
3.4 Evaluation 100
3.4.1 Experiment Setup 100
3.4.2 Results of Experiments 102
3.5 Summary 106

4 Priority Assignment for Priority Based Wormhole Switching 108
4.1 Priority Assignment and Motivation Case 109
4.2 Properties of Priority-Based Wormhole Switching 112

4.3 Lower and Upper Bounds of Worst-Case Network Latency 114
4.4 Branch and Bound Priority Assignment 119
4.4.1 Heuristic Functions 121
4.4.2 Backtracking policy 124
4.5 Evaluation 125
4.5.1 Evaluation of Different Heuristic Functions 125
4.5.2 Evaluation of Different Priority Assignment Schemes . . . 128
4.6 Summary 132

5 Priority Share Support in Priority Based Wormhole Switching 134
5.1 Wormhole Switching with Priority Share 135
5.1.1 Priority Share Structure 135
5.1.2 Blocking Problem 138
5.1.3 Traffic-flow Model and Inter-relationships 140
5.2 Schedulability Analysis with the Composite Model 142
5.2.1 Properties of Priority Share 142
5.2.2 Analysis with the Composite Model 144
5.2.3 An Example 149
5.3 Schedulability Analysis with Time Window 150
5.3.1 New Blockings Without the $D \leq T$ Constraint 151
5.3.2 Priority Window Model 153
5.3.3 Maximum Network Latency 156
5.3.4 Another Example 161
5.4 Group Priority Allocation 163
5.5 Evaluation 167
5.6 Summary 173

6 Simulation **174**

6.1 Application Model 174

6.2 Simulation Platform and Application Mapping 176

6.3 Theoretic Analysis 180

6.4 Experimental Results 182

6.5 Summary 186

7 Conclusion and Future Work **188**

7.1 Contributions of the Thesis 188

7.2 Future Work 190

List of References **191**

List of Figures

1.1 Network-on-Chip Architecture 17

1.2 Wormhole Switching in Pipeline 22

1.3 Blocking in Wormhole Switching 23

2.1 Network Topology 31

2.2 Three Packet Switching Techniques 33

2.3 Virtual Channel 36

2.4 Æthereal Contention Free Router 43

2.5 Architecture of SDM Router 46

2.6 A General Architecture of Priority Arbitration 49

2.7 ALG Scheduling Mechanism Structure 51

2.8 An Example Set of Traffic-Flows 59

2.9 Transitivity in Indirect Competing Relationship 61

2.10 Initial Timing Diagram 66

2.11 Final Timing Diagram 66

2.12 Contention Tree 68

2.13 Scheduling with Contention Tree Model 68

3.1 A Case of Indirect Interference 75

3.2 The Problem of Indirect Interference 75

3.3 Upper Bound Analysis of Indirect Interference 78

3.4 Unschedulable under Kim's Approach 83
3.5 A Modified Case of Traffic-Flows Set 85
3.6 Priority Level-5 Busy Period 86
3.7 Priority Level-i Busy Period with Multiple Packet Arrivals 87
3.8 Scheduling Sequence With Parallel Communication 94
3.9 A General Parallel Communication Model 97
3.10 The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 30 103
3.11 The Pass Ratio Under Varied Average Link Utilization With Flow Set 30 103
3.12 The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 60 104
3.13 The Pass Ratio Under Varied Average Link Utilization With Flow Set 60 104
3.14 The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 90 105
3.15 The Pass Ratio Under Varied Average Link Utilization With Flow Set 90 105

4.1 A Case of Real-Time Traffic-Flows 110
4.2 Timing Graph with Different Priority Ordering 111
4.3 Transformation of Priority Ordering 118
4.4 Heuristic Branch & Bound Search Algorithm 120
4.5 The Number of Priority Assignment Operations Under Varied Maximum Link Utilization 126
4.6 The Number of Priority Assignment Operations Under Varied Average Link Utilization 127

4.7 The Number of Priority Assignment Operations With Varied Number of Traffic-Flows Set Under 55% Maximum Link Utilization . 128
4.8 The Number of Priority Assignment Operations With Varied Number of Traffic-Flows Set Under 32% Average Link Utilization . . . 129
4.9 The Pass Ratio Under Varied Maximum Link Utilization 130
4.10 The Pass Ratio Under Varied Average Link Utilization 130
4.11 The Pass Ratio With Varied Number of Traffic-Flows Set Under 55% Maximum Link Utilization 131
4.12 The Pass Ratio With Varied Number of Traffic-Flows Set Under 32% Average Link Utilization 132

5.1 Output Arbitration with Priority Share 136
5.2 Traffic-flows with Distinct Priority 138
5.3 Traffic-flows with Priority Share 138
5.4 Two Possible Situations for Indirect Interference 146
5.5 The Blocking Problem 152
5.6 Priority Level-G_2 Window 154
5.7 Three Possible Relations Between Priority Window and Period . . 157
5.8 Priority Level-G_i Window with Self-blocking 160
5.9 Share Priority Assignment Algorithm 165
5.10 The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 30 168
5.11 The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 30 169
5.12 The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 60 170

5.13 The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 60 171
5.14 The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 90 171
5.15 The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 90 172

6.1 Automotive Vehicle Navigation and Stability Control System Model 175
6.2 Application Tasks 177
6.3 Inter-Task Traffic Flows 178
6.4 Mapping of the Application Tasks onto Processors Interconnected by a 4x4 NoC Mesh 179
6.5 Link Competing Relationship between the Traffic-Flows 181
6.6 Communication Latency Figures Obtained Using Theoretic Analysis (in microseconds) 182
6.7 Communication Latency Figures Obtained Using Analysis and Simulation (in microseconds) 184

List of Tables

2.1 Traffic-flows Description 65

3.1 Traffic-flow Description 90

3.2 Traffic-flows Description 93

4.1 Traffic-Flows Description 110

4.2 Priority Ordering with Different Policies 111

5.1 Traffic-flows Description 149

5.2 Traffic-flows Description 162

Acknowledgement

First of all I would like to thank my supervisor, Professor Alan Burns, for his guidance, advice, encouragement and his important support throughout my three years of research, which led to the submission of this thesis. His enthusiasm and precise view on research has made a deep impression on me and I have learned so much from him. I would also like to thank my assessor, Dr. Neil Audsley, for his great support in my work.

I would also like to thank Dr. Leandro Soares Indrusiak for all the supports in simulation evaluation and fruitful discusses on my research. I enjoyed every minute when we work together in this interesting research area.

In addition, thanks to all the rest of the colleagues in the Real-Time Research Group for giving me unforgettable helps and for sharing with me their precious experiences. In particular, I would like to thank Mr Paul Emberson for his very helpful advices on the simulate annealing algorithm and its implementation. I also thank Mrs Sue Helliwell, our group administrator, for always kindly caring for my work and life.

A special thanks to the person who without her I would not have finished this thesis, the one who enthusiastically encouraged me and supported me as much as she could, my wife, Lu Zhang.

Finally, I would like to express my sincere gratitude to my parents, Yuhai Shi and Xiuying Wang, for their permanent love and irreplaceable support. To my family-in-law, Chaojian Zhang and Meihua Shi, my many thanks for all your support and affection.

Declaration

I declare that the research described in this thesis is original work which I undertook between October 2006 and October 2009 at the University of York, with advices from my supervisor, Prof. Alan Burns. Certain parts of this thesis have been published previously as technical reports, conference and journal papers.

Chapter 3 is based on the journal paper "Schedulability Analysis for Fixed Priority Wormhole Switching in On-Chip Networks" (accepted by the International Journal Embedded and Real-Time Communication Systems (IJERTCS)) [75] and the same name technical report (Dept. of Computer Science, University of York, Technical Report YCS-2008-436, 2008) [73] as well as the conference paper "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching" (Proceeding of the 2nd IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Apr 2008) [72].

Chapter 4 describes the research previously published in the paper "Priority Assignment for Real-Time Wormhole Communication in On-Chip Networks" (Proceeding of the 29th IEEE Real-Time System Symposium (RTSS), Dec 2008) [71].

Parts of the material based on Chapter 5 is published in the paper "Real-Time Communication Analysis with a Priority Share Policy in On-Chip Networks" (Proceeding of the 21st Euromicro Conference on Real-Time Systems (ECRTS), Jul 2009) [74]. Other parts of the material based on Chapters 5 is descibed in the paper "Improvement of Schedulability Analysis with a Priority Share Policy in On-Chip Networks" (17th International Conference on Real-Time and Network Systems (RTNS), Oct 2009) [70].

The simulation presented in chapter 6 is based on the journal paper "Schedulability Analysis for Fixed Priority Wormhole Switching in On-Chip Networks"

(accepted by the International Journal Embedded and Real-Time Communication Systems (IJERTCS)) [75].

Chapter 1

Introduction

In the last two decades, we have witnessed an exciting development of modern electronic technologies and products being used in our daily life, from consumer electronics such as mobile phones, digital cameras and notebooks to industrial machinery, robotics and automotive electronics. These micro-electronic products have influenced the ways of human communication, activity and life. The key force driving the development of micro-electronic products is the System-on-Chip (SoC) technologies, where most or all of the functional elements of a complete electronic system are integrated onto single chips [57]. In such a system, two main components can be recognized - *intellectual properties* (IPs) entities and *communication* entities. The IPs entities normally indicate the heterogeneous pre-designed blocks and are in charge of data processing and computation. The communication entities, on the other side, provide the necessary on-chip communication infrastructure for data exchange between the IPs.

Early SoCs employed busses or a point-to-point (P2P) approach to fulfil the information exchange demands. The P2P communication scheme can provide the complete link capacity at the expense of dedicated channels amongst all the IP pairs. However, the number of wires around the IPs increase as the system scale

grows. Therefore, the P2P scheme has poor link utilization and scalability [21]. The bus scheme connects multiple IP cores with a cost efficient shared medium. At any time only one IP (source) is allowed to drive the bus and connect to one or more target IP. Good link utilization is its advantage. But the bus-based scheme still fails to satisfy the requirements of future SoC mainly due to two major drawbacks [36]. First, it is non-scalable, the bandwidth is shared by all IPs and thus the bus becomes the performance bottleneck when more and more IPs are connected. Second, it does not support multiple concurrent communications. The exclusive bus resource could block other potential parallel communications. Using separate busses interconnected by bridges may partially alleviate some of these problems, since different busses may offer different bandwidth capacities and increase communication parallelism. Nonetheless, scalability remains a problem for separate bus architectures [3]. With the rapid increase in the number of IP cores to be connected and the increase in performance demands, busses and point-to-point based platforms suffer from limited scalability and quickly become a communication bottleneck [30, 36].

Inspired by the success and scalability of off-chip networks (e.g. wide-area networks), the data switching and packet routing mechanisms was introduced into on-chip communications. Network-on-Chip [21, 33, 11, 39], (NoC), is emerging as a significant solution for the complex communication requirements of SoCs to overcome the limitation of today's P2P and bus-based communication infrastructure. Although NoCs share many characteristics with off-chip networks, the situation for NoC is radically different from the latter, so many of the techniques for network design have to be reinvented or adapted to cope with the constraints of this new scenario. A few distinctive limitations are unique for on-chip networks, such as minimal energy consumption [10] and small footprint [41] (both computation and storage functions implemented in a small silicon area). Such tight constraints

affect, for instance, the choice of routing and buffering policies, as the cost of storing and maintaining large routing tables or buffering a complete network packet could be too high in both area and power consumption. NoCs are also expected to be less non-deterministic [57], as its structural and topological features are known beforehand and would probably not change during operation.

The advantages of NoC include:

- Scalability [36, 12] : Since the physical wires increase as the number of IPs increase, the network offers more bandwidth and parallel communication opportunity. Hence, the NoC paradigm provides a highly scalable communication architecture.

- Reusability [21, 14]: NoCs decouple the computation from the communication by the definition of standard network interfaces. This makes communication services transparently available with a minor effort from the IPs. The reuse complexity is efficiently reduced and greater design productivity is achieved.

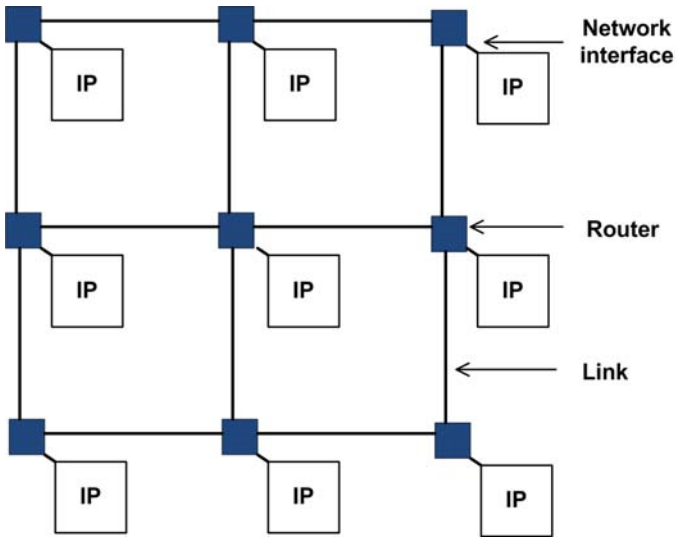


Figure 1.1: Network-on-Chip Architecture

Here we give an example of a common NoC architecture as outlined in Figure 1.1. A NoC consists of nine IP blocks and routers that are connected using physical links so that they are able to communicate with each other by sending and receiving messages. The IPs which undertake the computational data processing can be arbitrary, typical examples are embedded processors, dedicated hardware devices, or reconfigurable fabrics. Different IPs do not directly connect to each other; the communication between IPs has to go through a set of routers. The router nodes deliver the data from source to its destination according to the chosen switching protocols and routing policies. The links connect the routers, providing the raw bandwidth by physical channels. Between the router and IP, there is a connection component - the network interface to glue them together. Encapsulating the traffic in the underlying communication media and managing services provided by networks are the two major functions of the network interface [66]. All these components work together as a network to provide the data service for the applications.

1.1 Real-Time Communication Service

Multiple IP-cores based designs using NoC allow multiple applications to run in parallel. These applications execute data processing and exchange information through the underlying communication infrastructure. As an integration platform, the NoC should have the capability to provide different level of services for various applications on the same network. There is one kind of communication service, where the *correctness* relies on not only the logical communication result but also the completion time bound [45]. A data packet received by a destination too late could be useless or even cause a severe consequence. These critical communications are called *real-time* communications. Real-time communication is common

in safety critical systems, such as robot control or automotive electronics [47]. Recently, more and more requirements from multimedia [32] and system level verification [17] also require guaranteed timing delivery and thus a high degree of predictability.

Time is the central important characteristic for real-time systems differing from other types of computing systems. For on-chip communications, the time we concern with is the packet transmission duration which is defined as the *packet network latency*. The packet network latency is taken into account from the time at which the first bit of a packet is generated from the generater until the last bit is accepted by the receiver. The deadline is the boundary point which marks the latest time that a packet should arrive to its destination. Assigning an explicit deadline to each communication service distinguishes a real-time communication from a non real-time one. So being a real-time communication is about satisfying the timing constraints or in other words, meeting the deadlines. Normally, real-time communications are treated as service assurance that is provided by the network to the demanding applications. So we need a certain measure to quantify the service. According to the levels of service assurances required by the applications, the network communication service can be classified as hard real-time, soft real-time or non real-time (best effort) communications [45, 14].

1.1.1 Hard Real-Time Communication

Hard real-time communications are usually used in applications where it is absolutely imperative to meet deadlines otherwise the whole system might fail, leading to catastrophic results, like, for example, loss of life or serious damage to the environment. The fundamental requirement for hard real-time systems is predictability. It must be possible to guarantee that hard real-time services will meet their requirements in terms of both functionality and timeliness. The need

for predictability implies that the implementation policies used in hard real-time communications must be analyzable. The validation of hard real-time systems is usually reasoned by using precise mathematical models to assure each hard real-time service must be finished within a bounded time even in the worst case. Preferably, this timeliness validation must be carried out before the system is operational. Æthereal [67, 32], Nostrum [58] and Mango [15, 16] are examples of NoCs implementing hard real-time communication.

1.1.2 Soft Real-Time Communication

Meeting all the deadlines is desirable for performance reasons. Yet, a resulting problem is that it may lead to an over-constrainted system, complex and expensive to implement. In practical engineering contexts, if occasional missing a deadline can be tolerated or does not cause serious harm; only the system's overall performance becomes poor, this kind of service is defined as soft real-time communication. The soft real-time communication is rarely required to prove strictly that the service surely meets its real-time performance objective. In many cases, only an acceptable percentage of deadlines may be required by any probability analysis or extensive simulation and test rather than a worst-case analysis. In addition, system utility and efficiency are given preference over the worst-case behaviour and a good average network latency instead of a guaranteed worst-case one becomes one of the primary goals. A classic example of soft real-time communication is multimedia systems [52]. A frame of a video is delivered every thirtieth of a second, and the difference in the time when each image frame is displayed and when the accompanying sound is presented should be no more than 80 msec. However due to the congestion or over-load, the restriction might be violated and the quality of image and sound is degraded. In this case, the users can often tolerate a few glitches as long as the glitches occur rarely and for short lengths of time. QNoC

[19] and DiffServ NoC [34] provide prioritized soft real-time services.

1.1.3 Non Real-Time Communication

Non real-time communication, known as best effort service, refers to communication for which no assurance can be given whatever. In most NoC-related works, best effort service includes the communication service for which only correctness and completion are guaranteed [14], such as most data services like file sharing or background synchronization traffic. The network tries to deliver packets as fast as it can. A good average performance is exhibited with low traffic load, but the worst case can be much more than the average case [63]. It is very common that a on-chip network consists of both real-time and non real-time communication services. On the one hand, the timing constraints of real-time services can be met based on the validation analysis or statistics measures with higher level confidence. On the other hand, the overall system performance, resource utilization and efficiency can be improved by integrating with best effort traffic.

1.2 Wormhole Switching

Early multi-hop networks normally used store-and-forward packet switching [25]. In this approach, the entire packet is stored in a buffer when a packet reaches an intermediate node. But this approach inevitably brings larger size buffers and longer packet latency. To address these problems, wormhole switching [61] is proposed which utilizes the cut-through switching technique that a data packet arriving at an intermediate node is immediately forwarded to the next node without buffering if the next output channel is available. Currently, wormhole switching has been widely accepted for many on-chip networks [26].

The implementation mechanism is that each packet in a wormhole network

is divided into a number of fixed size flits. Wormhole switching allocates router buffers and physical channels at flit level instead of packet. The flit is the smallest unit on which flow control is performed. A packet in this case is decomposed into a header flit, body flit(s), and a tail flit. The header flit takes the routing information and governs the route. As the header advances along the specified path, the remaining flits follow in a pipeline way, possibly spanning a number of routers. If the header flit encounters a link already in use, it is blocked until the link becomes available. In this situation, all the flits of the packet will remain in the router along the path and only a small buffer is required in each router. Figure 1.2 illustrate a packet segmentation and pipeline transmission in wormhole switching. We note that in the extreme situation, each router only needs one flit size buffer capacity to support data transmission.

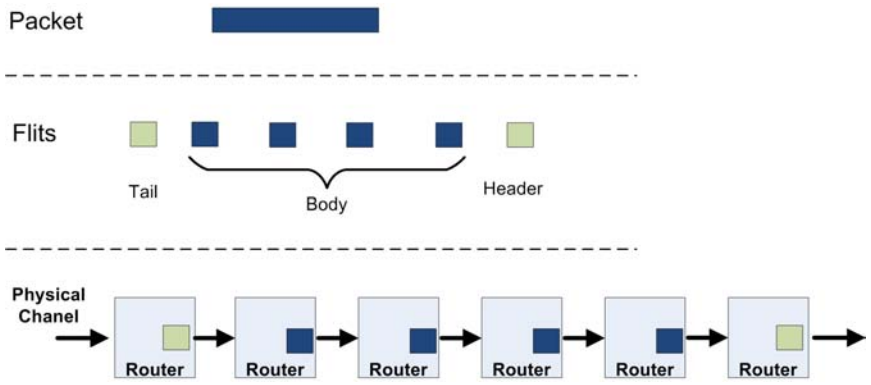


Figure 1.2: Wormhole Switching in Pipeline

On one side, wormhole switching brings low network latency with a low buffer cost, but on the other side, it also introduces chain-blocking. Normally, the resources, such as buffers or physical channels, are shared by all the packets transmitted on the network. When several packets try to access the same resource at the same time, contention occurs and the network only can serve one packet and suspend the others based on some arbitration policy. Once a packet becomes blocked

in wormhole switching, all the flits occupy multiple routers along its path. This will block other packets, which can in turn block other packets, and so on - a phenomenon called chain-blocking. Figure 1.3 illustrates this situation. When packet A is blocked at router 8, it holds buffers of routers 2, 3, 4 ,8. If packet B arrives at this time and requires the physical channel between routers 3 and 4, it is unable to make progress even though all physical channels it required are idle since packet A occupies all the buffer spaces. The blocking problem leads to packet delay and jitter, possibly leading to the violation of timing constraints of the packets. The exact timing predictability in this situation is hard [8, 76] due to the possibility of a packet becoming blocked spanning several routers during its journey. A virtual channel technique [20] can mitigate this problem, the detailed information will be discussed in Chapter 2.

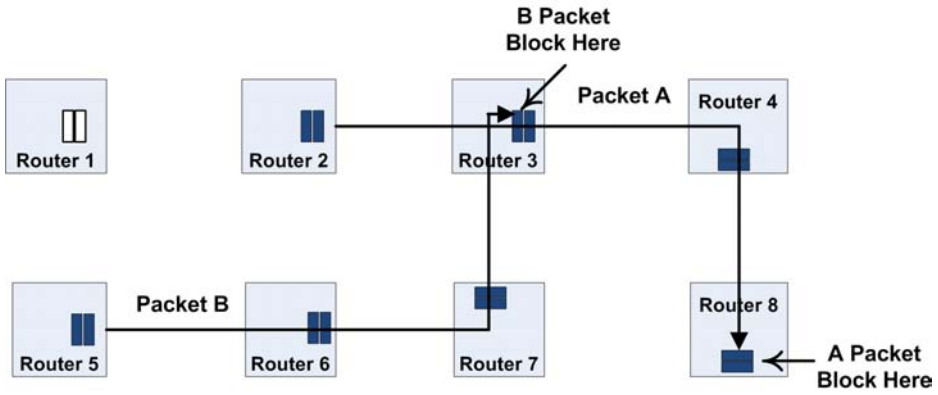


Figure 1.3: Blocking in Wormhole Switching

1.3 Motivation, Goal and Contributions

Wormhole switching brings many advantages, better performance, smaller buffering requirement, lower average latency and greater throughput [67]. All these features make wormhole switching the *de facto* standard in the NoC area. In

[26], Salminen *et al* give an overview of state-of-the-art regarding network-on-chip (NoC) proposals, in all packet based on-chip networks, 78% (28 out of 36) NoCs adopt the wormhole switching as the basic packet switching policy. However, up until recently, wormhole switching is only deployed to support best-effort [12, 59, 32, 21] or soft real-time services [19] and was not considered as a suitable scheme to support hard real-time communication because of potential unbounded worst-case delay. Some NoC designs have used a hybrid approach using two different mechanisms within the same router - one for best-effort and the other for real-time traffic. For example Æthereal[67, 32] hybrids wormhole switching with Time division multiplexing (TDM) and Mango hybrids wormhole switching with virtual circuit [17]. These hybrid approaches inevitably introduce extra overhead to combine two transmission mechanisms together. Therefore, it would be advantageous if we could use a single wormhole switching scheme to support all traffic classes. How to support the hard real-time service is the key issue. Some early works in interconnection networks [60, 76, 8, 44] and recently works in NoCs [53, 19, 34] have explored the use of priority combined with wormhole switching to handle the guaranteed services. We refer to these extensions as real-time wormhole switching. Priority order on the usage of the link is enforced by a priority based preemption structure which is constructed in each router. With this arbitration scheme, the physical link is in effect locally pre-emptable. The priority arbitration technique has two attractive features:

1. It can distinguish the different levels of communication services, the higher its priority, the more urgent the communication is.

2. It provides a mean of analyzing network timeliness.

However, these studies either do not deliver hard real-time service [76, 19, 34] or suffer pessimism [8, 44] or even defect[53, 60]. These points are considered later

in this thesis.

All these issues motivated our research which aims at exploring analyzable wormhole switching for hard real-time communication. Analysis and validation are essential for performance and predictability of guaranteed services. The analysis objective in this thesis is defined for a *traffic-flow* which is a generalized description for a communication service. The traffic-flow is a packet stream which traverses the same route from the source to the destination and requires the same grade of service along the path. For *hard real-time* traffic-flows, it is necessary that all the packets generated by the traffic-flow must be delivered before their deadlines no matter what the arrival order of the flow set. A set of hard real-time traffic-flows over the network are termed *schedulable* if all the packets belonging to these traffic-flows meet their deadlines even under worst-case scenarios. Logically, the worst-case scenario must be investigated, to show that deadlines are met in the worst case; if so then the deadlines will be met in all other cases. This is applied to wormhole switching using *worst-case network latency analysis*. The approach is to find the scenario which can induce the worst-case possible latency in the network and determine that latency.

This thesis concerns how wormhole switching can be analyzed quantitatively to satisfy hard timing constraints. The following statement synthesizes the central research proposition.

With a fixed priority based arbitration strategy and worst case network latency analysis, wormhole switching can be a solution for hard real-time on-chip communication.

This proposition will be achieved by a group of present research works in four parts:

1. In the first part, a novel analysis approach, which successfully emulates wormhole switching as a classic single processor scheduling model [51], is presented. By evaluating diverse inter-relationships and service attributes among the traffic-flows, this approach can predict the packet transmission latency for a given traffic-flow based on three quantifiable different interferences: direct interference from higher priority traffic-flows, indirect interference from higher priority traffic-flows and self-blocking produced by a previous packet from same traffic-flow. Due to the inevitable existence of parallel communication, we prove that the general problem of determining the exact schedulability of real-time traffic-flows over the on-chip network is NP-hard. We also prove that the real maximum network latency is bounded by the theoretical calculation in our model.

2. One of the common problems that need to be solved in the development of priority-based wormhole switching is, how to assign priority to each traffic-flow so that all the traffic-flows loaded on the network will meet their time constraints. So in the second part we will explore the priority assignment policy in real-time on-chip networks. We present a branch and bound search algorithm to find the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space on average.

3. As two powerful tools, the worst-case latency analysis and priority assignment ensure that the hard timing guarantees can be delivered by priority based wormhole switching. But this approach requires distinct priorities and an exclusive virtual channel for each traffic-flow in each router port. This restricted implementation structure results in higher area overhead. So in the third part, we propose an improved solution that utilizes a *priority*

share policy to reduce the resource overhead while still achieving hard real-time communication guarantees. The priority share policy permits multiple traffic-flows to contend for a single virtual channel and share the same priority level. Considering the extra blocking delay introduced by this priority share, novel analysis approaches are presented in two steps regarding to the state of the traffic-flow deadline. In the first step we restrict the deadlines to be less than or equal to the relative period, so only single blockings from other flows sharing the same priority are considered. In this state, we introduce the composite model schedulability analysis. This approach models all the flows sharing the same priority as a single scheduling entity and the packet transmission is analyzed holistically. In the second step we relax the restriction of having deadlines less than the relative period and show two additional blockings. In this case, we change the analysis view point from per flow to per priority. By checking the packet release at the corresponding priority window, we can verify the timing semantics of real-time flow with a simple yet efficient scheme. In addition, we also find that the previous result is just a special case covered by this new approach. Building on this static analysis, we furthermore explore several share priority allocation policies, in such a way that the timing requirements of all the traffic-flows are met with a reduced resource overhead.

4. To better validate and evaluate our analysis technique, in the last part, an autonomous vehicle control application is simulated on an actual NoC platform. By simulation, we are interested to know how realistic are the worst case latencies calculated using scheduling analysis against the simulation results. The simulation results provided evidence that the communication latency calculated using the real-time analysis approach is safe, closely matching the figures obtained from simulation.

1.4 Thesis Structure

The remaining chapters of this thesis are organized as follows: Chapter 2 gives a historical review of state-of-the-art research in real-time on-chip networks. The survey includes the existing research works in NoC architecture, implementation schemes for real-time communication and validation analysis. In Chapter 3, the novel approach on schedulability analysis which is based on worst-case network latency evaluation for each traffic-flow is presented. Priority assignment is very important part in any fixed priority based communication service. This is addressed in Chapter 4. In order to reduce the resource overhead, Chapter 5 proposes a priority share policy to solve this problem. The relevant schedulability analysis and share priority allocation scheme are discussed in this chapter. Chapter 6 gives a comparison between theoretic analysis and practical simulation for an application running on a NoC. Finally, Chapter 7 concludes the contributions of this thesis, provides an overall evaluation, and considers direction for future work.

Chapter 2

On-Chip Networks and Real-Time Service

Real-time communication services in on-chip networks is the central theme of this thesis. In this chapter, an overview of the state-of-the-art research is presented. This survey of related works is divide into three parts. We first review NoC architecture characteristics, particularly topology, switching, routing and flow control techniques. All these aspects jointly determine the performance and characteristic of NoCs. Then, we examine the existing implementation schemes for guaranteed service in on-chip networks, including both hard real-time and soft real-time services. Two classical implementation approaches, resource reservation and arbitration scheduling are reviewed. In the last part of this chapter we review the current works on schedulability analysis in priority based wormhole switching. The drawbacks and defects of these works are discussed.

2.1 On-Chip Communication Network

On-chip networks are designed using many similar principles as wide-area networks. Networks are characterized by topology and protocols. The topology embodies the structural relations among the network components such IPs, routers and physical channels. While the protocols specify the ways in which the network components operate with each other, which principally determines the *switching*, *routing* and *flow control* techniques. In the following, we describe these network characteristics one by one.

2.1.1 Topology

Topology concerns the layout and connectivity of routers and links. Usually the topology is represented as a graph. Although many network topologies have been proposed in terms of their graph-theoretical properties [25], very few of them have ever been implemented. Most NoCs implement regular forms of network topology that can be easily laid out on a chip surface.

As a regular topology, the fat-tree topology uses a tree structure with routers on the nodes and terminals on the leaves, except that every node has replicated fathers. The SPIN [2, 33] project uses a fat-tree topology: every node has four sons and the father is replicated four times at any level of the tree. So four fathers offer four equivalent paths, where the shortest path is through the nearest ancestor. Figure 2.1(A) shows the basic SPIN architecture with 16 IP blocks, where the IP blocks reside at the leaves and the routers reside at the vertices. Such topology produces a non-blocking network with a performance that scales gracefully with the system size. But from the wiring point of view, this will consume a large wiring resource and is therefore only good for small system.

Another interesting topology is presented in the Octagon NoC project [27], see

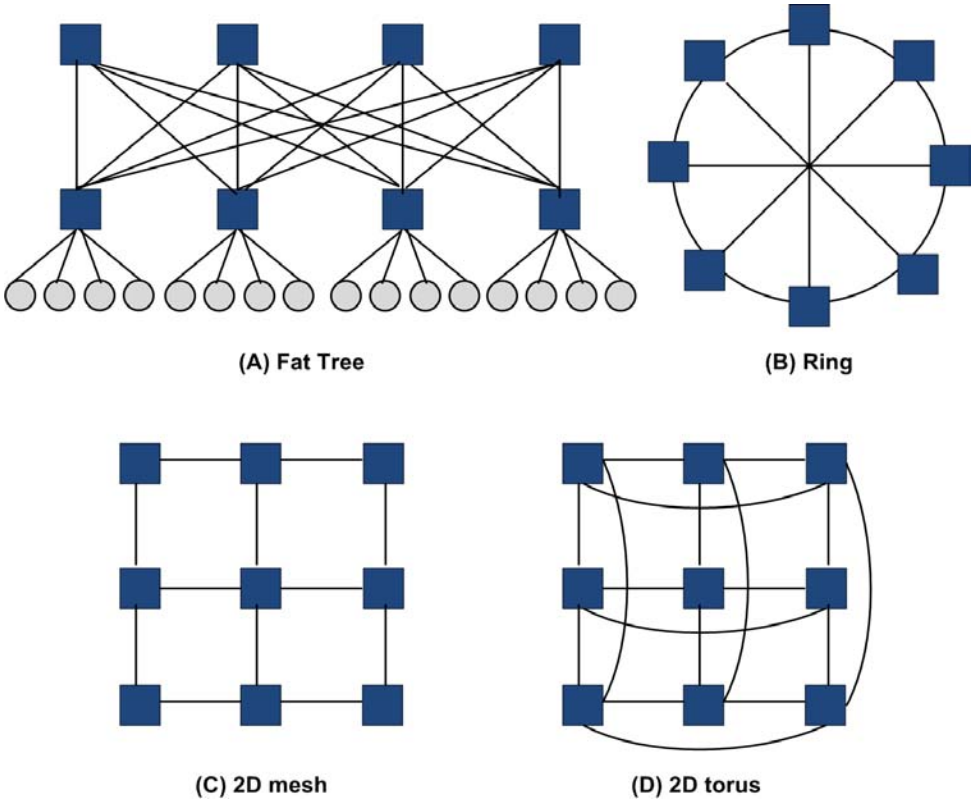


Figure 2.1: Network Topology

Figure 2.1(B), in which a ring of 8 nodes is connected by 12 bidirectional links. So traveling between any pair of routers takes no more than two hops.

Kumar *et al* [46] have presented a mesh-based interconnect architecture, which consists of an $m \times n$ mesh of routers interconnecting with a set of IPs. Figure 2.1(C) shows a 3×3 case. Every router has two, three or four neighbor routers according to its position.

A 2D torus [21, 22] has a similar architecture to a regular mesh; the only difference is that a router at the edges is connected to the router at the opposite edge through an extra wire. Figure 2.1(D) is a case of 3×3 2D torus. All the routers have four neighbor routers no matter where it is. The average number of hops between any pair of routers is smaller than the same size 2D mesh, but with an increase in cost and complexity. Both 2D mesh and 2D torus are popular on-chip

topologies because their layouts on a two-dimension chip plane use symmetric-length wires [41]. The regularity results in the predictability of wire delay with a simple routing algorithm.

2.1.2 Switching

Switching determines how and when data packets traverse through the routers. Packet switching and circuit switching are the two main switching technique used in on-chip networks.

Circuit switching reserves a dedicated end-to-end path from the source to the destination before starting to transmit the data. Circuit switching is connection-oriented and has an explicit connection establishment process. A more detailed introduction to circuit switching will be given in Section 2.2.1.1. By comparison, in packet switching, no link reservation is made, and the packets make their ways independently from source to destination, perhaps along different routes, and with different delays. There are three basic packet switching schemes: store and forward (SF), virtual cut through (VCT), and wormhole switching (WH).

2.1.2.1 Store and forward

Store and forward [25] is the simplest form of packet switching. The router must receive the complete packet before it can begin transmitting the packet to the next router. As a result, the latency per router and the buffer size are at least equal to the size of the packet. A time-space diagram of the progress of a packet across three routers is shown in Figure 2.2(A). From the figure we can see that the latency experienced by a packet is proportional to the distance between the source and destination nodes. Because of expensive buffer resource in NoCs, few NoCs proposals have used this technique [26]. The packet latency without contention is expressed by Eq.(2.1):

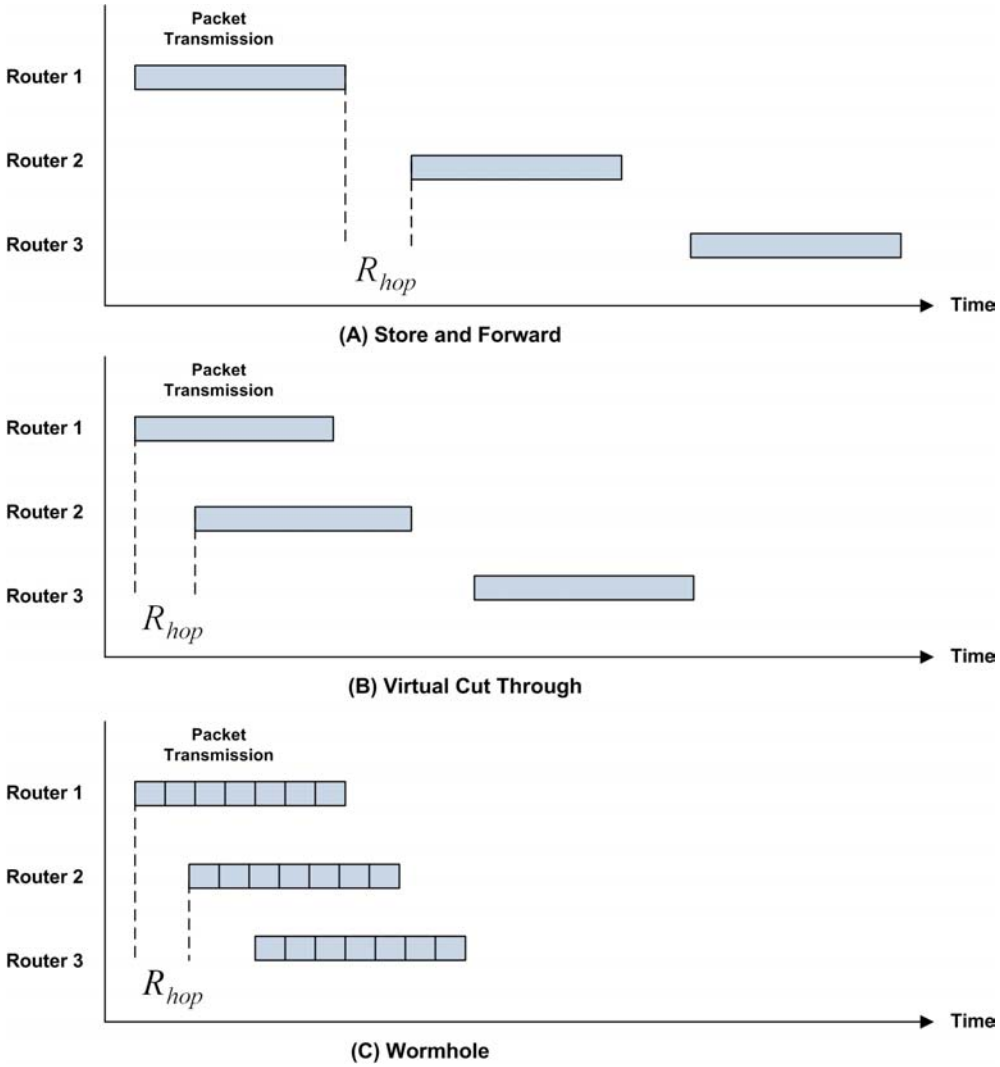


Figure 2.2: Three Packet Switching Techniques

$$Latency_{SF} = (\frac{Length_{packet}}{BW} + R_{hop}) \times Hops \tag{2.1}$$

where BW is the link bandwidth and R_{hop} the routing delay per hop; Hops is the number of hops from the source to the destination.

2.1.2.2 Virtual cut-through

Virtual cut-through switching [24] reduces the transmission latency by forwarding a packet as soon as a routing decision has been made and the next router guarantees that the completed packet will be accepted. The packet does not even have to be buffered at the output and cut through to the input of the next router before the completed packet has been received at the current router (see middle transmission in Figure 2.2(B)). The complete packet is stored at an intermediate router buffer only if the next required channel is busy (last transmission in Figure 2.2(B)). Virtual cut through can achieve a high throughput by its pipeline mechanism which overlaps switching, routing and flow control as much as possible.

Like store-and-forward, VCT allocates both link bandwidth and buffers at the packet level. So for the extreme case, where a packet is blocked at each router, the VCT switching approach gets reduced to the store and forward approach. Without contention, once the header of packet reaches the destination node, the time of a packet pipeline transmission is determined by the packet size and link bandwidth. So the network latency of a packet in VCT is shown as follows:

$$Latency_{VCT} = \frac{Length_{packet}}{BW} + Hops \times R_{hop} \tag{2.2}$$

2.1.2.3 Wormhole switching

Wormhole switching [61] and virtual cut-through both utilize a cut-through technique. Wormhole switching improves on virtual cut through by reducing the buffer requirement at the flit level. This results in the major different between them is in how they handle packet blocking. In wormhole switching, the buffer capacity of the router is a multiple of flits. If a packet is blocked, flits of the packet are stalled and may span a sequence of routers. With virtual cut through, a router, at which a packet is blocked, must receive and store all flits of this blocked packet.

Due to the pipelined transmission, in the absence of contention, wormhole and VCT switching have the same latency but wormhole needs smaller buffers than VCT.

$$Latency_{wormhole} = \lceil \frac{Length_{packet}}{Flit_{size}} \rceil \times \frac{Flit_{size}}{BW} + Hops \times R_{hop} \tag{2.3}$$

where $Flit_{size}$ is the flit size in the network.

Wormhole switching has nice properties but frequently is open to a high level of congestion and inefficient use of channel bandwidth due to chained blocking (a case has been illustrated in Section 1.2). One way to improve on wormhole switching's performance is by combining wormhole switching with virtual channels (VCs) technique [20]. A virtual channel is a resource allocation technique which provides multiple independent flit buffer queues for each physical channel. If a packet currently using the physical channel becomes blocked, other packets which are held in the VC will compete and the winner gets the right to use the physical channel, thus bypassing the blocked packet. In this way, several packets can share a physical channel. Figure 2.3 shows the concept of 4 virtual channels sharing one physical channel. Several benefits can be gained by deploying virtual channels with wormhole switching [14]:

- deadlock avoidance,
- more efficient network channel utilization,
- support of different service levels.

Because of these advantages, a combination of wormhole switching and virtual channels became the prevailing switching scheme advocated in the NoC community [41, 26]. It offers high flexibility and good resources utilization while the

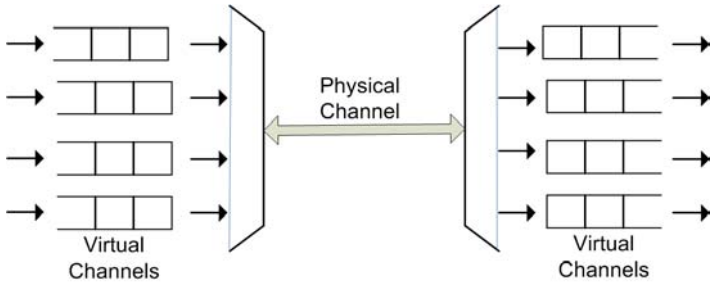


Figure 2.3: Virtual Channel

switch complexity remains moderate.

2.1.3 Routing

In a network, a node has the ability to transfer packets to another node. The routing algorithm determines which path the packet should take through the network [25]. Implementation complexity and performance requirements are two major concerns in selecting the routing strategy for an on-chip network [62].

