A Link-Level Communication Analysis for Real-Time NoCs

by

Sina Gholamian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

Abstract

This thesis presents a link-level latency analysis for real-time network-on-chip interconnects that use priority-based wormhole switching. This analysis incorporates both direct and indirect interferences from other traffic flows, and it leverages pipelining and parallel transmission of data across the links. The resulting link-level analysis provides a tighter worst-case upper-bound than existing techniques, which we verify with our analysis and simulation experiments. Our experiments show that on average, link-level analysis reduces the worst-case latency by 28.8%, and improves the number of flows that are schedulable by 13.2% when compared to previous work.

Acknowledgements

This thesis would not have been possible without the valuable advices and the help of several individuals who contributed and extended their valuable assistance in the fulfilment of this thesis.

First of all, I would like to thank my supervisor, Professor Hiren D. Patel, for his guidance, advice, encouragement and his important support throughout my master's program, which led to the submission of this thesis. His great motivation and accurate view on research made a priceless impression on me and I have learned so much from him. I would also like to thank my co-supervisor, Professor Sebastian Fischmeister, for his great support in my research.

In addition, thanks to all the rest of the colleagues in the Embedded Systems and Computer Architecture Group for giving me unforgettable help and for sharing with me their precious and inspiring experiences.

Finally, I would like to express my sincere gratitude to my parents, Ahmad Gholamian and Foroogh Mehdipoor, and also my sisters, Sara and Samira, for their permanent love and irreplaceable support throughout my life.

Dedication

This thesis is dedicated to my parents, Ahmad Gholamian and Foroogh Mehdipoor, who offered me unconditional love and support throughout my life. It is also dedicated to my sisters, Sara and Samira, who always supported me throughout my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern chip-multiprocessors (CMPs) connect a large number of embedded processing elements using a network-on-chip (NoC) communication interconnect. Unlike traditional bus-based interconnects, NoCs offer an extensible, and scalable interconnect solution [8]. While the use of NoCs is becoming widespread in general purpose computing, its adoption for hard real-time systems has been cautious. This is because of the need to provide provable guarantees that the hard real-time software always meets its timing requirements. However, traditional NoCs focus on improving the average-case performance of the interconnect through dynamic routing algorithms, and flow control policies. This makes analyzing the worst-case latencies of the communication difficult and pessimistic. Therefore, the deployment and worst-case latency analysis of hard real-time software on NoC-based CMPs remains a prominent challenge, and an impediment to adopting CMPs for hard real-time systems.

To address this challenge, researchers propose the use of resource reservation, and run-time arbitration as potential solutions. Resource reservation allocates resources before the start of the communication to ensure that there is no contention between any two packets for a resource. An example of resource reservation approach is the time-division multiplexing (TDM) [25, 19, 24]. This requires reserving resources for the flows, which prevents other flows from reusing and sharing these resources. Furthermore, low latency flows in TDM are tightly coupled with the bandwidth. This happens when a portion of the allotted slot is wasted resulting in over-allocation of bandwidth, which results in under-utilization of the network resources. Similarly, Kandlur et al. [15] present a connection-oriented packet-switched approach for multicomputers with point-to-point interconnect network. Their approach uses message scheduling to enforce certain arrival orders of the messages, which results in resource reservation issues similar to TDM.

Run-time arbitration, on the other hand, uses routers that arbitrate access to links at run-time. Hence, contention is expected, and the proper analysis accounts for possible contention to produce the worst-case latencies. Some of the approaches that use run-time arbitration are the following: Bjerregaard et al.'s MANGO NoC [4], Kavaldjiev et al.'s round-robin arbiter [18], and priority-aware arbiters by Shi and Burns [33, 34]. Even though the work by Bjerregaard et al., and Kavaldjiev et al. use run-time arbitration, they suffer from similar issues as TDM. That is, the low latency flows need to be over-allocated resulting in a tight coupling between the latency and the bandwidth.