According to how a path is defined to transmit packets, a routing algorithm can be classified as either *deterministic* or *adaptive*. In deterministic routing, the path is completely determined by the source and the destination address. Thus, deterministic routing always produces the same path (single path) between a given source/destination pair. Compared with deterministic routing, adaptive routing uses information about network traffic and/or channel status to avoid congested or faulty regions of the network. So the path is not only determined by the source and the destination but also by the network condition.

More than 70% of NoCs utilize deterministic routing [26]. The best feature of deterministic routing is simplicity. One of the popular deterministic routings is X-Y routing [61] which runs in 2D mesh topology. In X-Y routing, packets are transferred first along the X dimension. When it reaches the proper site, it turns to the Y dimension and arrives at destination in the same fashion. By satisfying

a strictly monotonic order on the dimensions traversed, deadlock-free routing is guaranteed. However, failure to dynamically respond to network congestion is one of the major problems in deterministic routing.

In contrast, adaptive routing has the ability to avoid congestion links and fault-tolerance by making dynamic decisions. This approach increases the efficiency of the network when network traffic is overloaded. But it also introduces complicated routing logic and the communication overhead of status updating. Another problem of adaptive routing is out-of-order packet delivery. Sufficient buffers are required by each network interface to number and reorder the arrival packets.

DyAD [37] combines the features of adaptive routing and deterministic routing. It keeps monitoring the local network and makes decisions based on this information. When the network is not congested, it works in a deterministic routing mode. Once a congestion occurs in the network, each router switches back to the adaptive routing mode thus leads to higher throughput.

2.1.4 Flow control

Flow control mechanisms in a NoC architecture are critical for fast packet propagation across the network and for high link utilization. Buffer management and allocation are fundamental tasks of each flow control scheme [64]. In NoC, each node needs to employ buffer resources to send or receive data. Flow control deals with the allocation of channels and buffers to a packet as it traverses this path and prevents data being sent to any full buffer. The common flow control schemes include Handshake [81], ACK/NACK [64], and the Credit based scheme [23, 66, 18].

The handshake scheme is a simplest flow control protocol. When a sender puts data on the link, it activates the related valid (VALID) signal. When the receiver is ready to consume the validated data, it responds with the corresponding acknowl-

edge (ACK) signal. The handshake scheme has a low cost implementation but poor link utilization. When the sender tries to transfer a flit, it faces a long handshake cycle spanning two interactive nodes. Proteo [78] and SoCIN [81] adopt the handshake flow control protocol.

In ACK/NACK scheme, when the upstream router sends flits whenever it becomes available, this sender will keep a copy in its buffer locally. If the downstream router has available buffer space, it accepts the flit and replies with an acknowledgment (ACK) to the upstream router. Otherwise, the downstream router drops the flit and sends a negative acknowledgment (NACK). Upon receiving an ACK, the sender deletes the local copy of the flit; upon receiving a NACK, the sender rechecks its output queue and starts re-sending the dropped flit. In order to improve the link utilization and avoid the long waiting time for the acknowledgment, the ACK/NACK is always combined with a GO-BACK-N policy. Flits are transmitted continuously and the sender does not wait for an ACK after sending a flit. When a NACK is received, the sender backs up to N flits from that negative acknowledgment and re-sends it in addition to the N-1 succeeding flits that were transmitted during the round-trip delay. At the receiver, the N-1 received flits following the corrupted one are discarded regardless of whether they were received error-free or not. The ACK/NACK policy provides better throughput and latency but more buffering resources are required. Besides this, frequently re-sending up to N flits will cause a heavy performance hit. Thus, ACK/NACK only adapts to the low errors situation. XPipes [12] adopts the ACK/NACK flow control scheme.

In Credit based flow control, an upstream router keeps a counter of the available space in each downstream router. When a flit is sent from an upstream router, the counter is decremented. So if the sender has enough credit, it can send multiple flits continuously. When the flit is consumed by the downstream router, the credit is produced in a sender's counter to indicate that more free space in the receiver is

available. If the counter reaches zero, the receiver buffer is full and the sender's request is blocked until new credits are received from the receiver node. In this scenario, there will never be sent more credits back than the receiver can accept. Consequently, no flit is dropped due to a full buffer.

Both ACK/NACK and Credit flow control introduce multiple in-flight flits in the link and outperform the Handshake protocol. Considering the extra buffer cost at the output port to store the outgoing flits waiting for the acknowledgement, the ACK/NACK flow control is less efficient compared with the Credit based flow control [22]. So currently in NoC projects, such as Ætheral [23, 66], QNoC [19], Chain [29], SPIN [33] and Dally [21], the Credit based flow control scheme is adopted.

In this section we have reviewed several key techniques, including topology, switching, routing and flow control, in on-chip networks. In the next section, we present how these techniques are used to support real-time communication services.

2.2 Implementation Schemes for Real-Time Services

On-chip networks, as a significant solution for complex communication of SoCs, outperform the traditional busses or point-to-point approach in many ways. Packet-routing and segmented link communication structure offers maximum flexibility and scalability [14]. Normally, the network resources, like buffer, physical or logical channel, are shared by all the IP blocks on the chip, then local performance is not degraded when scaling. Resource sharing brings higher utilization but also introduces unpredictable network delays due to contention [63]. In on-chip networks, multiple parallel tasks running on different IP cores exchange information. When more than one packet tries to access the shared resource at the same

time, contention occurs. The contention problem, which leads to packet delay and even missing deadlines, has become the major influence factor of network predictability. Hard real-time communication, on the contrary, requires the timing to be predictable even under the worst-case situation. Besides this, network resource allocation and scheduling for real-time traffic-flows should be analyzable during the design phase. Predictability of performance is essential for NoCs designs to take early decisions before actual implementation. Fortunately, the communication patterns of a SoC is determined during a pre-configuration period; structural and topological features are foreseeable. Therefore, we need an implementation policy and evaluation approach to ensure the packet network latency never violates its timing bound. Utilizing this approach, we also can plan and explore the distribution of real-time applications over the network to produce an effective mapping.

Design choices and implementation structures to support real-time services in on-chip networks are the central theme of this section. To address the contention problem, two well known disciplines, *resource reservation* and *run-time arbitration* are presented in the next two sections, including a brief discussion about the advantage and disadvantage for each scheme.

2.2.1 Resource Reservation Discipline

Resource reservation discipline considers that contention is avoidable by trying to pre-arrange and reserve a set of resources before the start of communication, so that packets are injected at the network's edge in such a way that they never access the same resource at the same time. These resources, such as physical wires, time-slots and virtual channels, etc, construct logical independent end-to-end communications, thus avoiding contention. Circuit switching [79, 69, 80], time division multiplexing (TDM) [67, 23, 32, 58] or spatial division multiplexing (SDM) [1] are several common resource reservation schemes.

2.2.1.1 Real-Time Communication Supported by Circuit Switching

Circuit switching provides connection-oriented end-to-end communication with exclusive circuit resources for data transmission. SoCBUS [79, 69] and Smart Chips for Smart Surroundings (4S) project [80] employed this scheme.

In circuit switching, a circuit with a fixed physical path is set up between the sender and the receiver, and all the packets from the same flow are sent over this circuit. The circuit switching includes three phases: an explicit circuit establishment phase, a data transmission phase and a circuit tearing down phase. The circuit establishment process is as follows: a request including routing information and service level requirement is sent from the source in the network. This request packet is routed by the router to its destination. Each intermediate node receiving this request will execute the acceptance test. If this request passes this test, the router will temporarily lock the resource which cannot be used by any other connections and pass this request to the next router. When the request reaches its destination, an acknowledge is sent back along the route and the resources are changed to permanent locks. When the acknowledge has returned to the source node, the circuit connection is established and data can be transferred from the source along this connection. During the process of circuit establishment, if the request is rejected by one router, this router will return a negative acknowledge to the source. Each intermediate router along the return path receiving this message will release the temporary locked resources. If the application does not require this circuit any more, a tearing-down request is sent that releases all the resources as it follows the route. The latency of circuit switching is determined by the time to setup a path and the subsequent data transmission time:

$$Latency_{circuit} = Latency_{setup} + Latency_{data} \tag{2.4}$$

$$Latency_{data} = \frac{Length_{Packet}}{BW} \tag{2.5}$$

The time of path setup process is unknown in principle. Once the path has been setup, the propagation delay for each packet is determined by the allocated path bandwidth.

This scheme has the advantage that it guarantees tight temporal timing and bandwidth bounds for individual flows with minimal protocol and buffer cost. But one of the disadvantages of circuit switching is inefficient bandwidth usage. This is because the resource allocation in the circuit is based on a worst-case scenario, and during the live time of circuit connection, the resource is occupied exclusively, even if there is no data transmission available. Another problem is unpredictable setting-up process. The circuit establishment is normally based on best effort service and thus may take quite long time at runtime. Recently, Jerger *et al* [40] proposed a new circuit switching scheme where the control network is separated from the data transmission network. The circuit setup process uses a special control channel and hence reduces the contention and average latency of circuit establishment.

2.2.1.2 Real-Time Communication Supported by TDM

Differing from circuit switching which allocates whole physical transmission capacity for a single traffic-flow, TDM divides the usage of the expensive resource into fixed time slots along the timing axis. As a static schedule, TDM globally allocates these time slots to individual applications to ensure each flow is independent of other flows and contention is free. Æthereal [68, 67, 32] and Nostrum [58] are examples of systems implementing hard real-time services with TDM.

In the following, we use the Æthereal project as an example to illustrate the concept of TDM. The output port of each router in the network uses a *slot table*

to manage the resources. Every slot table T has S time slots. The time slot is the minimum time unit for a router to read and write data blocks. We assume the time period for a time slot is T_{slot} and thus, in each time slot, the total size of data transmission is $W_{slot} = BW \cdot T_{slot}$. Each flow can be assigned m slots based on its required bandwidth. There is a logical notion of synchronicity that all routers in the network occupy the same fixed-duration slot. Each entry of the slot table is a map between an input port and an output port; $T(s, o) = i$, which means that a block from input i is passed to output o at time $s + k \cdot S$, $k \in N$. So no congestion can occur as one output port can be assigned to no more than one input port at any time. During the connection establishment process, the slot will be reserved along the path from source to destination increase by one (modulo S). If slot s is reserved at one router, slot $(s + 1) mod\ S$ must be reserved in the next router on the path.

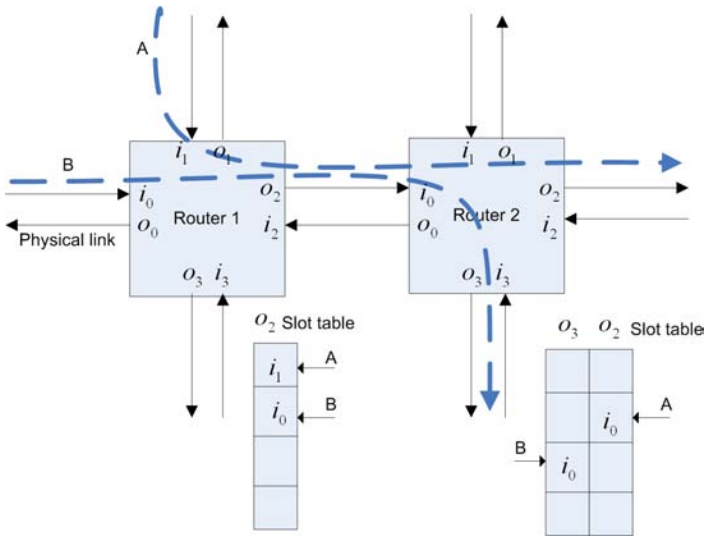


Figure 2.4: Æthereal Contention Free Router

Figure 2.4 shows two packet A and B which belong to two separate flows pass through router 1 and router 2. The size of slot table is S=4. At slot=0, packet A transmits from i_1 to o_2 in router 1 as indicated by slot table $T_{router_1}(0, o_2) = i_1$.

After one slot, packet A will be switched from i_0 to o_2 in router 2 as stated by the slot table $T_{router_2}(1, o_2) = i_0$. At the same time, packet B will be transferred in the same fashion from i_0 to o_2 in router 1 ($T_{router_1}(1, o_2) = i_0$). So the usage of the physical link between router 1 and route 2 is time multiplexed by packets A and B.

Fundamentally, with TDM, all schedules are determined in advance and therefore the resource allocation information is available to each router a priori. This predictability is an important benefit of TDM and real-time behavior is very easy to reason about. It is easy to find worst-case network latency and show which deadlines will be met through off-line analysis [31]. We can formulate the throughput bounds for every flow as:

$$BW_{TDM\_flow} = \frac{m}{S} \times BW \tag{2.6}$$

Similarly, the latency bounds can be summarized as:

$$Latency_{TDM\_flow} = \lceil \frac{Length_{packcet}}{m \cdot W_{slot}} \rceil T + Hops \times T_{slot} \tag{2.7}$$

Note that even though TDM brings many benefits, it is not without problem. Actually, TDM suffers similar problems to circuit switching, e.g. low resource utilization. Worst-case requirement satisfaction and time slot granularity always result in excessive resource reservation [4]. This will waste link transmission capability and energy and lead to degradation of average performance for best effort traffic. In order to get better link utilization, Æthereal [68, 67, 32] adopts a hybrid approach by mixing best effort packets with TDM. The best effort packets can be forwarded in a real-time service reserved time slot whenever real-time packets are not present. But this requires two independent router components to support both TDM and best effort which increases the hardware cost. Andreasson and Kumar

[4] have explored in utilizing the idle capacity of TDM circuit. A slack time aware concept was proposed where a producer can inject best effort packets during the slacks in time-slots reserved for real-time service packets, thereby mixing real-time and best effort traffic at the source instead of each intermediate routers. But this requires the control messages to inform the routers over the reserved path and hence introduces extra overhead. Another problem for TDM is that it introduces the coupling between delay bound and bandwidth allocation granularity. The delay is determined by T, so in order to get a small delay, a small value of T is preferred. On the other hand, in order to have more flexibility in allocating bandwidth, having smaller bandwidth allocation granularity, a larger T is preferred. It is clear that an TDM strategy cannot implement low-latency and low-bandwidth flow allocation simultaneously.

2.2.1.3 Real-Time Communication Supported by SDM

The SDM [1] approach also uses a resource reservation to construct an end-to-end connection, similar to TDM or circuit switching techniques, to implement the service quality guarantee. The major different with the other techniques is that this time the transmission channel is divided into a set of physical wires along the spacial axis. This exploits the fact that the on-chip network links are physically made of a set of pin wires, as showed in Figure 2.5. The packet transferred in SDM are digit-serialized on a number of wires proportional to the bandwidth allocated to the flow. For example, each port of router in SDM is N bits width. When an application establishes a connection from source to destination, the SDM calculates the bandwidth and allocates the relevant sub-set of wires to this application for the whole connection lifetime. If a link in SDM offers maximum M connections simultaneously, the according N-bit link is divided in M individually switchable group of wires. In SDM a finer bandwidth granularity means more

wires per link. In the extreme case, when an application can be assigned to only one wire, the corresponding bandwidth granularity is $\frac{1}{N}$ of link total bandwidth. The SDM has similar properties to circuit switching except the bandwidth granularity is depended on the width of the wire.

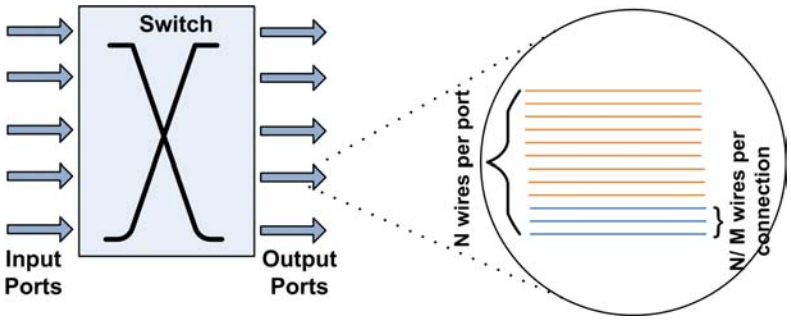


Figure 2.5: Architecture of SDM Router

2.2.2 Run-Time Arbitration Discipline

Another way to address the contention problem is via arbitration and scheduling which normally utilize the arbiter to determine the access of resource at run time. In this way, the contention is deemed to be unavoidability but the worst-case situation is predictable. Two crucial components are required to facilitate the scheduling and arbitration. First, an appropriate arbiter in each router is deployed to allocate the transmission capacity dynamically according to some pre-configured policy. Second, an admission control entity decides whether a new traffic-flow request should be accepted. The admission control will accept a new arrival flow if there is enough resource to meet the request without jeopardizing the flows which have already been accepted. This can be achieved by some on-line/off-line analysis related to the scheduling algorithm. Now we turn our attention to the different arbitration policies that can be used to select packets for transmission on a link. Round-robin [43, 42] and priority related arbitrations [17, 28, 19, 9] are the

two most common arbitration schemes in on-chip networks.

2.2.2.1 Round-Robin (RR) Arbitration

With round-robin arbitration services all the traffic-flows in a sequence are repeated indefinitly. In [43, 42], at each router port, a set of virtual channels are multiplexed with a physical channel. All the arrival packets are waiting in one of the VCs. At any time, the scheduler polls each non-empty VC in a cyclic order and serves a flit which is at the head of the VC. When there are N not empty VCs, each VC gets one cycle service time every N cycles, that is, every round. So the physical channel is time shared between its VCs on a cycle-by-cycle basis in a repeating fashion. An idle VC does not occupy service cycle. This policy ensures that each VC has a service rate of at least $\frac{1}{N}$ link capacity.

If a physical channel has N multiplexed VCs, then each of N VCs is guaranteed throughput greater than or equal to:

$$BW_{RR} = \frac{BW}{N} \tag{2.8}$$

This is the worst-case throughput for one VC. A real-time service flow may reserve a series of VCs from its source to destination. The minimal bandwidth of this flow is determined by a VC with a minimal throughput bound. Given a lower bound of bandwidth on a connection, $min(BW_{RR})$, the corresponding maximum latency for a packet is:

$$Latency_{RR} = Hops \times R_{hop} + \frac{Length_{packet}}{min(BW_{RR})} \tag{2.9}$$

The round-robin approach is commonly used for scheduling with fairness and gives equal chances to all competitors. But it cannot differentiate between the quality of service desired by different traffic-flows. It also encounters the similar

problem with TDM that a delay bound on a flow is closely coupled to bandwidth.

2.2.2.2 Priority Arbitration

In conventional wormhole routers, data flits held by VCs access the output link based on a first-come, first-service service discipline. This scheme is suitable for non real-time traffic since it is fair, simple and produces good average performance. But in real-time communication, the network must ensure each real-time packet meets its deadline bound. Priority scheduling is a well known and widely used technique in the real-time system area, and makes it possible to provide and assess the timeliness of guaranteed real-time communication. The objective of priority based arbitration is to add the ability to provide differentiated services to the flows, using a resource allocation mechanism based on priority. In priority-driven scheme, each packet is assigned individual priority during the transmission. When multiple packets arrive simultaneously at the router, the packet with the highest priority is served first, even if other packets are waiting to be served. According to the resource allocation granularity, the priority arbitration principle can be classified into fine granularity (flow based) and coarse granularity (class based) arbitration. The fine granularity scheme assigns each flow a distinct priority and virtual channel. So a potential real-time guarantee can be given to individual flows with the cost of greater hardware complexity. By comparison, a coarse granularity scheme groups traffic flows into different classes with different service levels and prioritizes for each class. This scheme offers better adaptation to varying network traffic with smaller hardware cost. However, hard real-time service is difficult to guarantee. In the following, we review some schemes based on priority arbitration.

Flow Based Priority Arbitration

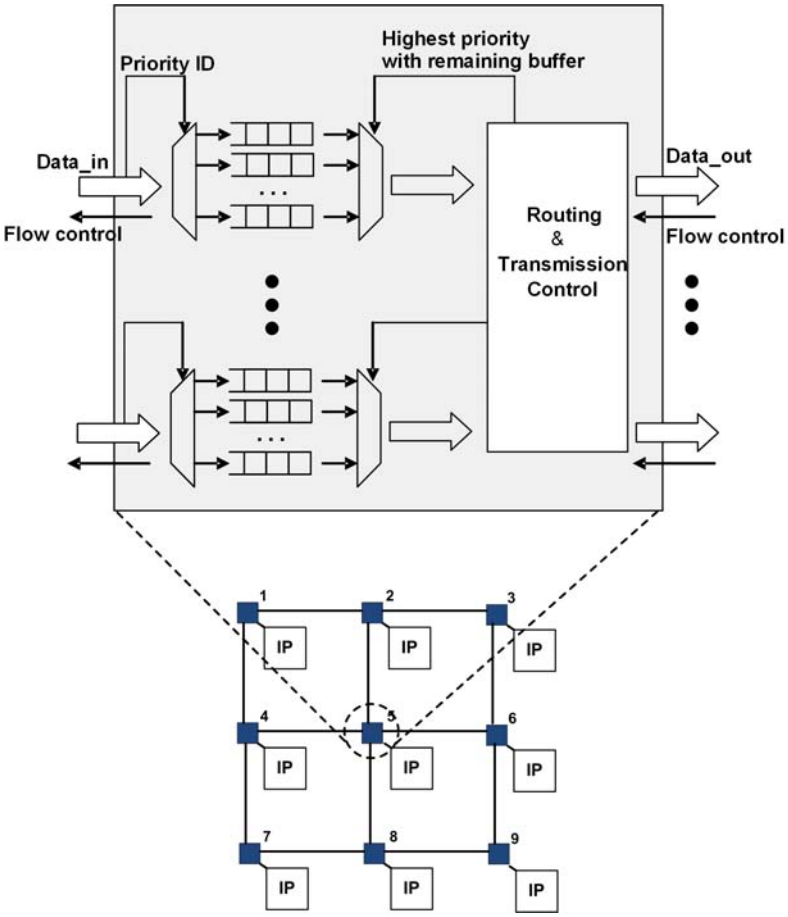


Figure 2.6: A General Architecture of Priority Arbitration

The ANoC [9] and Chain [29, 28] NoC projects and some early works [35, 44] in interconnection networks deploy the flow based fixed priority arbitration structure. Such scheme can be implemented using a virtual channel technique which multiplexes every link among different communication flows with multiple dependent buffers and assigns a priority level. Likewise, every communication flow is assigned a priority level (which is inherited by all packets of that flow), and that level defines which VC it should use. Thus, a packet with priority i can only request the VCs associated with priority value i. This one-to-one relation can

ensure that any VC is requested by no more than one traffic flow. So a logical virtual circuit is established from source to destination by reserving a sequence of VCs for a traffic flow along its path. The flow control protocol is deployed to prevent packets being sent to any full buffer.

Figure 2.6 shows a general internal structure to support priority arbitration. In each input port, a different FIFO buffer stores flits of packets arriving through different virtual channels (i.e. different priorities). The router assigns an output port for each incoming packet according to their destination node. At any time, a flit of a given packet will be sent through its respective output port if it has the highest priority among the packets being sent out through that port, and if it has buffer space on the respective buffer of the neighboring node connected to that output port (This can be supported by some flow control protocol, e.g. Credit flow control). If the highest priority packet can't send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link. Once a higher priority packet appears, transmission of the current packet is preempted and the higher priority packet gets through. Transmission of the lower priority packet is resumed only after all higher priority packets are serviced. Therefore, the allowable service time for a traffic flow comprehends all the time intervals at which no higher priority traffic flow competes for the same physical link. This flit-level preemption with priority arbitration can provide delivery guarantees for hard deadline packets. The related latency analysis approaches are discussed in the next section.

The above schemes employ a fixed priority policy which require a distinct priority for each flow before the communication and this priority never changes during the service period. In contrast, a dynamic priority policy defines the priority of individual packet dynamically, as the function of current status. The Mango NoC project [15, 16] designs a new dynamic arbitration discipline, Asynchronous

Latency Guarantee arbiter (ALG), to support hard real-time communication. The ALG scheduling mechanism consists of a set of virtual channels and priority adjustment and arbitration component (SPQ) in each switch. The structure of ALG mechanism is illustrated by Figure 2.7.

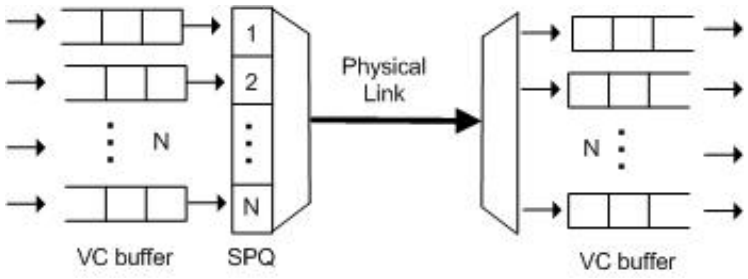


Figure 2.7: ALG Scheduling Mechanism Structure

In traditional fixed priority scheduling, the higher priority flit will always preempt the lower priority flit link resources and leads to latter starvation. So in order to treat different flows more fairly, the ALG scheduler modifies the common fixed priority policy by ensuring that a flit on a given (high priority) VC can block a flit in another (lower priority) VC only once. This policy is implement by cooperation between the VCs and SPQ modules. For instance, the switch has N prioritized VCs (1 indicates the highest priority and N is the lowest priority). When the first flit $f_{P,1}$ from priority P VC tries to access the link, it will wait no more than $P-1$ flit time to get service supposing all the higher priority flits (P -1 in this case) arrive at the same time. When the second flit $f_{P,2}$ coming from the same VC wants to access the output channel, it must wait until all the other flits (up to $N-P$) which have been blocked by flit $f_{P,1}$ are transmitted. In the worst case, a maximum of $P-1$ flits, the number of VCs of higher priority than P, can take priority over the $N-P$ flits that must be transmitted before $f_{P,2}$ gets permission to access the output channel. Thus, the maximum holding time which the flit waits until it is served by the switch is $(N-P)+(P-1)+P = N+P-1$. The service

guarantee is characterized by the priority level of each VC reserved for a flow and the total number of VCs on each link. In order to avoid the flits pending and save the buffer, the traffic condition must be restricted. For any traffic, if the interval time of any two flits with the same priority is more than $N + P - 1$ flit time, the flit will be sure to grant access to the output channel in no more than P flit time. In this situation, only one flit buffer for each priority is enough to ensure the validity of the ALG scheduling mechanism. The guaranteed bandwidth and delay for each priority level P in the switch is $BW/(N + P - 1)$ and P flit time respectively if the source respects the traffic interval condition.

Class Based Priority Arbitration

Schemes QNoC [19] and DiffServ NoC [34] deploy classified traffic-flows with the fixed priority arbitration policy and have a similar arbitration structure as flow based arbitration. In QNoC [19], the author identifies four different types of communication requirements and defines appropriate service levels to support them: Signaling, Real-time, Read/Write and Block-Transfer. There are separate buffers (VCs) for each of the four service levels. A priority ranking is established among the four service levels, where Signaling is given the highest priority and Block-Transfer the lowest. Priority preemption scheduling ensures that data of a higher priority packet is always transmitted before that of a lower service level.

DiffServ NoC [34] classifies the packets before entering the network by performing a selected classification algorithm. Depending on the assigned service ID, the packets will be forwarded to different priority queues in the routers. The classified algorithm firstly identifies and marks the different service level for each packet at the edge router. Each service level has separate virtual channel and priority level at the output port. Thus, differentiated service levels are simply implemented by

means of a priority mechanism. Additional service levels may be defined if desired, as long as a priority ranking is adhered to.

The classified priority arbitration schemes only provides for one class of traffic to receive better service than another class of traffic at a single link. When multiple flows with the same priority compete for the same resources, class based priority service has behavior similar to BE service, with no real-time guarantee [55, 56].

2.2.3 Summary

We have reviewed in this section a number of research works concerned with real-time on-chip networks in this section. To address guaranteed service, two basic approaches: resource reservation and scheduling arbitration are discussed. In the resource reservation approach, all the allocation and schedule are fixed and planned for the peak load demand with exclusive resources. This provides accurate predictability but always suffers lower link utilization. Scheduling arbitration, on the contrary, determines the access to the resource at run-time and offers higher flexibility. Priority based wormhole switching has been proposed as a possible solution for real-time on-chip communication and has received considerable attention. But the major difficult is how to analyze and validate its correctness. In the next section, we will review some existing works about schedulability analysis with priority based wormhole switching.

2.3 Static Analysis of Wormhole Switching

Performance analysis and validation are crucial for real-time network systems to ensure that packet latencies always meet the timing bounds. There are two alternative validation approaches: *simulation* or *algorithm analysis* [52]. The simulation approach constructs a network model and traffic patterns and then the net-

work behavior is simulated by injecting the traffic load into the network. The simulation based method can offer very accurate network behavior but may be very time-consuming. Each simulation run can take considerable time and evaluate only a single network configuration and traffic pattern. Some priority based NoC proposals, such as Chain [29, 28], QNoC [19] and DiffServ NoC [34] use simulation based methods to validate the guaranteed service. But it is very difficult, or impossible, to cover all the system states. So it is appropriate for soft real-time services but it is difficult to validate hard real-time systems. The algorithm analysis approach assumes the worst-case traffic properties and abstracts the network communication models, e.g. the network delivery characteristics and router arbitration behavior. Based on the models, the algorithm is developed to conduct the performance evaluation. It is very fast compared with simulation. The performance bound can be tight enough if accurate models of traffic and network behavior are abstracted. Examples using algorithm analysis approach can be found in [8, 44, 53, 60]. In this thesis, we will especially focus on algorithm analysis. Although in Chapter 6, we undertake some simulation experiments and compare the outcome with that precalculated by the analysis.

2.3.1 Network Model and Constraints

Before analyzing the inter-task communication latency introduced by multi-hop data transmission over the NoC, we need to describe the critical properties of the application and network models.

As a starting point we assume that there are a set of applications which include multiple parallel tasks mapped onto a NoC with a regular topology. The NoC has bidirectional links with uniform bandwidth between nodes. A priority based wormhole switching is implemented with the support of any deterministic routing (e.g. X-Y routing). The flow control protocol (e.g. credit based) ensures

that a router node never sends data what cannot be accepted by the downstream node. Flit-level priority preemption arbitration is assumed, the architecture and operation behavior have been described in Section 2.2.2.2. Here we assume that the tasks allocating to different nodes has already been undertaken. It is evident that different mappings of an application onto a NoC may result in scenarios where the timing constraints may or may not be met, so the static analysis discussed in this section can be used as an acceptance test for a given mapping. The issue of application mapping is beyond the scope of this thesis.

Two schedulability tests for real-time communication networks under fixed priority policy are the *network latency analysis* and the *utilization test*. The evaluation of network latency is based on packet scheduling on the physical link. The concept of *worst-case network latency* is introduced here which is inspired by *worst-case response time* proposed in real-time system scheduling [48]. Under the network latency analysis, the system can be schedulable if all its flows meet their relative deadlines, and the flow meets its deadline if its worst-case network latency is less than or equal to its relative deadline. The research reported in [8, 44, 53] is based on the network latency analysis approach. With the utilization test, the system can be schedulable if the overall utilization of the specific network system is less than a pre-defined upper bound. Mutka's work [60] is an example of utilization based schedulability test.

For analysis and validation, several key constraints and assumptions are introduced in early works [8, 44, 53]:

1. The traffic-flows considered are periodic (or sporadic) and independent. There is no data dependency among the traffic-flows so that each packet can be periodically released.

2. The maximum packet size of each traffic-flow can be found.

3. There are a sufficient number of virtual channels and priority levels at each router port. Each flow will be allocated an exclusive virtual channel. So that blocking due to VC unavailability does not occur. Blocking appears in communication service where the physical resource (virtual channels in this case) is not enough to handle the practical service requests.

4. Priority arbitration determines that at any time a packet will be sent if it has the highest priority among the pending packets and it has remaining buffer in the downstream node. In addition, a higher priority packet can preempt a lower priority packet during its transmission. A flit cannot be preempted, it represents an atom entity that is either fully transfered or not at all.

5. The worst-case release scenario is assumed to occur when all the higher priority flows are released into the network at the same time.

These constraints and assumptions are the foundations to drive the subsequent schedulability analysis. Some of them will in later chapters be altered, remedied or lifted. In Chapter 3, we will discuss new conditions for trigger the worst-case release scenario as these assumptions are not validate in some situations. In Chapter 5, to reduce the resource overhead, more than one flow can be allocated to the same VC and hence share the priority. These improvements either provide more exact schedulability analysis or offer a more realistic network model.

2.3.2 Traffic-Flow Model

The basic model that is considered in this thesis is a network system Γ that consists of N real-time traffic-flows, $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$. Each traffic-flow τ_i is characterized by a set of attributes $\tau_i = (P_i, C_i, T_i, D_i, J_i^R)$.

- Priority P. Each traffic-flow has a distinct priority value P. All the packets that belong to τ_i inherit the same priority P_i. The value 1 denotes the highest

priority and larger integers denote lower priorities. We assume the traffic-flow is prioritized by any possible global priority assignment policy. The issue of priority assignment will be discussed in Chapter 4.

- Maximum basic network latency C. The *basic network latency* occurs when no traffic-flow contention exists. The basic network latency is determined by its source/destination routing distance, packet size, link bandwidth and some additional protocol control overheads. For real-time evaluation purposes, we must consider the *maximum basic network latency*. Let the maximum packet size belonging to τ_i be $Length_i^{max}$, the maximum basic network latency C_i is given by:

$$C_i = \lceil \frac{Length_i^{max} + L^{add}}{Flit_{size}} \rceil \cdot Flit_{size}/BW + Hops \cdot R_{hop} \qquad (2.10)$$

where L^{add} is the additional data for wormhole switching such as header and tail flit information. The second term in the equation counts the routing delay experienced by the header flit. Once the header flit reaches the destination, the transmission time of this pipelined packet is determined by the packet size and network bandwidth which is accounted by the first term. Here we only consider the longest possible, or maximum, basic network latency for evaluation. If the real-time traffic-flow can meet its deadline with the maximum basic network latency, it will meet the deadline for any other basic network latency scenario.

- Period T. We assume that all the traffic-flows which require timely delivery are periodic or sporadic. For periodic traffic-flow, the packet inter-arrival time is known and fixed. The sporadic flow model is the same as the periodic flow model except that packet generation time occur with a minimum interval time, rather than exactly every period. For the sake of generalization, the

attribute period T is defined as the lower bound interval on the time between releases of successive packets. This can be done because in the worst-case sporadic tasks can be seen as periodic.

- Deadline D. Each real-time traffic-flow has deadline D constraint, which means all the packets belonging to this traffic-flow have the restriction that they should be delivered from a source router to a destination router within a certain delay bound even in the worst-case situation. As an initial restriction on the model, all the analysis approaches in this section suppose that each traffic-flow's deadline must be less than or equal to its period, $D_i \leq T_i$ for all $\tau_i \in \Gamma$. This model restriction will be removed in Chapters 3 and 5.

- Jitter J^R. The attribute J_i^R is the release jitter [7] of flow τ_i and denotes the maximum deviation of successive packets released from its period. A periodic traffic-flow τ_i has to generate the packet within a fixed time which is its period, T_i, then it will be released as soon as it generated. However, when this periodic flow τ_i is subjected to release jitter, its generated time becomes under some circumstances different from its release time. So, τ_i is not become strictly periodic and a variation in its release times has arisen. So, release jitter of a flow is defined as the maximum variation in its release times. If an expected release time of packet from a periodic traffic-flow τ_i is a, then the real release time may occur later than normal, by the time no more than $a + J_i^R$.

2.3.3 Inter-Relationships between Traffic-Flows

To capture the relations between traffic-flows and the physical links of the network, we formalize the mesh network topology defined as a directed graph $\mathbb{G} : V \times E$. V is a set, whose elements are called nodes, each node v_i de-

notes one router in the mesh network. E is a set of ordered pairs of vertices, called edges. An edge $e_{x,y} = \{v_x \rightarrow v_y\}$ is considered to be a real physical link from router v_x to router v_y; v_x is called the source and v_y is called the destination. We define a mapping space from the traffic-flow set to the physical links $\Gamma \rightarrow \mathbb{G}$. Given a set of n traffic-flows Γ, we can map them to the target network. The routing $\Re_i$ of each traffic-flows τ_i is denoted by the ordered pairs of edges, $\Re_i = \{e_{1,2}, e_{2,3}, \ldots, e_{n-1,n}\}$. If a traffic-flow τ_i shares at least one link with τ_j, the intersection set for τ_i and τ_j is computed as $\Re_i \cap \Re_j$. If $\Re_i \cap \Re_j = \emptyset$, τ_i and τ_j are disjoint (share no physical link).

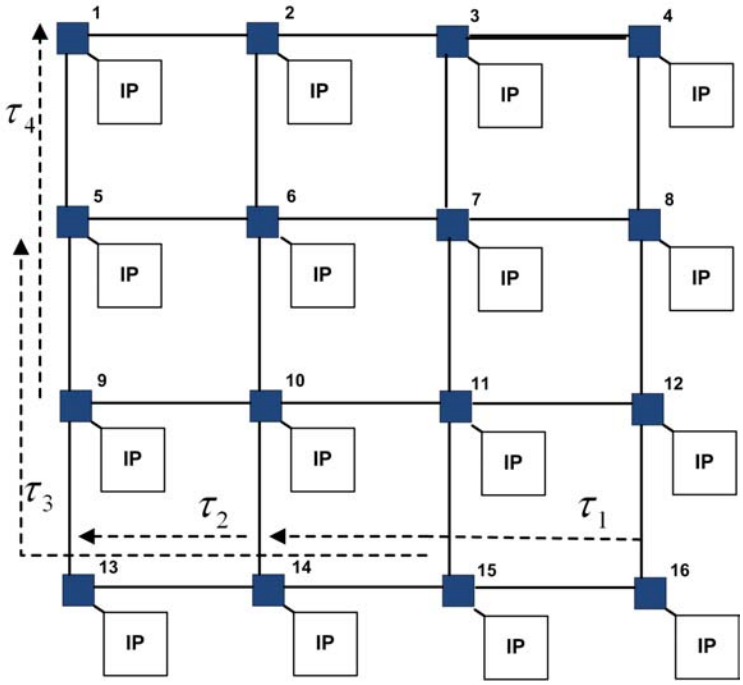


Figure 2.8: An Example Set of Traffic-Flows

For the example in Figure 2.8, the routing of τ_1, τ_2, τ_3 and τ_4 are:

$$\Re_1 = \{e_{16,15}, e_{15,14}\},$$

$$\Re_2 = \{e_{14,13}\},$$

$$\Re_3 = \{e_{15,14}, e_{14,13}, e_{13,9}, e_{9,5}\}$$

$\Re_4 = \{e_{9,5}, e_{5,1}\}$.

The intersection sets between them are:

$\Re_1 \cap \Re_2 = \Re_1 \cap \Re_4 = \Re_2 \cap \Re_4 = \emptyset$,

$\Re_1 \cap \Re_3 = \{e_{15,14}\}$,

$\Re_2 \cap \Re_3 = \{e_{14,13}\}$

$\Re_3 \cap \Re_4 = \{e_{9,5}\}$.

Based on whether they share the same physical links, we categorize the competition relationship between the traffic-flows into two different types: *direct competing relationship*, and *indirect competing relationship*. The direct competing relationship means a traffic-flow has at least one physical link in common with the other traffic-flow. For any two traffic-flows τ_i and τ_j, if a direct competing relationship exists between them, then $\Re_i \cap \Re_j \neq \emptyset$. With the indirect competing relationship, by comparison, the two traffic-flows do not share any physical link but there is (are) intervening traffic-flow(s) between the given two traffic-flows. For example, if there are three flows τ_i, τ_j, τ_k meeting the following situation, $\Re_k \cap \Re_i = \emptyset$, $\Re_k \cap \Re_j \neq \emptyset$ and $\Re_i \cap \Re_j \neq \emptyset$; τ_j is the intervening flow in this case, then there is an indirect competing relationship between τ_i and τ_k. Note that the indirect competing relation is transitive. If more than one intermediate flow exists, the indirect competing relationship still holds. Following the above case, if there is a new flow τ_a which has an indirect competing relationship with τ_j (τ_k is intermediate flow between τ_a and τ_j), and τ_a does not share any physical link with τ_i, then there is an indirect competing relationship between τ_a and τ_i (both τ_k and τ_j are intermediate flows). Figure 2.9 shows the situation.

Assuming that the priorities have been assigned to each traffic-flow, if a packet is released, the possible delay it suffered before completion consist of all the possible interferences from higher priority traffic-flows. Based on different priority

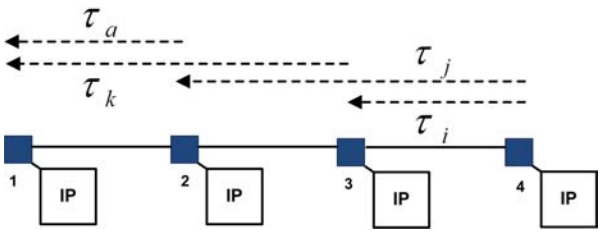


Figure 2.9: Transitivity in Indirect Competing Relationship

levels and the competing relationships, two different interferences are introduced [8, 44]:

- **Direct interference from higher priority traffic-flow**

  When two traffic-flows τ_i and τ_j have a direct competing relationship and meet the condition $P_j > P_i$, τ_j will force a direct contention with the observed traffic-flow τ_i. For flow τ_i, we define a **direct interference set** S_i^D which includes all the traffic-flows meeting the above conditions, S_i^D = $\{\tau_j | \Re_j \cap \Re_i \neq \emptyset$ *and* $P_j > P_i$ *for all* $\tau_j \in \Gamma\}$.

- **Indirect interference from higher priority traffic-flow**

  When two traffic-flows τ_i and τ_k have an indirect competing relationship and meet the condition $P_k \geq P_j > P_i$, where τ_j is the intervening traffic-flow, τ_i may suffer an indirect interference from τ_k even when they do not share any physical link. For flow τ_i, we define an **indirect interference set** S_i^I which includes all the traffic-flows meeting the above conditions, S_i^I = $\{\tau_k | \tau_k$ has indirect competing relationship with τ_i *and* $P_k \geq P_j > P_i$ *for all* $\tau_k \in \Gamma$, where τ_j is any intermediate flow$\}$.

Return to the example in Figure 2.8. The four prioritized traffic-flows sorted from high to low priority are τ_1, τ_2, τ_3 and τ_4. Flows τ_1 and τ_2 have no shared links with any other higher priority traffic-flow so no direct or indirect interference,

$$S_1^D = S_1^I = S_2^D = S_2^I = \emptyset,$$

Flow τ_3 shares the link resources with τ_1 and τ_2 and gets

$$S_3^D = \{\tau_1, \tau_2\}, \text{ and } S_3^I = \emptyset.$$

Flow τ_4 directly contends with τ_3 and indirectly competes with τ_1 and τ_2, so

$$S_4^D = \{\tau_3\} \text{ and } S_4^I = \{\tau_1, \tau_2\},$$

With the definition of system model and relationship between traffic-flow, we will review current schedulability analysis works in the following sections.

2.3.4 Mutka's Contribution

Mutka [60] was the first to address the analysis in priority based wormhole switching. The global fixed priorities are assigned to all the flows that share a common link based on the Rate Monotonic (*RM*) algorithm [51], i.e., flows with shortest period have highest priority. A channel arbitrator at each router port uses priority to decide which flits should get transmission next. As a classic algorithm for fixed priority single processor scheduling, Liu and Layland [51] gave an utilization bound test for RM to guarantee the schedulability of the task set. For n tasks that share the same processor, if the total processor utilization $\sum_{i=1}^{n} \frac{C_i}{T_i}$ is no more than the utilization bound which is given by $n(2^{1/n} - 1)$, then all the tasks are all schedulable. The RM test is given as:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.11}$$

where $n(2^{1/n} - 1)$ decreases monotonically from 0.83 when $n = 2$ to $ln2$ (≈ 0.69) when $n \to \infty$.

Mutka applied this important result to wormhole switching networks where the schedulability of each flow is checked using the RM utilization bound at each physical link along the flow's path. For n traffic-flows that have their paths overlap

on a physical link, if the total link utilization is no more than the specific utilization bound, then all the flows which share this physical link can meet their deadlines. However, we must point out that this test is not adequate for guaranteed delivery data packets under wormhole switching, since it assumes that each link can be analyzed independently in terms of link bandwidth utilization. This is true only if each VC has enough space to buffer the entire packet at intermediate hops. Obviously this is not the case of wormhole switching within which only small flit space are available in each VC and one packet can be distributed across several routers. We will give a contradiction example in Chapter 3 to show that the RM test is not adequate for determining the schedulability.

2.3.5 Balakrishnan's Contribution

The priority based wormhole switching in Balakrishnan's [8] model does not allow packet interleaving and the preemption must happen at packet boundaries. Based on this assumption, an off-line analysis techniques, a lumped link model, is presented. In the lumped link model, all the links the traffic-flow travels are lumped as one shared resource and the interference due to all higher priority flows using one or more of the links along the path of the flow is calculated.

The algorithm [8] firstly tries to find the maximum interference set. For an observed flow τ_i, any higher priority flow which shares one or more physical links with τ_i are grouped into set S_i^D. These flows contribute the direct interference to τ_i packet delay. Any flow which does not share a link with τ_i but may have indirect competing relationshion with τ_i will be grouped into set S_i^I. The author define a new set S including all the possible flows which directly or indirectly contends with τ_i, $S = S_i^D + S_i^I$. With the higher priority flow set, the worst-case interference

is derived by following Eq.(2.12)

$$F(t) = \sum_{\forall \tau_j \in S} \lceil \frac{t}{T_j} \rceil C_j \tag{2.12}$$

where $F(t)$ is the time demand function of t. The sum $F(t)$ is the required service time of all the higher priority flows that contend with the flow τ_i up to time t. This equation can be solved using an iterative technique [7]. Let t^n be the n^{th} iterative value generated from the equation:

$$F(t^n) = \sum_{\forall \tau_j \in S_i} \lceil \frac{t^n}{T_j} \rceil C_j \tag{2.13}$$

The iteration starts with $t^0 = \sum_{\forall \tau_j \in S_i} C_j$ and terminates when $F(t^n) = t^n$, which is the maximum interference of τ_i. If $t^n + C_i \leq D_i$, then τ_i meets the deadline.

The analysis based on the lumped link model is efficient but the analysis result is quite pessimistic. Since all the direct and indirect flows that contend with the observed flow are lumped into a single unit in this model. The indirect flows will therefore be treated in the same way as direct flows. This introduces excessive interference to the current flow since all the competing flows no mater direct or indirect must be scheduled in sequence [53]. Balakrishnan [8] also admits this over-calculation is not necessarily the case.

2.3.6 Kim's Contribution

Kim's [44] model permits flit-level preemption and evaluates worst-case network latency by a timing line diagram. Kim's algorithm consists of generating a direct initial timing diagram and then modifying this diagram by deleting all the indirect time slots that might not contend.

The first step of the algorithm is to construct a higher priority flow set S for the

observed flow. The process of construction is the same as Balakrishnan's [8] where $S = S_i^D + S_i^I$. After generating higher priority flow set S, the algorithm creates the initial timing diagrams for each flow. The time slot is the basic unit of the timing diagram which indicates a flit level block transmission in the network. The slots in the timing diagram can be set a value to either Allocated, Waiting, Busy or Free. Allocated means that time slot is allocated to a flow for transmission. Waiting means preempted state and Busy indicates that a higher priority packet is using that time slot. Free means that time slot is usable.

For easy description, we consider a case illustrated in Figure 2.8. The attributes of the traffic-flows are showed in Table 2.1. The time units are not necessary in this analysis as long as all the traffic-flows use the same base.

Real-Time Traffic-flow	P	C	T	D
τ_1	1	1	5	5
τ_2	2	2	7	7
τ_3	3	2	9	9
τ_4	4	4	12	12

Table 2.1: Traffic-flows Description

An initial timing line diagram for τ_4 is established in Figure 2.10 with the assumption that all the flows (τ_1, τ_2, τ_3) in S directly contend with τ_4. As the time advances, the free time slots that are not used by flows in set S can be used by τ_4. The delay upper bound of τ_4 is the time when the summation of all these free time slots equals the basic network latency. The initial network latency upper bound is 18.

The initial result is pessimistic since it does not distinguish between the direct and indirect high priority flows. If any indirect higher priority flow exist, we need to go back to the initial time diagram and check all the allocated and busy time slots of the indirect flows. A time slot used by an indirect flow can be freed if all the intermediate flows do not require that time slot. A released time slot can be

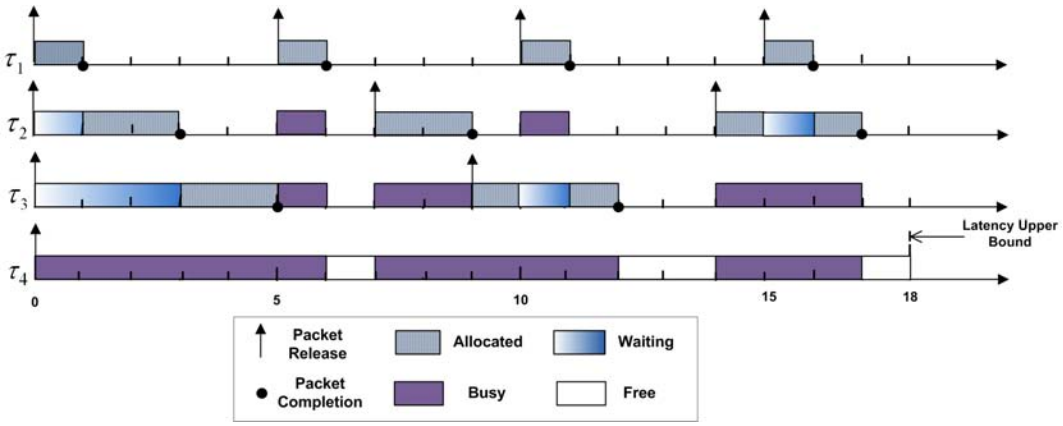


Figure 2.10: Initial Timing Diagram

reused by other flows.

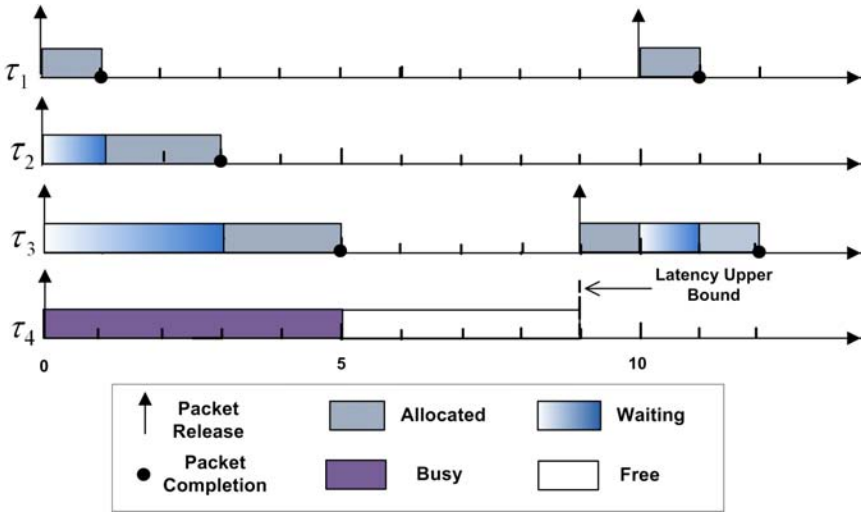


Figure 2.11: Final Timing Diagram

The final timing diagram is showed in Figure 2.11. Flows τ_1 and τ_2 indirect contend with τ_4. Compared with initial timing graph in Figure 2.10, during time slots [5,6], [7,9] and [14,16], there is no active packet service from intermediate τ_3. So the time slots of second and forth instances from τ_1 and second and third instances from τ_2 are removed. Because of the released time slots, the network latency for τ_3 and τ_4 are finally computed to be 5 and 9.

Kim's algorithm consider the different between direct and indirect higher priority flows. But it still introduces excessive interference to the current flow because when the first instance of τ_1 or τ_2 access the physical links, only τ_3 gets blocked. In this case, τ_4 may transmit while τ_3 is blocked, resulting in an smaller latency upper bound. In Chapter 3, we will present new analysis which can distinguish between direct and indirect interference properly. The indirect interference in that analysis is treated as interference jitter and a tighter network latency upper bound is achieved.

2.3.7 Lu's Contribution

Lu *et al* [53] utilized a similar timing diagram as Kim to find the worst-case network latency. The improvement of Lu is in proposing a novel contention tree model to distinguish direct and indirect interference and capture concurrent usage of links. The contention tree is defined as a directed graph: a flow τ_i is represented as a node τ_i in the tree; an edge $E_{i,j}$ directs from node τ_i and τ_j, τ_i is the parent, τ_j is the child. Given a set of traffic-flows, the corresponding contention tree is built as following:

1. Sort the traffic-flows with a priority sequence.

2. Construct the tree from the highest priority flow. If τ_i shares at least one link with τ_j, (τ_i gets higher priority than τ_j), an edge $E_{i,j}$ is created between them.

In the contention tree, a direct contention is represented by a direct edge between them while an indirect contention is implied by a path via a parent node(s). The case in Figure 2.8 is presented by the contention tree in Figure 2.12. Three edges $E_{1,3}$, $E_{2,3}$ and $E_{3,4}$ represent the direct contentions. Two indirect contention

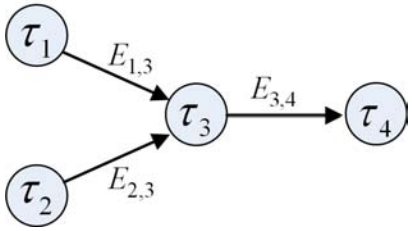


Figure 2.12: Contention Tree

relations for τ_4 are implied by two paths, $E_{1,3} \rightarrow E_{3,4}$ and $E_{2,3} \rightarrow E_{3,4}$, from τ_1 and τ_2 via τ_4 parent's τ_3.

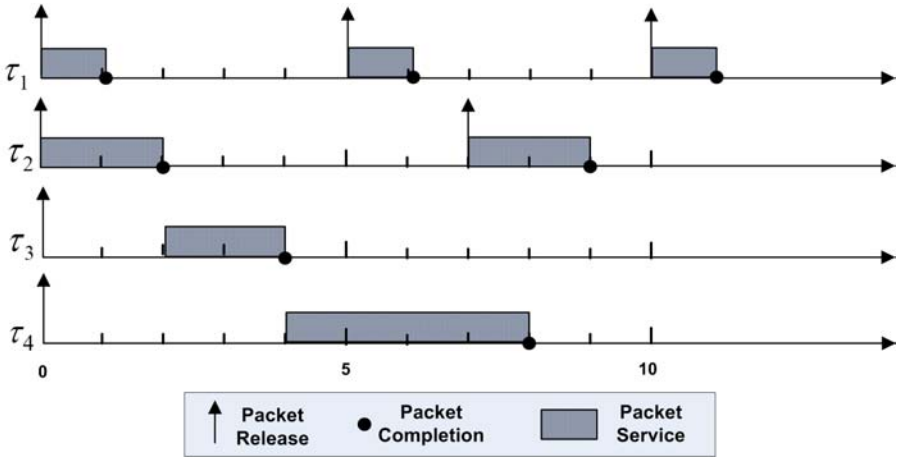


Figure 2.13: Scheduling with Contention Tree Model

Utilizing the contention tree, the authors addressed a parallel communication phenomenon in the timing diagram. From Figure 2.13, we can observe that

1. τ_1 and τ_2 are served by the network in parallel. This concurrency is reflected by the disjoint nodes in the contention tree since they do not share any physical link;

2. τ_3 is transmitted on the overlapped free time slots between [2, 4] after τ_1 and τ_2 completion since it has direct contention relation with τ_1 and τ_2;

3. τ_4 is scheduled only after τ_3 completes transmission at time 4. The service

time slices [0, 2] from τ_1 and τ_2 is a direct contention delay for τ_3 but an indirect contention for τ_4.

Compared with Kim's approach, the contention tree model considers possible parallel scheduling in disjoint communication links and thus reduces the possible interference from higher priority flows. The worst-case latency of τ_3 and τ_4 analyzed by this model are 4 and 8 which is smaller than Kim's result. However, the contention tree model assumes that the worst-case scenario happens when all the flows are released at the same time. This is not true when indirect interference is considered. In the next chapter we will give a contradictory case to show that worst-case network latency does not always happen at that point. As a result, Lu's analysis is optimistic. In addition, we prove that finding the worst-case scenario when parallel communication occurs is NP-hard.

2.4 Summary

This chapter has presented a review of background material, including on-chip network structures, current implementation schemes for real-time services and schedulability analysis of priority-based wormhole switching. As can been seen from the covered contributions, the work that is related to schedulability analysis with wormhole switching is either pessimistic or improper. This motivates the design of a novel analysis approach. In the next chapter, by identifying diverse inter-relationships among the traffic-flows, an approach is derived that can evaluate the packet network latency for all practical situations.

Chapter 3

Schedulability Analysis for Priority Based Wormhole Switching

As seen in the last chapter, priority based wormhole switching has been proposed as a possible solution for real-time on-chip communication. Also, several existing schedulability analysis approaches and their weaknesses were addressed. In this chapter, novel, tighter, worst-case network latency analysis is presented, which successfully emulates wormhole switching as a classic single processor scheduling model. By evaluating diverse inter-relationships and service attributes among the traffic-flows, this approach can predict the packet transmission latency for a given traffic-flow based on three quantifiable different interferences: direct interference from higher priority traffic-flows, indirect interference from higher priority traffic-flows and self-blocking produced by the previous packet of the same traffic-flow. Section 3.1 discuss the analysis of direct and indirect interferences; then the analysis is extended to include the self-blocking interference in Section 3.2. Due to the inevitable existence of parallel communication, in Section 3.3, we prove that the general problem of determining the exact schedulability of real-time traffic-flows over the on-chip network is NP-hard. We also prove that the real

maximum network latency is bounded by the theoretical calculation in our model. Section 3.4 provides an evaluation of randomly generated traffic-flow sets to show how the network latency test is better than the previous published one. A summary of the chapter is provided in Section 3.5.

3.1 Basic Network Latency Analysis

In priority based wormhole switching, the packet advances when it has the highest priority and remaining buffer in each router along its path. So the whole set of links used by a traffic-flow are treated as one resource and other traffic-flows using one or more links along the path of the observed flow produce contention interference. To determine the upper bound of network latency for a real-time traffic-flow, the maximum basic network latency and maximum contention interference need to be calculated. The maximum basic network latency can be calculated by static analysis of the traffic-flow pattern and the network features. Eq.(2.10) gives an upper bound of packet latency without contention in the network. The important factor dominating the latency upper bound is interference. In this section, we assume that the analysis is based on the single packet instance, which means for a traffic-flow, before the current packet instance is accepted by the destination, the successive packet can not be generated from its source node. In another word, the restriction condition $D \leq T - J^R$ is always true. So under the fixed priority arbitration discipline and single packet instance assumption, only traffic-flows with higher priority than the current one can cause interference. In order to evaluate these possible competing interferences, we should find all the higher priority traffic-flows which could affect the observed one and calculate the interference upper bound by analyzing the characteristics of these higher priority traffic-flows. Also, we notice that there is a resource competition relationship between the ob-

served flow and its relevant higher priority flows. This is similar to the processor resource model [51, 7, 77] in real-time scheduling, in which all the tasks contends for a single shared processor resource. In wormhole switching networks, the shared physical communication links are also contended for by the associated traffic-flows. Utilizing priority preemption, we can analyze this model following the real-time scheduling approach for single-processors.

The analysis algorithm is based on the computation of the worst-case network latency R for each traffic flow. If $R_i \leq D_i$, the analysis ensures that flow τ_i always meets its time boundary. We have listed several key constraints and assumptions introduced by early works in Section 2.3.1 for the analysis of wormhole switching. These assumptions and constraints are still valid in our analysis. Besides that, we need to find the condition which can trigger the traffic-flow's worst-case network latency.

Inspired by Liu and Layland's seminal paper [51], two major conditions to achieve the worst-case latency are identified:

Condition 1. *All the higher priority traffic flows release packets at their maximum rate with maximal packet size.*

Condition 2. *The packet from the observed traffic flow is released simultaneously with all the higher priority packets (referred to as the critical instant).*

These conditions have been used by a series of research works [8, 35, 53] to obtain the maximum network latency with their scheduling analysis techniques. Following the same foundations, we will quantify the new network latency analysis based on two distinguishing interferences:

- Direct interference from higher priority traffic-flows.
- Indirect interference from higher priority traffic-flows.

3.1.1 Direct Interference From Higher Priority

This section covers the network latency analysis of a traffic-flow assuming only direct higher priority traffic-flows exist. The direct and indirect higher priority traffic-flows sets S_i^D and S_i^I used in this chapter have the same definition as in Chapter 2. The worst-case latency occurs when the two critical conditions are satisfied.

Theorem 1. *The maximum network latency R_i for traffic-flow τ_i can be found by:*

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R}{T_j} \rceil C_j + C_i \tag{3.1}$$

when only direct higher priority traffic-flows exist.

Proof. For an observed traffic-flow τ_i, the network latency R_i of a packet released from τ_i is computed as:

$$R_i = C_i + I_i \tag{3.2}$$

where I_i is the interference summation from the higher priority traffic-flow(s). The maximum basic network latency C_i is constant and known in advance by static analysis. Supposing that we can find an upper bound of the total network latency R_i, the maximum number of instances released from any higher priority flow τ_j is no more than:

$$\lceil \frac{R_i + J_j^R}{T_j} \rceil \tag{3.3}$$

assuming the worst-case release scenario of τ_j: the first packet release starts J_j^R later than the normal arrival, and the subsequent releases are at the maximum release rate of $1/T_j$, and are of maximum size. These higher priority packets will immediately preempt τ_i's packet transmission and delay the latter. Consequently, the interference summation from all the flows in S_i^D is thus:

$$I_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R}{T_j} \rceil C_j \tag{3.4}$$

Using Eq.(3.4) to substitute I_i in Eq.(3.2), we can produce:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R}{T_j} \rceil C_j + C_i \tag{3.5}$$

□

We note the variable R_i appears on both sides of the Eq.(3.1). This equation can be solve using an iterative technique [7]. Let R_i^{n+1} be the $(n+1)^{th}$ iterative value generated from the equation:

$$R_i^{n+1} = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^n + J_j^R}{T_j} \rceil C_j + C_i \tag{3.6}$$

The iteration starts with $R_i^0 = C_i$ and terminates when $R_i^{n+1} = R_i^n$. This iteration also should halt if $R_i^{n+1} > D_i$, which denotes the deadline miss for this packet. By this iterative technique, the worst-case network latency with direct interference can be calculated (R_i=R_i^{n+1}=R_i^n).

3.1.2 Indirect Interference From Higher Priority

The model indicated by Eq.(3.1) only considers the direct interference from higher priority traffic-flow since their activities directly delay the completion time of the observed flow. However, when indirect higher priority traffic-flow exits, some extra interference also needs to be taken into account.

Consider the following scenario: $\tau_1, \tau_2, \ldots, \tau_k$, τ_j and τ_i are loaded on the network with the inter-relations of traffic-flows illustrated in Figure 3.1. Traffic-flows are sorted with descending priority, τ_1 with highest priority, τ_i with lowest

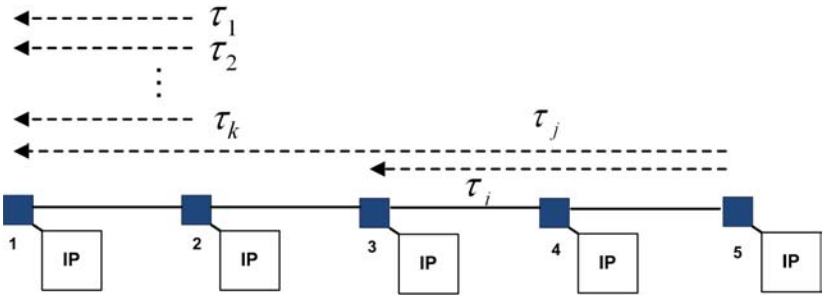


Figure 3.1: A Case of Indirect Interference

priority. Here we follow the old fashion (reverse alphabetical order) [51, 7, 77] that τ_i as the observed flow has the lowest priority and τ_j, τ_k have higher priority than τ_i, $P_k > P_j > P_i$. We examine the competing relation of τ_i and τ_j and get S_i^D = $\{\tau_j\}$, S_i^I = $\{\tau_1, \ldots, \tau_k\}$, S_j^D = $\{\tau_1, \ldots, \tau_k\}$ and S_j^I = $\emptyset$. Without considering the possible indirect interference from $\tau_1, \ldots, \tau_k$, the worst-case network latency of τ_i occurs when τ_j and τ_i are released at the same time.

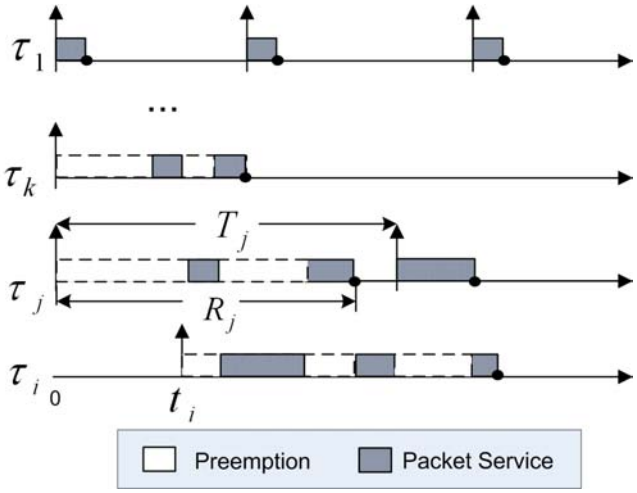


Figure 3.2: The Problem of Indirect Interference

When $\tau_1, \ldots, \tau_k$ as indirect higher priority traffic-flows are taken into account, even though they do not share any physical link with τ_i, we find their services still can impose an extra interference on τ_i. The time-line graph in Figure 3.2

shows such a situation. The solid up arrow in the graph indicates the release time of a packet from a traffic-flow. A packet served by the network for some time is depicted as a shaded rectangle. The preemption of a packet is depicted as a white rectangle. The bold circle denotes the time when the complete packet is received at the destination. We assume τ_j is released with the other higher priority packets at the same time 0, the packets from $\tau_1, \ldots, \tau_k$ will contend with τ_j. This contention delays the start transmission time of τ_j until a time instant t_i, t_i is the start time of τ_j first transmission service. At the time $0 + t_i$, τ_i is released, the packet from τ_j immediately preempts τ_i. It is easy to find that τ_j imposes the interference C_j upon the τ_i during the time interval $[t_i, R_j]$. At the time $0+T_j$, τ_j is released again but this time all the higher priority traffic-flows $\tau_1, \ldots, \tau_k$ only send a very small packet or even do not send any packet. Flow τ_j in this situation does not suffer any interference from them and gets the network service immediately. From the view point of τ_i, the time interval between two successive releases from τ_j is only (T_j - t_i). Consider our original assumption, the worst-case network latency occurs when all the higher priority packets are released simultaneously with their maximum packet sizes and the minimum packet release interval T. The maximum interference a packet from τ_i suffered from a higher priority traffic-flow is calculated by $\lceil \frac{R_i+J_j^R}{T_j} \rceil C_j$. But in this case, the minimum interval between subsequent preemptions from τ_j is only (T_j - t_i) which is less than the original minimum interval assumption T_j. Note that this phenomenon can only occur when considering indirect interferences.

Theorem 2. *The upper bound of interference suffered by τ_i from direct higher priority traffic-flow τ_j is:*

$$\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \rceil C_j \tag{3.7}$$

when the indirect interference is considered.

Proof. Let s_j denote the packet release time from τ_j. In this analysis, assume without loss of generality, that the first packet is released at time 0. Therefore, each packet from τ_j is generated periodically at the time instant 0, T_j, $2T_j$, ..., qT_j. Let $s_{j,q} = (q-1)T_j$, where q is the sequential number of the packets. But in the practical environment, an application does not always follow the strict periodic releases and has application release jitter. More specifically, the release time $s_{j,q}$ satisfies:

$$(q-1)T_j \leq s_{j,q} \leq (q-1)T_j + J_j^R \tag{3.8}$$

In addition, in Figure 3.2, we observe that the possible interference from higher priority packets also defers its starting transmission time. If the worst-case network latency for τ_j is R_j, the upper bound of the start transmission time is $R_j - C_j$. The real service transmission start time for each packet satisfies:

$$(q-1)T_j \leq s_{j,q} \leq (q-1)T_j + J_j^R + R_j - C_j \tag{3.9}$$

Now we evaluate the maximum interference suffered by τ_i from τ_j in a given time interval. Here we assume the start transmission time of τ_j is $a = R_j - C_j + J_j^R$, this is the upper bound since it was released. A packet from τ_i is released simultaneously with τ_j and b is the corresponding completion time of this release. The worst-case interference occurs when the packets from τ_j are released at their maximum frequency since τ_i is released. Figure 3.3 illustrates this situation.

The number of preemptions by τ_j is given by the positive integer number g within the interval $[a, b]$, $g \in \mathbb{N}$. The last release of τ_j should fall into the interval before the completion of τ_i, therefore g is the largest value that satisfies:

$$a - (R_j - C_j) - J_j^R + (g-1)T_j < b \tag{3.10}$$

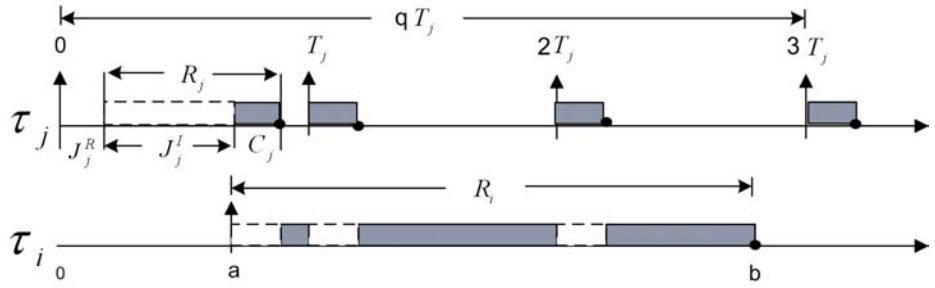


Figure 3.3: Upper Bound Analysis of Indirect Interference

or, equivalently,

$$g < \frac{J_j^R + R_j - C_j + b - a}{T_j} + 1 \tag{3.11}$$

The largest positive integer number satisfying this inequality is given by

$$g = \lceil \frac{J_j^R + R_j - C_j + b - a}{T_i} \rceil \tag{3.12}$$

The interval $[a, b]$ marks the worst-case network latency of τ_i, $b - a = R_i$. Therefore, the interference upper bound from τ_j is:

$$\lceil \frac{R_i + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{3.13}$$

□

The packet from τ_i will experience worse latency than that predicted by Eq.(3.1) on account of the indirect interference from $\tau_1, \ldots, \tau_k$ which delays τ_j and may cause more hits on τ_i. Therefore, the worst-case network latency does not occur when the packet is released simultaneously with all the higher priority packets (at time 0) but at the point when the packet from the observed traffic-flow is released at the same time as all the direct higher priority packets finish waiting and start transmission. So we modify our critical instant (Condition 2) with:

Condition 3. *The critical instant for each flow happens at the time that the packet from the observed traffic-flow is released exactly when all the direct higher priority packets are ready to start transmission.*

Actually, different traffic flows in NoC may share different sets of physical links according to their routes and hence have their own direct and indirect higher priority flows. So each flow could has its own critical instant. It is unlikely that time 0 will be the critical instant for all the traffic-flows. For example, the case in Figure 3.2 shows that τ_j's critical instant is 0 since it is released with all its higher priority flows $\tau_1, \ldots, \tau_k$. But for τ_i, it's critical instant is t_i. In Chapter 6, by a practical application simulation, we will show further evidence that the critical instant for all the flows is not necessarily the same.

The deviation induced by indirect interference from higher priority flows between consecutive releases is called *interference jitter*, using symbol J^I to denote it. The interference jitter of a traffic-flow is the maximum deviation between two successive packet start transmission times which can be obtained by computing the difference between its maximum and minimum values. Consider the situation that no higher priority packet is sent in a period, the minimum packet start transmission time becomes zero. Accordingly, the upper bound of interference jitter for a traffic-flow is given by:

$$J_j^I \leq R_j - C_j \tag{3.14}$$

Note that not all the traffic-flows suffer interference jitter, this only happens when the observed traffic-flow τ_i has indirect interference, namely, J_j^I exists if and only if $S_j^D \cap S_i^I \neq \emptyset$.

Theorem 3. *The network latency upper bound R_i for traffic-flow τ_i can be found by:*

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R + J_j^I}{T_j} \rceil C_j + C_i \tag{3.15}$$

when both direct and indirect higher priority traffic-flows exist.

Proof. I_i is the maximum interference from flows whose priorities are higher than the current flow τ_i. I_i can be presented by a summation of all $I(j)$; where $I(j)$ is the maximum interference from τ_j, $\tau_j \in S_i^D$:

$$I_i = \sum_{\tau_j \in S_i^D} I(j) \tag{3.16}$$

According to Theorem 2, the maximum amount of interference from all flows that have higher priority than τ_i is given by:

$$I_i = \sum_{\tau_j \in S_i^D} \lceil \frac{R_i + J_j^R + J_j^I}{T_j} \rceil C_j \tag{3.17}$$

As a result, the worst-case network latency in case of interference jitter and release jitter is calculated as follows:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j^R + J_j^I}{T_j} \rceil C_j + C_i \tag{3.18}$$

□

Solving Eq.(3.15) is achieved by a recurrence relation as in Eq.(3.19).

$$R_i^{n+1} = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \rceil C_j + C_i \tag{3.19}$$

where R_i^0 = C_i until we get $R_i = R_i^{n+1} = R_i^n$ ($n = 0, 1, ..$). However, if R_i^{n+1} becomes greater than the deadline, we say that the flow is unschedulable.

We modify our traffic-flow attributes with the six-tuple $(P_i, C_i, T_i, D_i, J_i^R, J_i^I)$ for τ_i. Differing from the previous works [8, 44, 53], here we address the new critical instant condition for each flow and introduce interference jitter to address the affect from indirect high priority traffic-flow. A tighter upper bound for worst-case network latency is obtained with our new approach. Since a schedulable real-time system is a system where all the flows meet their relative deadlines, we can utilize this new analysis approach, provided by Eq.(3.1) and Eq.(3.15), as a test tool to determine the schedulability of the whole system.

3.1.3 Numeric Examples

Revisiting the example given in Figure 2.8 with the attributes in Table 2.1 (page 59). Since the higher priority traffic-flow always forces interference to the lower priority traffic-flow and extends the latter's network latency, we sort the traffic-flows based on their priority and proceed to analyze them from the highest priority one by one.

Traffic-flows τ_1 and τ_2 do not suffer any contention and receive the worst-case network latency equal to their maximum basic latency, R_1=C_1= 1, R_2= C_2=2. Thus τ_1 and τ_2 are schedulable. Flow τ_3 shares the physical link with the higher priority traffic-flows τ_1 and τ_2, S_3^D= $\{\tau_1, \tau_2\}$, S_3^I = $\emptyset$. The network latency for τ_3 according to Eq.(3.1):

$$R_3 = C_3 + \lceil \frac{R_3}{T_2} \rceil C_2 + \lceil \frac{R_3}{T_1} \rceil C_1$$

$$R_3^0 = 2$$

$$R_3^1 = 2 + \lceil \frac{2}{5} \rceil 1 + \lceil \frac{2}{7} \rceil 2 = 2 + 1 + 2 = 5$$

$$R_3^2 = 2 + \lceil \frac{5}{5} \rceil 1 + \lceil \frac{5}{7} \rceil 2 = 2 + 1 + 2 = 5$$

The recurrence stops at R_3 = 5 so the worst-case network latency of τ_3 is 5. Note that τ_3 maximum latencies based on Kim's and Lu's analysis are 5 and 4 respec-

tively (see Section 2.3). It seems that Lu's result is more exact than the one presented here. But we will show in Section 3.3 that Lu's analysis is actually optimistic (i.e. unsafe).

Flow τ_4 suffers both direct and indirect interferences with S_4^D=$\{\tau_3\}$, S_4^I = $\{\tau_1, \tau_2\}$. Based on the principle proposed in Section 3.1.2 when indirect interference exists, we treat indirect interference as interference jitter and therefore update the attributes of our example. The interference jitter of traffic-flow τ_3 referred to τ_4 equals $R_3 - C_3$ = 3. Eq.(3.15) becomes

$$R_4 = C_4 + \lceil \frac{R_4 + J_3^I}{T_3} \rceil C_3$$

$$R_4^0 = 4$$

$$R_4^1 = 4 + \lceil \frac{4+3}{9} \rceil 2 = 6$$

$$R_4^2 = 4 + \lceil \frac{6+3}{9} \rceil 2 = 6$$

which results in at R_4 = 6. So the worst-case network latency of τ_4 is 6 with both direct and indirect interference. Again, the traffic-flow meets its deadline. Compared with Kim's and Lu's results (9 for Kim and 8 for Lu), our result is more tight.

If we extend the basic network latency of flow τ_4 to 5 and keep all other parameters unchanged, we find that flow τ_4 is unschedulable under Kim's analysis since the worst-case latency is 13 which exceeds its deadline 12, Figure 3.4 illustrate the timing diagram. Compared with Kim's analysis, the worst-case latency calculated by our approach is as follows:

$$R_4 = C_4 + \lceil \frac{R_4 + J_3^I}{T_3} \rceil C_3$$

$$R_4^0 = 5$$

$$R_4^1 = 5 + \lceil \frac{5+3}{9} \rceil 2 = 7$$

$$R_4^2 = 5 + \lceil \frac{7+3}{9} \rceil 2 = 9$$

$$R_4^2 = 5 + \lceil \frac{9+3}{9} \rceil 2 = 9$$

The iteration stops at 9 which means τ_4 is schedulable in fact.

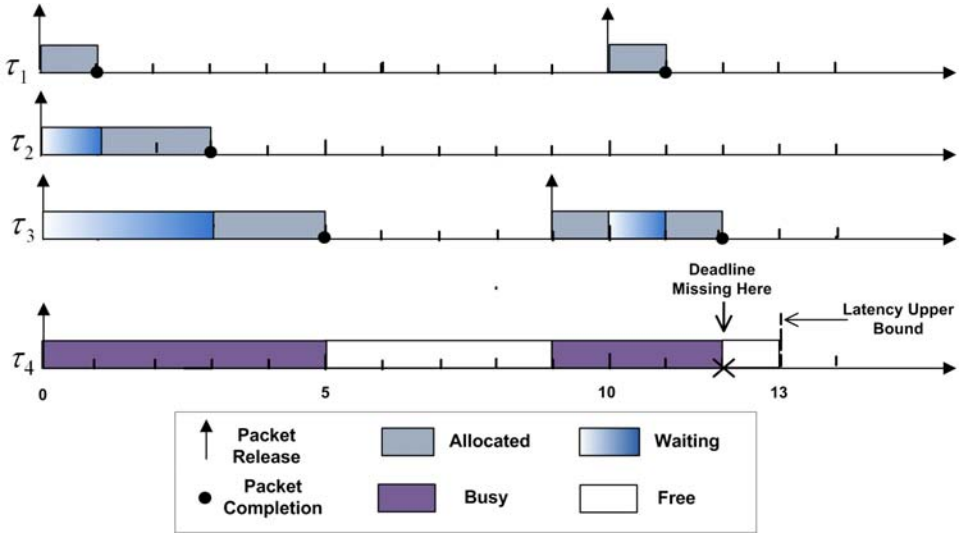


Figure 3.4: Unschedulable under Kim's Approach

The major different between our approach and Kim's and Lu's analysis is how to handle the indirect interference from higher priority flow. Ideally, at any time only the highest priority entity can be processed in a fixed priority preemption system under a single resource assumption. And any higher priority task execution will delay the completion time of any low priority task. So in Kim's and Lu's approaches, the analysis supposes a priority consistency between the traffic-flows. Priority consistency means a low priority flow cannot get service when a higher priority flow is still waiting. However, wormhole switching is different from classic real-time system where the link resources are distributed and the network supports multiple concurrent transmission. For example, in Kim's analysis in Figure 3.4, τ_4 starts transmission only after τ_3 completes since τ_4 has physical links in common with τ_3. Following the same fashion, τ_3 starts only after τ_1 and τ_2 complete. However, we note that τ_4 never shares any link with τ_1 and τ_2, so when τ_1 and τ_2 transmit data packets, they only can block τ_3 but cannot block τ_4. In this

situation, τ_4 may transmit while τ_3 is blocked. The time slots [0, 3] and [10, 11] can be used for τ_4's transmission in this case. Actually the priority consistency is not valid when indirect higher priority traffic-flows exist, where a higher priority flow blocks a middle priority flow which results in a reduced network latency to the lower priority flow. Kim's and Lu's approaches fail to address this situation and thus introduce excessive interference from indirect higher priority flows. In Section 3.4, by evaluation comparison, we investigate the performance of both worst-case network latency analysis proposed in this chapter and Kim's analysis with randomly generated flows set.

3.2 Extending the Analysis with Self-Blocking

The analysis in the last section relies on finding the worst-case amount of interference from higher priority traffic-flows which can preempt the observed one. However, this approach is valid only when network latency is less than period. In a situation, with a long transmission journey or heavy network congestion, the network latency for a packet can be quite large. It is possible that while the end flits from a previous packet are being delivered, the start flits of the next packet are already introduced. Since the data packets within the same virtual channel are served in first-in-first-out (FIFO) order, when a packet has to wait for the transmission of previous packets from the same traffic-flow, blocking occurs – we refer to this as *self blocking*. In this situation, the possible latency interference suffered by the newly arrived packet comes from not only higher priority traffic-flows but also by the previous packet instance. None of the current analysis techniques [8, 44, 53] can work in this case because all these approaches only consider the single packet instance situation, with the explicit constraint $D \leq T - J^R$, and treat this multiple packet instance as the evidence of a missing deadline. However, it can be very well

happen that the deadline is longer than the period in the case of NoC communication. This constraint of $D \leq T - J^R$ brings inflexibility to network exploration and design. Here we lift this constraint by permitting the packet deadline to be more than its period and extend our analysis model to cover this multiple instances case. Again we draw on analysis derived for processor scheduling [77].

3.2.1 Analysis with Self Blocking

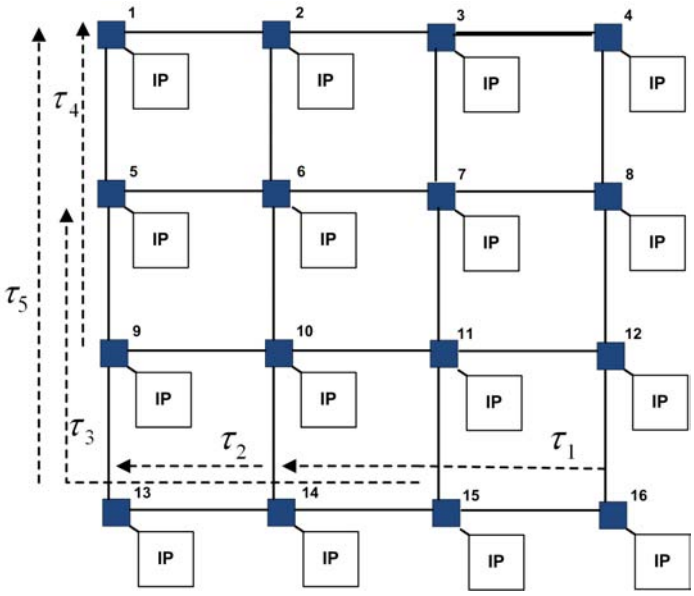


Figure 3.5: A Modified Case of Traffic-Flows Set

The concept of a level-i *busy period*, firstly introduced by Lehoczhy [48] in processor scheduling, indicates a time interval within which only tasks of priority i or higher are processed. A modified version of the priority level-i busy period, used in analysis of network latency with multiple instances in on-chip networks, is defined as a contiguous time interval during which a set of link resources are keep busy and serve all the traffic-flows of priority higher than or equal to priority i. We modify our previous case in Figure 2.8 and add a new flow τ_5 with the lowest priority. The relation between τ_5 and other existing flows is illustrated in

Figure 3.5. In this new case, τ_5 directly contends with τ_3 and τ_4 and indirectly suffers interference jitter from τ_1 and τ_2. For the link resources between router 1 and 13 (τ_5's path), the priority level-5 busy period is the contiguous time duration which the corresponding link resource keeps serving all the queueing packets with priority 5 or higher (3 or 4 in this case). Figure 3.6 illustrates this level-5 busy period.

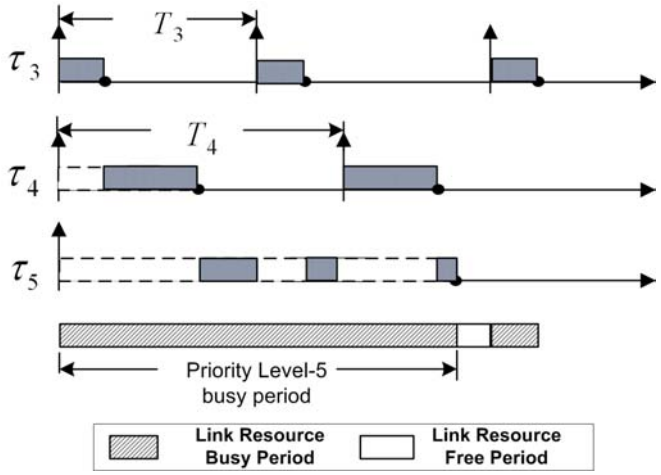


Figure 3.6: Priority Level-5 Busy Period

The current analysis techniques proposed in Section 3.1 has the constraint $D_i \leq T_i - J_i^R$. It implies that if τ_i is schedulable then the current priority level-i busy period will end at or before the time instant of the next packet arrival from τ_i. However if we allow $D_i > T_i - J_i^R$, a current packet can be waiting transmission without missing its deadline when a successive packet from τ_i arrives, and thus the busy period can be extended beyond T_i. More than one packet instance from τ_i is served during a priority level-i busy period. In this case, the packet network latency would include extra blocking from the previous packet arrival. So, the analysis in this section is concerned with analyzing the blocking from the analyzed traffic-flow itself as well as interference from other traffic-flows in the network.