Consequently, Shi and Burns [33] propose a wormhole switching policy with priority-based arbiters that allow higher priority flows to preempt lower priority flows. Wormhole switching promotes reduced communication latencies, smaller buffers, and a simpler implementation. Moreover, this approach overcomes the tight coupling of latency and bandwidth that by TDM and TDM-like approaches suffer from, and it allows for a variety of traffic flow types with its use of priorities. Shi and Burns [33, 34] assume a given mapping of source and destination nodes of the flows, and the routes the flows take. They pioneer an elegant flow-level analysis (FLA) that determines the worst-case communication latencies by incorporating direct and indirect interferences of the flows. They use FLA to ensure communication flow schedulability. However, FLA assumes that the flows are indivisible units of communication. As a consequence, FLA does not incorporate the effects of pipelining and parallel transmission of data in the network. This thesis aims to incorporate these characteristics in an analytical method to provide tighter estimates than FLA.

Neglecting the parallel and pipelining effects in data transmission through NoCs result into loose communication latency upper-bound. More importantly, since of the loose latency upper-bound, the flows assume to be unschedulable; however they can be schedulable with more accurate analysis. These observations give us insights to present a more accurate technique that captures the parallel and pipelining effects in NoCs, and results to lower upper-bound latency and higher schedulability compared to FLA.

In this work, we present a novel link-level latency analysis to determine worst-case latencies of communication on real-time network-on-chip interconnects that use priority-based wormhole switching. The challenges of this analysis are to incorporate the pipelining and parallelism in NoC framework to achieve tighter communication latency. We develop an approach that captures direct and indirect interferences from communication flows as well as pipelining and parallel transmission of data across the links. The results of this approach provide tighter worst-case upper-bounds than existing techniques. Our experiments confirm that link-level analysis provides tight upper-bounds when compared to a flow-based analysis.

1.1 Main Contribution

The main contribution of this work is a link-level analysis (LLA) for a NoC supporting wormhole switching with priority-based arbiters. LLA analyzes the interference traffic flows suffer in a NoC at the link-level by considering the pipelining, and parallel transmission of packets. We show that LLA provides tighter worst-case bounds than FLA through analysis, and simulation experiments. We implement a cycle-accurate simulation model of the NoC using SystemC, and we deploy a set-top application on the NoC.

1.2 Thesis Structure

The remaining chapters of the thesis are organized as follows: Chapter 2 gives the related background of real-time communication for NoCs. Chapter 3 illustrates the system model that we applied. Chapter 4 gives the required theoretical background about flow level communication analysis (FLA) that forms a primary base for our work in the following chapter. Chapter 5 provides our contribution for proposing a higher-granularity link-level analytical method for analysing deployed hard real-time communication flows on a NoC. We call this Link-Level Analysis (LLA). We present our cycle-accurate SystemC simulator in Chapter 6, which helps us perform experiments that compares FLA and LLA. By utilizing this simulator, in Chapter 7 we present a simulation of a dual-channel set-top box case study deployed on a 4 by 4 network-on-chip which shows LLA provides tighter results compared to FLA. Chapter 8 provides experimental comparison of FLA and LLA. Finally, Chapter 9 concludes our contributions and also provides some interesting insights about the future work.

Chapter 2

Background and Related Work

Many electronic devices are used in everyday life ranging from handhold cellphones, laptops and cars to industrial applications such as automotive and robotics industry. These electronic devices utilize the *system-on-chip* (SoC) design methodology, which means that the system designer embeds a section or the entire of computational and/or communication of the system into a single chip[2].

Typically, a SoC consists of two major parts: processing units and communication medium. The processing units, called *intellectual properties* (IPs), are data processor blocks that perform data processing and computation. The communication elements provide a proper substrate for IPs to exchange data and control signals among each other.