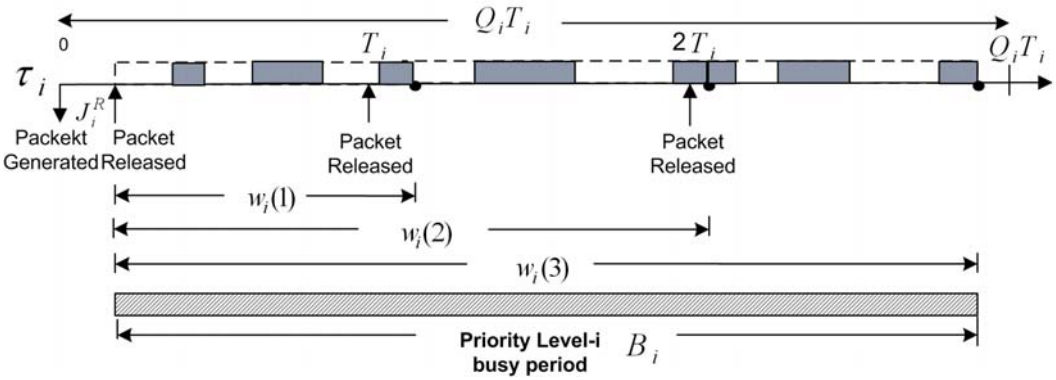


Figure 3.7: Priority Level-i Busy Period with Multiple Packet Arrivals

Figure 3.7 shows multiple packets arriving during a priority level-i busy period B_i. Consider the existence of release jitter, the packet release could start J_i^R later. The busy period illustrated in this figure begins when the first packet from τ_i or higher is released and ends when the network is idle or a lower priority packet less than i is able to get a chance for transmission.

Theorem 4. *The width of priority level-i busy period B_i can be found by:*

$$B_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{B_i + J_j^R + J_j^I}{T_j} \rceil C_j + \lceil \frac{B_i + J_i^R}{T_i} \rceil C_i \tag{3.20}$$

Proof. According the definition of busy period, all the arrival packets of priority i or higher before the end of the busy period must be transmitted during the busy period. So the width of the busy period is equal to the service requirement made by traffic-flow τ_i in addition to all the higher priority packet arrivals in the busy period.

Supposing that we can find an upper bound for the total busy period B_i, the maximum number of instances Q_i of a flow τ_i that falls into this busy period is calculated by:

$$Q_i = \lceil \frac{B_i + J_i^R}{T_i} \rceil \tag{3.21}$$

if the packet release, from τ_i, is simultaneous with the start of the busy period. Consequently, the maximum service requirement from τ_i is thus:

$$\lceil \frac{B_i + J_i^R}{T_i} \rceil C_i \tag{3.22}$$

On the other hand, the maximum service time requirement generated by all the higher priority traffic-flows during the busy period should be no more than:

$$\sum_{\tau_j \in S_i^D} \lceil \frac{J_j^R + J_j^I + B_i}{T_j} \rceil C_j \tag{3.23}$$

As a resolution, the width of the priority level-i busy period B_i can be found by:

$$B_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{B_i + J_j^R + J_j^I}{T_j} \rceil C_j + \lceil \frac{B_i + J_i^R}{T_i} \rceil C_i \tag{3.24}$$

□

The result of B_i is obtained by the iteration technique, where iteration starts with $B_i^0 = C_i$ and finishes when $B_i^n = B_i^{n+1}$.

We note that if $B_i \leq T_i - J_i^R$, then the busy period ends at or before the second packet from τ_i is generated. This means that only one packet instance of τ_i is served during the busy period. Eq.(3.20) degrades to Eq.(3.15). If $B_i > T_i - J_i^R$, then more than one packet instance of τ_i is generated during a priority level-i busy period. Some successive generated packets might be blocked by the previous packets. In this situation, to determine the worst-case network latency, we need to check the network latency for each generated packet during the busy period. The maximum of these values gives the worst-case network latency.

Theorem 5. *Having the traffic-flow with $D_i > T_i - J_i^R$, the maximum network*

latency for τ_i is given by:

$$R_i = \max_{q=1,\ldots,Q_i} (w_i(q) - (q-1)T_i + J_i^R) \tag{3.25}$$

where q is the index of packet instance, and $w_i(q)$ is given by:

$$w_i(q) = qC_i + \sum_{\forall \tau_j \in S_i^D} \lceil \frac{w_i(q) + J_j^R + J_j^I}{T_j} \rceil C_j \tag{3.26}$$

Proof. Suppose we have calculated the priority level-i busy period and $B_i > T_i - J_i^R$, so more than one packet instance falls in the busy period. We need check each release to find the maximum latency.

We use the index variable q to denote a packet instance of τ_i. The first packet in the busy period corresponds to $q = 1$ and the final one is $q = Q_i$. We assume a time window $w_i(q)$ which denotes the time interval from the beginning of the busy period until the completion of the q^{th} packet transmission. The time windows for 1^{st}, 2^{nd} and 3^{rd} packets in the busy period are showed in Figure 3.7 as $w_i(1)$, $w_i(2)$ and $w_i(3)$ respectively. The time window $w_i(q)$ is given by:

$$w_i(q) = qC_i + \sum_{\forall \tau_j \in S_i^D} \lceil \frac{w_i(q) + J_j^R + J_j^I}{T_j} \rceil C_j \tag{3.27}$$

The variable qC_i accounts for the transmission service time of the first q packets instances of τ_i during the busy period and $\sum_{\forall \tau_j \in S_i^D} \lceil \frac{w_i(q)+J_j^R+J_j^I}{T_j} \rceil C_j$ denotes the service requirements from all the higher priority flows which fall in this time window $w_i(q)$. The value of $w_i(q)$ can be found by iteration starting with $w_i^0(q)$ = $C_i + qT_i$ and ending when $w_i^{n+1}(q)$ = $w_i^n(q)$. The generated time of q^{th} packet happens at instant $(q-1)T_i$ relative to the start of busy period so the network

latency of the q^{th} instance is given by:

$$R_i(q) = w_i(q) - (q-1)T_i + J_i^R \tag{3.28}$$

As mentioned above, the worst-case network latency can occur at any one of these packet releases during the busy period, and thus the worst-case network latency is given by:

$$R_i = \max_{q=1,\ldots,Q_i} (w_i(q) - (q-1)T_i + J_i^R) \tag{3.29}$$

□

3.2.2 Another Example

The case showed in Section 3.1.3 only considers the single packet instance situation. This could induce an optimistic result when self blocking occurs. We introduce a new case with the additional flow τ_5 in Figure 3.5 (page 85). The attributes of τ_5 are given by Table 3.1. It is easy to see that the worst case latency of τ_1, τ_2, τ_3 and τ_4 are unchanged (they have higher priority).

Real-Time Traffic-flow	P	C	T	D	J^R
τ_5	5	3	8	12	0

Table 3.1: Traffic-flow Description

For τ_5, because of the situation of $D_5 > T_5 - J_5^R$, we need to consider the possible affect of self blocking. Utilizing the busy period analysis, first we calculate the priority level-5 busy period B_5. Eq.(3.20) becomes:

$$B_5 = \lceil \frac{B_5 + J_3^I}{T_3} \rceil C_3 + \lceil \frac{B_5}{T_4} \rceil C_4 + \lceil \frac{B_5}{T_5} \rceil C_5$$

$$B_5^0 = 3$$

$$B_5^1 = \lceil \frac{3+3}{9} \rceil 2 + \lceil \frac{3}{12} \rceil 4 + \lceil \frac{3}{8} \rceil 3 = 9$$
$$B_5^2 = \lceil \frac{9+3}{9} \rceil 2 + \lceil \frac{9}{12} \rceil 4 + \lceil \frac{9}{8} \rceil 3 = 14$$
$$B_5^3 = \lceil \frac{14+3}{9} \rceil 2 + \lceil \frac{14}{12} \rceil 4 + \lceil \frac{14}{8} \rceil 3 = 18$$
$$B_5^4 = \lceil \frac{18+3}{9} \rceil 2 + \lceil \frac{18}{12} \rceil 4 + \lceil \frac{18}{8} \rceil 3 = 23$$
$$B_5^5 = \lceil \frac{23+3}{9} \rceil 2 + \lceil \frac{23}{12} \rceil 4 + \lceil \frac{23}{8} \rceil 3 = 23$$

The busy period calculation iteration stops at $B_5 = 23$. Obviously $B_5 > T_5$; more than one packet instance is generated during the busy period. Q_5 is the total amount of packets in the busy period:

$$Q_5 = \lceil \frac{B_5}{T_5} \rceil = 3$$

Next, we must check the network latency of these three packets separately to determine the worst-case latency. The time windows for the first packet, the first two packets and the first three packets are $w_5(1)$, $w_5(2)$ and $w_5(3)$ respectively. Using Eq.(5.22), we get:

$w_5(1) = 10$ and $R_5(1) = 10 - 0 = 10$

$w_5(2) = 20$ and $R_5(2) = 20 - 8 = 12$

$w_5(3) = 23$ and $R_5(3) = 23 - 16 = 7$

The worst-case network latency is thus $max(R_5(1), R_5(2), R_5(3)) = 12$; the second packet instance experiences the maximum latency. If we use the analysis approach proposed in Section 3.1 which does not considers self-blocking problem, only first packet release is analyzed. The maximum latency for τ_4 is 10 which is overoptimistic in fact.

3.3 Parallel Communication and Tightness Analysis

By abstracting the communication resources and finding the inter-relations among the traffic-flows, we have succeeded in transforming real-time wormhole communication into an analyzable model. In this section, we show that our analysis approach can produce an upper bound for network latency in all situations. We also show that the exact worst-case network latency evaluation in the parallel communication situation, as a general problem, is NP-hard. During the discuss of parallel communication, we address two defects in the previous published analysis approaches.

3.3.1 Parallel Communication

Parallel communication is the phenomenon that when two or more higher priority flows do not share any link resource but only contend with lower priority flows. The three flows, τ_1, τ_2 and τ_3, in Figure 3.5 (page 85), are an example of parallel communication where τ_1 and τ_2 are independent of each other and only share a link with τ_3. We keep the attributes of τ_1 and τ_2 unchanged and modify τ_3 with new attributes in Table 3.2. According to the critical instant assumption, flow τ_3 experiences its maximum network latency when released simultaneously with τ_1 and τ_2. The worst-case network latencies for τ_1, τ_2 and τ_3 are 1, 2, 6.5 according to our analysis model in Section 3.1. Flow τ_3 misses its deadline in our analysis. Figure 3.8(A) shows the timing diagram with theoretical calculation for τ_3. But we find that during the analysis of τ_3, τ_2 is forced to compete with τ_1 in this model even though they do not share any physical link; and this will force an extra delay on τ_3. The problem is that when we schedule all the traffic-flows with the resource competing relationship, we always ideally assume at any time instant only one traffic-flow can be served by the subset of the network which hosts the

interfering flows.

Real-Time Traffic-flow	P	C	T	D	J^R
τ_1	1	1	5	5	0
τ_2	2	2	7	7	0
τ_3	3	2.5	6	6	0

Table 3.2: Traffic-flows Description

Lu *et al* in his paper [53] found this phenomenon that real-time transmission scheduling can be parallel for disjoint concurrent contentions (see Section 2.3.7). Figure 3.8(B) illustrates this concurrency. If all traffic-flows release the packets simultaneously, τ_1 and τ_2 are executed at the same time in this scheduling sequence. This parallel communication reflects the fact that no real link resource is shared between the direct higher priority traffic-flows of the observed one. For τ_i, this parallel communication phenomenon exists when the following condition is met: $\tau_j, \tau_k \in S_i^D \; and \; \Re_j \cap \Re_k = \emptyset$.

A Defect in Lu Analysis

Lu *et al* [53] proposed the contention tree model to analyze the network latency when parallel communication exist. The worst-case network latency is assumed to occur when all the traffic-flows are released simultaneously and the disjoint traffic-flows are scheduled in parallel. The corresponding network latency for τ_1, τ_2 and τ_3 are 1, 2, 4.5 in Lu's model which are smaller than our results. However we must point out the analysis by Lu *et al* [53] is defective. Even though this parallel communication can potentially reduce the intervention to the lower priority traffic-flow, since they may overlap along the time axis, the worst-case latency does not always occur when all the traffic-flows are released at the same time. Figure 3.8(C) is a contradictory case against this assumption. The network releases τ_1 and τ_3 at

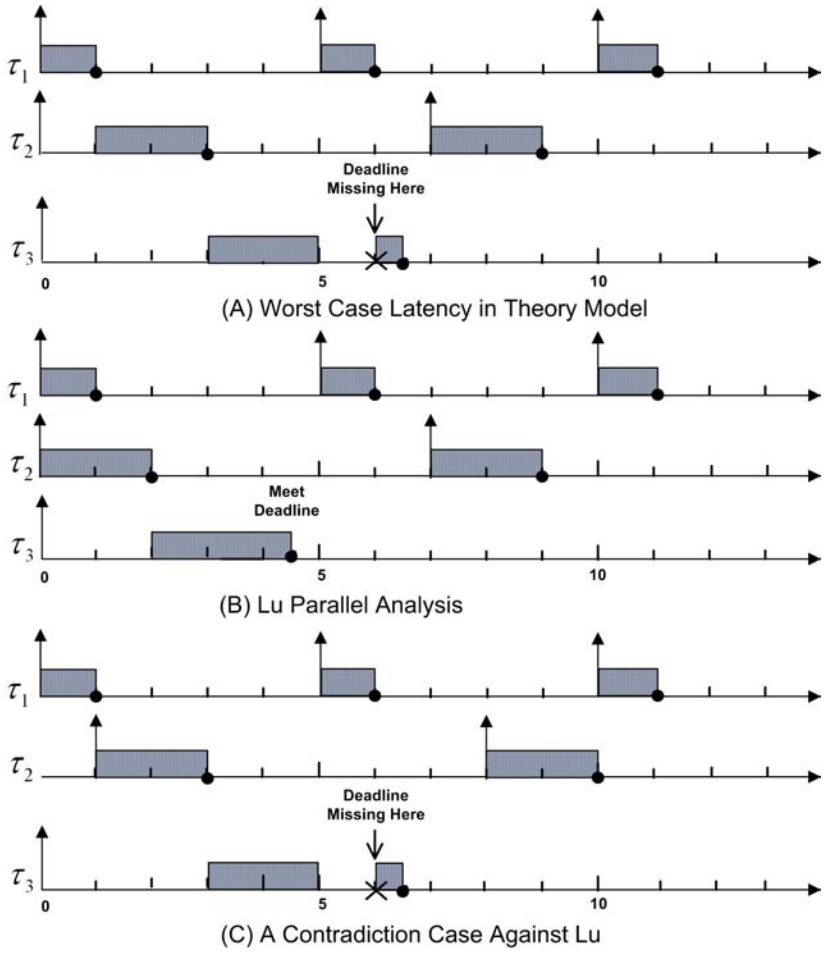


Figure 3.8: Scheduling Sequence With Parallel Communication

the same time, τ_2 is ready just before τ_1 accomplishes its transmission. Flow τ_2 consequently holds the link resource until service completion. After that time, no more new packet instances from τ_1 and τ_2 arrive, and the links are available for τ_3. In this situation, the lower priority traffic-flow τ_3 suffers more interference. The network latency of τ_3 is 6.5 more than the upper bound value produced in Lu's analysis and τ_3 is unschedulable in fact.

A Defect in Mutka Analysis

We have showed a defect in Lu's analysis when parallel communication exists. We will next show that the analysis by Mutka described in Section 2.3.4 also cannot

be used to guarantee packet deadlines.

Mutka [60] utilized the RM scheduling utilization inequality to determine the schedulability of traffic-flow. The test is executed link by link to calculate the schedulable utilization upper bound. In our case, Figure 3.5, the link between router 14 and 15 and the link between router 13 and 14 both only get two active flows. So the real utilization for link between router 14 and 15 is:

$$\frac{1}{5} + \frac{2.5}{6} \approx 0.6167$$

The real utilization for link between router 13 and 14 is:

$$\frac{2}{7} + \frac{2.5}{6} \approx 0.7024$$

The RM utilization bound for each link is given by Eq.(2.11):

$$2(2^{1/2} - 1) \approx 0.828$$

Thus, flow τ_3 is determined to be schedulable according to the RM test. However, as showed in Figure 3.8(C), flow τ_3 actually misses its deadline. The RM test is insufficient since it treats each physical link as an independent scheduling unit. Obviously, this is not the case of wormhole switching where one packet can span multiple routers and the packet with highest priority at a router may not be sent since it can be blocked at some other link.

3.3.2 Tightness Analysis of Our Model

The parallel communication phenomenon does not appear in all the traffic-flow sets loaded on the network. However, this possible parallelism clearly complicates the analysis progress. In general scheduling, when the worst-case release conditions are not easily determined, the analysis is usually intractable. Let us re-visit our analysis model, for τ_i, the total interferences generated by all direct higher priority traffic-flows are the sequence summation of each traffic-flow,

I_i=$\sum_{j\in S_i^D}\lceil\frac{R_i+J_j^R}{T_j}\rceil$. This assumption can produce a tight result for each traffic-flow when there is no parallel communication between direct higher priority traffic-flows. However, when parallel communication exists, the following theorem shows that the approach is not always tight but is only an upper bound for network latency.

Theorem 6. *The real worst-case network latency is no more than the calculation result accomplished by Theorems 1, 3 and 5 when parallel communication exists.*

Proof. Similar to the scheduling in the single processor model, our model also implies that at any time only one traffic-flow can win the access right of the shared resource and execute the service. Let t_1, t_2 indicates time instants after network startup. During any time interval [t_1, t_2] where $t_1 < t_2$, the maximum required service time of all the higher priority traffic-flows (namely, interference to the observed traffic) is no more than $\sum_{\forall j\in S_i^D}\lceil\frac{t_2-t_1+J_j^R+J_j^I}{T_j}\rceil C_j$. Since parallel services exist, more service opportunity can be supported by the network in any time interval. This may accelerate consumption of the required service from higher priority traffic-flows and finally shortens the whole interference imposing on the observed one. Consequently, the worst-case network latency in the real network is no more than the calculation result under the assumption of no parallel communication. □

Theorem 6 implies the model and analysis proposed in this thesis is only sufficient but not necessary when parallel communication exists. In another words, if a traffic-flow can pass this schedulability test then it will meet its deadline. But if it fails this test, the inverse conclusion cannot be made. Next, we give a proof that it is not possible to produce a polynomial-time necessary and sufficient test unless P=NP.

Suppose the parallel communication instance in Figure 3.9, let set Γ include $N+1$ traffic-flows distributing across the network with attributes τ_i=(P_i, C_i, T_i, D_i, J_i^R, J_i^I, s_i); s_i is the release time of τ_i. The first N traffic-flows are disjoint

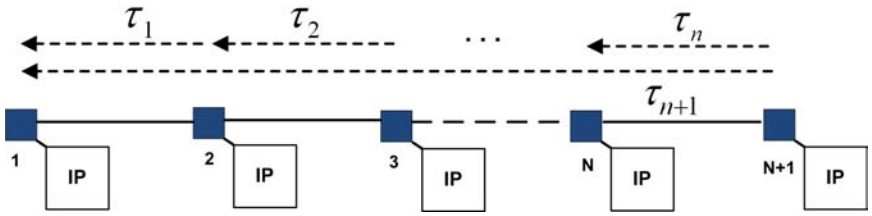


Figure 3.9: A General Parallel Communication Model

with each other and communication in parallel. The observed one τ_{n+1} with lower priority contends with the first N higher priority traffic-flows, so S^D_{n+1} = $\{\tau_1, \tau_2, \ldots, \tau_n\}$. In order to determine the network latency, we need to take account of all the possible free gap intervals for the observed traffic-flow. The case in Figure 3.8(C) implies the worst-case network latency no longer occurs when all the traffic-flows are released simultaneously. Thus, we need to examine all the possible packet release sequences to achieve the worst-case network latency calculation. We now show that to solve this problem when parallel communication exists is NP-hard.

Lemma 1. *For traffic-flows set* Γ *meeting the conditions showed in Figure 3.9, the observed traffic-flow* τ_{n+1} *is schedulable on the network if and only if all the deadlines are met during time interval* $[0, t_1]$*, where* $t_1 = s+T$*,* $s = max\{s_1, \ldots, s_{n+1}\}$ *and T=lcm*[1]$\{T_1, \ldots, T_{n+1}\}$.

Proof. $\Rightarrow$: If the observed traffic-flow is schedulable on the network, relying on the schedulable condition, all the deadlines are met since it was released. Thus, all deadlines in the interval [0, t_1] are met.

$\Leftarrow$: At any time instant t_i, let the network state be denoted by $E(e_1, \ldots, e_n)_{t_i}$, where e_i is the amount of time for τ_i to finish its transmission service since its last release. The time interval [0, $s + T$] is separated into two parts [0, s) and [s, s+T]. After s, all the traffic-flows are released and executed in parallel except τ_{n+1}. Let

[1]least common multiple

$t_i \in [s, s+T]$ and $t_j = t_i + j \times T$ for j is positive integer and $j > 0$, it is not difficult to see that the network state $E_{t_i} = E_{t_j}$ is always true. This means the schedule progress repeats itself every P units of time after s. Since all deadline of τ_{n+1} in [s, s+T] are met, all the deadline after $s + T$ should also be met. As the assumption, the deadlines of τ_{n+1} in [0, s] are met. Thus τ_{n+1} is schedulable in this network. □

Strictly speaking, we need to check all the infinite scheduling sequences after packet releases of the traffic-flow set to ensure the deadline is always met. Lemma 1 gives us an easy method to estimate whether the traffic-flow set is schedulable in a finite schedule time interval $[0, s + T]$. If all the deadlines in $[0, s + T]$ are met, then this traffic-flow set is a valid schedule. If we know the period, maximum basic network latency and release time of all packets in a traffic-flow set in advance, we can find all the available gap intervals in [0, s+T] for observed traffic-flow with a polynomial time algorithm. Determining the schedulability of the observed traffic-flow, namely finding the worst-case network latency when knowing all the available gap intervals in advance is also taking polynomial time. Now we prove that if the traffic-flow set with an arbitrary release time, finding the worst-case network latency is intractable. Utilizing K Simultaneous Congruences [49] which has been showed to be NP-complete, our problem can be proved to be intractable by reducing the current problem to a known NP-complete problem in polynomial time.

K Simultaneous Congruences: Given N ordered pairs of positive integers (a_1,b_1), ..., (a_n, b_n) and a positive integer K ($2 \leq K \leq N$). Is there a subset of $\ell \geq K$ ordered pairs $(a_{i,1}, b_{i,1}), \ldots, (a_{i,\ell}, b_{i,\ell})$ such that there is a positive integer x with the property that $x = a_{i,j} + p_j \times b_{i,j}$ for each $1 \leq j \leq \ell$ (p_j is nonnegative integer)?

Theorem 7. *For a traffic-flow set with arbitrary release time, the problem of determining schedulability when parallel interference exists is NP-hard.*

Proof. The proof includes two steps, we first prove that finding all the gap intervals for the observed traffic-flow is NP-hard. Suppose n sorted pairs of positive integers (a_1,b_1), ..., (a_n,b_n) with constraint $a_i \geq b_i - 1$ and a positive integer K, we construct n parallel traffic-flows communications $(\tau_1, \ldots, \tau_n)$ with attributes $T_i = D_i = b_i$, $C_i = b_i - 1$, and $s_i + C_i = a_i$. All the n traffic-flows have the higher priority levels and never share any physical link each other. In our construction, each traffic-flow in each period only exports one free time unit. For this traffic-flow set, finding the gap intervals on the network if and only if all the n traffic-flows output free time units simultaneously. Thus, the K Simultaneous Congruences problem has a solution if and only if the free time unit can be found in the constructed model. Now we assume an observed traffic-flow τ_{n+1} which has the lowest priority comparing with $\tau_1, \ldots, \tau_n$. Since this construction progress can be done in polynomial time, the problem of finding all the gap intervals is as hard as the K Simultaneous Congruences problem. Determining the schedulability of τ_{n+1} when knowing all the gap intervals is polynomial time complexity, so the problem of determining schedulability in the parallel interference situation is NP-hard. □

Theorem 7 shows that no tractable necessary and sufficient schedulable test exists for wormhole fixed priority scheduling. In this chapter, we have derived a sufficient evaluation, that is tight in many situations and is an improvement over previously published techniques. This statement is supplied by the evaluation comparisons presented in the next section.

3.4 Evaluation

In this section, we want to quantify the schedulability improvement that may be achieved by using the analysis proposed in this chapter against Kim's approach [44]. The evaluation is presented as experiments in two parts, the first part shows how each experiment is created, the second part shows the results of the experiments.

3.4.1 Experiment Setup

Considering the fact that a low dimensional mesh is quite common in current on-chip networks, we conduct our evaluation based on a 4×4 2D mesh. The dimension-order X-Y routing is used since it is simple and deadlock free. Although we focus on the architectures inter-connected by 2D mesh networks with X-Y routing schemes, our algorithm can be adapted to other network architectures with different topologies or different deterministic routing schemes. The evaluation software is developed with standard C++ and all the evaluations were taken under the Intel Core 2 Duo (3GHz) and 2G bytes memory.

The first step of the experiment is traffic-flow generation. Each traffic-flow is characterized by the transmission path and a set of properties, such as period T, deadline D, basic communication latency C and priority P. The random source/destination nodes pairs are chosen for each traffic-flow. Then the X-Y routing algorithm gives the deterministic routing from source to destination. The packet size $Packet\_Size$ is chosen from the range [16, 1024] with a uniform probability distribution function where the size is the number of flits of a packet. With the assumptions of one flit per cycle link transmission and one cycle per node delay, the basic network latency C could be easy to calculate by Eq.(2.10). Another important metric period T is calculated by a random utilization variable u_i. We em-

ploy the uniform distribution algorithm [13] to generate a set of random utilization variables. Utilizing this utilization variable, each flow's period is calculated as T_i = $Packet\_Size_i/u_i$. Note that the $D > T$ situation is not considered by the Kim's approach, so in these experiments only the evaluation under the $D \leq T$ constraint is investigated. The traffic-flow deadline D is set to be T for all the traffic-flows. For a set of generated traffic-flows, we ensure the flow set does not violate predefined constraints: maximum link utilization U_{max} or average link utilization U_{avg}. The link utilization U_{link_j} for a single link j can be found by summing the usages of all the m traffic-flows assigned to this link.

$$U_{link_j} = \sum_{i=1}^{m} Packet\_Size_i/T_i \tag{3.30}$$

The maximum link utilization is the maximum number of all the links in this network, $U_{max} = max(U_{link_j})$ where $\forall link_j \in mesh$. The average link utilization is given by:

$$U_{avg} = (\sum_{j=1}^{M} U_{link_j})/M \tag{3.31}$$

where M is the total number of links in the network.

The evaluation strategy is: first we randomly generate a traffic-flow set under the predefined network load constraints and prioritize each flow with a distinct priority level. The T/H (period/hops) monotonic priority assignment policy [2] [35] is chosen in this case, where a traffic-flow with the smallest calculation value of period/hops is given the highest priority. Then two different schedulability analysis tests are executed for this set to evaluate the performance. The comparison measure used in this experiment is *pass ratio*, namely, the percentage of the number of traffic-flow sets that pass the schedulability test versus the number of original traffic-flow sets. Only if all the traffic-flows in a generated set pass the schedula-

[2] see next Chapter for discussion of priority assignment algorithm

bility test, do we say that the set is schedulable. The experiment investigates the performance of two schedulability analysis approaches with varied link utilization. Both the maximum and average link load schemes are considered. For each test, we count the percentage of the number of schedulable sets out of 1000 ones that are randomly generated. In general, the more traffic-flow sets meet their timing constraints, the more effective is the analysis approach.

3.4.2 Results of Experiments

Figures 3.10, 3.12 and 3.14 show, for each maximum link utilization, the percentage of the schedulable sets achieved by our approach against Kim's approach. Figures 3.11, 3.13 and 3.15 do similar comparison with average link utilization. The generated test set reported in Figures 3.10 and 3.11 contains 30 flows each time; then this number increases to 60 and 90 in Figures 3.12 and 3.13, and Figures 3.14 and 3.15.

Figures 3.10, 3.11, 3.12, 3.13, 3.14 and 3.15 show that when the overall link utilization is very low, 0.1 for maximum and average utilization, both of the schedulability tests give the similar performance of near 100% pass ratio. While when the utilization is very high, i.e. more than 0.8 in maximum utilization or 0.4 in average utilization, although our approach is better than Kim's one, the pass ratio of both tests is very low. During the ranges of [0.1, 0.8] for maximum utilization and [0.1, 0.4] for average utilization, our approach significantly outperforms Kim's approach. This is because Kim's approach can not properly address the indirect higher priority traffic-flows and thus over-calculates the maximum latency. This overestimation causes a low pass ratio; only 47.6% of flow sets pass the schedulability test comparing with 97.8% with our approach when the network maximum link load reaches 0.4 in Figure 3.10. Similar result can also be found in Figure 3.11.

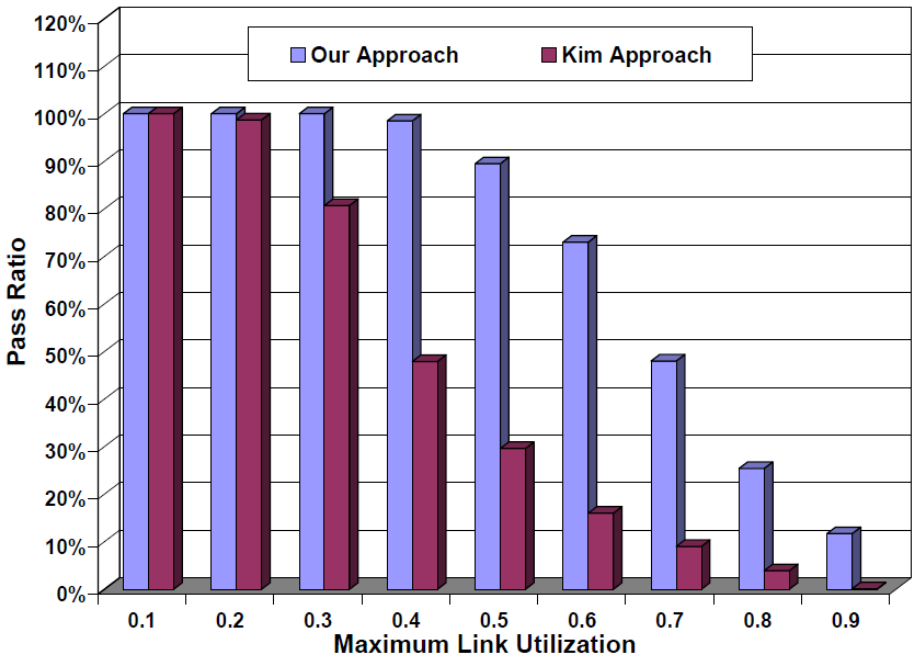


Figure 3.10: The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 30

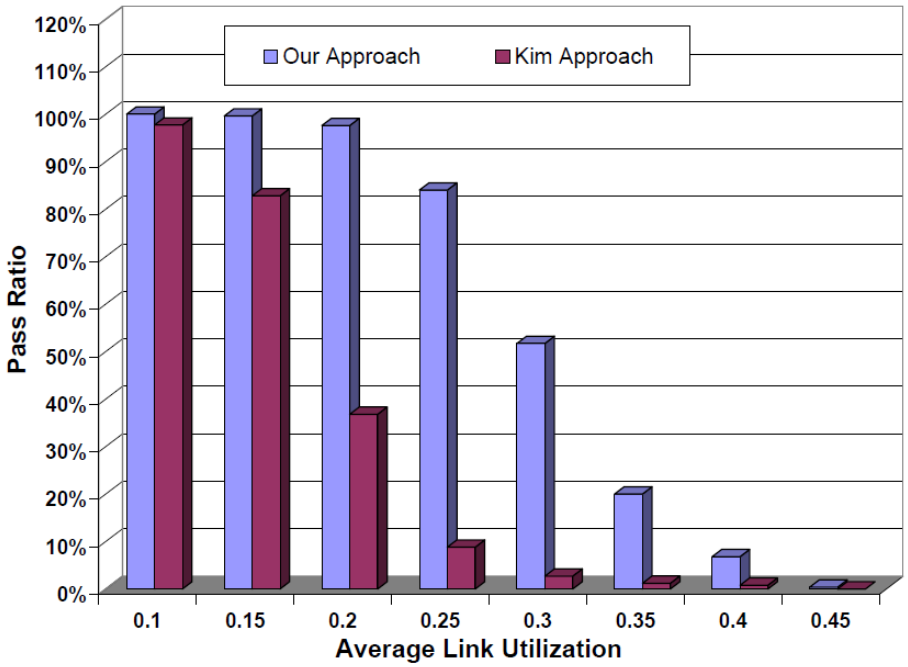


Figure 3.11: The Pass Ratio Under Varied Average Link Utilization With Flow Set 30

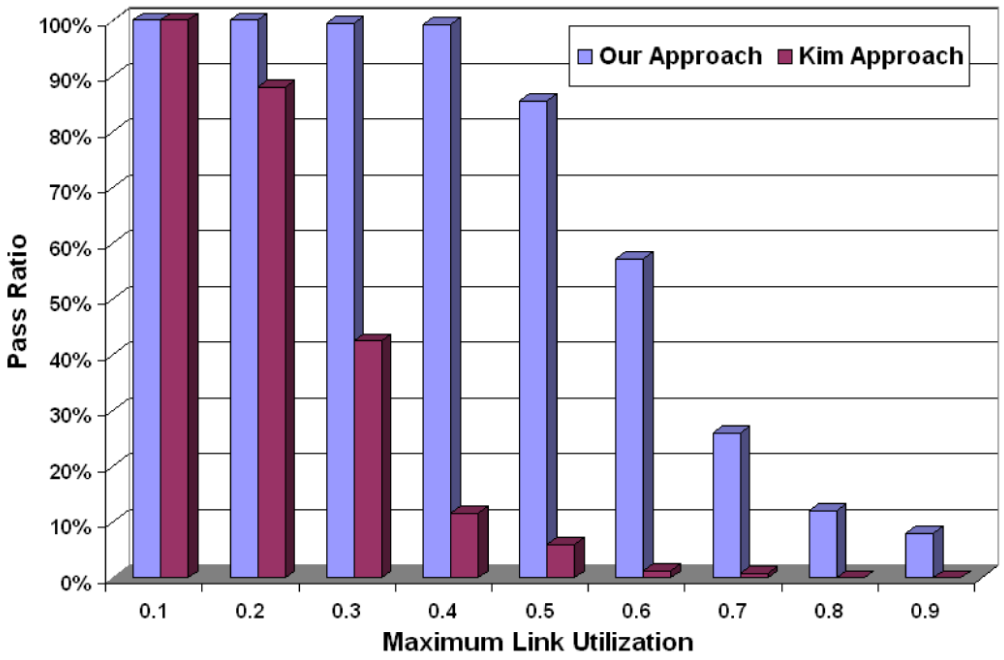


Figure 3.12: The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 60

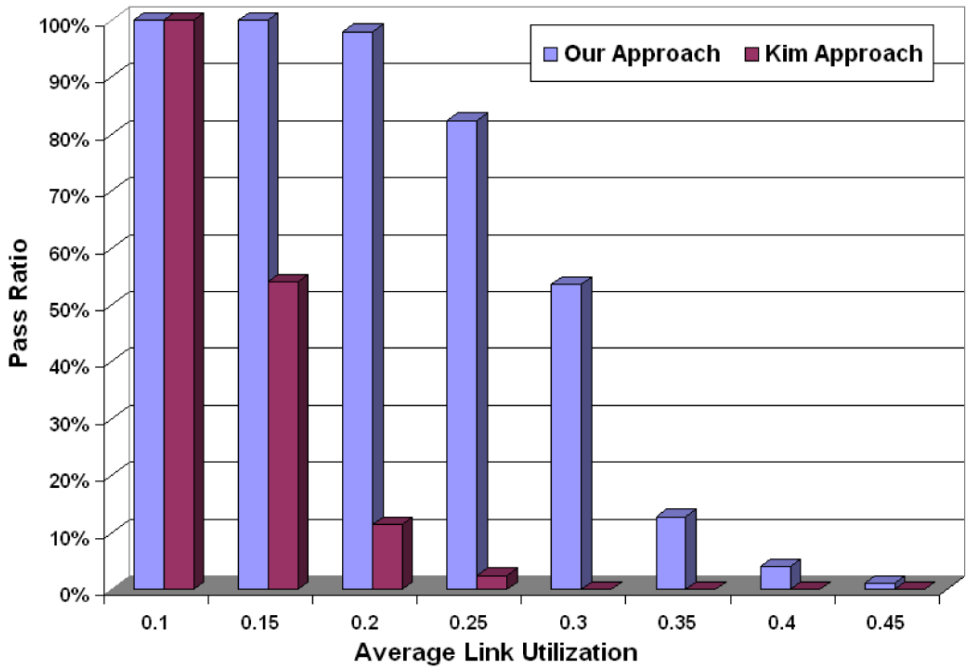


Figure 3.13: The Pass Ratio Under Varied Average Link Utilization With Flow Set 60

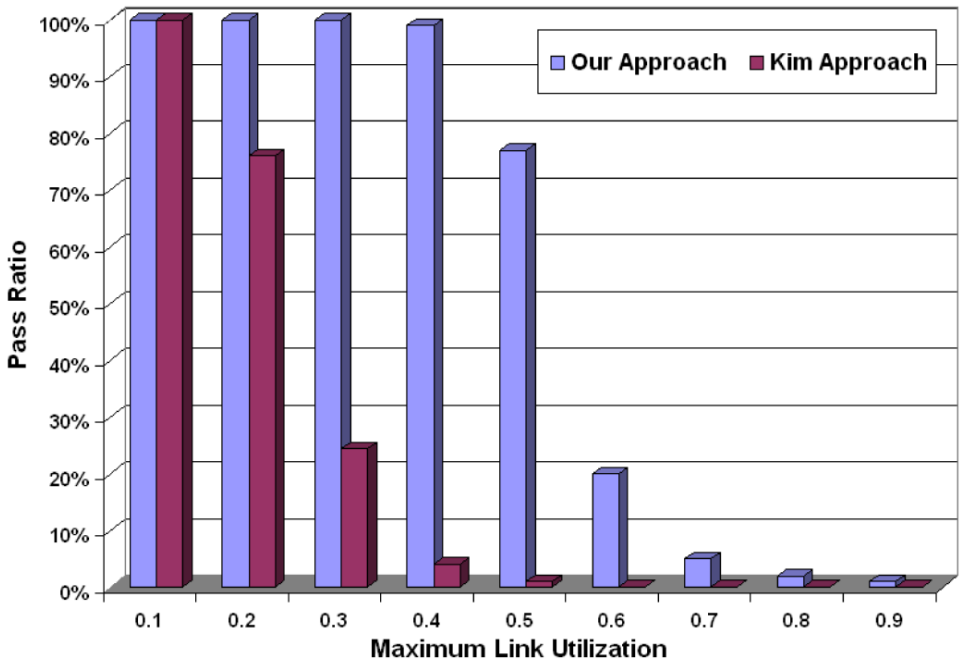


Figure 3.14: The Pass Ratio Under Varied Maximum Link Utilization With Flow Set 90

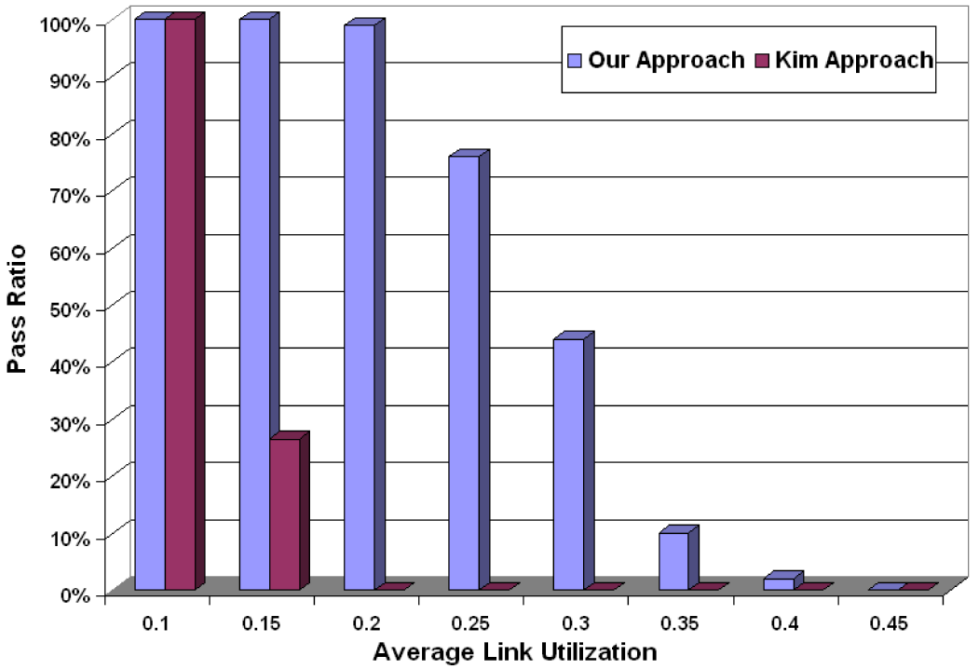


Figure 3.15: The Pass Ratio Under Varied Average Link Utilization With Flow Set 90

All graphs show that when maximum link utilization is no more than 40% or average link utilization is no more than 20%, the performance of our approach stays higher, more than 90% pass ratio for all evaluated flow sets. While by comparison, Kim's approach only achieve a 5% pass ratio in this situation, for example see Figure 3.14).

The comparisons between three selected flow sets: 30, 60 and 90 are showed in Figures 3.10, 3.12 and 3.14 for maximum link utilization and Figures 3.11, 3.13 and 3.15 for average link utilization. The pass ratio for both tests fall off with the increase of flow size. But it seems that Kim's approach is very sensitive to the size of flow set since the gap between the two approaches becomes progressively larger under the flow set with bigger size. For example, there is 51% better performance than Kim's test when the average utilization of network is 0.2 and flow set is 30 in Figure 3.11. While this standard of performance rises to 82% and 94% with the size of set reaches 60 (Figure 3.13) and 90 (Figure 3.15).