As the number of cores integrated into a System-on-Chip (SoC) increases, the role of the interconnection system becomes more and more important. The on-chip communication issues are the limiting factors for performance and power consumption in current and next generation SoCs. Design in the era of ultra-deep submicron (UDSM) silicon is mainly dominated by issues concerning the communication infrastructure. While SoCs consisting of tens of cores were common in the last decade,the next generation of many-core SoCs will contain hundreds or thousands of cores. With the evidence of multi-core systems, as the number of cores residing on the same SoC increases significant, the communication infrastructure also need to change drastically in order to support the new inter-core communication demands. One of the widely used and recognized strategies nowadays is Network-on-Chip (NoC) architectures which represent the most viable solution to cope with scalability issues of future many-cores systems and to meet performance, power and reliability requirements [35].

2.1 Networks on Chip

The network-on-chip (NoC) is a promising solution for complex communication of $SoCs$ and outperforms the traditional buses or a point-to-point approach in many ways [7]. This section provides the required background about NoC paradigm.

2.1.1 NoC Basics

NoC improves the on-chip communication and brings drastic improvements over conventional bus and point-to-point connection. It also improves the scalability of system-on-chips (SoC). Due to NoC multiple degrees of parallelism both in computation and communication, using NoC in real-time systems requires new theoretical frameworks and concepts [6]. The major goal of communication-centric design and NoC paradigm is to achieve greater design productivity and performance. Greater design productivity results to a computer system that has higher performance compared to traditional designs. By handling the increasing pipelining and parallelism which is achievable with utilizing NoC framework, a higher design productivity is the target. A typical NoC consist of the following fundamental components.

- **Network interfaces** implement the interface by which cores (IP blocks or simply the Nodes) connect to the NoC. Their function is to decouple computation (the cores) from communication (the network).
- **Routing nodes** route the data according to chosen protocols. NoCs implement the routing strategies such as XY routing, west first, etc.

- **Links** are physical connections between the nodes that provide raw bandwidth. They may consist of one or more logical or physical channels.

Figure 1 shows a commonly used NoC topology. It represents a 4 $\times$ 4 2D mesh grid, which provides an on-chip communication infrastructure for sixteen cores. All the components as discussed are shown in the figure. The figure consists of sixteen IPs, and routers adjacent to IPs.

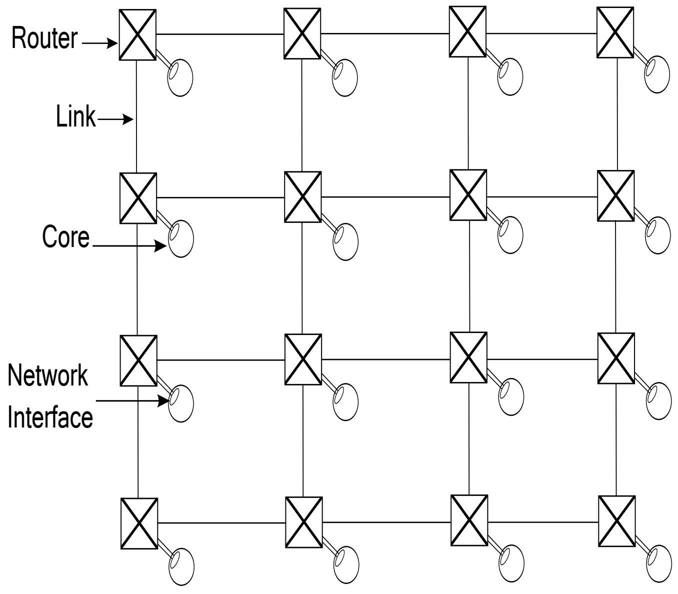

Figure 2.1: A simple NoC example showing the topological aspects.

2.1.2 Classifications

NoCs are classified based on different aspects such as:

- **Topology:** defines nodes logical layout (connections) in the NoC. The most common topologies are 2-D mesh and torus. Both have connections between 4 neighbour nodes but torus has wraparound links connecting the nodes on network edges and mesh does not have these edges. Network adapters implement the interface by which cores (IP blocks or simply the Nodes) connect to the NoC. Their function is to decouple computation (the cores) from communication (the network) [29]. Figure 2.2 shows different topologies for a NoC.