3.5 Summary

The new on-chip communication architectures need to provide different levels of service for various components on the same network. The requirement of real-time applications needs a scheduling strategy and analysis approach to predict whether all the real-time packets can meet their timing bounds. In this chapter, we introduced a novel analysis approach for real-time on-chip communication with wormhole switching and fixed priority arbitration. The major contributions of this chapter are: first, we derive worst-case network latency analysis based on three possible situations, direct interference from higher priority flows, indirect interference from higher priority flows and self-blocking interference. Secondly, we observe that when parallel communication exists in a real network, the general

problem of determining the exact schedulability of real-time traffic-flows over the on-chip network is NP-hard. However the results presented in this chapter do form an effective upper bound. In addition, the defects in two previous published works are illustrated. Evaluations are also included. A clear improvement is demonstrated over the best alternative (not optimistic) approach.

Chapter 4

Priority Assignment for Priority Based Wormhole Switching

As illustrated in the previous chapter, with the support of fixed priority preemption in on-chip networks, for a set of traffic-flows with a known priority ordering, the schedulability analysis algorithm can find the network latency upper bound for each flow. So hard deadlines can be guaranteed by this approach. But one of the common problems that need to be solved in the development of priority-based communication is, how to assign priority to each traffic-flow so that all the traffic-flows loaded on the network will meet their time constraints. The main purpose of this chapter is to explore priority assignment policies in real-time on-chip networks. We present at the beginning a case to show that none of the current priority assignment policies work well in wormhole switching networks. Then a heuristic branch and bound search algorithm is introduced to find the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space on average.

This chapter is organized as follows: Section 4.1 reviews the current priority assignment policies. Sections 4.2 and 4.3 describe several important properties of

priority-based wormhole switching. Based on these properties, in Section 4.4, a branch and bound search algorithm for priority assignment is developed. In order to improve the search efficiency, we design six heuristic functions which try to guide the search to the right branch and produce a schedulable priority assignment with the shortest time. In Section 4.5, the evaluation results of our algorithms against several current priority assignment policies are presented. Finally, Section 4.6 gives a summary of this chapter.

4.1 Priority Assignment and Motivation Case

Priority assignment policies for fixed priority scheduling on single-processor system have been studied widely. The early work, proposed by Liu and Layland [51], introduced the *Rate Monotonic (RM)* priority assignment policy, such that the task with shortest period is given the highest priority. However, RM policy requires a restrictive system model, deadline and period must be identical for each task in the system. Later, Lenug and Whitehead [49] lifted this constraint and defined the *Deadline Monotonic (DM)* priority assignment policy, allowing a task's deadline to be less than the period; where the shorter deadline the task has, the higher priority value is assigned. Audsley in [5, 6] designed a general assignment technique with polynomial time complexity that is guaranteed to find a schedulable priority ordering if one exists in a system. The basis of this algorithm is that the response time of each task is only determined by its higher priority tasks but not the relative priority ordering of those tasks. Recently, Zuhily and Burns [82] introduced the *Deadline minus Jitter Monotonic (D-JM)* priority ordering for a task set with non-zero release jitter.

Priority assignment in wormhole switching networks was firstly addressed by Mutka [60], which is based on the RM approach. Li and Mutka [50] later pre-

sented a *Laxity Monotonic (LM)* priority assignment policy and a related run-time adjustment method. The laxity is defined as the difference between the deadline and maximum basic network latency, which corresponds to the amount of delay a packet can tolerate. Priorities are assigned based on laxity with the highest priority going to the traffic-flow with the smallest laxity. Hary [35] improved the original LM and RM algorithms by accounting for the numbers of hops a traffic-flow travels and proposed a set of priority assignment approaches: Period/Hops RM(T/H), Laxity/Hops LM(L/H) and two modified versions: Period/ln(Hops+e-1) and Laxity/ln(Hops+e-1). Here the priority of a traffic-flow which travels a long distance is increased.

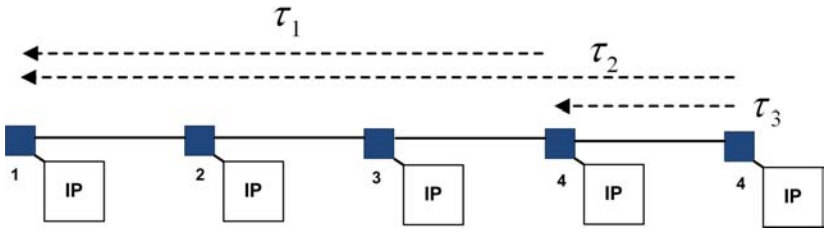


Figure 4.1: A Case of Real-Time Traffic-Flows

However, none of these priority assignment polices is referred to as *optimal*. An optimal priority assignment approach means it can provide a schedulable priority ordering whenever such an ordering exists. The following case shows that none of RM, DM, LM, RM(T/H) or LM(L/H) can produce a schedulable priority ordering corresponding to a set of traffic-flow with relationships illustrated in Figure 4.1 and attributes in Table 4.1.

Real-Time Traffic-Flow	C	T	D
τ_1	2	5	5
τ_2	3	7	7
τ_3	4	9	9

Table 4.1: Traffic-Flows Description

Real-Time Traffic-Flow	Priority Ordering				
	RM	DM	LM	RM(T/H)	LM(T/H)
τ_1	1	1	1	1	1
τ_2	2	2	2	2	2
τ_3	3	3	3	3	3

Table 4.2: Priority Ordering with Different Policies

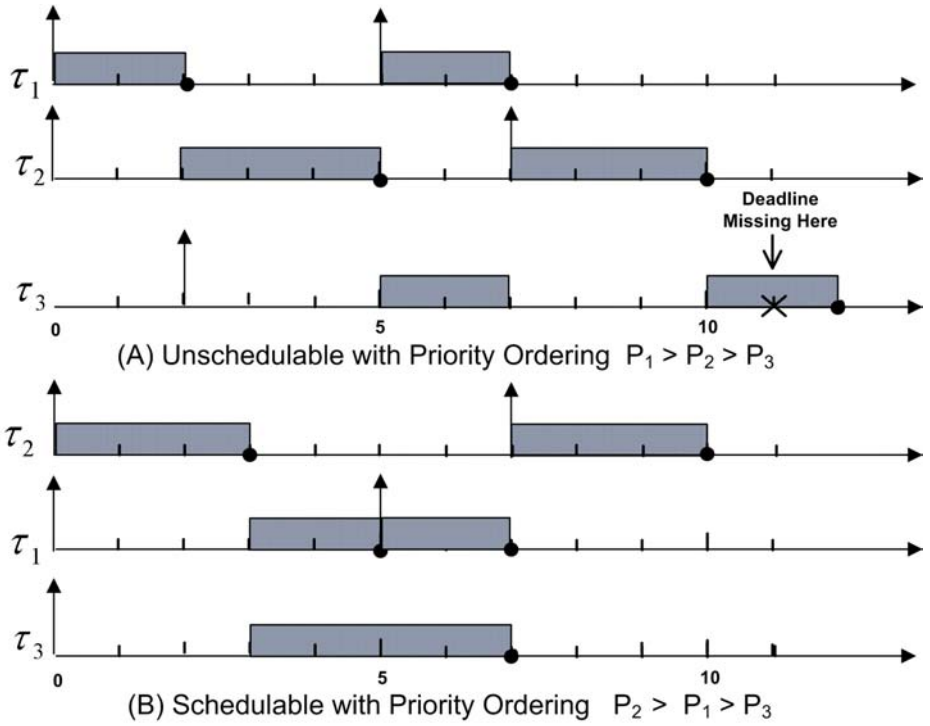


Figure 4.2: Timing Graph with Different Priority Ordering

The priority orderings produced by RM, DM, LM, RM(T/H) or LM(L/H) policies are showed in Table 4.2. We note that all the priority ordering are the exactly same in this case; τ_1 gets the highest priority, then τ_2; τ_3 is assigned the lowest priority. The analysis method given in the last chapter calculates the worst-case network latency of τ_1, τ_2 and τ_3 as 2, 5, 10 respectively. The worst-case network latency for τ_3 occurs when τ_3 is released at the same time τ_2 finishes waiting and starts transmission. Figure 4.2 (A) shows this timing graph (it is a real possible behavior of the traffic-flows). Flow τ_3 misses its deadline and so the system is

unschedulable with this priority ordering. If these priority assignment policies are optimal then we should not be able to find another priority ordering which results in all the deadlines being met. However if we assign the traffic-flows with the following priority ordering, priority $\tau_2 >$ priority $\tau_1 >$ priority τ_3, then the corresponding worst-case network latency for each traffic-flow are 3 for τ_2, 5 for τ_1 and 7 for τ_3, Figure 4.2 (B) shows this timing graph. All the traffic-flows are schedulable. This case shows that the existing priority assignment policies are not optimal for priority-based real-time on-chip communication. It may happen that a schedulable priority ordering does exist for a set of traffic-flows, but none of the current approaches can find it.

4.2 Properties of Priority-Based Wormhole Switching

Using a motivation example in the last section, we have showed that priority assignment in wormhole switching is different from the classic real-time system area where only one scheduling resource (CPU) is assumed. The special property of wormhole switching determines that a novel priority assignment approach is desirable.

Assuming that priorities have been assigned to each traffic-flow, the schedulability of a traffic-flow set can be determined by the worst-case network latency analysis shown in the last chapter. During the analysis process, we note some special properties. These properties can help us reason about the effect of changing priority ordering of traffic-flows.

Theorem 8. *The network latency of a traffic-flow is not dependent on lower priority traffic-flows.*

Proof. Under fixed priority preemptive arbitration, only packet release from any higher priority flow can preempt the current flow service and delay it. So the network latency of a flow is not dependent on any lower priority flow. □

Theorem 9. *The network latency of a traffic-flow is dependent on the higher priority traffic-flows and their relative priority ordering.*

Proof. In priority-based preemption scheduling model, the observed traffic-flow τ_i suffers two different interferences, direct interference from higher priority traffic-flows and indirect interference from higher priority traffic-flows. For any traffic-flows τ_j meeting the conditions: $\tau_j \in S_i^D$ and $P_j > P_i$, τ_j can force a direct competing interference. For any traffic-flows τ_k meeting the conditions: $\tau_k \in S_i^I$, $\tau_j \in S_i^D$, $\tau_k \in S_j^D$ and $P_k > P_j > P_i$, the activity of τ_k can produce an indirect competing interference. Thus, the interference suffered by the observed traffic-flow is determined by the set of all the higher priority traffic-flows. In addition, it is not hard to find if we swap two higher priority traffic-flows, the indirect interference relation may be changed. A typical case has been shown in Figure 4.1, with priority descending ordering τ_1, τ_2 and τ_3, τ_3 has direct competition with τ_2 and indirect competition with τ_1. However, if we swap priority of τ_1 and τ_2, τ_3 in this situation only suffers direct interference from τ_2. Thus, the priority ordering of higher priority traffic-flows also determines the network latency of lower priority traffic-flows. □

Theorem 9 show that priority ordering is the major factor in the analysis of worst-case network latency for real-time on-chip communications. Unfortunately, the general priority assignment policies of Audsley [5, 6] which require that the response time is not dependent upon the higher priority ordering, is not applicable in this case.

4.3 Lower and Upper Bounds of Worst-Case Network Latency

There is no known efficient algorithm for finding a schedulable assignment of priorities that takes less than exponential time. Consider a set of n real-time traffic-flows, each traffic-flow is mapped to a unique priority level by some priority assignment policy. There are $n!$ distinct priority orderings over this set. Except for a very small size of traffic-flow set, we cannot check all the possible orderings because of exponential time complexity. However, we notice an interesting property that not all the traffic-flows can meet their deadlines with a specific priority level. Let us revisit the case shown in Figure 4.1. If τ_2 is assigned the lowest priority, no matter what the ordering of τ_1 and τ_3, τ_2 is determinately not schedulable. If we can find these 'unsuitable' traffic-flows at each corresponding priority, we can reduce the search space significantly.

Two useful concepts: *lower bound* and *upper bound* of worst-case network latency are introduced. The lower bound of worst case network latency $R_i^{'}$ is the network latency of a traffic-flow τ_i at some priority level when only direct interference is considered. On the contrary, the upper bound of worst-case network latency R_i^* can be obtained for a traffic-flow τ_i when maximum interference is considered. Actually, the real network latency cannot be worked out before we know the priority assignment for all the traffic-flows (Theorem 9). But the possible range can be evaluated aforehand.

The proposed priority assignment algorithm starts from the lowest priority and moves up to the highest priority. During the process of priority assignment, we use an assignment status to mark each traffic-flow, namely, *assigned* or *unassigned*. Thus, all the assigned traffic-flows have lower priorities and all the unassigned traffic-flows are assumed to be higher priorities than the assigned ones. When we

try to assign priority k to an unassigned traffic-flow τ_i, the lower bound of network latency can be obtained by the following approach:

1. Except for the assigned traffic-flows, all the unassigned traffic-flows which share resource link with τ_i are inserted into set S_i^D.

2. Utilizing recurrence relation Eq.(4.1) which comes from the Eq.(3.1) in Chapter 3 to find the lower bound of worst-case network latency.

$$R_i^{'} = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^{'} + J_j^R}{T_j} \rceil C_j + C_i \tag{4.1}$$

Here, all the unassigned traffic-flows which have a resource competing relationship with τ_i are treated as direct interferences.

Additionally, the upper bound of network latency also can be obtained by a similar approach:

1. Except for the assigned traffic-flows, all the unassigned traffic-flows which share resource link with τ_i are inserted into set S_i^D.

2. For any traffic-flow $\tau_j \in S_i^D$, if an unassigned traffic-flow τ_k is found which is in common with τ_j and not in common with τ_i, τ_j will produce an interference jitter and the jitter J_j^I upper bound is $D_j - C_j$.

3. Utilizing the following relation Eq.(4.2) which comes from the Eq.(3.15) in Chapter 3 to find R_i^*

$$R_i^* = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i^* + J_j^R + D_j - C_j}{T_j} \rceil C_j + C_i \tag{4.2}$$

Here we evaluate the network latency based on the worst-case interference jitter scenario and obtain the upper bound of R_i.

Theorem 10. *The worst-case network latency evaluated by the approach presented in the last chapter is never less than its lower bound, for τ_i, $R_i \geq R_i^{'}$ is always true.*

Proof. For a set of n traffic-flows $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ loaded on the network, we assume that priority levels between $n, \ldots, k+1$ have been assigned and k is the available lowest priority. Let priority k be assigned to τ_i . After that, all the unassigned traffic-flows will be allocated priorities between $k-1, \ldots, 1$ in a non-particular order, but have higher priority than τ_i. Thus all the unassigned traffic-flows which share resource link with τ_i will force a direct competition upon it; Eq.(3.1) shows this situation. Besides that, some unassigned traffic-flows also have a relation of indirect interference with τ_i. This can force an extra 'hit' on τ_i. So the worst-case network latency can be achieved in this situation by Eq.(3.15) which takes both direct and indirect interferences into account. By comparison, the lower bound of network latency only considers direct interference. Thus, the network latency evaluated by Eq.(3.1) and Eq.(3.15) for τ_i is never less than its lower bound $R^{'}$. □

So for a flow τ_i, if its network latency lower bound is greater than its deadline, $R_i^{'} > D_i$, then $R_i > D_i$ is a reasonable conclusion. By evaluating the lower bound of network latency, Theorem 10 gives an approach to eliminate all the 'unsuitable' traffic-flows where $R_i^{'} > D_i$ at the specific priority.

Theorem 11. *The worst-case network latency evaluated by our approach presented in last chapter is no more than its upper bound, for τ_i, $R_i \leq R_i^*$ is always true.*

Proof. Follow a similar argument as the proof in Theorem 10, at each priority level, any unassigned flow τ_j which shares the same link with τ_i will contribute direct interferences. And when interference jitter exists, it may introduce the extra

hits. The condition for the interference jitter happens when any unassigned flow τ_k exists that it is in common with τ_j and not in common with τ_i with the priority ordering $P_k > P_j > P_i$. Obviously not all the direct higher priority flows meet this condition. Besides that, even τ_j has interference jitter, the jitter upper bound should be no more than $R_j - C_j$ which is less than $D_j - C_j$ in Eq.(4.2). So the worst-case network latency evaluated by Eq.(3.1) and Eq.(3.15) for τ_i is no more than its upper bound R^*. □

Next we will use the upper bound of worst-case network latency to find the traffic-flow which is guaranteed for schedulablility at the specific priority.

Theorem 12. *If a traffic-flow τ_i is assigned the lowest priority and satisfies the condition $R_i^* \leq D_i$, then if a schedulable priority ordering exists for Γ, a schedulable ordering also exists with τ_i assigned the lowest priority.*

Proof. Follow the approach exhibited by Audsley [5], let us assume that a schedulable priority ordering exists over a traffic-flow set Γ with τ_i assigned priority i. Since τ_i meets the condition $R_i^* \leq D_i$ when it is assigned the lowest priority, so we move τ_i to the bottom of the priority ordering. The traffic-flows originally assigned priority $i+1, \ldots, n$ are promoted one level, Figure 4.3 shows this transformation process. Clearly, the traffic-flows assigned priority $1, \ldots, i-1$ remain schedulable as nothing has changed to affect their schedulability. The traffic-flows set which originally are assigned priority $i+1, \ldots, n$ must remain schedulable because the interference on them has decreased or at least no change due to τ_i now moving to the lowest priority. Since τ_i is schedulable at the lowest priority level, a schedulable ordering exists with τ_i assigned the lowest priority.

□

Theorem 12 implied that when a traffic-flow satisfies the condition $R_i^* \leq D_i$ at the lowest priority, there must be a schedulable priority ordering with τ_i assigned

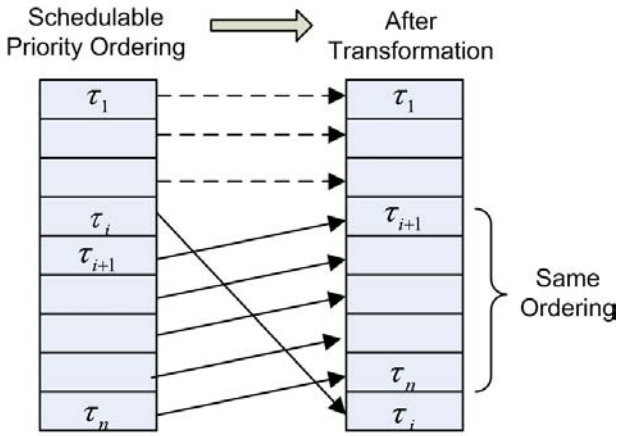


Figure 4.3: Transformation of Priority Ordering

the lowest priority no matter what the ordering of the rest of the traffic-flows. Each time when we find a traffic-flow meeting the above condition we can assign this flow the lowest priority. We can apply this process repeatedly and reduce the search space of priority assignment.

The next Theorem will tell us how to determine unschedulablility before a full priority assignment is given. Let $\Theta(n,k)^{\Gamma}$ denote a partial priority ordering from priority n to k (n > k) for a set Γ with n traffic-flows. $\Theta(n,1)^{\Gamma}$ is a full priority ordering for n traffic-flows.

Theorem 13. *During the process of priority assignment, at the priority level k, a priority assignment policy has produced a partial priority levels ordering $\Theta(n,k-1)^{\Gamma}$ for a traffic-flows set Γ. If no unassigned traffic-flow meets the condition: $R_i^{'} \leq D_i$, $\tau_i \in \Gamma$, at the priority level k, then no schedulable priority ordering exists that assigns the same traffic-flows with priority n to k as $\Theta(n,k-1)^{\Gamma}$.*

Proof. For any priority assignment policy, each priority needs to map to a specific traffic-flow. During the process of priority assignment, at the priority level k, there are $k!$ distinct priority orderings exist. Consider a unassigned traffic-flow τ_i, there should be $(k-1)!$ priority orderings with τ_i at priority level k. If τ_i cannot meet

the condition $R_i^{'} \leq D_i$, no matter what the priority ordering of other unassigned traffic-flows, τ_i is un-schedulable at priority level k according to Theorem 10. If all the unassigned traffic-flows cannot meet the condition $R_j^{'} \leq D_j$ at the priority k, $\tau_j \in \Gamma$, all the $k!$ priority orderings are all un-schedulable. Then, no schedulable priority ordering exists that assigns the same traffic-flows with priority n to k as $\Theta(n, k-1)^{\Gamma}$. □

The properties of Theorem 10, Theorem 12 and Theorem 13 can help to reduce the possible search space for finding the schedulable priority ordering. Therefore, we will use these properties as guidelines while designing an algorithm for priority assignment.

The intuition for the algorithm is as follow. At each priority level k, if any traffic-flow τ_j exists which $R_j^* \leq D_j$, then assign τ_j to level k; otherwise compute the set of traffic-flows with $R' \leq D$. If this set is empty, level k can not be assigned a flow. If the set is non-empty, choose one member τ_i using a heuristic and assign τ_i to level k. If this choice does not lead to a schedulable system, try the other members of the set. Repeat for all priority levels.

4.4 Branch and Bound Priority Assignment

We give a heuristic branch & bound search algorithm (BBSA) (in Figure 4.4) to find a schedulable priority assignment if it exists. The search algorithm proceeds by performing a heuristic guided search to produce a possible priority ordering. When this priority ordering is completed, the schedulability analysis presented in the last chapter is executed to test this ordering. If this priority ordering passes this test, we have found a schedulable priority assignment. If not, the algorithm backtracks to generate another one.

The priority search algorithm starts from the lowest priority. The upper bound

```
procedure LowerBoundAnalysis(Priority P)
 for each unassigned τ_i ∈ Γ
   do { R'_i ← LowBoundofNetworkLatency(τ_i, P)                    (1)
      { if R'_i ≤ D_i                                              (2)
      {   then Insert(τ_i, Priority_P_CandidateList)               (3)

procedure UpperBoundAnalysis(Priority P)
 for each unassigned τ_i ∈ Γ
   do { R*_i ← UpperBoundofNetworkLatency(τ_i, P)                  (4)
      { if R*_i ≤ D_i                                              (5)
      {   then { assigned(τ_i, P)                                  (6)
               { P ← P − 1                                         (7)
               { return ( true )                                   (8)
      {   else return ( false )                                    (9)

main
 for P ← n to 1
   do { if UpperBoundAnalysis(P)                                   (10)
      {   then Continue                                            (11)
      { LowerBoundAnalysis(P)                                      (12)
      { while Priority_P_CandidateList is EMPTY                    (13)
      {   do { P ← P + 1                                           (14)
             { if P > n
             {   then return (FAIL)                                (15)
             { τ_i ← GetTrafficFlowWithPriority(P)                 (16)
             { unassigned(τ_i)                                     (17)
      { τ_i ← GetBestCandidate(Priority_P_CandidateList)           (18)
      { comment: Select the traffic-flow with largest heuristic value next
      { assigned(τ_i, P)                                           (19)
      { P ← P − 1                                                  (20)
      { if P < 1                                                   (21)
      {   then { if (SchedulableTest() == true )                   (22)
               {   then return (SUCCESS)                           (23)
               {   else { P ← P + 1                                (24)
                        { GoTo (13)                                (25)
```

Figure 4.4: Heuristic Branch & Bound Search Algorithm

of worst-case network latency for each unassigned traffic-flow is evaluated at the current priority P (line 10), $P = n$ at the beginning. Any unassigned traffic-flow which meets the condition $R_i^* \leq D_i$ is assigned priority P (lines 4 - 8). At the specific priority, if no traffic-flow with the condition $R_i^* \leq D_i$ exists (line

9), the algorithm starts calculating the lower bound of worst-case network latency (line 12). We construct n prioritized candidate lists *Priority_P_CandidateList*, $1 \leq P \leq n$, each list is related to a priority level P. Initially, each candidate list is empty. Theorem 10 implies only the traffic-flows which meet the condition $R_i' \leq D_i$ might be potentially schedulable at the specific priority. All the unassigned traffic-flows meeting the condition $R_i' \leq D_i$ are inserted into the corresponding priority level P candidate list (lines 2 - 3). After that, if priority P candidate list is not empty, an appropriate traffic-flow is selected from the candidate list to assign to priority P and it is removed from this list (lines 18 - 20). When more than one traffic-flow is available at priority P, to make the search more efficient, we select the "most promising" candidate first. A set of heuristic functions is proposed, the details of which are described below. The algorithm iterates this process at each priority until $P < 1$ (line 21) which means all the traffic-flows are assigned priority. Finally, the complete schedulable test is called.

4.4.1 Heuristic Functions

The heuristic policy helps select the most likely candidate when more than one traffic-flow meets the condition $R' \leq D$. Our heuristic policy is based on an assumption of *tolerating additional interference*, namely, how much additional interference can be tolerated by a traffic-flow at a priority level without missing the deadline. This is a quite natural assumption, the more interference can be tolerated for a traffic-flow, the lower priority level is assigned. This policy leaves the chance for the traffic-flows which have less tolerable additional interference to get higher priorities. How to quantify the tolerable interference for each traffic-flow at a specific priority is an issue.

Here we propose two different baseline evaluation schemes. The first scheme $H1$ is the simple intuitive scheme which considers the absolute difference between

D_i and R'_i as the tolerable interference for a traffic-flow τ_i. The traffic-flow with largest difference value is selected from the corresponding candidate list. Let H_i denote the heuristic value for τ_i.

H1 Heuristic Function :

$$H_i = D_i - R'_i \tag{4.3}$$

The second baseline scheme $H2$ utilizes the sensitivity analysis technique [65] to evaluate the maximum tolerable additional interference. Sensitivity analysis can capture the bounds within which a parameter (basic network latency C in this case) can be varied without violating the timing constraints. Let $R'(C)$ denote the calculation function of lower bound of worst-case network latency, $R'_i = R'(C_i)$. If a traffic-flow τ_i meets the condition $R'_i \leq D_i$, utilizing sensitivity analysis, we can obtain maximum permissible increase of C_i without violating system schedulability. Let ΔC denote the tolerable increase.

H2 Heuristic Function :

$$H_i = \max(\Delta C)\ where\ R'(C_i + \Delta C) \leq D_i \tag{4.4}$$

Besides this, we also consider the fact that the traffic-flow which travels a long distance likely encounters more blocking interference than a short distance. Thus the distance factor also should be taken into account. The following two heuristic functions are improved versions of $H1$ and $H2$ which consider the number of hops a traffic-flow travels.

H3 Heuristic Function :

$$H_i = \frac{D_i - R'_i}{hops} \tag{4.5}$$

H4 Heuristic Function :

$$H_i = \frac{\max(\Delta C)}{hops} \; where \; R'(C_i + \Delta C) \leq D_i \tag{4.6}$$

The distance is inverse proportional to the tolerable additional interference so that the heuristic value calculated by $H3$ and $H4$ decreases when a traffic-flow traverses more hops. Accordingly, this traffic-flow has less possibility to be choosen at the lower priority level, which is as desired. When traffic-flow traverse a single hop, the heuristic value produced by $H3$ and $H4$ is equal to the result calculated by $H1$ and $H2$ respectively.

Also, we notice that for any given flow τ_i, all the higher priority traffic-flows in S_i^D will take a proportion of available network communication capacity $\sum_{\forall j \in S_i^D} C_j/T_j$. Thus, only a fraction of the network service capacity 1-$\sum_{\forall j \in S_i^D} C_j/T_j$ remains available to accommodate the current flow τ_i. The flow with higher free network service capacity is a better candidate to be served with lower priority. Thus the available network capacity is also modelled as a heuristic metric. The following two heuristic functions are the improved versions of $H1$ and $H2$ which consider network service capacity.

H5 Heuristic Function :

$$H_i = \frac{D_i - R'_i}{\sum_{\forall j \in S_i^D} C_j/T_j} \tag{4.7}$$

H6 Heuristic Function :

$$H_i = \frac{\max(\Delta C)}{\sum_{\forall j \in S_i^D} C_j/T_j} \; where \; R'(C_i + \Delta C) \leq D_i \tag{4.8}$$

4.4.2 Backtracking policy

It is not difficult to see that the search proceeds like a depth first search tree, each possible priority ordering is a branch of the search tree. However, the heuristic functions do not always select the right branch which induces a schedulable priority assignment. Thus, backtracking becomes an important part of the algorithm to help locate the correct search path. During the search iteration process, backtracking occurs at two possible places.

Middle unschedulable branch : Sometimes there is no traffic-flow satisfying the condition: $R'_i \leq D_i$ at the priority level P. In this situation, Theorem 13 implies that there is no schedulable priority ordering based on the traffic-flows which have been selected. The algorithm in this situation backtracks its search process by unassigning the selected traffic-flow at the priority $P + 1$ and tries to find another appropriate one, pseudo code lines 13 - 19 describe this process. Therefore, we prune some middle unschedulable branches to make the search more efficient.

End unschedulable branch: When a complete priority ordering has been found (line 21), the full schedulable test is called. If the priority ordering does not pass the schedulability test, the algorithm has to backtrack to find other appropriate priority orderings by following the same procedure to the middle unschedulable branch (lines 24 - 25).

Because of the restricted condition and pruning branch policy, only a small search space is produced compared with the original $n!$. The algorithm utilizes the backtracking policy to search the possible priority orderings. Since the search space is determined by the upper and lower bounds of each flow which heavily rely on the competing relationship between them. So the complexity of algorithm is still an open issue, but the worst case should be no more than $n!$.

4.5 Evaluation

This section describes experiments that have been conducted to quantify the performance of the branch and bound search algorithm (BBSA). The evaluations utilize the same simulation platform introduced in the last chapter and the generation of basic attributes for each traffic-flow also follows the same fashion. The following two sections show the scope of the experiments: the first section discuss the effect of different heuristic functions in BBSA while the second section presents the performance of BBSA against a set of existing priority assignment approaches: RM(T) and improvements of RM (T/Hops and T/ln(e+Hops-1)).

Except for the evaluation with a varied size of flows set, each generated test set contains 30 flows and each investigation level on the result diagrams is the average of 1000 randomly generated flow sets.

4.5.1 Evaluation of Different Heuristic Functions

The efficiency of the BBSA algorithm depends heavily on its heuristic search policy, so in this experiment, we examined the performance of the six different heuristic functions. During the heuristic search processing, if a promising candidate is found, it will be assigned the current priority and the search moves to the higher priority level; otherwise, none of the current flows meeting the condition, the search algorithm backtracks to a lower priority level. So a reasonable metric to compare the efficiency of different heuristic functions is a measure of the number of priority assignments, namely, how many priority assignment operations are required to find a schdulable priority ordering. We design two groups of experiments, first we compare these heuristic functions with different levels of overall link utilizations. And then, we varied the number of traffic-flows from 40 to 100 with a fixed link load. All the experiment measures are taken under the maximum

and average link load schemes separately.

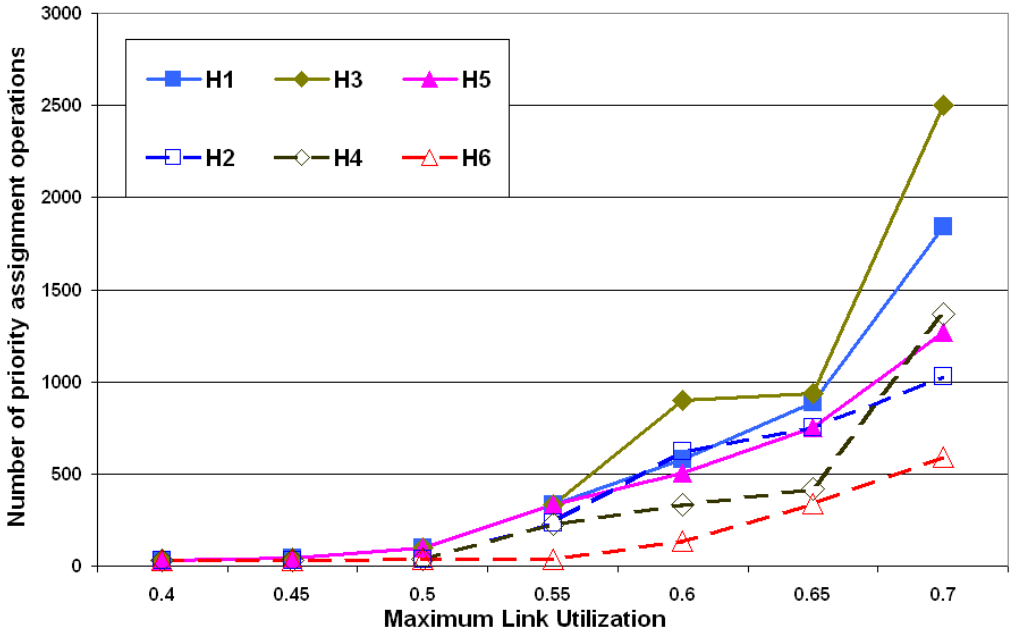


Figure 4.5: The Number of Priority Assignment Operations Under Varied Maximum Link Utilization

Figure 4.5 and Figure 4.6 show the average number of priority assignment operations for several heuristic functions under varied link load. As shown in both figures, all of the heuristic policies perform comparatively well and find a schedulable priority ordering quickly at the lower network load (no more than 0.28 in average load or 0.5 in maximum load). This is because in many cases, no matter what the priority ordering, the traffic-flow set is always schedulable. However, with the increase of network load, the iteration operation increases rapidly. The difference among these heuristic functions becomes clear. As can be seen in both figures, there is a significant reduction in the average number of assignment operation for heuristic algorithms with the sensitivity analysis technique (H2, H4 and H6) than the intuitive ones (H1, H3, H5) which consider the absolute distance between deadline and worst-case latency. The average operation numbers from H2,

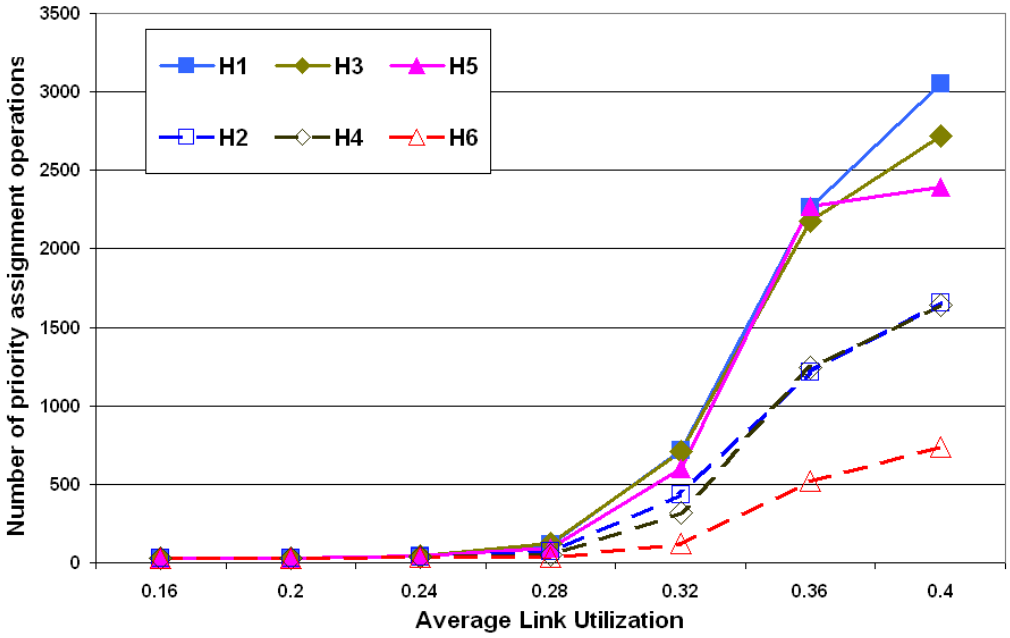


Figure 4.6: The Number of Priority Assignment Operations Under Varied Average Link Utilization

H4 and H6 sometimes are even no more than half of the corresponding H1, H3 and H5. Two considered heuristic factors, distance and network service capacity, give comparable results. From the observations, the operation numbers of H5 and H6 are reasonable reduced compared with the baseline functions H1 and H2 when the network service capacity is considered. But the distance as heuristic factor (H3 and H4) does not present more advantage over H1 and H2.

Figure 4.7 and Figure 4.8 show the variation of priority assignment operations as a function of the size of traffic-flows set for fixed link utilization scenarios. With the increase of the flows set size, the gap between H1, H3, H5 and H2, H4, H6 becomes progressively larger. It is evident that the sensitivity analysis technique can achieve the best performance as flow set increase. Finally, in all cases, the experiment result shows that the H6 scheme which combines sensitivity analysis and network load capacity performs equally well and is highly effective

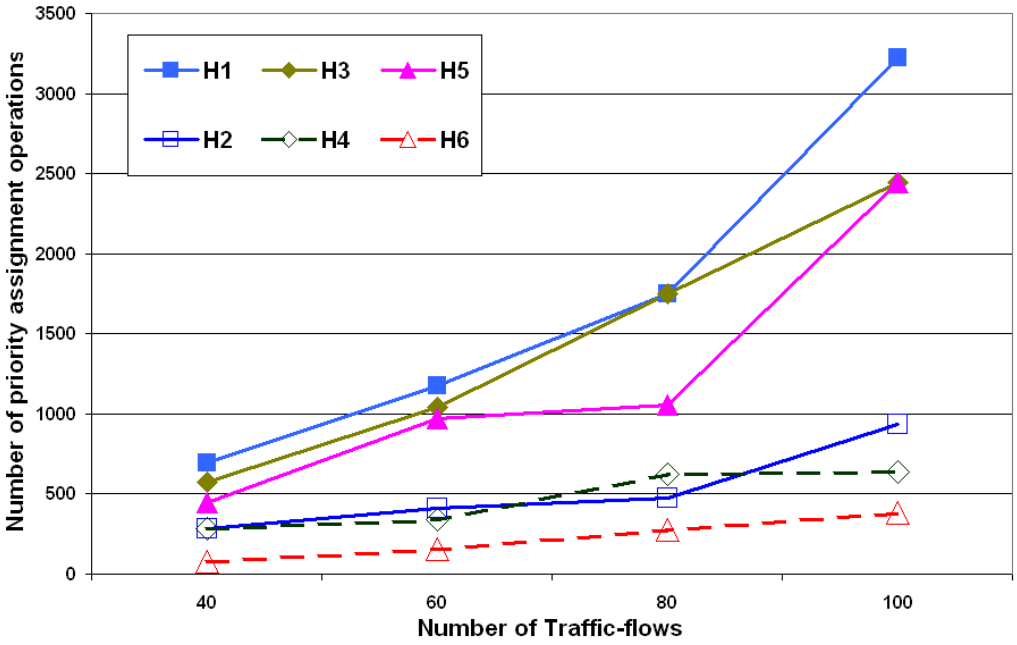


Figure 4.7: The Number of Priority Assignment Operations With Varied Number of Traffic-Flows Set Under 55% Maximum Link Utilization

in reducing the number of heuristic search operations. The average timing cost to find a schedulable priority order with 100 traffic-flows is no more than 1 second with the Intel Core 2 Duo (3GHz) and 2G bytes memory.

4.5.2 Evaluation of Different Priority Assignment Schemes

This experiment investigated the performance of different priority assignment policies. Hary *et al* [35] found that laxity as the priority assignment criteria has similar performance to the period based ones, therefore, in our evaluation, only the period related priority assignment criterions are considered. Three existing schemes that are compared with BBSA are : RM(T) [60], improvements of RM (RM(T/H) and RM(T/ln(e+H-1))) [35]. The evaluation is aimed at the question "is it worth undertaking the heuristic scheme". The comparison measure used in this part is *pass ratio*, namely, the percentage of the number of traffic-flows set

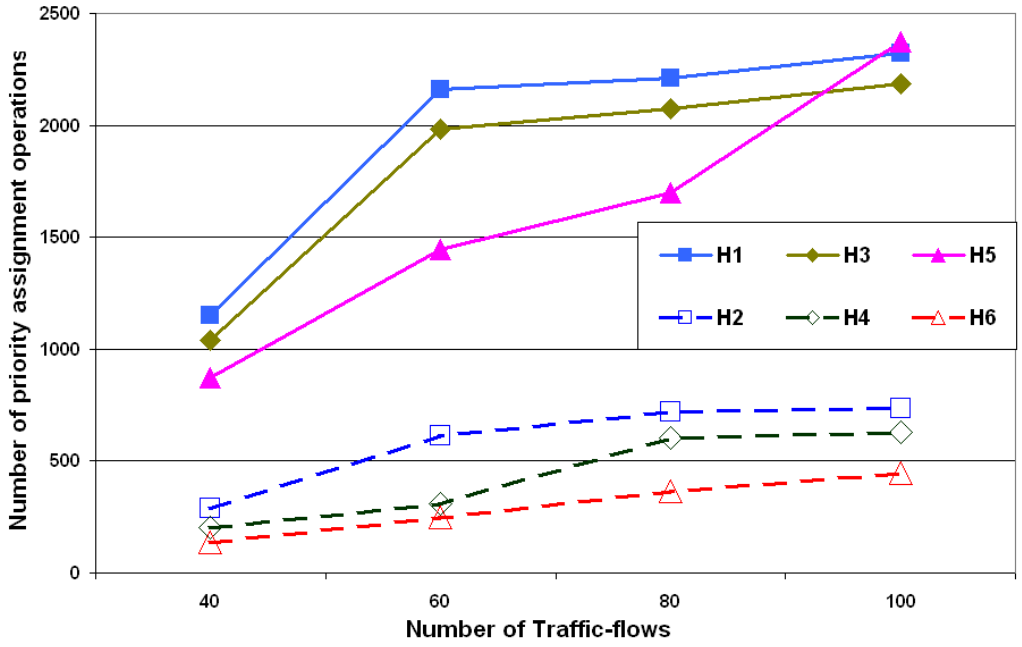


Figure 4.8: The Number of Priority Assignment Operations With Varied Number of Traffic-Flows Set Under 32% Average Link Utilization

that pass the schedulability test versus the number of original traffic-flows set with different priority assignment policies. Only if all the traffic-flows in a generated set pass the schedulability test, do we say that the set is schedulable. The H6 heuristic function is selected as the default for the BBSA. For evaluation reasons, we do not try to check all the possible branches but set a stoping criteria for the BBSA with the maximum number of iterations. When the algorithm reaches the maximum iterations without finding a schedulable priority ordering for a set, this set is treated as unschedulable. In general, the more traffic-flow sets meet their timing constraints, the higher performance for the assignment policy.

Figure 4.9 and 4.10 plot, for each utilization level, the percentage of traffic-flow sets that were schedulable according to each priority assignment scheme. We find, under the lower link utilization, there is no big performance gap between different priority assignment polices. All the schemes achieve nearly 100% pass

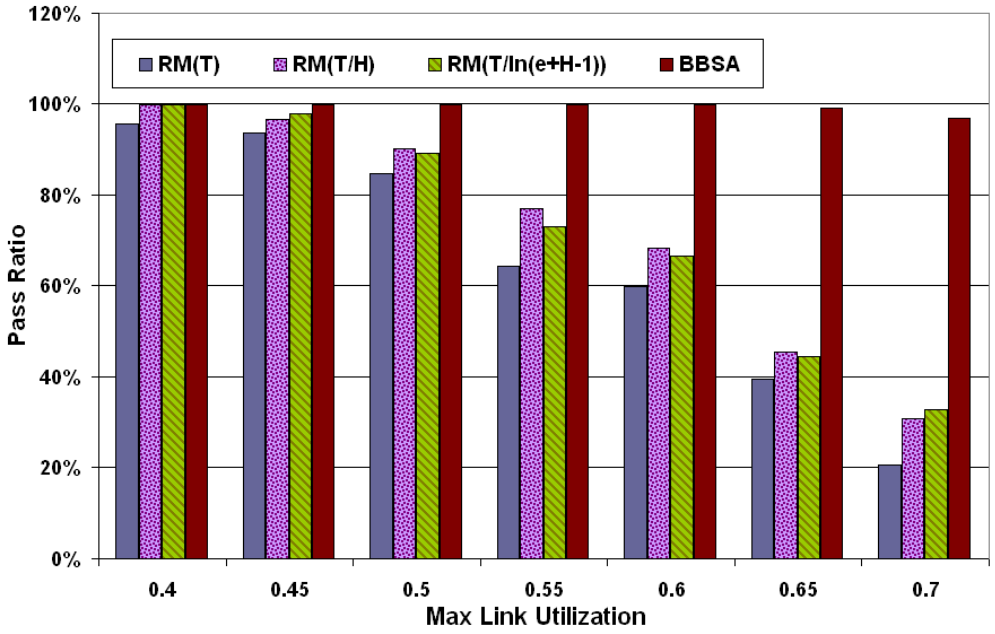


Figure 4.9: The Pass Ratio Under Varied Maximum Link Utilization

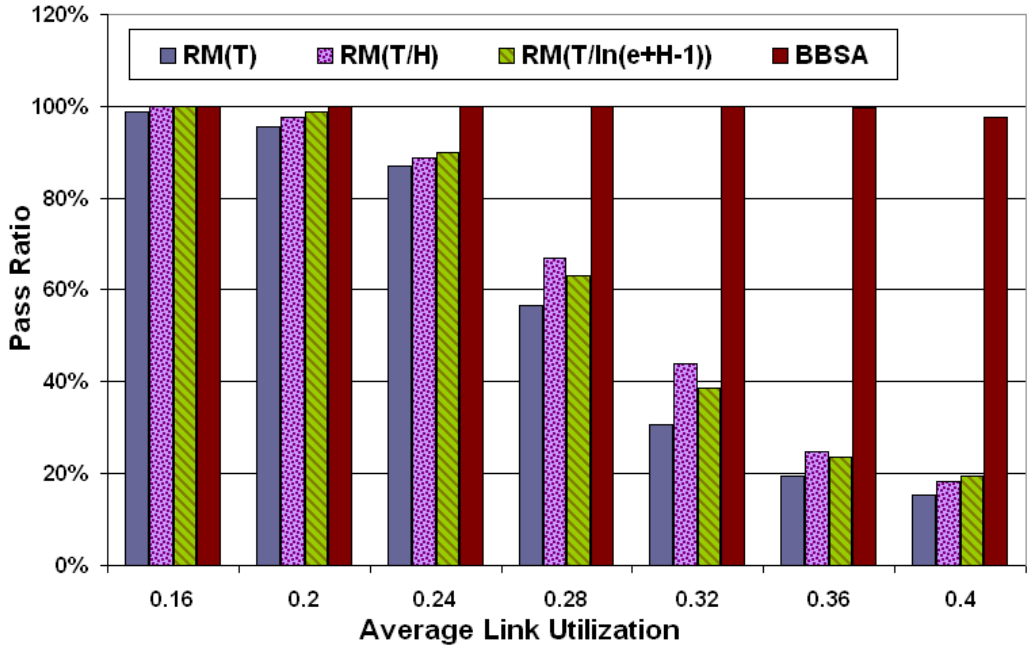


Figure 4.10: The Pass Ratio Under Varied Average Link Utilization

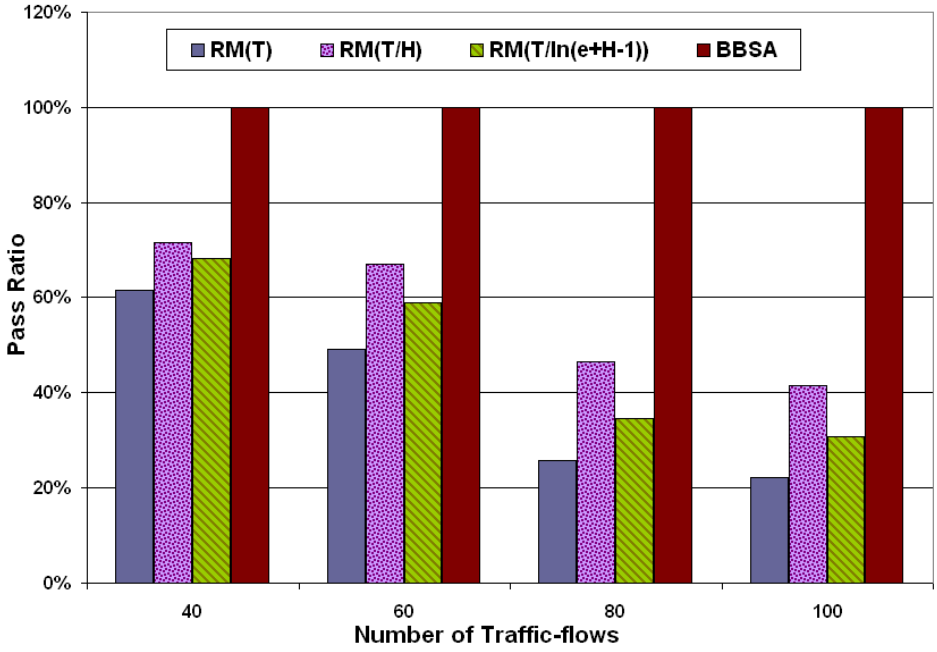


Figure 4.11: The Pass Ratio With Varied Number of Traffic-Flows Set Under 55% Maximum Link Utilization

ratio. When the link load increases, pass ratio decreases as expected. The heuristic search algorithm significantly outperforms the RM and its related schemes under higher network load. It still can find a schedulable priority assignment for a set even when the network maximum link load reaches 0.6, comparing with RM which only gets 59.8% pass ratio in Figure 4.9. Also, from Figure 4.10, there is around 80% better performance of the pass ratio than RM; when the average utilization of the network is 0.36.

Figure 4.11 and Figure 4.12 show the pass ratio variation with the increase in the traffic-flow set size. BBSA shows its robustness and achieves the highest pass ratio. In addition, the number of flow sets does not greatly affect BBSA performance, and the pass ratio remains nearly constant. By comparison, RM and related policies perform poorly and their performances are sensitive to the size of flow set. When the number of flow set increases from 40 to 100, the average

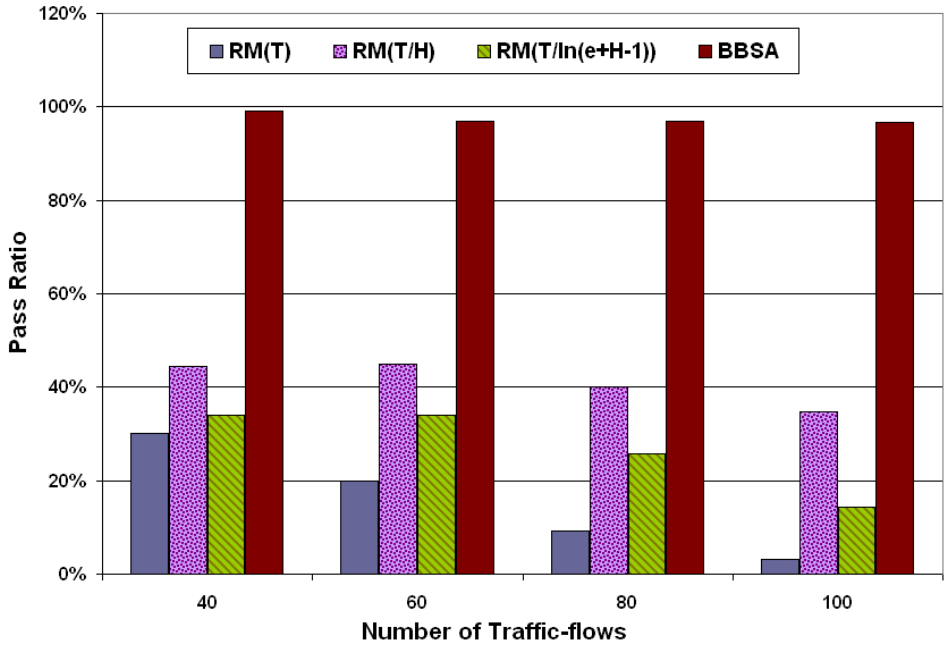


Figure 4.12: The Pass Ratio With Varied Number of Traffic-Flows Set Under 32% Average Link Utilization

pass ratio falls to almost 35% and 20% for RM related policies under the two test scenarios.

From an engineering view-point, when the network load is not high (e.g. no more than 40% for maximum link utilization or 20% for average link utilization with 30 flows in a 4×4 mesh NoC), RM related priority assignment policies are a good choice since they are simple and achieve acceptable schedulable pass ratio. When the network load is higher, to find a possible priority ordering, BBSA is a much better alternative.

4.6 Summary

In this chapter we discuss a general problem in priority-based network systems: how to assign priority to each flow so that all the flows loaded on the network will

meet their time constraints. By a motivation example, we show at the beginning of the chapter that none of the current priority assignment techniques can support priority based wormhole switching properly. And then, the upper and lower bound network latencies are introduced which can be found even without exact priority ordering. These properties help to reduce the possible search space for finding the schedulable priority ordering. Based on these properties, a heuristic branch and bound search algorithm is developed. The results of evaluation show that this search algorithm significantly outperforms the traditional priority assignment policies.

Chapter 5

Priority Share Support in Priority Based Wormhole Switching

The last two chapters presented worst-case network latency analysis and priority assignment techniques in priority-based wormhole switching. The hard timing bound is delivered by this approach with the support of a priority based router infrastructure which allocates each traffic-flow a distinct priority and virtual channel independently. But the drawback of this approach is precisely that this *distinct priority* per traffic-flow implementation policy results in higher virtual channel and priority overhead. To overcome this problem, in this chapter, we propose a solution that utilizes a *priority share policy* to reduce the resource overhead while still achieving hard real-time communication guarantees. The priority share policy permits multiple traffic-flows to contend for a single virtual channel and share the same priority level.

This chapter is organized as follows: the next section introduces the basis model of wormhole switching with support for priority sharing. The related attributes and flow contention relationships are discussed. Considering the extra blocking delay introduced by this priority share, composite model-based schedu-

lability analysis is presented in Section 5.2. This approach models all the traffic-flows sharing the same priority as a single scheduling entity. Hence, the packet transmission is analyzed holistically and the computational complexity is kept low. But this approach requires that all the flows must meet the constraint that network latency is no more than period. This is a strong restriction. To remove this restriction, in Section 5.3, we change the view of analysis from per flow to per priority, that is, it computes the total time window at each priority level instead of each traffic-flow. By checking the packet release instance of each traffic-flow at the corresponding priority window, we can verify the timing semantics of each real-time traffic-flow with a simple yet efficient mechanism. The deadline no more than period constraint is successfully removed by this approach. In addition, we also find that the previous result is just a special case covered by this new analysis. Building on the static analysis, in Section 5.4, we furthermore propose a set of share priority allocation policies, in such a way that the timing requirements of all the traffic-flows are met with a reduced resource overhead. By evaluations based on enumeration, Section 5.5 presents the results of this approach. Finally, Section 5.6 summarises this chapter.

5.1 Wormhole Switching with Priority Share

5.1.1 Priority Share Structure

Real-time communication in priority-based wormhole switching networks has been studied widely. Based on the distinct priority per traffic-flow assumption, a group of schedulability analysis approaches have been explored; the related works have been reviewed in Chapter 2. In Chapter 3, we also give a novel analysis technique, which successfully emulates wormhole switching as a classic single processor scheduling model. However, all these works have the basis assumption

of a distinct priority and exclusive virtual channel per traffic-flow in order to keep each flow independent and avoid complex interference. This should be reasonable for a general network in terms of cheap storage and computation resources. But the realistic concern in implementation cost and energy consumption of on-chip networks always means that the number of virtual channels supported by an interconnection infrastructure is kept as small as possible. To address this problem, we propose a priority share scheme by mapping multiple traffic-flows onto a single virtual channel. This optimization is simple but effective.

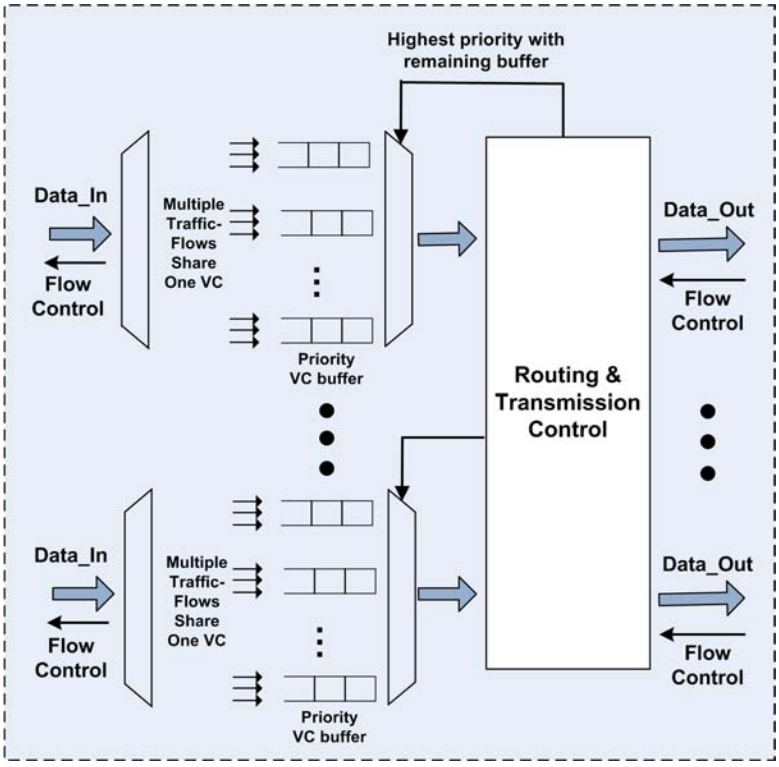


Figure 5.1: Output Arbitration with Priority Share

Figure 5.1 shows a router structure in wormhole switching networks with the priority share support. There are a number of prioritized virtual channels available at each router input port. Differing from the previous model discussed in Chapter 2, the distinctive characteristic of this scheme is that multiple traffic-flows per virtual channel are supported. Assume each port of a network router only sup-

ports limited a limited number of distinct virtual channels and priority levels in terms of the restrictions of hardware. Multiple traffic-flows loaded in the wormhole network are allocated the same virtual channel and thus are mapped to the same priority. Each packet generated by the traffic-flow inherits this priority. A packet with priority G_i can only request the virtual channel associated with priority G_i. With priority arbitration, at any time, a flit of a given packet will be sent out through its respective outport if it has the highest priority and it has sufficient buffer in subsequent routers. In addition, a higher priority packet can also preempt a lower priority packet during its transmission (flit level preemption). If the highest priority packet cannot send data because it is blocked elsewhere in the network, the next highest priority packet wins the output arbitration. As a hybrid solution, best-effort traffic-flows also can be multiplexed on the same links with lowest priority (any real-time flow has higher priority than best-effort flows). In the case where no real-time flow is available, best-effort flows make use of spare bandwidth.

To show why the priority share scheme can save the resource cost, we give a case in Figure 5.2, which illustrates five traffic-flows τ_1, τ_2, τ_3, τ_4 and τ_5 loaded on a NoC platform. Under the traditional distinct priority policy, each flow must be allocated an unique priority and exclusive virtual channel. So the total number of priority levels and the number of virtual channels for this case are 5 and 14 (2 VCs for τ_1 and τ_2 respectively, 3 VCs for τ_4 and τ_5 respectively and 4 VCs for τ_3). But when priority share is supported, better mapping are possible. Figure 5.3 illustrates a such approach where flows τ_1, τ_2 and τ_3 share the same priority G_1, τ_4 and τ_5 share the same priority G_2 and G_1 is higher priority than G_2. The whole priority and virtual channel costs are only 2 and 10 (5 VCs for priority G_1's flows where flows τ_2 and τ_3 share one VC at router 4, flows τ_1 and τ_3 share VCs at routers 2 and 1; 5 VCs for priority G_2's flows where flows τ_4 and τ_5 share VCs at

routers 4 and 5) which are significantly smaller than the previous scheme. This is a substantial resource saving with no performance degradation in terms of missed deadlines.

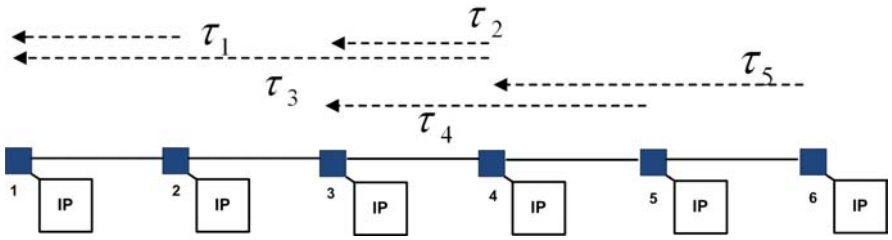


Figure 5.2: Traffic-flows with Distinct Priority

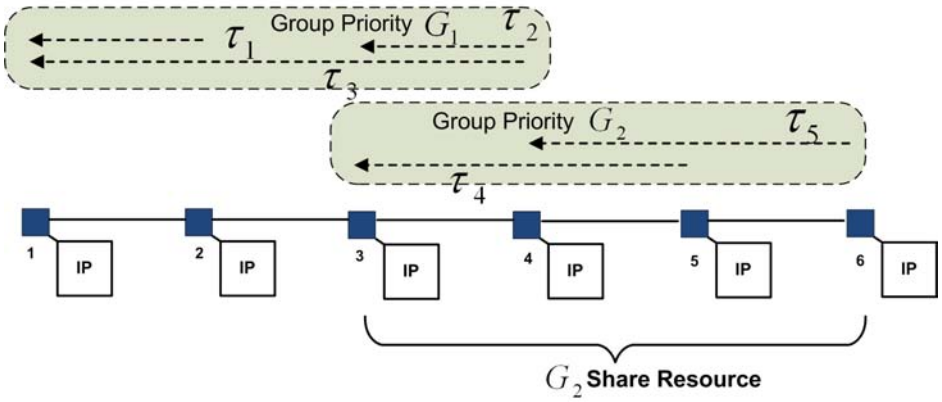


Figure 5.3: Traffic-flows with Priority Share

5.1.2 Blocking Problem

The priority share scheme can dramatically reduce the resource overhead compared with the original distinct priority per traffic-flow scheme, but on the other side, it may lead to significantly blocking and unpredictable network latency. Consider the fact that traffic-flows within the same virtual channel are served in the first-in-first-out (FIFO) order because priority preemption is only available between the different virtual channels. When a traffic-flow has to wait for the trans-

mission of another traffic-flow in the same buffer due to priority share, blocking occurs. The blocking appears in real-time communication where the physical resource is not enough to handle the practical service requests. Therefore, a packet can be blocked by every packet with the same priority which arrives before it. Once a packet is blocked by another packet with the same priority which holds a virtual channel for a prolonged duration, it can block other packets, which can in turn block other packets, and so on.

As a simple example to motivate the blocking problem, let us revisit the case showed in Figure 5.3. We assume that a packet from τ_5 is released, because of priority share τ_5 can be blocked by τ_4 if it arrives just after τ_4. During τ_4's transmission, it can be preempted by the packet releases from higher priority flows τ_2 and τ_3. Note that when τ_2 or τ_3 is active, τ_4's packet service will be suspended but still occupy the link resources. In this situation, only after τ_4's completion, can the packet from τ_5 resume its transmission service. So the interference suffered by τ_4 actually extends the possible completion time of τ_5. Besides that, flow τ_1 in this case also can introduce some interference (see section 5.2) which delays the network latency of τ_5 further. Even though flows τ_1, τ_2 and τ_3 never share any physical link with τ_5, they still can play the major role in determining τ_5's transmission latency. This phenomenon does not exist in the distinct priority per flow policy. The exact analysis in this situation is very hard due to the complicated blocking inter-relationship between the flows. The blocking leads to packet delays and even missed deadlines. Therefore, a novel schedulability analysis approach to predict whether all the real-time packets can meet their timing requirements is desirable.

5.1.3 Traffic-flow Model and Inter-relationships

When the network supports priority share, more than one flow can share the same priority level. In order to distinguish from the distinct priority model discussed in previous chapters, here we introduce a new term "*group priority*" G. The value 1 denotes the highest priority and larger integers denote lower priorities. All the traffic-flows competing for the same virtual channel will be mapped to the same group priority. Therefore, each traffic-flow τ_i here is characterized by the new six-tuple attributes $\tau_i = \{G_i, C_i, T_i, D_i, J_i^R, J_i^I\}$. All the flow with the same group priority G_i can be regarded as a set denoted by $S(i)$. In order to avoid run time complexity, here we require any flow if it is assigned a group priority at a router's port, it must have been assigned the same group priority at each router's port throughout its path. The group priority should be assigned off-line and remain constant at run-time. We also define a function $G(\tau_i)$ to obtain the corresponding group priority for a traffic-flow τ_i. It follows that $\tau_i \in S(G(\tau_i))$.

Under the priority share policy, assuming that the group priorities have been assigned to each traffic-flow, if a packet is released, the possible delays it suffered before completion consist of all the interferences from higher priority traffic-flows and the blocking from the traffic-flows with the same priority. The direct and indirect interferences from higher priority flows have been presented in Chapter 2. The blocking is a new delay which is introduced by the priority share policy. Based on different competing relationships, we categorize the blockings into two different types:

- **Direct blocking from traffic-flow with same priority**

  When two traffic-flows τ_i and τ_j have the direct competing relationship and meet the condition $G(\tau_j) = G(\tau_i)$, if the packet from τ_j is released just before τ_i, τ_j will block τ_i. For flow τ_i, we define a direct blocking set S_i^{SD}

which includes all the traffic-flows meeting the above conditions, S_i^{SD} = $\{\tau_j | \Re_j \cap \Re_i \neq \emptyset$ *and* $G(\tau_j) = G(\tau_i)$ *for all* $\tau_j \in \Gamma\}$.

- **Indirect blocking from traffic-flow with same priority**

  When two traffic-flows τ_i and τ_k have an indirect competing relationship and meet the conditions $G(\tau_k) = G(\tau_j) = G(\tau_i)$, τ_j is the intervening flow, τ_i may suffer an indirect blocking from τ_k even they do not share any physical link. An indirect blocking example has been shown in Figure 5.3 where τ_1 blocks τ_3 and further blocks τ_2. For flow τ_i, we define an indirect blocking set S_i^{SI} which includes all the traffic-flows meeting the above conditions, S_i^{SI} = $\{\tau_k | \tau_k$ has indirect competing relationship with τ_i *and* $G(\tau_k) = G(\tau_j) = G(\tau_i)$ *for all* $\tau_k \in \Gamma$, where τ_j is any intermediate flow$\}$.

Returning to the example in Figure 5.3, five traffic-flows $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5 are mapped into two sets, the set with priority G_1 includes τ_1, τ_2 and τ_3, $S(1) = \{\tau_1, \tau_2, \tau_3\}$, the set with priority G_2 includes τ_4 and τ_5, $S(2) = \{\tau_4, \tau_5\}$; $G_1 > G_2$. Traffic-flows τ_1, τ_2 and τ_3 have no shared links with any higher priority flow so no direct or indirect interference,

$$S_1^D = S_1^I = S_2^D = S_3^I = S_3^D = S_3^I = \emptyset.$$

Due to sharing the same group priority, the direct and indirect blocking sets for τ_1, τ_2 and τ_3 are:

$$S_1^{SD} = \{\tau_3\}, S_1^{SI} = \{\tau_2\},$$
$$S_2^{SD} = \{\tau_3\}, S_2^{SI} = \{\tau_1\},$$
$$S_3^{SD} = \{\tau_1, \tau_2\}, S_3^{SI} = \emptyset.$$

Flow τ_5 does not have any higher priority flow, so

$$S_5^D = S_5^I = \emptyset.$$

Flow τ_4 directly competes with τ_3 and τ_2 and indirectly competes with τ_1 and hence

$$S_4^D = \{\tau_2, \tau_3\}, S_4^I = \{\tau_1\}.$$

Besides that, τ_4 and τ_5 share the same group priority level, thus

$$S_4^{SD} = \{\tau_5\}, S_4^{SI} = \emptyset,$$
$$S_5^{SD} = \{\tau_4\}, S_5^{SI} = \emptyset.$$

With the definition of the new priority share model and traffic-flow relationship, we will derive, in the next section, an upper bound on the schedulability of real-time flows.

5.2 Schedulability Analysis with the Composite Model

One of the main challenging issues of NoC is how to use the minimum available on-chip resource while achieving considerable transmission quality of service. The on-chip resource overhead can be effectively reduced with a priority share policy. But this scheme inevitably introduces blocking which heavily complicates the analysis process. None of current schedulability analysis techniques (presented in Chapters 2 and 3) can handle this case because all these approaches assume there are sufficient priority levels so that blocking due to sharing virtual channels does not occur.

5.2.1 Properties of Priority Share

When multiple traffic-flows share the same priority with the $D \leq T$ constraint, we notice some special properties. These properties can help us reason about the effect of blocking and interference.

Theorem 14. *When a set of traffic-flows with $D \leq T$ and the same group priority are all schedulable, one flow cannot block any other flow more than once.*

Proof. Let us assume a set of traffic-flow $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with same priority level G_i. Suppose τ_a and τ_b are two flows in set $S(i)$, $T_a \geq T_b$, and both meet the timing bound, the theorem is proved by the following two separated statements,

1. τ_b cannot be blocked by τ_a more than once while still meeting its deadline.

2. τ_a cannot be blocked by τ_b more than once while still meeting its deadline.

With the priority share policy, any flow can be blocked by other flows with the same priority just because of early arrival. Thus, the interference from higher priority and blocking from other same priority flows jointly determine the maximum latency of the current one. Suppose during any interval $[t_1, t_2]$, $t_1 \leq t_2$, the maximum interference suffered by a traffic-flow τ_i is $I_i(t_1, t_2)$. According to the property of priority share, if τ_b is released ahead of τ_a, the interference suffered by τ_b will be treated as blocking for τ_a, and vice versa.

With the condition $T_b \leq T_a$, statement (1) can be proved by contradiction. Let r_i denote the network latency of any flow τ_i. If τ_b is blocked by τ_a more than once, r_b must be greater than T_a, $r_b > T_a$. However, in order to meet the deadline, the relation $r_b \leq D_b \leq T_b \leq T_a$ must hold, which contradicts with the above.

Statement (2) is proved by a similar contradiction. For τ_b, during the time interval [0, r_b], the total service requirement from τ_b, τ_a and all the other flows sharing the same system priority is no more than $C_a + C_b + \sum_{\tau_i \in \{S(i) - \tau_a - \tau_b\}} \lceil \frac{r_b}{T_i} \rceil C_i$. The total service requirement from all the higher priority flows is no more than $\sum_{\tau_i \in S(i)} I_i(0, r_b)$. The network latency is hence derived by:

$$r_b = C_a + C_b + \sum_{\tau_i \in \{S(i) - \tau_a - \tau_b\}} \lceil \frac{r_b}{T_i} \rceil C_i + \sum_{\tau_i \in S(i)} I_i(0, r_b) \tag{5.1}$$

Since τ_b is schedulable with $r_b \leq D_b \leq T_b \leq T_a$, and thus a value r_b where $r_b \leq T_b$ must exist to fit Eq.(5.1).

On the other hand, if τ_a is blocked by τ_b more than once, $r_a > T_b$ must be true. The time interval [0, r_a] can be subdivided into [0, T_b], and [T_b, r_a]. During [0, T_b], the service requirement must be greater than the network transmission capacity because until r_a, τ_a just completes its packet service. Thus, the following relation must be hold

$$C_a + C_b + \sum_{\tau_i \in (S(i) - \tau_a - \tau_b)} \lceil \frac{t'}{T_i} \rceil C_i + \sum_{\tau_i \in S(G_i)} I_i(0, t') > t' \tag{5.2}$$

where t' is any time instant during the time period [0, T_b]. But this contradicts Eq.(5.1) where a value no more than T_b can be found to fit this equation. Hence it is impossible that τ_b may block τ_a more than once while still meeting its deadline.

□

5.2.2 Analysis with the Composite Model

The schedulability analysis technique is the key to guarantee that the deadlines are always met in the hard real-time communication system. The worst-case network latency analysis discussed in Chapter 3 is based on per traffic-flow evaluation and tries to find the maximum interference for each observed flow. However, the analysis process becomes complicated when the priority share policy is supported. Both blocking and interference need to be taken into account. To solve this problem, here we propose a novel scheme - the *composite traffic-flow model*. The composite model scheme tries to collapse all the flows sharing the same group priority as a single scheduling entity and hence, all the link resources required by the flows with the same priority level are treated as a single resource competing model. The motivation behind this approach is that by composite analysis, all the flows sharing the same priority will be scheduled as a holistic unit, so the complicated blocking analysis is effectively avoided and the computational complexity is

kept sufficiently low.

Consider a set of traffic-flow $S(i)$ with the same priority level G_i, the following steps are used to derive a composite task $\hat{\tau}_i$:

1. For the composite task $\hat{\tau}_i$, the maximum basic network resource usage is assigned equal to the summation of the all the flows in $S(i)$.

$$\hat{C}_i = \sum_{\forall \tau_m \in S(i)} C_m \tag{5.3}$$

2. Define the higher priority set $hp(i)$; all the flows in S_m^D are the members of set $hp(i)$,

$$hp(i) = \bigcup_{\forall \tau_m \in S(i)} S_m^D \tag{5.4}$$

where $\bigcup$ is the union operation of the flow sets.

3. The deadline for composite task $\hat{\tau}_i$ is equal to the minimum one of the original flows set.

$$\hat{D}_i = \min_{\forall \tau_m \in S(i)} (D_m) \tag{5.5}$$

In this composite model, all the original flows in $S(i)$ are integrated into a single transmission unit $\hat{C}_i$. Theorem 14 shows that no traffic-flow can be blocked more than once by the other flow with the same priority level. This actually implies the fact that the total service requirement from priority level G_i during [0, $max(R_1, R_2, \ldots, R_n)$] is no more than $\sum_{\tau_m \in S(i)} C_m$. We have showed in Section 5.1.2, due to blocking, any packet release from higher priority flow may extend lower priority flow completion time even when they never share any link resource. Therefore all the original higher priority flows of each τ_m, $\tau_m \in S(i)$ are treated as the interference for this new composite flow, $hp(i) = \bigcup_{\forall \tau_m \in S(i)} S_m^D$. We assume the start service time for composite traffic-flow is released simultaneously with all

the higher priority traffic-flows. Without loss of generality, we assume this time instant is at time 0. Until the time instant $\hat{R}_i$ when all the transmission service from priority level G_i complete, during the time interval [0, $\hat{R}_i$], the upper bound of interference produced by any direct higher priority traffic-flow τ_j, $\tau_j \in hp(i)$, during this time interval is:

$$\lceil \frac{\hat{R}_i + J_j^R}{T_j} \rceil C_j \tag{5.6}$$

Note that, here we do not consider the interference jitter problem, but it does occur when indirect higher priority flows exist. This implies that the interference evaluated from Eq.(5.6) may be less than the real worst-case scenario. The relative analysis and proof have been discussed in Chapter 3. But the analysis in that Chapter only considers distinct priorities per traffic-flow. When the network supports priority share, the possible interference jitter comes from two potential situations.

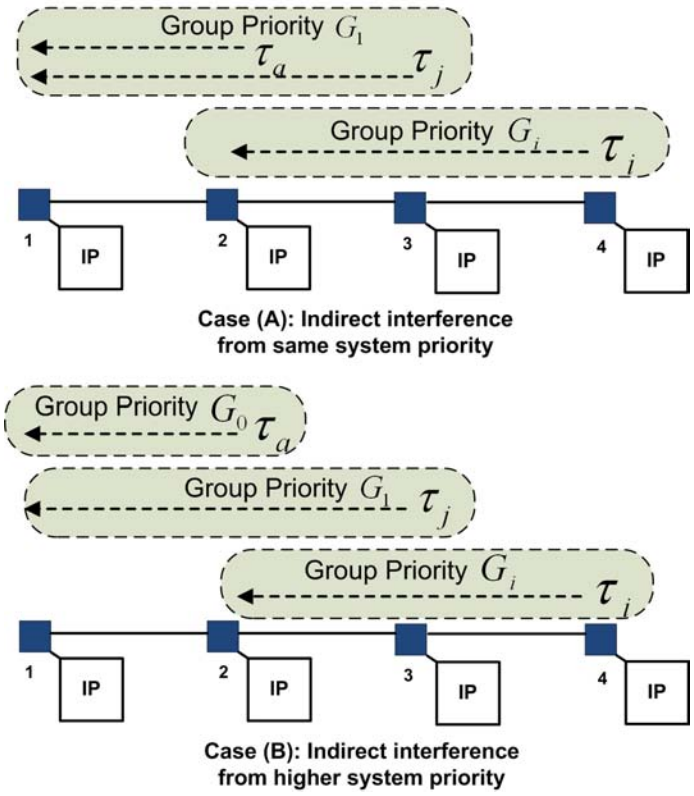


Figure 5.4: Two Possible Situations for Indirect Interference

Figure 5.4 illustrates these two situations which induce interference jitter. Flow τ_a shares the same priority with τ_j in Case (A) and higher priority than τ_j in Case (B). The transmission service from τ_a will force in τ_j an un-expected deferral. From the view of priority level-G_i flows, no matter what scenario occurs, the final result is similar: the indirect interference or blocking results in the real packet minimum arrival interval from direct higher priority traffic-flow τ_j is less than the original assumption T_j, this is modeled as interference jitter. So the upper bound of interference produced by τ_j when interference jitter exists is:

$$\lceil \frac{\hat{R}_i + R_j - C_j + J_j^R}{T_j} \rceil C_j \tag{5.7}$$

where $R_j - C_j$ is the maximal possible jitter upper bound and R_j is the worst-case latency of τ_j. Theorem 15 below will prove that the network latency upper bound of τ_j can be found by $\hat{R}_{G(\tau_j)}$. The interference jitter phenomenon only happens when a flow in $S(i)$ has indirect higher priority flows. Consider the fact that there are two possible indirect interferences scenarios when the network permits priority shares, either of which can result in interference jitter, the corresponding conditions are $S_j^D \cap S_m^I \neq \emptyset$ or $S_j^{SD} \cap S_m^I \neq \emptyset$, where $\tau_m \in S(i)$ and $\tau_j \in S_m^D$.

As a result, an upper bound of network latency for the composite flow $\hat{\tau}_i$ in the case of interference jitter and release jitter is calculated as follows:

$$\hat{R}_i = \sum_{\forall \tau_j \in hp(i)} \lceil \frac{\hat{R}_i + R_j - C_j + J_j^R}{T_j} \rceil C_j + \hat{C}_i \tag{5.8}$$

The value of $\hat{R}_i$ can be found using the usual iterative technique. The iteration starts with $\hat{R}_i^0 = \hat{C}_i$ and terminates when $\hat{R}_i^{n+1} = \hat{R}_i^n$. By this iterative technique, the maximum latency for $\hat{\tau}_i$ can be calculated ($\hat{R}_i = \hat{R}_i^{n+1} = \hat{R}_i^n$).

Theorem 15. *The network latency evaluated by the composite approach is no less*

than the real maximum network latency experienced by any flow sharing the same priority level G_i:

$$max(R_i) \leq \hat{R}_i \;\; \tau_i \in S(i) \tag{5.9}$$

Proof. We assume the real network latency for any flow sharing the priority G_i can be obtained by some exact approach. The theorem is proved by guaranteeing the network latency calculated based on the composite traffic-flow model is always greater than the real one. The network latency for any flow includes the basic network latency, the interferences from higher priority traffic-flows and the blockings from other same priority traffic-flows. Suppose τ_i is the flow which has the maximum latency in the set $S(i)$, $R_i = max(R_1, R_2, \ldots, R_n)$, next we will prove that the $\hat{R}_i \geq R_i$ is always true.

Theorem 14 shows that no traffic-flow can be blocked more than once by the other flow with the same priority level. This actually implies the fact that the total service requirement from the priority level G_i before τ_i completion is no more than $\sum_{\tau_m \in S(i)} C_m$. Thus, $\hat{C}_i \geq \sum_{\tau_m \in S(i)} C_m$ is true.

The analysis for the composite model supposes that any higher priority packet release will finally delay the composite flow $\hat{\tau}_i$. Let t_1, t_2 indicate time instants after network startup. During any time interval [t_1, t_2] where $t_1 < t_2$, the maximum required service time from all the flows with priority higher than G_i is no more than $\sum_{\forall \tau_j \in hp(i)} \lceil \frac{t_2 - t_1 + R_j - C_j + J_j^R}{T_j} \rceil C_j$. However, a possible situation is that some parallel service opportunities can be supported by the network when some flows do not share any physical links. An example has been shown in Figure 5.3, if τ_2 and τ_5 are released at the same time, and τ_3 and τ_4 are not active, τ_2 and τ_5 will be served in parallel. This parallel chance may accelerate consumption of the required service and finally shortens the whole delay time. Consequently, the network latency evaluated by composted traffic-flows is always greater than, or equal

to the real maximum latency.

□

Theorem 16. *A sufficient condition of schedulability for a set of traffic-flows* $\{\tau_1, \tau_2, \ldots, \tau_n\}$ *with same group priority* G_i *is:* $\hat{R}_i \leq \hat{D}_i$.

Proof. Theorem 15 shows that the composite approach may take extra interference and blocking into account than the real one, hence $\hat{R}_i \geq \max(R_1, R_2, \ldots, R_n)$. On the other hand, if $\hat{R}_i \leq \min(D_1, D_2, \ldots, D_n)$, this induces $\max(R_1, R_2, \ldots, R_n) \leq \min(D_1, D_2, \ldots, D_n)$. So the traffic-flows $\{\tau_1, \tau_2, \ldots, \tau_n\}$ are all schedulable.

□

5.2.3 An Example

We use the case in Figure 5.3 (page 138) to show the analysis approach proposed in this section. The inter-relations between these traffic-flows have been examined in Section 5.1.3. The attributes of the traffic-flows are showed in Table 5.1.

Real-Time Traffic-flow	G	C	T	D	J^R
τ_1	1	1	11	11	0
τ_2	1	2	6	6	0
τ_3	1	3	16	16	0
τ_4	2	3	12	12	0
τ_5	2	1	30	30	0

Table 5.1: Traffic-flows Description

Flows τ_1, τ_2 and τ_3 share the same priority G_1. Utilizing the composite model analysis approach, the composite task $\hat{\tau_1}$ is constructed as following:

$\hat{C_1} = C_1 + C_2 + C_3 = 1 + 2 + 3 = 6$

$hp(i) = S_1^D + S_2^D + S_3^D = \emptyset$

$\hat{D_1} = min(D_1, D_2, D_3) = 6$

$\hat{\tau_1}$ does not suffer any interference from higher priority flow, so $\hat{R_1} = \hat{C_1} = 6$. $\hat{R_1} \leq \hat{D_1}$ and three flows τ_1, τ_2 and τ_3 are all schedulable.

Following the same fashion, we do the analysis for priority level G_2 traffic-flows. the composite task $\hat{\tau_2}$ is constructed as following:

$$\hat{C_2} = C_4 + C_5 = 3 + 1 = 4$$
$$hp(i) = S_4^D + S_5^D = \{\tau_2, \tau_3\}$$
$$\hat{D_2} = min(D_4, D_5) = 12$$

We note that $\hat{\tau_2}$ suffers direct interference from higher priority flows τ_2 and τ_3 and indirect interference from τ_1, which delays τ_3 and may introduce an extra hit to $\hat{\tau_2}$. The interference jitter of τ_3 equals $R_3 - C_3 = 3$. Then we apply Eq.(5.8) and get:

$$\hat{R_2} = \hat{C_2} + \lceil \frac{\hat{R_2}}{T_2} \rceil C_2 + \lceil \frac{\hat{R_2} + R_3 - C_3}{T_3} \rceil C_3$$
$$\hat{R_2}^0 = 4$$
$$\hat{R_2}^1 = 4 + \lceil \frac{4}{6} \rceil 2 + \lceil \frac{4+3}{16} \rceil 3 = 9$$
$$\hat{R_2}^2 = 4 + \lceil \frac{9}{6} \rceil 2 + \lceil \frac{9+3}{16} \rceil 3 = 11$$
$$\hat{R_2}^3 = 4 + \lceil \frac{11}{6} \rceil 2 + \lceil \frac{11+4}{16} \rceil 3 = 11$$

which iteratively stops at $\hat{R_2} = 11$. $\hat{R_2} \leq \hat{D_2}$ and hence τ_4 and τ_5 meet their deadlines.

5.3 Schedulability Analysis with Time Window

In the last section, we have proposed a priority share policy to reduce the resource overhead in priority-based wormhole switching networks. The hard real-time communication guarantees can be achieved with the composite model analysis scheme. The advantage of this approach is it simplicity and provides a sufficient condition to validate hard real-time on-chip communications. But this approach requires that all the traffic-flows must meet the constraint that network latency is no

more than period. This is a strong restriction. The more complex the system, with long communication delays over several hops, the greater the global delays will become.

In this section, another analysis approach is introduced to handle priority share, which is based on “per-priority analysis”, that is, it computes the total time window at each priority level instead of traffic-flow. This new technique completely changes the analysis view point from “per-flow basis" to “per-priority basis". By checking the packet release instance of each traffic-flow at the corresponding priority window, we can verify the timing semantics of real-time traffic-flow with a simple yet efficient mechanism. The ’deadline no more than period’ constraint is successfully removed by this approach with a low computational complexity. In addition, we also find that the composite model analysis is just a special case covered by this new approach.