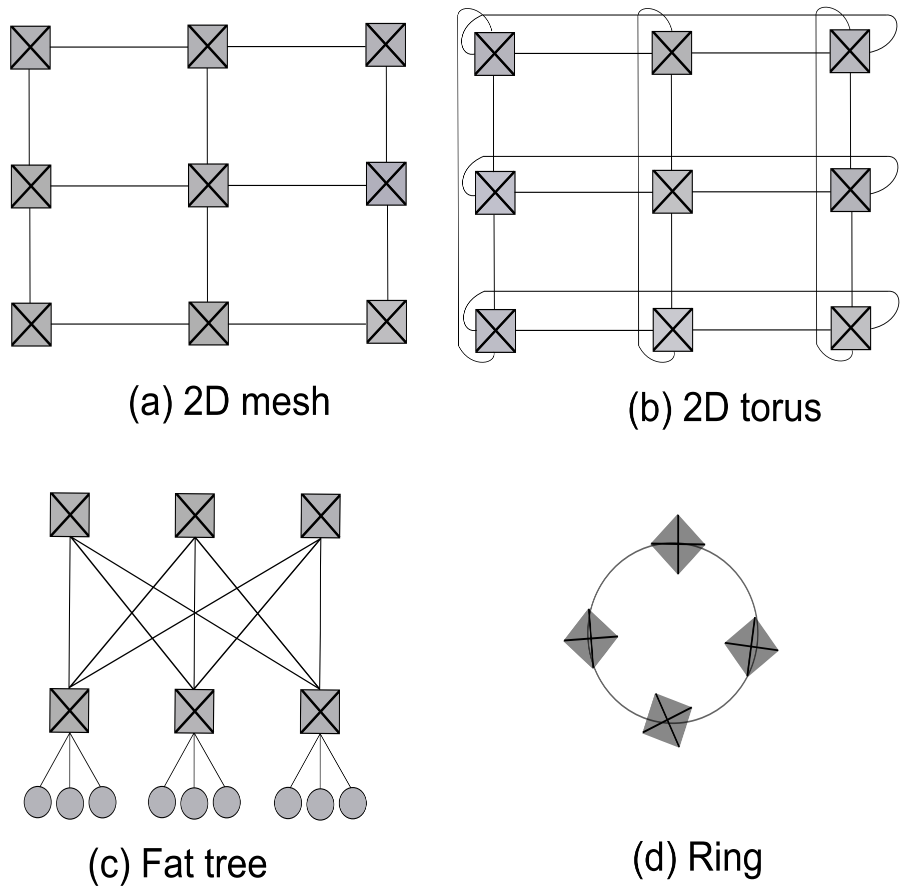

Figure 2.2: A simple NoC example showing different topologies.

- **Routing:** decides the path taken from source to the destination. Routing nodes route the data according to chosen protocols, which can be deterministic or adaptive. In deterministic routing, the path of communication is solely defined by the address of the source and the destination. On the other hand, adaptive routing decides based on the dynamic traffic of the network at the routing time. Therefore, deterministic routing always generates the same path for a given source and destination, but adaptive routing might choose different path based on the current traffic of the network.

The highlighted feature of deterministic routing is its simplicity. Some examples of deterministic routing policy are like XY routing, west first, etc. On the other hand, adaptive routing can prevent congested areas in networks by choosing alternative paths dynamically. This scheme results in higher efficiency compared to deterministic routing when the network is congested. DyAD [14] is an example of adaptive routing.

- **Switching:** decides the timing and control states of communication packets through the routers within the network. A packet is a stream of bits and traverses through the physical links of on-chip network starting from its source to the destination. A packet may contain data or control information. The two most commonly used switching techniques are circuit switching and packet switching.

  Circuit switching is a connection-oriented technique. It establishes the connection before starting the transmission. This method reserves the required buffer and control states through the path of message from the source to the destination. Because of this pre-reservation requirement, this scheme results in low utilization of system resources, such as routers buffer space and control states. Hence, designers prefer to use the other scheme, which is packet switching.