5.3.1 New Blockings Without the $D \leq T$ Constraint

Blocking severely affects the schedulability of each flow when priority share is supported. To simplify the blocking problem, the composite model analysis imposes a deadline no more than period restriction so that one traffic-flow cannot be blocked by another flow with the same priority more than once; this is termed **single blocking**. Figure 5.5(A) shows such a situation. With this property, all the flows sharing the same priority are transformed into a single scheduling unit and the maximum network latency is addressed by this new model. However, without this enforced constraint, we find two additional blocking phenomena may appear which also need to be considered.

Multiple blocking : A set of traffic-flows share the same priority, one could block another more than once. Figure 5.5(B) shows such a situation. The solid up arrows indicate the packet release instances. The packet’s latency is depicted as a

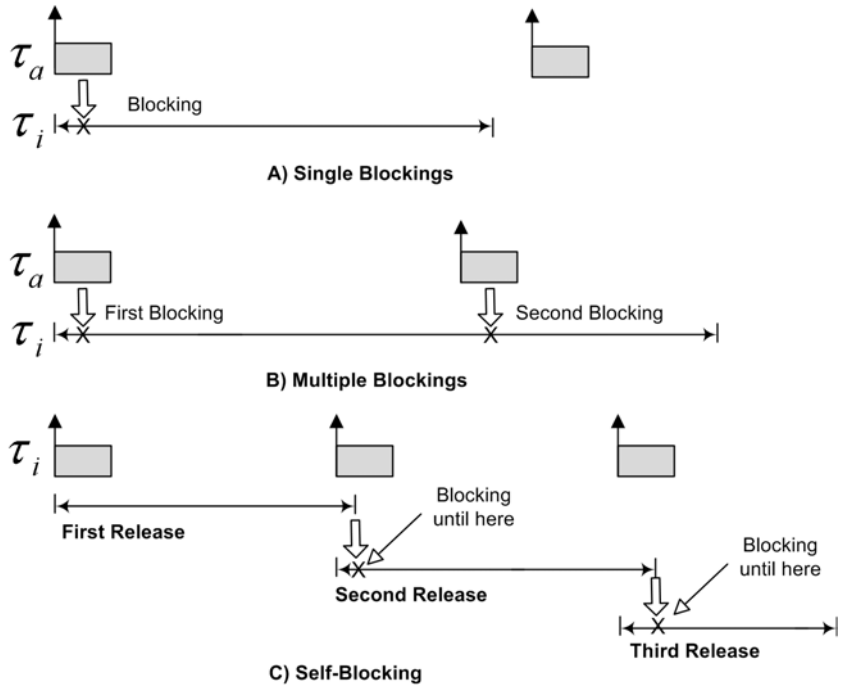


Figure 5.5: The Blocking Problem

horizontal arrow line. Flow τ_a shares the same priority as τ_i; if the transmission latency of τ_i is bigger than the period of τ_a, τ_i could be blocked by τ_a more than once.

Self-blocking : In this situation while the end flits from a previous packet are being delivered, the start flits of the next packet from the same flow are already introduced. Therefore, the possible blocking delay suffered by the new arrival packet comes from not only the other flows with the same priority but also the flow itself, we refer to this as *self-blocking*. Figure 5.5(C) shows such a situation. In this example, the second packet released from τ_i is blocked by the first one until its completion. Similarly, the third packet is also blocked by the second one, etc.

5.3.2 Priority Window Model

The correctness of design and development of practical real-time applications in priority-based wormhole switching relies on efficient schedulability analysis. To overcome the limitation of the composite model analysis, we introduce a new model. The term *priority level-G_i share resource* is defined as all the link resources required by the flows with the same priority G_i. This share resource is modeled as a single competing unit. A *priority window* $W(i)$ is used to define a contiguous time interval during which this priority level-G_i share resource keeps the network busy and serves all the traffic-flows of priority higher than or equal to the priority G_i. The priority window will continue until the time when the share resource becomes idle, ready for the next transmission and yet there is no service requirement from priority level G_i or higher waiting to be transmitted. As shown in Figure 5.3, for the priority level G_2 share resource, the links between router 3 and 6, the corresponding priority window is the contiguous time duration which the shared resource keeps serving all the queueing packets with priority G_2 or higher (G_1 in this case). Figure 5.6 illustrates a priority level-G_2 window. The bold circle denotes the time the packet is received completely at the destination node. In this example, the total window $W(2)$ at priority level-G_2 share resource is the time span from the first release of τ_2 to the completion time of the second instance of τ_5.

For a set of traffic-flow $S(i)$ with the same group priority level G_i, next, we show how to compute the corresponding priority window $W(i)$.

Lemma 2. *The priority level-G_i window $W(i)$ upper bound can be calculated by the following relation:*

$$W(i) = E(i) + I(i) \tag{5.10}$$

where $E(i)$ denotes the summation of service requirements generated by all the

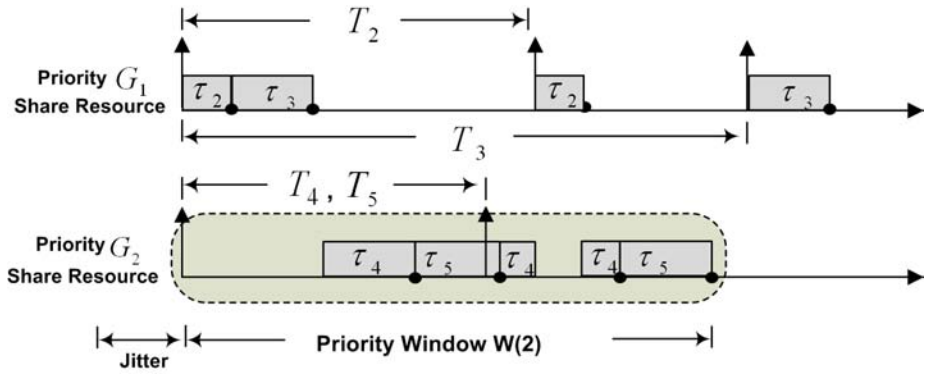


Figure 5.6: Priority Level-G_2 Window

traffic-flows with the priority G_i and $I(i)$ accounts all the interferences from the higher priority traffic-flows which contend the level-G_i share link resource during this window.

Proof. According the definition of priority window, all the arrival packets of priority G_i or higher before the end of the priority window must be transmitted during the window. Besides that, any packet with priority lower than G_i is unable to delay the current window. Therefore, the width of the priority window is equal to the time interval taken to serve the transmission requirements, $E(i)$, made by all the traffic-flows with priority G_i and all interferences, $I(i)$, from higher priority traffic-flows which contend the level-G_i share link resource during this window.

□

The quantities $E(i)$ and $I(i)$ determine the priority window for the level-G_i shared resource. So if we can find an upper bound of $E(i)$ and $I(i)$, the maximum priority window $W(i)$ is then trivially computed. For convenience, here we use the higher priority set $hp(i)$ which has been defined in the last section. Note that, when we explain how to calculate the priority window for G_i priority level, we assume that analysis for all the higher priority $G_1, G_2, \ldots, G_{i-1}$ has already been completed.

Theorem 17. *The maximum priority window $W(i)$ for priority level G_i shared resource can be found by:*

$$W(i) = \sum_{\forall \tau_m \in S(i)} \lceil \frac{W(i) + J_m^R}{T_m} \rceil C_m + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{5.11}$$

Proof. Supposing that we can find an upper bound of the total priority window $W(i)$, the maximum number of instances of a flow τ_m with priority G_i that can delay this window is computed as follows:

$$\lceil \frac{W(i) + J_m^R}{T_m} \rceil \tag{5.12}$$

assuming the worst-case release scenario of τ_m: the first packet release starts J_m^R later than the first arrival, and the subsequent releases are maximum packet size with the maximum release rate of $1/T_m$. Consequently, the service requirement summation from all the flows in $S(i)$ is thus:

$$E(i) = \sum_{\forall \tau_m \in S(i)} \lceil \frac{W(i) + J_m^R}{T_m} \rceil \cdot C_m \tag{5.13}$$

On the other hand, the interferences produced by all the higher priority flows which compete with the level-G_i shared resource also delay the corresponding priority window. The maximum interference analysis has been discussed in the last section. During any time interval, an upper bound of interference produced by any higher priority traffic-flow τ_j when interference jitter exists is:

$$\lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{5.14}$$

where $R_j - C_j$ is the maximal possible jitter upper bound and R_j is the worst-case

latency [1] of τ_j.

Any packet release from the higher priority flows which compete with the priority level-G_i share resource will finally extend the corresponding priority window $W(i)$. Thus, the maximum interferences produced by these flows can be computed as follows:

$$I(i) = \sum_{\tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \tag{5.15}$$

Combining Eq.(5.10), Eq.(5.13) and Eq.(5.15) , an upper bound of priority level-G_i window in case of interference jitter and release jitter is given by:

$$\begin{aligned} W(i) = & \sum_{\forall \tau_m \in S(i)} \lceil \frac{W(i) + J_m^R}{T_m} \rceil \cdot C_m + \\ & \sum_{\forall \tau_j \in hp(i)} \lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \rceil C_j \end{aligned} \tag{5.16}$$

□

The result of $W(i)$ can be solved using the iterative technique. The iteration starts with $W(i)^0 = \sum_{\forall \tau_m \in S(i)} C_m$ and terminates when $W(i)^n$ no longer increases, it has converged. By this iterative technique, the maximum priority window can be calculated ($W(i) = W(i)^{n+1} = W(i)^n$).

5.3.3 Maximum Network Latency

Based on the maximum priority window of G_i, the next step is how to find the maximum network latency for each flow in $S(i)$. For any observed flow τ_i, $\tau_i \in S(i)$, the maximum delay occurs when it is released with all the higher priority flows simultaneously and all the other flows sharing the same priority as τ_i start their services just before τ_i, this will produce the maximum service requirements

[1] Theorems 18, 19 and 20 below will show that the R_j can be found by calculating the corresponding priority window.

on the shared resource. So the earliest starting time of τ_i is the same as the priority level G_i window beginning. To calculate the worst case network latency, we need to find the latest completion time. Due to the multiple blocking and self-blocking problems, if more than one packet instance is released from the same flow during a priority level G_i window, then it is necessary to check these instances in order to find the overall worst-case network latency of the traffic-flow.

Motivated by the observation of the relation between priority window and period, we check the priority window at three different situations as showed in Figure 5.7

- $W(i) \leq min(T_m - J_m^R)$
- $min(T_m - J_m^R) < W(i) \leq T_i - J_i^R$
- $W(i) > T_i - J_i^R$

where τ_m is any flow in $S(i)$.

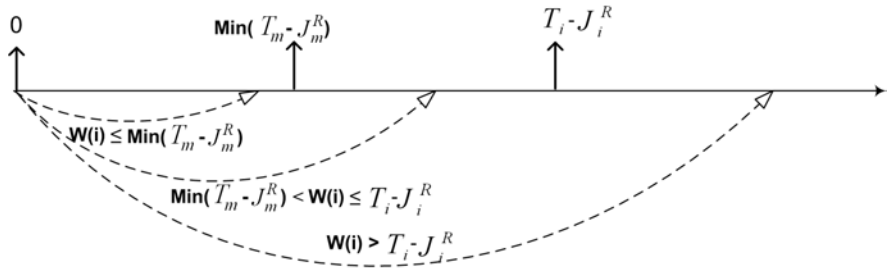


Figure 5.7: Three Possible Relations Between Priority Window and Period

Theorem 18. *The maximum network latency R_i for τ_i is given by:*

$$R_i = W(i) + J_i^R \tag{5.17}$$

when $W(i) \leq min(T_m - J_m^R)$

Proof. If condition $W(i) \leq min(T_m - J_m^R)$ for $\forall \tau_m \in S(i)$ is true, then the priority window ends at or before any repeated release from the flow with priority G_i. This means that no traffic-flow can be blocked by any other flow sharing the same priority more than once. The multiple blocking and self blocking discussed in Section 5.3.1 do not occur. In the worst case, τ_i will be the last flow getting transmission opportunity in this priority window. So the ending of the priority window is the completion time of τ_i's packet instance. The maximum network latency R_i for τ_i is hence given by:

$$R_i = W(i) + J_i^R \tag{5.18}$$

□

We also find that for $\forall \tau_m \in S(i)$, the relation $\lceil \frac{W(i)+J_m^R}{T_m} \rceil = 1$ is always true. Eq.(5.13) in this situation is simplified as $\sum_{\forall \tau_m \in S(i)} C_m$ and the priority window analysis scheme is degraded to the composite analysis presented in the last section. Actually, the composite model analysis is only a specific case in the priority window analysis when $W(i) \leq min(T_m - J_m^R)$.

Theorem 19. *The maximum network latency R_i for τ_i is given by:*

$$R_i = W(i) + J_i^R \tag{5.19}$$

when $min(T_m - J_m^R) \leq W(i) \leq T_i - J_i^R$

Proof. If $min(T_m - J_m^R) < W(i) \leq T_i - J_i^R$ is true, this implies that only the first packet instance of τ_i is served during this window and no self-blocking occurs. But multiple packet instances from any other flow in $S(i)$ may fall into the current window because of $\exists \tau_m \in S(i)$, $\lceil \frac{W(i)+J_m^R}{T_m} \rceil > 1$. These multiple blockings will delay the completion time of the current packet, but the worst-case latency still can

be found by checking the priority window. In this case, only one packet instance is released by τ_i, hence the ending of the priority window is the completion time of τ_i's packet instance. The maximum network latency R_i for τ_i is hence given by:

$$R_i = W(i) + J_i^R \tag{5.20}$$

□

If $W(i) > T_i - J_i^R$, then more than one packet instance of τ_i is generated during a priority level-G_i window. Some successive generated packets from τ_i might be blocked by the previous ones. In this situation, the delay from self-blocking also needs to be taken into account.

Theorem 20. *The maximum network latency R_i for τ_i is given by:*

$$R_i = \max_{q=1,\ldots,\lceil \frac{W(i)+J_i^R}{T_i} \rceil} (w_q(i) - (q-1)T_i + J_i^R) \tag{5.21}$$

where q is the index of packet instance, and $w_i(q)$ is given by:

$$\begin{aligned} w_q(i) = qC_i + & \sum_{\forall \tau_m \in S(i), \tau_m \neq \tau_i} \lceil \frac{w_q(i) + J_m^R}{T_m} \rceil C_m + \\ & \sum_{\forall \tau_j \in hp(i)} \lceil \frac{w_q(i) + R_j - C_j + J_j^R}{T_j} \rceil C_j \end{aligned} \tag{5.22}$$

when $W(i) > T_i - J_i^R$.

Proof. The number of packets that could be released from τ_i before the end of the priority window is given by:

$$\lceil \frac{W(i) + J_i^R}{T_i} \rceil \tag{5.23}$$

To determine the worst-case network latency, we must check all the packet instances during the priority window. Following the general approach used in Chap-

ter 3 for distinct priority analysis, here we give a similar approach to find the worst-case network latency when priority share is supported.

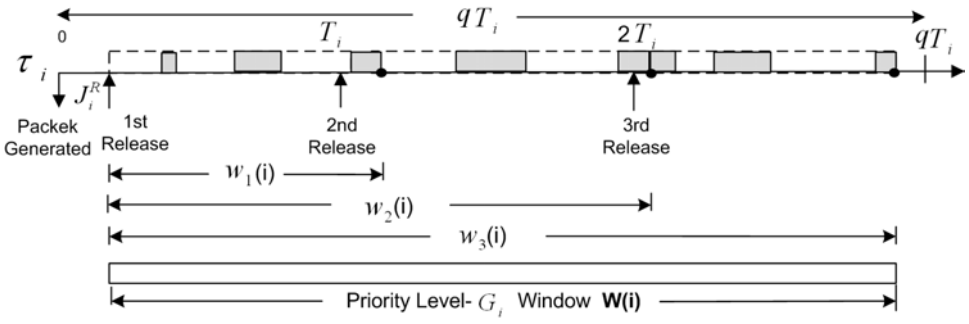


Figure 5.8: Priority Level-G_i Window with Self-blocking

Figure 5.8 shows a self-blocking during a priority level-G_i window. We use the index variable q to denote a packet instance of τ_i. The first packet in the window corresponds to $q = 1$ and the final one is $q = \lceil \frac{W(i)+J_i^R}{T_i} \rceil$. Therefore, the time from the first release of τ_i until achieving the q^{th} transmission is given as a collection of service requirements from all the flows which compete the priority-G_i share resource. We assume a new time phase $w_q(i)$ which denotes the time interval from the beginning of the priority window until the completion of the q^{th} packet transmission. The time phase for the 1^{st}, the 2^{nd} and the 3^{rd} packet in the window are shown in Figure 5.8 as $w_1(i)$, $w_2(i)$ and $w_3(i)$ respectively. The time phase $w_q(i)$ is given by:

$$w_q(i) = qC_i + \sum_{\forall \tau_m \in S(i), \tau_m \neq \tau_i} \lceil \frac{w_q(i) + J_m^R}{T_m} \rceil \cdot C_m + \sum_{\forall \tau_j \in hp(i)} \lceil \frac{w_q(i) + R_j - C_j + J_j^R}{T_j} \rceil C_j \tag{5.24}$$

The variable qC_i accounts for the transmission service time of the first q packet instances of τ_i during the priority window. The final part of the right hand side of

this equation include all the service requirements from priority level G_i or higher flows which fall in this time windows $w_q(i)$. The value of $w_q(i)$ can be found by the similar iteration policy but starting with $w_q(i)^0 = C_i + qT_i$ and ending when $w_q(i)^{n+1} = w_q(i)^n$. The generation time of the q^{th} packet happens at instant $(q-1)T_i$ relative to the start of priority window so the network latency of the q^{th} instance is given by:

$$R_i(q) = w_q(i) - (q-1)T_i + J_i^R \tag{5.25}$$

The maximum network latency can occur at any one of these packet releases during the priority window. We will consecutively analyze each release until τ_i stops blocking itself; which means the packet transmission service finishes within the same period as it is released, $W(i) \leq qT_i - J_i^R$. Thus maximum network latency is given by:

$$R_i = \max_{q=1,\ldots,\lceil \frac{W(i)+J_i^R}{T_i} \rceil} (w_q(i) - (q-1)T_i + J_i^R) \tag{5.26}$$

□

Finally, flow τ_i is schedulable, if and only if $R_i \leq D_i$.

5.3.4 Another Example

Revisiting the example given in Figure 5.3 (page 138). The original case only considered single blocking with the constraint of $D \leq T$. Now we change some attributes of the flows to introduce two new blockings: multiple blocking and self-blocking. We update flow τ_4 with new attributes in Table 5.2 (just decease τ_4's period from 12 to 9) and keep other parameters unchanged.

Flows τ_1, τ_2 and τ_3 share the same priority G_1. Utilizing priority window

Real-Time Traffic-flow	G	C	T	D	J^R
τ_1	1	1	11	11	0
τ_2	1	2	6	6	0
τ_3	1	3	16	16	0
τ_4	2	3	**9**	12	0
τ_5	2	1	30	30	0

Table 5.2: Traffic-flows Description

analysis, first we calculate $W(1)$ according to Eq.(5.11):

$$W(1) = \lceil \frac{W(1)}{T_1} \rceil C_1 + \lceil \frac{W(1)}{T_2} \rceil C_2 + \lceil \frac{W(1)}{T_3} \rceil C_3$$

Utilizing the iterative technique,

$$W(1)^0 = 1 + 2 + 3 = 6$$
$$W(1)^1 = \lceil \frac{6}{11} \rceil 1 + \lceil \frac{6}{6} \rceil 2 + \lceil \frac{6}{16} \rceil 3 = 1 + 2 + 3 = 6$$

The recurrence stops at $W(1)$ = 6 which is less than $min(T_1, T_2, T_3)$. So R_1= R_2 = R_3 = $W(1)$ =6.

Flows τ_4 and τ_5 share the priority level G_2. The maximum window for G_2 not only contains the blocking but also the interference from higher priority flows. In this case, τ_2 and τ_3 contend with the priority level G_2 shared resource and hence contribute direct interference. Besides that, the activity of indirect higher priority flow τ_1 also can introduce some extra interference which is treated as interference jitter. So the window for G_2 is given by:

$$W(2) = \lceil \frac{W(2)}{T_2} \rceil C_2 + \lceil \frac{W(2) + R_3 - C_3}{T_3} \rceil C_3 + \lceil \frac{W(2)}{T_4} \rceil C_4 + \lceil \frac{W(2)}{T_5} \rceil C_5$$
$$W(2)^0 = 10$$
$$W(2)^1 = \lceil \frac{10}{6} \rceil 2 + \lceil \frac{10+3}{16} \rceil 3 + \lceil \frac{10}{9} \rceil 3 + \lceil \frac{10}{30} \rceil 1 = 14$$
$$W(2)^2 = \lceil \frac{14}{6} \rceil 2 + \lceil \frac{14+3}{16} \rceil 3 + \lceil \frac{14}{9} \rceil 3 + \lceil \frac{14}{30} \rceil 1 = 19$$
$$W(2)^3 = \lceil \frac{19}{6} \rceil 2 + \lceil \frac{19+3}{16} \rceil 3 + \lceil \frac{19}{9} \rceil 3 + \lceil \frac{19}{30} \rceil 1 = 24$$

$$W(2)^4 = \lceil \frac{24}{6} \rceil 2 + \lceil \frac{24+3}{16} \rceil 3 + \lceil \frac{24}{9} \rceil 3 + \lceil \frac{24}{30} \rceil 1 = 24$$

which iteratively results in $W(2) = 24$.

For τ_5, $W(2) < T_5$, so only one instance is released during this window. According to Theorem 19, $R_5 = W(2) = 24$.

For τ_4, $\lceil \frac{W(2)}{T_4} \rceil = 3$, so there are three released packet instances during the priority window. We need check all these three packet instances to determine the worst-case network latency. Utilizing Theorem 20, the window phases for the first packet, first two packets and the first three packets are $w_1(2)$, $w_2(2)$ and $w_3(2)$ respectively. Using Eq.(5.22) and Eq.(5.25), we get:

$w_1(2) = 11$ and $R_4(1) = 11 - 0 = 11$

$w_2(2) = 21$ and $R_4(2) = 21 - 9 = 12$

$w_3(2) = 24$ and $R_4(3) = 24 - 9 * 2 = 5$

The maximum network latency is thus $max(R_4(1), R_4(2), R_4(3)) = 12$. All the flows meet their deadlines and the second packet instance experiences the maximum latency in our analysis.

5.4 Group Priority Allocation

Wormhole switching with fixed priority arbitration has been proposed as a possible solution for real-time on-chip communication. However, the current approach emphasizing the independency between the traffic-flows introduces excessive virtual channel overheads, in such a way that resource costs become unaffordable. Mapping more than one flow to a single virtual channel is a possible solution. The relevant blocking analysis and schedulability test have been represented in the previous two sections. However, these approaches assume that the group priorities have already been allocated to each traffic-flow. Consider that n

traffic-flows are loaded on the network and there are only N priorities available, where $N \leq n$, the total number of ways that n flows can be mapped into N priorities is $((n)!)/((N)!(n-N)!)$. This implies that it is impractical to determine the optimal solution by brute-force techniques, except for a very small number of traffic-flows. Hence in this section, we design a greedy priority allocation policy which ensures schedulability with reduced time complexity.

Two important properties for priority-based wormhole switching need to be clear before discussion of priority allocation:

Theorem 21 (see Chapter 4, Theorem 8)**.** *The network latency of a traffic-flow is not dependent on the lower priority traffic-flows.*

Theorem 22 (see Chapter 4, Theorem 9)**.** *The network latency of a traffic-flow is dependent on the higher priority traffic-flows and their relative priority ordering.*

It is obvious that the above two theorems still hold when a priority share policy is considered. Actually, each flow with distinct priority is only a special case for priority share where no flow shares the same group priority. Assuming that a schedulable priority ordering exists for a traffic-flow set under the distinct priority per flow policy, now we give an assignment policy to find a corresponding flow ordering when the network supports priority share.

The intuition for the algorithm is as follow: Starts at the lowest priority, at each group priority G_k, if any traffic-flow τ_i exists that when τ_i is mapped to priority G_k, all the flows which have been assigned group priority G_k or less are still schedulable, τ_i will be assigned priority G_k. If no additional flow mapped to G_k can lead to a schedulable system, the group priority is increased.

Algorithm in Figure 5.9 shows the process of group priority allocation. The algorithm maintains two flow sets. All the flows with the initial priority assignment are inserted into set Γ'. Γ'' includes all the flows which have been allocated a

```
procedure SCHEDULABILITYTEST(G)
 for G_i ← G to G_n
   do { if PassSchedulabilityTest(τ_i)! = True
        then return ( false )
 return ( true )

procedure GETFLOWWITHOUTCHECKED(G, Γ')
 if (Policy 1)
   then return (LowestPriorityFlow(Γ'))
 if (Policy 2)
   then { if (S(G) is EMPTY)
          then return (LowestPriorityFlow(Γ'))
          else { MaxResourceFlow ← NULL
                 MaxResource ← 0
                 for each unchecked τ_j ∈ Γ'
                   do { if (MaxResource < ShareResource(τ_j, S(G))
                        then MaxResourceFlow ← τ_j
                 return (MaxResourceFlow) }

main
 G ← G_n
 while Γ' is not EMPTY
   do { SetAllUnChecked(Γ')
        repeat
          { τ_i ← GetFlowWithoutChecked(G, Γ')
            TryPriorityAssignment(τ_i, G)
            if SCHEDULABILITYTEST(G)
              then { Assigned(τ_i, G)
                     Move(τ_i, Γ'') }
              else FreePriorityAssignment(τ_i)
            SetChecked(τ_i) }
        until AllChecked(Γ')
        G ← G − 1 }
```

Figure 5.9: Share Priority Assignment Algorithm

group priority by our algorithm. At the beginning, set Γ'' is empty. The priority assignment algorithm starts from the lowest group priority. At each priority level, the algorithm firstly needs to find a 'promising' flow τ_i in Γ'. There we introduce two different selection policies to find the right flow in set Γ', the details of which are described below. Then τ_i is assigned with the current group priority G, $G = G_n$ at the beginning, and is checked with the schedulability test. Theorem 22 implies that all the flows with current priority G or lower need to be checked since the network latency may change due to the possible blocking and interference jitter. If τ_i and all the flows in Γ'' are all schedulable, we can assign G to τ_i and move it to Γ''. If τ_i cannot be assigned to G, we do the same check for the next candidate flow in Γ'. During the process of schedulability checking, we use a check status to mark each traffic-flow, namely, *checked* or *unchecked*. This ensures that each flow only can be checked once during a specific priority allocation. We repeat the procedure for all the flows in Γ' to find that any flows can be mapped to G. If none of the flows in Γ' can be assigned to the current group priority G, then G is increased and the algorithm repeats the above processing.

When considering the next candidate flow to allocate group priority, all the flows in Γ' should be checked in turn. To make the priority assignment more efficient, here two greedy selection policies help find the most likely candidate when more than one flow is available in set Γ'.

- The first policy always returns the lowest priority flow in Γ' at each selection process.
- The second policy selects the flow which shares the maximum network resource with all the flows at the current priority level in Γ''. If there is no flow at the current priority level, the algorithm automatically returns the lowest priority one.

Theorem 23. *If a schedulable priority ordering exists under distinct priority per flow policy, Algorithm in Figure 5.9 can find a schedulable priority ordering under priority share.*

Proof. At each group priority assignment process, the priority allocation algorithm in Figure 5.9 ensures that all the flows moved in set Γ'' are schedulable. In addition, we find that all the flows in set Γ' still remain schedulable, because the interference on them has decreased or at least unchanged since some higher priority flows have been moved to set Γ''. So after each step of group priority assignment, the whole flow set remains schedulable. The priority assignment algorithm will stop when all the flows are allocated group priorities. Thus, the algorithm guarantees that if a schedulable priority ordering exists under distinct priority per flow policy, a corresponding schedulable priority ordering under priority share can be found. □

In the worst case, the number of group priority levels required is equal to the number of traffic-flows which means that no traffic-flows can share the same priority. The greedy algorithm includes inner and outer loops and thus its time complexity is $O(N^2)$.

5.5 Evaluation

This section presents our evaluation results concerning the resource saving improvement of the priority share approach. The traditional distinct priority per traffic-flow approach is selected as the benchmark.

The evaluation utilizes the same platform introduced in Chapter 3 and the generation of basic attributes for each traffic-flow follows the same fashion. The major implementation overhead in supporting priority-based wormhole switching includes the total priority levels and the amount of virtual channels. So we use the

ratio of required resource cost between priority share policy and the distinct priority per traffic-flow policy as a performance metric. We run each evaluation 1000 times; for each chosen number of traffic-flow and utilization, in four steps as following. Firstly, we generated a traffic-flow set under the predefined network load constraints and set sizes (30, 60 and 90 for each point in our evaluation). Secondly, we utilize the branch and bound priority search algorithm proposed in the last section to find the distinct priority ordering for this flow set and calculate the resource costs. Thirdly, for the same parameters of the flow set, we do the resource allocation based on our new schedulability analysis and priority assignment under the priority sharing policy, which tries to reduce the cost without missing any deadline. Lastly, we count the percentage of the resource cost under the two different priority policies.

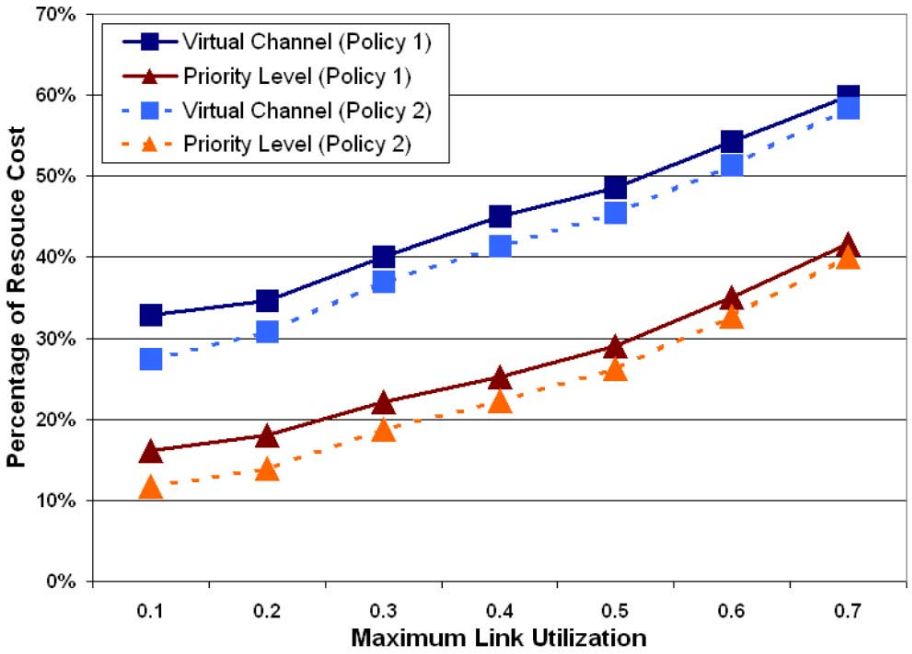


Figure 5.10: The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 30

From the utilization point of view, the investigations are carried out with varied

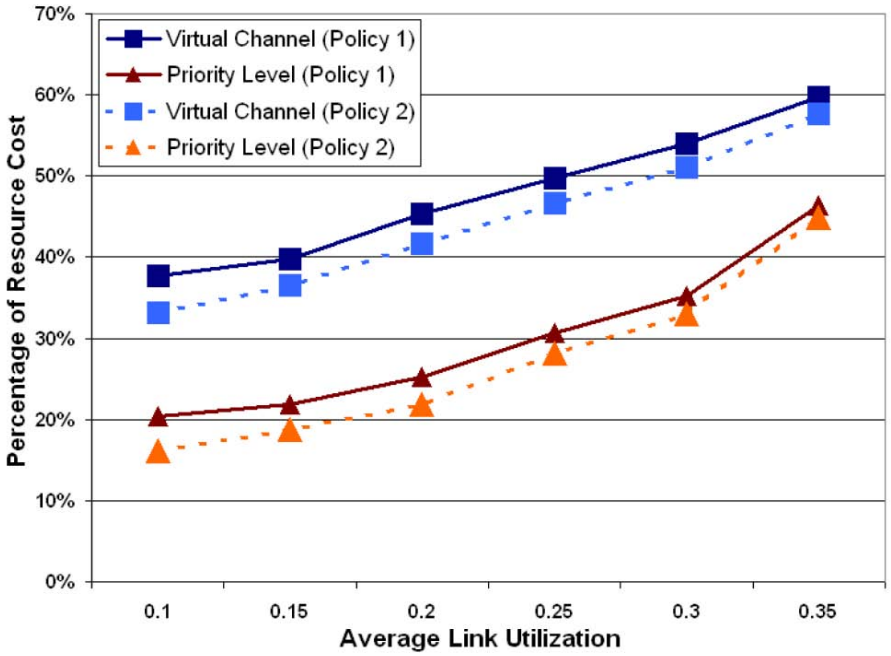


Figure 5.11: The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 30

maximum link utilization and average link utilization schemes. Figures 5.10, 5.12 and 5.14 show the percentage of the resource cost versus the overall maximum link utilization of the network regarding three different sizes of flow set, 30, 60 and 90. Figures 5.11, 5.13 and 5.15 do similar comparison with average link utilization. Each graph has four lines, two solid lines indicate the group priority allocation with the heuristic policy 1 and the dashed lines indicate the policy 2. Two lines with square box show the virtual channels cost and the priority levels are demonstrated as triangle box.

As expected, the priority share policy significantly outperforms the simple approach under all the link load situations. Especially at lower link utilization, the priority share scheme exhibits a remarkable cost saving; consuming only 12.3% of priority levels and 27.4% of virtual channels compared with the original approach when the network maximum link load is 0.1 in Figure 5.10. Similar results can also

be found in Figure 5.11 with the average link load. This is because at the lower link load, each flow can tolerate a bigger blocking delay without missing its deadline so many flows may share the same virtual channels and priorities. With the increase of link load, our priority sharing approach still achieves a good resource saving but needs more resource cost compared with low load. Around 60% virtual channels and 41% priority levels against the benchmark are required to make the flow set schedulable when the network maximum link load reaches 0.7 in Figure 5.10. The reason is that the priority share may reduce the schedulability, at the higher link load, more and more resources are needed to make the flow set meet their deadlines.

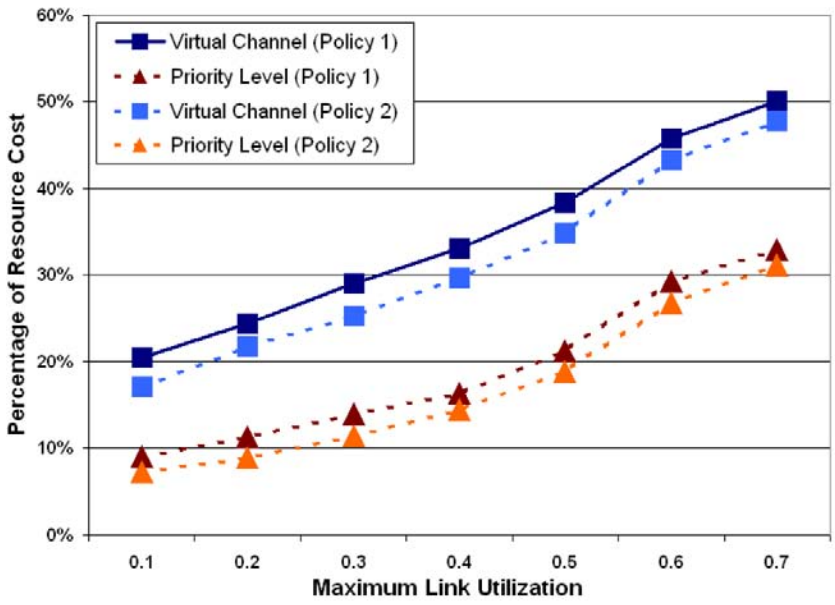


Figure 5.12: The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 60

The comparison between three selected flow sets: 30, 60 and 90 are showed in Figures 5.10 and 5.11 for 30, Figures 5.12 and 5.13 for 60, and Figures 5.14 and 5.15 for 90. It is not difficult to find that with the increase in the flow set size, the difference between the priority share policy and the distinct priority per

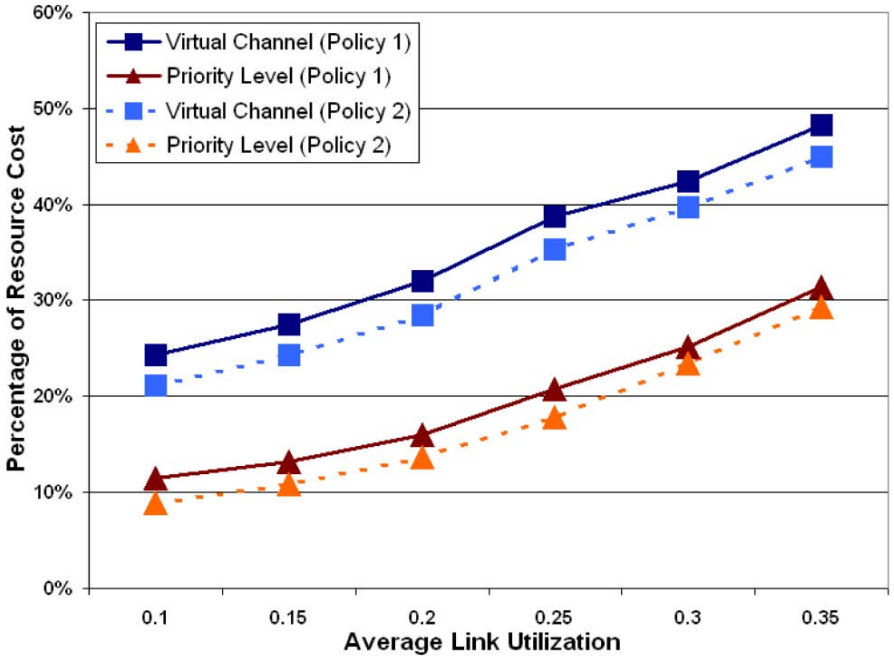


Figure 5.13: The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 60

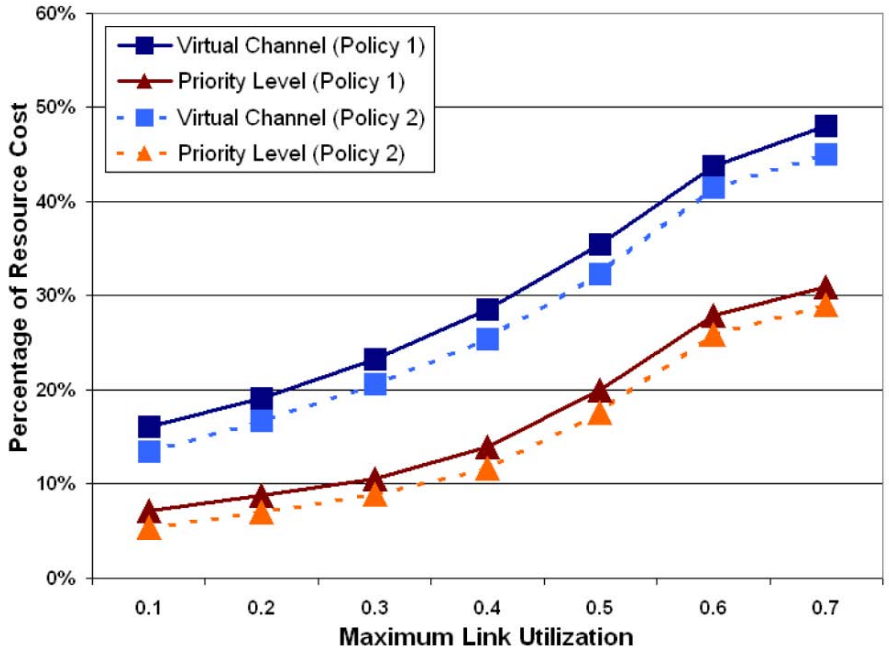


Figure 5.14: The Ratio of Resource Cost Under Varied Maximum Link Utilization With Flow Set 90

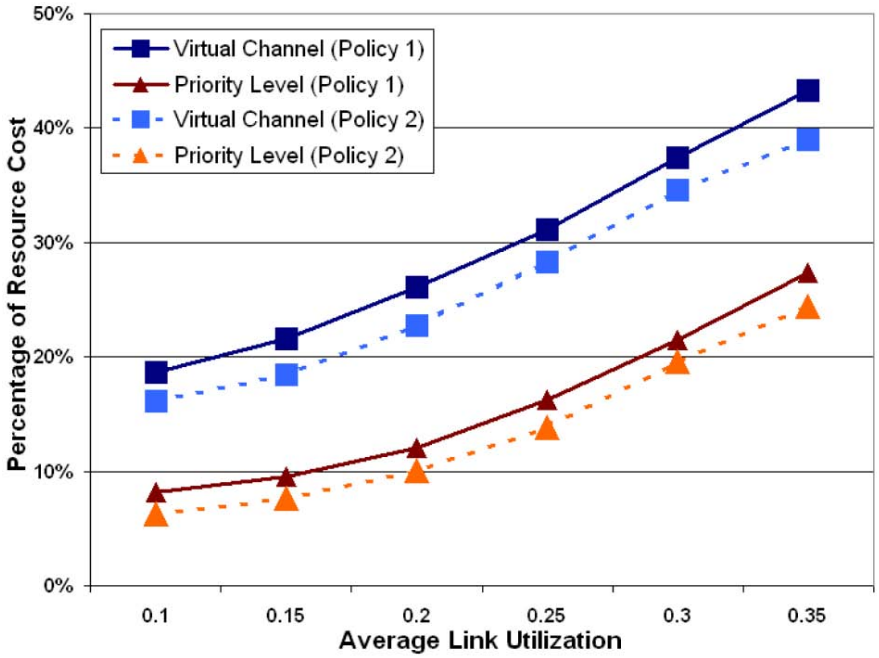


Figure 5.15: The Ratio of Resource Cost Under Varied Average Link Utilization With Flow Set 90

traffic-flow policy becomes progressively larger. Our approach shows its power and achieves better resource saving as the flow set size increases. For example, to make each flow meet its deadline under the average link load 0.35, for 30 flows, the ratios between the priority share and distinct priority for virtual channels and priority levels are 57% and 44%. However, when the size of flow set is increased to 90, the ratios fall to 39% and 24% respectively. This is because when more and more traffic-flows compete for the same physical link, there is more opportunity for multiple flows to share the same resource and hence decrease the overall overhead.

In all cases, the evaluation shows the second selection policy which returns the flow with the maximum shared resource performs better than the first policy.

5.6 Summary

To reduce the resource cost introduced by the distinct priority policy, in this chapter, we proposed a novel priority sharing policy to permit more than one flow to share the same priority level. Considering the extra blocking delay introduced by this priority share, novel analysis approaches were presented in two steps regarding to the state of the traffic-flow deadline.

In the first step we restricted the deadlines to be less than or equal to the relative period, so only single blocking from other flows sharing the same priority were considered. In this state, we introduced composite model schedulability analysis. This approach models all the flows sharing the same priority as a single scheduling entity and the packet transmission is analyzed holistically.

In the second step we relaxed the restriction of having deadlines less than the relative period and analyzed two additional blockings. In this case, we changed the analysis view point from per flow to per priority. By checking the packet release at the corresponding priority window, we can verify the timing semantics of the real-time flow with a simple yet efficient algorithm. In addition, we found that the previous result is just a special case coved by this new approach.

Finally, in this chapter we have considered a priority assignment scheme for the priority share model. The evaluation results show that, on average, the number of virtual channels and priorities can be reduced by 50% and 70% respectively.

Chapter 6

Simulation

To better validate our analysis technique, in this chapter, we compare the theoretic results with those obtained from the simulation of an actual NoC platform under the communication load imposed by a realistic application. The chosen application is part of an autonomous vehicle [54], as it includes both large volume data flows as well as control loops with short messages. This kind of application normally requires a guaranteed communication service with bounded latency. By simulation, we are interested to know how realistic are the worst-case latencies calculated using scheduling analysis against the simulation results. Of course, the simulation results may themselves not be an accurate reflection of the actual system's behaviors.

6.1 Application Model

The autonomous vehicle model we consider comprises a navigation and a vehicle control system. The navigation part is designed to recognize an unknown space by populating a database of obstacles, obtained through stereo photogrammetry and ultrasonic sensors. Also included in this example is the direction and

stability control for the vehicle. Figure 6.1 illustrates a basic structure of this application model.

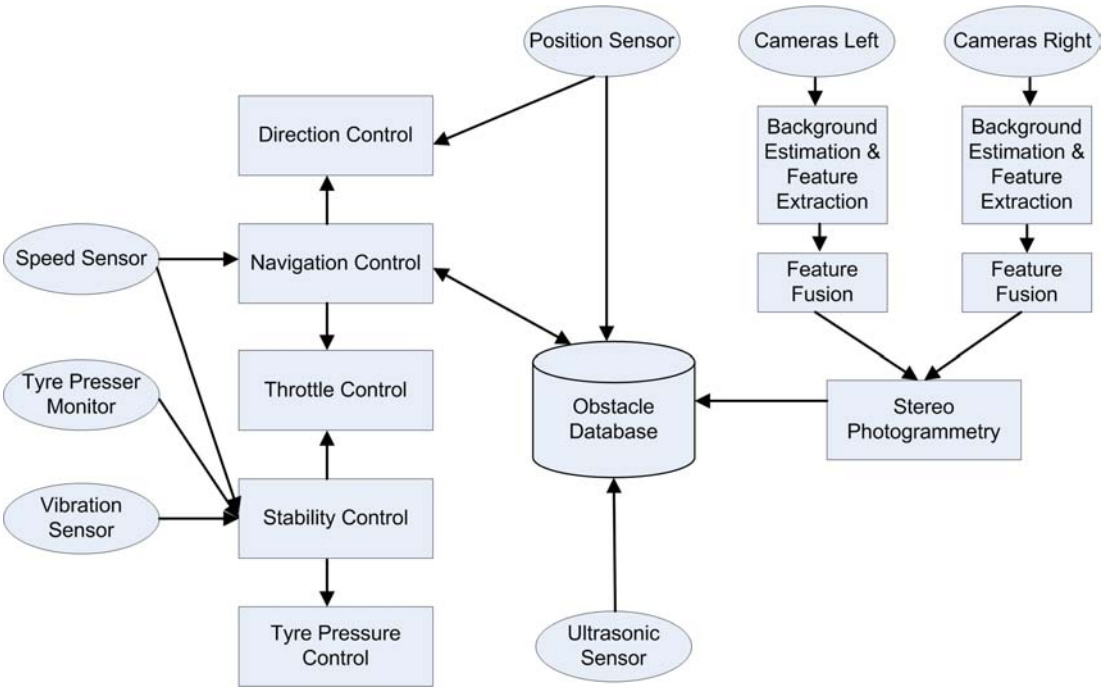


Figure 6.1: Automotive Vehicle Navigation and Stability Control System Model

Two cameras play key roles to gather the obstacle information of the unknown space. Each of them feeds the system with 25 frames/second, each frame with 640x480 resolution and 8 bit colour representation. Each frame is partitioned in 4 quadrants which are loaded from the camera's frame buffer concurrently. Each quadrant of the frame is processed for background estimation and feature extraction, then the features of all quadrants of a frame are merged. Next, a stereo photogrammetry routine analyzes the features from both cameras and estimates the distance from the vehicle to the obstacle, calculates the obstacle absolute location (by using its own position obtained from a positioning sensor) and adds it to the database. Ultrasonic sensors also add obstacle information to the database, but since they have much shorter range, they are mainly used to refine the entries obtained with photogrammetry. The vehicle uses the obstacle database to guide its

navigation process. By analyzing the vehicle's current position, current direction and the results from a query to the obstacle database, the navigation system controls the speed and the direction of the vehicle. The stability system part gathers current tyre pressure and vibration parameters to adjust the speed and maintains the cruise attitude. This is done according to the information from the vibration and tyre pressure sensors. The vehicle can therefore better adapt to the surface on which it is moving and minimize vibrations.

Figure 6.2 shows the function of each task that composes the application in detail. Figure 6.3 shows the traffic flows that these tasks use to communicate (activation periods shown in seconds, payload in flits with 32 bits).

6.2 Simulation Platform and Application Mapping

A flit-accurate NoC platform simulation is chosen for the application which reuses part of the NoC component library presented in [38]. Based on this platform, the strategy to evaluate the communication latency for a particular application mapped onto a NoC platform has been presented in [54], which uses actors that are mapped to producers and consumers plugged to the NoC simulation model. However, the major limitation of this simulation platform is that it does not support priority. All the packets are treated equally and a centralized round-robin arbitration grants access to incoming packets in each router. To solve this problem, Dr. Leandro Soares Indrusiak created a set of new components including multiple virtual channels, flit level priority arbitration, credit-based flow control and gave full support for priority based wormhole switching [75] [1]

The simulation platform is designed as a 4x4 mesh topology with X-Y routing as showed in Figure 6.4. The topology contains 16 processor nodes and each

[1] Subject to the progress of current project, our simulator only supports distinct priority per flow; priority share will be implemented as part of future work.

Tasks	Description
TPMS	Tyre pressure monitoring system
VIBS	Vibration sensor
SPES	Speed sensor
POSI	Position sensor interface
USOS	Ultasonic sensor
FBU1	Frame buffer - Left camera, upper-left quadrant
FBU2	Frame buffer - Left camera, upper-right quadrant
FBU3	Frame buffer - Left camera, lower-left quadrant
FBU4	Frame buffer - Left camera, lower-right quadrant
FBU5	Frame buffer - Right camera, upper-left quadrant
FBU6	Frame buffer - Right camera, upper-right quadrant
FBU7	Frame buffer - Right camera, lower-left quadrant
FBU8	Frame buffer - Right camera, lower-right quadrant
STAC	Stability control
TPRC	Tyre pressure control
DIRC	Direction control
NAVC	Navigation control
OBDB	Obstacle database
BFE1	Background estimation and feature extraction 1
BFE2	Background estimation and feature extraction 2
BFE3	Background estimation and feature extraction 3
BFE4	Background estimation and feature extraction 4
BFE5	Background estimation and feature extraction 5
BFE6	Background estimation and feature extraction 6
BFE7	Background estimation and feature extraction 7
BFE8	Background estimation and feature extraction 8
FDF1	Feature data fusion 1
FDF2	Feature data fusion 2
STPH	Stereo photogrammetry
THRC	Throttle control
VOD1	Visual odometry 1
VOD2	Visual odometry 2
OBMG	Obstacle database manager

Figure 6.2: Application Tasks

node is placed on a separated network router. The operation frequency of the NoC simulator is 100 MHz. So the throughput for each router port equals 3.2 Gbit/s.

In order to perform a validation of the application, in other words, to simulate each inter-task communication of the application model on a platform, it is necessary to map the tasks of the application model onto the chosen NoC platform. Generally, the mapping implementation is a crucial step in the SoC design and has critical influence on the communication performance. But in our case, the

Flow	Source	Dest	Flits	Period (s)	Deadline (s)
1	POSI	NAVC	1024	0.5	0.5
2	NAVC	OBDB	2048	0.5	0.5
3	OBDB	NAVC	16384	0.5	0.5
4	OBDB	OBMG	16384	0.5	0.5
5	NAVC	DIRC	512	0.1	0.1
6	SPES	NAVC	512	0.1	0.1
7	NAVC	THRC	1024	0.1	0.1
8	FBU3	VOD1	38400	0.04	0.04
9	FBU8	VOD2	38400	0.04	0.04
10	VOD1	NAVC	512	0.04	0.04
11	VOD2	NAVC	512	0.04	0.04
12	FBU1	BFE1	38400	0.04	0.04
13	FBU2	BFE2	38400	0.04	0.04
14	FBU3	BFE3	38400	0.04	0.04
15	FBU4	BFE4	38400	0.04	0.04
16	FBU5	BFE5	38400	0.04	0.04
17	FBU6	BFE6	38400	0.04	0.04
18	FBU7	BFE7	38400	0.04	0.04
19	FBU8	BFE8	38400	0.04	0.04
20	BFE1	FDF1	2048	0.04	0.04
21	BFE2	FDF1	2048	0.04	0.04
22	BFE3	FDF1	2048	0.04	0.04
23	BFE4	FDF1	2048	0.04	0.04
24	BFE5	FDF2	2048	0.04	0.04
25	BFE6	FDF2	2048	0.04	0.04
26	BFE7	FDF2	2048	0.04	0.04
27	BFE8	FDF2	2048	0.04	0.04
28	FDF1	STPH	8192	0.04	0.04
29	FDF2	STPH	8192	0.04	0.04
30	STPH	OBMG	4096	0.04	0.04
31	POSI	OBMG	1024	0.5	0.5
32	USOS	OBMG	1024	0.1	0.1
33	OBMG	OBDB	4096	1	1
34	TPMS	STAC	2048	0.5	0.5
35	VIBS	STAC	512	0.1	0.1
36	STAC	TPRC	2048	1	1
37	SPES	STAC	1024	0.1	0.1
38	STAC	THRC	1024	0.1	0.1

Figure 6.3: Inter-Task Traffic Flows

mapping heuristic is irrelevant to the approach presented in this thesis, which aims to verify if, for a given mapping, the timing constraints imposed on all inter-task communications are always met. Even so, the manual mapping has been carefully chosen to make sure it can illustrate certain properties of a practical network

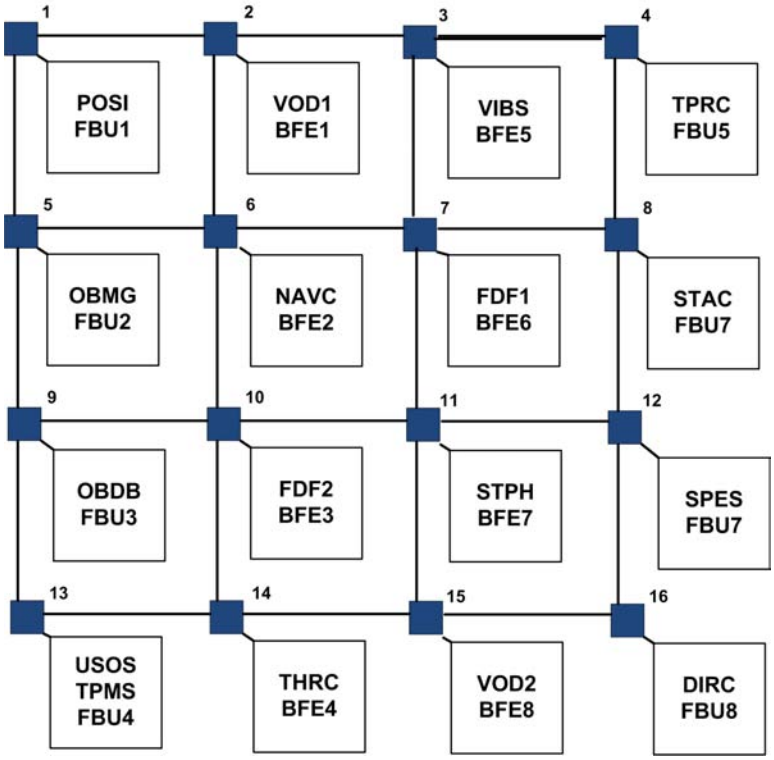


Figure 6.4: Mapping of the Application Tasks onto Processors Interconnected by a 4x4 NoC Mesh

system. Figure 6.4 gives a chosen mapping relationship between the tasks and processors. Two basic rules are applied during the mapping procedure: first, all the processors must keep the basic load balanced; the maximum number of tasks mapped to a processor does not exceed three and the minimum is not less than two. Second, the mapping should provide convenient and flexible interaction with the outside of the system (on board or off board). As a result, all the monitoring or sensor systems, e.g. speed sensor, position sensor or ultrasonic sensor, are located at the edge of the NoC platform. Besides that, since the computational cost for obstacle recognition is very heavy, the whole recognition process is divided into many largely independent parts including pre-processing, feature fusion, distance calculation and placed on the different part of a NoC.

In the simulation, we are not interested in the content that is being transmitted,

thus, the autonomous vehicle model will be implemented at the abstraction level of the traffic-flow. The processor nodes do not perform computations practically, but will periodically output data as if they were. The data information will be packaged at the flits level with the size according to the parameter showed in Figure 6.3. For our simulation, we further assume that there will be 4 clock cycles delay for a packet when it passes through a router. So the total network latency for each packet without contention can be calculated easily from the maximum packet size and the number of hops between source and destination.

6.3 Theoretic Analysis

In order to estimate the network latency we need to associate a priority with each traffic-flow. A priority scheme must be decided beforehand. We have discussed in Chapter 4, when the network utilization is not high, all the priority assignment schemes achieve the same good performance. So the most common priority scheme: Rate Monotonic (RM) is chosen for priority ordering. The RM algorithm gives a flow a priority according to its period. The flow with the shortest period will get the highest priority.

For easy of explanation, each router in the network has its unique identity (ID) marked as number (from 1 to 16) in Figure 6.4. And the link between the routers can be indicated as the vector from X -> Y, where X and Y are the IDs of source and destination routers. Since the contention between the traffic-flows is the most important issue in latency analysis, all the physical link competing relationships under the given mapping are showed in Figure 6.5.

After addressing the priority ordering and traffic-flow competing relationship, we could use the proposed analysis approach in Chapter 3 to calculate the maximum basic network latency (calculated from Eq.(2.10)) and the worst-case net-

Links	Competing Traffic-Flows
Link 1 -> 2	1, 12
Link 2 -> 3	20,
Link 3 -> 4	16, 35,
Link 5 -> 6	13,
Link 6 -> 7	5, 21,
Link 7 -> 8	5, 17,
Link 9 -> 10	3, 8, 14,
Link 10 -> 11	22, 29,
Link 11 -> 12	18,
Link 13 -> 14	15, 34,
Link 14 -> 15	23, 34
Link 15 -> 16	19, 34,
Link 1 -> 5	31,
Link 5 -> 9	2, 33
Link 9 -> 13	
Link 2 -> 6	1, 10, 24,
Link 6 -> 10	7, 24, 25, 38,
Link 10 -> 14	7, 38,
Link 3 -> 7	20,
Link 7 -> 11	28,
Link 11 -> 15	
Link 4 -> 8	35,
Link 8 -> 12	5,
Link 12 -> 16	5,
Link 2 -> 1	
Link 3 -> 2	24,
Link 4 -> 3	
Link 6 -> 5	2,
Link 7 -> 6	25, 38,
Link 8 -> 7	38,
Link 10 -> 9	30,
Link 11 -> 10	6, 26, 30
Link 12 -> 11	6,
Link 14 -> 13	
Link 15 -> 14	11, 27,
Link 16 -> 15	9,
Link 5 -> 1	
Link 9 -> 5	4, 30, 32,
Link 13 -> 9	32,
Link 6 -> 2	8,
Link 10 -> 6	3, 6, 8, 11,
Link 14 -> 10	11, 27,
Link 7 -> 3	
Link 11 -> 7	22, 23,
Link 15 -> 11	23,
Link 8 -> 4	36,
Link 12 -> 8	34, 37,
Link 16 -> 12	34,

Figure 6.5: Link Competing Relationship between the Traffic-Flows

work latency (calculated from Eq.(3.1) and Eq.(3.15)). Figure 6.6 presents the communication latency analysis results for the autonomous vehicle application model with the given priority ordering.

Flow	Priority	Network Latency Theorical Analysis (us)	
		Basic	Worst Case
1	31	10.36	814.90
2	32	20.60	56.80
3	33	163.96	1342.38
4	34	163.92	1147.60
5	24	5.32	25.82
6	25	5.28	1229.74
7	26	10.36	461.52
8	1	384.16	384.16
9	2	384.08	384.08
10	3	5.20	5.20
11	4	5.26	394.62
12	5	384.08	768.24
13	6	384.08	394.54
14	7	384.08	768.24
15	8	384.08	384.08
16	9	384.08	384.08
17	10	384.08	384.08
18	11	384.08	384.08
19	12	384.08	768.16
20	13	20.60	409.88
21	14	20.56	425.24
22	15	20.60	445.84
23	16	20.64	466.48
24	17	20.64	409.92
25	18	20.60	425.32
26	19	20.56	445.88
27	20	20.60	471.74
28	21	82.00	486.68
29	22	82.00	568.68
30	23	41.12	61.68
31	35	10.32	620.16
32	27	10.36	435.56
33	37	41.04	445.72
34	36	20.72	472.00
35	28	5.24	25.88
36	38	20.56	415.08
37	29	10.32	404.92
38	30	10.44	830.16

Figure 6.6: Communication Latency Figures Obtained Using Theoretic Analysis (in microseconds)

6.4 Experimental Results

In the simulation, we measure the network latency for each flow that is the time for the packet(s) corresponding to that flow spends on the network. The network

latency consists of the time during which the packets are either routed between routers or waiting for routing in a buffer. For the chosen application mapping and priority ordering, we design three different simulation scenarios:

1. all the tasks are released simultaneously without jitter;
2. all the tasks are released asimultaneously with up to 10% release jitter;
3. all the tasks are released asimultaneously with up to 50% release jitter.

In each scenario, the simulation results of each flow include minimum, maximum and average latencies. When the simulation starts, each traffic-flow is released according to its release scenario, and then each of them is repeated periodically. Each scenario is simulated for a significant time (To simulate 1.002 seconds of the application (38 communication flows, cameras with 25 frames per second, 640x480 resolution, 8 bit colour representation), it took 10 hours, 34 minutes and 38 seconds on the PC machine (Intel Core Duo at 3.0 GHz, 3.23 Gb RAM). However, the simulator uses only a single core of the machine.) The simulation results with the three release scenarios are illustrated in Figure 6.7. For convenience, the last column in the result figure gives the percentage between the maximum latency observed by simulation against the result calculated by the analysis.

In the first scenario, all traffic flows are released simultaneously upon simulation start without any jitter. It was proved by Liu and Layland [51] that such a situation is the worst-case scenario for fixed-priority tasks sharing a single resource, as it will impose the maximum possible load to the system. This condition has been used by a number of research works [8, 35, 53] to obtain the maximum possible latency with a finite amount of simulation time. However, as we have pointed out in Section 3.1.2, this is not always the worst case from the view of hard real-time analysis in NoCs. As a distributed network system, different traffic flows may share different sets of physical links according to their routes and has

Flow	Communication Latency (us)										Percentage (%)
	Theoretic Analysis	Simulation no jitter			Simulation jitter 10%			Simulation jitter 50%			Max / Worst Case
	Worst case	**Max**	Min	Avg	Max	Min	Avg	Max	Min	Avg	
1	814.90	**778.14**	15.48	396.81	10.36	10.36	10.36	10.36	10.36	10.36	95.49
2	56.80	**35.98**	25.93	30.96	20.60	20.60	20.60	20.60	20.60	20.60	63.35
3	1342.38	**941.98**	179.32	560.65	305.11	163.96	169.84	163.96	163.96	163.96	70.17
4	1147.60	**1095.70**	337.74	716.72	202.58	163.92	165.53	163.92	163.92	163.92	95.48
5	25.82	**5.46**	5.32	5.39	5.32	5.32	5.32	5.32	5.32	5.32	21.15
6	1229.74	**399.44**	5.28	202.36	369.68	5.28	18.63	159.25	5.28	7.85	32.48
7	461.52	**394.34**	15.50	204.92	10.38	10.38	10.38	204.25	10.38	13.01	85.44
8	384.16	**384.16**	384.16	384.16	384.16	384.16	384.16	384.16	384.16	384.16	100.00
9	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	384.08	384.08	384.08	100.00
10	5.20	**5.20**	5.20	5.20	5.20	5.20	5.20	5.20	5.20	5.20	100.00
11	394.62	**389.24**	389.24	389.24	385.31	5.28	13.62	331.48	5.28	8.59	98.64
12	768.24	**768.08**	768.08	768.08	768.08	384.08	413.02	768.08	384.08	388.03	99.98
13	394.54	**394.32**	394.32	394.32	394.32	384.08	384.49	389.20	384.08	384.17	99.94
14	768.24	**768.08**	768.08	768.08	768.05	384.08	395.48	768.05	384.08	386.35	99.98
15	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	384.08	384.08	384.08	100.00
16	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	384.08	384.08	384.08	100.00
17	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	384.08	384.08	384.08	100.00
18	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	384.08	384.08	384.08	100.00
19	768.16	**768.08**	768.08	768.08	768.08	384.08	408.99	768.08	384.08	387.26	99.99
20	409.88	404.56	404.56	404.56	**404.60**	20.60	36.89	404.60	20.60	24.64	98.71
21	425.24	**425.04**	425.04	425.04	389.21	20.56	30.59	404.56	20.56	22.99	99.95
22	445.84	**445.52**	445.52	445.52	376.36	20.60	34.43	367.42	20.60	26.13	99.93
23	466.48	**466.00**	466.00	466.00	404.64	20.64	29.49	212.49	20.64	21.41	99.90
24	409.92	25.72	25.72	25.72	**383.68**	20.64	30.13	294.41	20.64	21.56	93.60
25	425.32	41.08	41.08	41.08	**396.14**	20.60	30.75	150.35	20.60	21.13	93.14
26	445.88	61.52	61.52	61.52	**404.56**	20.56	32.18	181.89	20.56	21.10	90.73
27	471.74	82.00	82.00	82.00	**392.69**	20.60	30.53	218.87	20.60	21.52	83.24
28	486.68	**466.00**	466.00	466.00	466.00	82.00	104.31	466.00	82.00	84.14	95.75
29	568.68	**547.92**	547.92	547.92	541.46	82.00	97.43	428.72	82.00	85.14	96.35
30	61.68	**61.58**	61.58	61.58	61.51	41.12	41.35	41.12	41.12	41.12	99.84
31	620.16	51.45	25.85	38.65	**390.27**	10.32	10.32	10.32	10.32	10.32	62.93
32	435.56	**394.36**	10.36	202.36	231.09	10.36	12.52	343.26	10.36	13.24	90.54
33	445.72	**424.98**	424.98	424.98	49.29	41.04	41.73	41.04	41.04	41.04	95.35
34	472.00	**415.13**	35.96	225.55	347.13	20.72	34.32	88.98	20.72	23.69	87.95
35	25.88	**25.63**	5.24	15.44	5.24	5.24	5.24	5.24	5.24	5.24	99.03
36	415.08	**414.74**	414.74	414.74	20.56	20.56	20.56	20.56	20.56	20.56	99.92
37	404.92	**399.38**	15.48	207.43	115.79	10.32	11.37	10.32	10.32	10.32	98.63
38	830.16	**404.58**	20.70	212.64	309.37	10.44	11.34	379.46	10.44	20.82	48.74

Figure 6.7: Communication Latency Figures Obtained Using Analysis and Simulation (in microseconds)

its own critical instant. Therefore, as explained before, finding the critical instant when considering multiple resources is far from trivial, so it is unlikely that simultaneous release will be the critical instant for all the traffic-flows. Actually, the simulation results show that some of the traffic flows had worst-case latencies which are equal (or very close to) the values found by our analysis, but flows such as 24, 25, 26, 27 and 31 had latencies that were between 5-20% against those from

the analysis, while traffics 5,6 and 38 were between 20-50%.

The second and third simulation scenarios aimed to capture the results of different interference patterns, so we introduced a release jitter for each packet of every traffic flow. In the second scenario, the upper bound of the release jitter of a packet was equal to 10% of the period of the respective traffic flow, while in the third scenario the upper bound was equal to 50% of the period. The release jitter of each packet was chosen randomly between zero and the upper bound with a uniform distribution. It can be seen that the results for traffic flows 24, 25, 26, 27 and 31 in the second scenario are much closer to the worst-case values found using our analysis than the first one. The comparison test presented enough evidence that the worst-case scenario for releasing all flows is not necessarily simultaneously.

The fact that the results for some flows are still well below the worst-case values found using analysis is because our simulations couldn't cover every possible release scenario, and thus every possible interference pattern, due to the huge size of the exploring space. For example, the margin between the analysis result (1229.74) and the maximum simulation result (399.44) for flow 6 is more than 67%. To confirm this hypothesis, we empirically chose a particular release scenario that would force flow 6 to have a latency which is closer to the worst-case value found with our analysis. By observation, flow 6 is affected by higher priority flows 8, 10, 11, 13, 18, 26 and 30, of which, 8, 13 and 18 have the bigger basic network latency and thus dominate the worst latency of flow 6. So the new scenario comprehends the releases of all flows but flows 8 and 13 upon simulation start. Flows 8 and 13 were configured to have offsets of 380 and 760 microseconds, respectively. In this circumstance, the latency of flow 6 was found to be 1149.20 microseconds, which corresponds to 93% of the worst-case value found using the proposed analysis. Note, this forced behavior is a valid (i.e. possible) behavior of the system.

In all cases, the results found using simulation were consistently less or equal to the worst-case bound found by our analysis, which further attests to the safety of the analysis. In some cases, we can say that the analysis is tight - the difference between the figures obtained using both methods is zero, and in many cases the difference is very small.

Considering the fact that real-time analysis figures can be calculated very quickly, they can be used early on the design flow to verify if a given platform configuration is able to fulfil communication latency requirements of the hard real-time tasks of an application. By quickly providing latency figures, the analysis can be used to support design space exploration of the platform and to guide the mapping of application tasks onto platform processors. At that stage of the design flow, the fact that the analysis may be conservative (because it may consider packet release scenarios that would not occur in the real system and the analysis approach itself has pessimism, e.g. parallel communication) is less relevant than the fact that it is safe, because it can guarantee that a given mapping and a given platform will always meet the hard real-time constraints of an application.

Lengthy simulation runs can help us to understand the behavior of the application and evaluate its average performance for the chosen platform and mapping, and to verify how conservative the analysis results were for typical execution scenarios. However, it is very difficult, or impossible, to cover all the system states and thus has only limited utility to verify the timing behaviour of the communication among tasks that have hard real-time constraints.

6.5 Summary

This chapter presented a comparison between theoretic analysis and practical simulation for an application running on a NoC. A model of an autonomous ve-

hicle control system is constructed and the network is simulated with this traffic model. The simulation results provided evidence that communication latency calculated using the real-time analysis approach is safe; sometimes, closely matching the figures obtained from simulation. This gives more flexibility in evaluating the timing property of real-time traffic flows in a NoC platform at design time, allowing the impact of communication latency to be properly considered during application development before a system prototype is available. Currently, our NoC simulation platform only supports the distinct priority per traffic-flow scheme. As future work, we will extend the current simulator to support priority share and the other models described in this thesis.

Chapter 7

Conclusion and Future Work

Wormhole switching as a cut-through switching technique, brings many advantages, smaller buffering requirement, lower average latency and greater throughput. All these features mean that wormhole switching has been accepted widely for most on-chip networks. This thesis explores how to utilize wormhole switching to support hard real-time service for on-chip networks. The contributions of this thesis and an overview of some ideas for future work are presented in this chapter.

7.1 Contributions of the Thesis

The main contribution of the thesis is the development of suitable analysis technique for verifying that wormhole switching can satisfy hard timing constraints. We stated the thesis proposition in Chapter 1 as follows:

With the fixed priority based arbitration strategy and worst-case network latency analysis, wormhole switching can be a solution for hard real-time on-chip communication.

This proposition has been achieved by a series of results developed in this thesis.

In Chapter 3, we presented a novel worst-case network latency evaluation approach for priority-based wormhole switching, which successfully emulates wormhole switching as a classic singe processor scheduling model. By evaluating diverse inter-relationships and service attributes among the traffic-flows, this approach can predict the packet transmission latency for a given traffic-flow based on three quantifiable different interferences: direct interference from higher priority traffic-flows, indirect interference from higher priority traffic-flows and self-blocking produced by the previous packet from the same traffic-flow. Then we pointed out that when parallel communication exists in a real network, the general problem of determining the exact schedulability of real-time traffic-flows over the on-chip network is NP-hard. However the results presented in this chapter do form an effective upper bound. In addition, two defects in previous published analysis approaches were addressed. Evaluations have also been included. A clear improvement can be achieved by our approach against the best alternative approach (Kim's approach [44]) for randomly generated traffic-flow sets.

Chapter 4 discussed another general problem in priority-based communication, how to assign priority to each flow so that all the traffic-flows loaded on the network will meet their time constraints. The special property of wormhole switching determines that the current priority assignment techniques cannot work properly. So we introduced a heuristic branch and bound search algorithm to find the possible priority ordering. Differing from the traditional exhaust algorithm which costs exponential complexity, our algorithm can effectively reduce the search space on average. The results of evaluation have showed that our approach outperforms the traditional priority assignment policies significantly.

To reduce the resource cost introduced by the distinct priority policy, in Chap-

ter 5, we proposed a novel priority sharing policy to permit more than one flow to share the same priority level. Considering the extra blocking delay introduced by this priority share, the novel analysis approaches were presented in two steps regarding to the state of the traffic-flow deadline. In the first step we restricted the deadlines to be less than or equal to the relative period, so only single blockings from other flows sharing the same priority were considered. In the second step we relaxed the restriction of having deadlines less than the relative period and showed a new solution to handle two additional forms of blocking. Finally, in this chapter we have considered priority assignment for this priority share model. The evaluation results have showed that, on average, the number of virtual channels and priorities can be reduced by 50% and 70% respectively.

To better validate our analysis technique, in Chapter 6, an autonomous vehicle control application is simulated on an actual NoC platform. By simulation, we are interested to know how realistic are the worst case latencies calculated using scheduling analysis against the simulation results. The simulation results provided evidence that communication latency calculated using the real-time analysis approach is safe, closely matching the figures obtained from simulation.

7.2 Future Work

Although we have addressed several problems in priority-based wormhole switching, there are some issues, which are related to what has been done in this thesis, that are still open to be solved. Solving these problems could form the main part of future work.

First of all, the analysis model used in this thesis supposes a set of independent real-time traffic-flows. In an actual network system, there would be precedence relations between the flows, the schedulability analysis with precedence depen-

dencies is a possible research area in the future.

The schedulability analysis and priority assignment considered in this thesis suppose that a set of tasks have been mapped onto the IPs that are interconnected with a NoC and the communication paths between the tasks have been determined by some routing algorithm. The mapping heuristic is irrelevant to the approach presented here, which simply aims to verify if, for the given mapping, the timing constraints imposed on all inter-task communication are always met. In the future, we could combine task mapping, routing selection, priority assignment and schedulability analysis together to explore system performance, improved throughput and reduced resource cost in a holistic way.

Currently we use fixed priority-based preemption arbitration in each router. Fixed priority arbitration is easy to implement but has a bound on utilization for schedulability. Dynamic priority arbitration (e.g. EDF [51]) is a possible alternative which is an optimal scheduling algorithm on preemptive uniprocessors and brings higher resource utilization. The related router structure and analysis approach could be explored and developed.

Utilizing the iterative technique, the network latency calculation is known to have pseudo-polynomial complexity. With the increase of network size, the schedulability analysis may take a considerable delay. However, in practice, the costumers may only care whether a set of traffic-flows meet the deadlines within a reasonable time frame. So an approximate approach with lower computational complexity is also desired.

In conclusion, priority based wormhole switching could be a solution for hard real-time on-chip communication. This thesis has offered a contribution to this field.

Bibliography

[1] P. Marchal A. Leroy, A. Shickova, F. Catthoor, F. Robert, and D. Verkest. Spatial division multiplexing: a novel approach for guaranteed throughput on nocs. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 81–86, 2005.

[2] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. Spin: A scalable, packet switched, on-chip micro-network. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 20070, Washington, DC, USA, 2003. IEEE Computer Society.

[3] A.Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual channels in networks on chip: implementation and evaluation on hermes NoC. In *SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 178–183, New York, NY, USA, 2005. ACM Press.

[4] D. Andreasson and S.Kumar. On improving best-effort throughput by better utilization of guaranteed throughput channels in an on-chip communication system. In *Proceedings of Norchip Conference, 2004.* , pages 265– 268, 2004.

[5] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. In *YCS164, Dept. Computer Science, University of York*, 1991.

[6] N. C. Audsley. On priority asignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, 2001.

[7] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[8] S. Balakrishnan and F. Ozguner. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Trans. Parallel Distrib. Syst.*, 9(7):664–678, 1998.

[9] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin. An asynchronous noc architecture providing low latency service and its multi-level design framework. In *ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 54–63, Washington, DC, USA, 2005. IEEE Computer Society.

[10] L. Benini and G. D. Micheli. Powering networks on chips: Energy-efficient and reliable interconnect design for socs. In *Proceeding of the International Symposium on System Synthesis*, pages 33–38, 2001.

[11] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, 2002.

[12] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine*, 4(2):18–31, 2004.

[13] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time System Journal*, 30(1-2):129–154, 2005.

[14] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computer Survey*, 38(1):1, 2006.

[15] T. Bjerregaard and J. Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1226–1231, Washington, DC, USA, 2005. IEEE Computer Society.

[16] T. Bjerregaard and J. Sparsø. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 34–43, 2005.

[17] T. Bjerregaard and J. Sparsø. Implementation of guaranteed services in the mango clockless network-on-chip. *IEE Proceedings: Computing and Digital Techniques*, Vol. 153, (4)(217-229), 2006.

[18] T. Bjerregaard and J. Sparso. Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Norchip Conference, 2004. Proceedings*, pages 269–272, Nov. 2004.

[19] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(105-128), 2004.

[20] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3(2):194–205, 1992.

[21] W. J. Dally. Route packets, not wires: On-chip interconnection networks. *Proceedings of the 38th Design Automation Conference (DAC)*, pages 684–689, 2001.

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

[23] J. Dielissen, A. Radulescu, K. Goossens, and E. Rijpkema. Concepts and implementation of the philips network-on-chip, 2003.

[24] J. Duato, A. Robles, F. Silla, and R. Beivide. A comparison of router architectures for virtual cut-through and wormhole switching in a nowenvironment. In *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pages 240–247, Washington, DC, USA, 1999. IEEE Computer Society.

[25] J. Duato and S. Yalamanchili. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, 2002.

[26] T. Hamalainen E. Salminen, A. Kulmala. Survey of network-on-chip proposals. White paper, OCPIP, April 2008. Available online (13 pages).

[27] K. Faraydon, N. Anh, and D. Sujit. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, 2002.

[28] T. Felicijan and S. B. Furber. Quality of service (QoS) for asynchronous on-chip networks. In *Formal Methods for Globally Asynchronous Locally Synchronous Architecture (FMGALS 2003)*, September 2003.

[29] T. Felicijan and S. B. Furber. An asynchronous on-chip network router with quality-of-service (QoS) support. In *Proceedings IEEE International SOC Conference*, pages 274–277, September 2004. ISBN 0780384458.

[30] S. Furber and J. Bainbridge. Future trends in SoC interconnect. In *IEEE International Symposium on VLSI Design, Automation and Test*, pages 183–186, 2005.

[31] O. P. Gangwal, A. Rădulescu, K. Goossens, G. Pestana, and E. Rijpkema. Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip. In Peter van der Stok, editor, *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, volume 3 of *Philips Research Book Series*, chapter 1, pages 1–36. Springer, 2005.

[32] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, 2005.

[33] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 250–256, New York, NY, USA, 2000. ACM Press.

[34] M. Harmanci, N. Pazos, P. Ienne, and Y. Leblebici. Providing QoS to Connection-less Packet-switched NoC by Implementing DiffServ Functionalities. In *International Symposium on System-on-Chip*, 2004.

[35] S. L. Hary and F. Ozguner. Feasibility test for real-time communication using wormhole routing. *IEE Proceedings - Computers and Digital Techniques*, 144(5):273–278, 1997.

[36] J. Henkel, W. Wolf, and S. Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. In *VLSID '04: Proceedings of the 17th International Conference on VLSI Design*, page 845, 2004.

[37] J. Hu and R. Marculescu. Dyad - smart routing for networks-on-chip. *DESIGN AUTOMATION CONFERENCE*, ISSU 41:260–263, 2004.

[38] L. S. Indrusiak, L. Ost, L. Moller, F. Moraes, and M. Glesner. Applying UML interactions and Actor-Oriented simulation to the design space exploration of Network-on-Chip interconnects. In *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pages 491–494, 2008.

[39] A. Jantsch and H. Tenhunen. *Networks on Chip*. Springer, 2003.

[40] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Circuit-switched coherence. In *NOCS '08: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 193–202, Washington, DC, USA, 2008. IEEE Computer Society.

[41] N. Kavaldjiev and G.J.M. Smit. A survey of efficient on-chip communications for SoC. In *4th PROGRESS Symp. on Embedded Systems*, pages 129–140, 2003.

[42] N. Kavaldjiev, G.J.M. Smit, P. T. Wolkotte, and P. G. Jansen. Routing of guaranteed throughput traffic in a network-on-chip. In *CTIT technical report series ; no. 05-42, ISSN 1381-3625*, number 2005-42, 2005. Kav05c Imported from DIES.

[43] N. Kavaldjiev, Gerard J. M. Smith, P. G. Jansen, and P. T. Wolkotte. A virtual channel network-on-chip for GT and BE traffic. In *ISVLSI '06: Proceedings*

of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, page 211, Washington, DC, USA, 2006. IEEE Computer Society.

[44] B. Kim, J. Kim, S. J. Hong, and S. Lee. A real-time communication method for wormhole switching networks. In *ICPP '98: Proceedings of the International Conference on Parallel Processing*, pages 527–534, 1998.

[45] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[46] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J.P. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. *ISVLSI*, 00:117, 2002.

[47] I. Lee, J. Y-T. Leung, and S. Son. *Handbook of Real-Time and Embedded Systems*. CRC Press, 2007.

[48] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

[49] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[50] J.P. Li and M.W. Mutka. Real-time virtual channel flow control. *J. Parallel and Distributed Computing*, 32(1):49–65, 1996.

[51] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[52] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[53] Z. Lu, A. Jantsch, and I. Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 960–964, 2005.

[54] S. Maatta, L. S. Indrusiak, L. Ost, L. Moller, J. Nurmi, M. Glesner, and F. Moraes. Validation of executable application models mapped onto network-on-chip platforms. In *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*, pages 118–125, 2008.

[55] A. Mello and N. L. V. Calazans. Rate-based scheduling policy for qos flows in networks on chip. In *VLSI-SoC*, pages 140–145, 2007.

[56] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Evaluation of current qos mechanisms in networks on chip. In *System-on-Chip, 2006. International Symposium on*, pages 1–4, Nov. 2006.

[57] G. De Micheli and L. Benini. *Networks on Chips: Technology and Tools*. Morgan Kauffman, 2006.

[58] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, page 20890, February 2004.

[59] F. Moraes, N. Calazans, A. Mello, L. Muller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1):69–93, 2004.

[60] M.W. Mutka. Using rate monotonic scheduling technology for real-timecommunications in a wormhole network. In *Proceeding of the Second*

Workshop on Parallel and Distributed Real-Time Systems, pages 194–199, 1994.

[61] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[62] U. Y. Ogras, J. Hu, and R. Marculescu. Key research problems in NoC design: a holistic perspective. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 69–74, 2005.

[63] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8):1025–1040, 2005.

[64] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 224–229, New York, NY, USA, 2005. ACM Press.

[65] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 160–169, Washington, DC, USA, 2005. IEEE Computer Society.

[66] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 878–883, 2004.

[67] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 350–355, March 2003.

[68] Edwin Rijpkema, Kees Goossens, and Paul Wielage. A router architecture for networks on silicon. In *Proceedings of Progress 2001, 2nd Workshop on Embedded Systems*, Veldhoven, the Netherlands, October 2001.

[69] S. Sathe, D. Wiklund, and D. Liu. Design of a switching node (router) for on-chip networks. In *ASIC, 2003. Proceedings. 5th International Conference Vol.1*, pages 75– 78, 2003.

[70] Z. Shi and A. Burns. Improvement of schedulability analysis with a priority share policy in on-chipnetworks. In *17th International Conference on Real-Time and Network Systems (RTNS), will appear)*.

[71] Z. Shi and A. Burns. Priority assignment for real-time wormhole communication in on-chip networks. In *Proceeding of the 29th IEEE Real Time System Symposium (RTSS)*, pages 421–430, 2008.

[72] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Proceeding of the 2nd ACM/IEEE International Symposium on Networks-on-Chip(NoCS)*, pages 161–170, 2008.

[73] Z. Shi and A. Burns. Schedulability analysis for fixed priority wormhole switching in on-chip networks. Technical report, 2008.

[74] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 3–12, 2009.

[75] Z. Shi, A. Burns, and L. S. Indrusiak. Schedulability analysis for real time on-chip communication withwormhole switching. *submitted to the International Journal Embedded and Real-Time Communication Systems (IJERTCS)*.

[76] H. Song, B. Kwon, and H. Yoon. Throttle and preempt: A new flow control for real-time communications in wormhole networks. In *ICPP '97: Proceedings of the international Conference on Parallel Processing*, pages 198–202, 1997.

[77] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time System Journal*, 6(2):133–151, 1994.

[78] D. S. Tortosa. *Proteo: The Development of a Practical Network-on-Chip*. PhD thesis, Tampere University of Technology, 2005.

[79] D. Wiklund and D. Liu. Socbus: Switched network on chip for hard real time embedded systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 78.1, Washington, DC, USA, 2003. IEEE Computer Society.

[80] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 155.1, Washington, DC, USA, 2005. IEEE Computer Society.

[81] C. A. Zeferino and A. A. Susin. SoCIN: A parametric and scalable network-on-chip. In *SBCCI '03: Proceedings of the 16th symposium on Integrated circuits and systems design*, page 169, Washington, DC, USA, 2003. IEEE Computer Society.

[82] A. Zuhily and A. Burns. Optimality of (D - J)-monotonic priority assignment. *Information Processing Letters*, 103(1):247–250, 2007.