  By comparison, packet switching is a connectionless technique and packets of the same message are routed independently, likely through different paths from the source to the destination. Some examples of packet switching approaches are store-and-forward and virtual cut-through. Virtual cut-through also could be done at a higher level of granularity than packet. Each packet is divided to separate units called flits. Wormhole switching applies this optimization to achieve higher throughput and efficiency compared to virtual cut-through. In the following we explain the wormhole switching technique. We also explain the advantages of this scheme that makes wormhole switching the most preferable switching technique for on-chip communication domain.

  - **Wormhole Switching.** The early approaches for routing and switching in packet switching networks utilized store-and-forward switching scheme [9]. In this technique, each router along the path must receive the entire packet before forwarding it to the next router of the path. Hence, this approach requires larger buffers and results in longer communication latencies. To address these drawbacks, the virtual cut-through scheme is applied. This switching policy forwards the packet as soon as the routing decision has been made. Hence, it does not require waiting to receive the entire packet before forwarding it to the next router, and therefore, this method reduces the communication latency.

    The wormhole switching scheme [26] has been proposed to achieve higher network resource usage efficiency, The method utilizes the cut-through switching technique and requires smaller buffer size and results to tighter communication latency compared to store-and-forward and virtual cut-through approaches. Wormhole technique performs the switching in the flit level instead of the packet level. Flits are smaller data units compared to packets. This results in even reduced buffer size and latency compared the cut-through switching. These highlighted features of wormhole switching make this method preferable to use for real-time communication switching and analysis [34]. Our analysis is based on wormhole switching method. In fact, both FLA and LLA analyses work based on wormhole switching.

- **Flow control:** determines how network resources (channel bandwidth, buffer capacity, and control state) are allocated to a packet traversing the network. It concerns with the allocation and deallocation of channels and buffers to a packet along its route from the source to the destination.

  Flow control may be buffered or bufferless [12], which determines the amount of buffer inside the switching nodes in the network. Links connect the nodes, providing the raw bandwidth. They may consist of one or more logical or physical channels. Bufferless flow control has more latency and lower throughput than buffered flow control. Buffered flow control can be further categorized into credit-based flow control, Ack/Nack flow control, etc. Switching and flow control policies together define the timing of transfers. Circuit-switching forms a path from source to destination prior to transfer by reserving the routers (switches) and links. All data follow that route and path is torn down after the transfer has completed. Packet-switching performs routing per-packet basis. Flow control assures the enough amount of buffers and control states through the communication path.

- **Quality-of-Service (QoS):** refers to the levels of services guaranteed for data transfers. Guarantees are related to timing (such as min. latency, max. latency, max. jitter, etc.), integrity (such as max. error rate, max. packet loss), and packet delivery (in-order or out-of-order). Most of the related research in QoS promises guarantees on performance and average-time latency. Recently, some researches applied QoS for real-time system and worst-case latency [33]. Three methods are employed to give timing guarantees: network dimensioning, circuit-switching, and prioritized packet scheduling [31]. The quality of service can be further categorized into best effort and guaranteed service schemes.

2.2 Real-Time Service Implementation Related Work

NoC platforms result to the tighter communication upper-bound compared to traditional buses or a point-to-point approach in many ways [7]. Several communication schemes have been proposed for NoCs to implement the message delivery policy within the on-chip network. Most of these scheme are inspired from off-chip communication networks. Packet-routing and segmented link communication offers maximum flexibility and scalability [3]. Normally, the network resources, like buffers, physical or logical channels, are shared by the IP blocks on the chip, and local performance is not degraded when scaling. Resource sharing brings higher utilization but also introduces unpredictable network delays due to contention. 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 delays and even missed deadlines, has become the major influence factor of network predictability. Hence, the way to solve the contention problem is a key issue in implementing a guaranteed service in a NoC design [34].

Resource reservation and run-time arbitration are two well known disciplines that address the contention problem.

2.2.1 Resource reservation

Resource reservation approach considers that contention is avoidable by trying to pre-arrange and allocate resources before the start of the communication, so that two packets never access the same resource at the same time. Time division multiplexing (TDM) and circuit switching are two common resource reservation schemes. Goossens et al. [19] and Millberg et al. [25] apply the TDM approach. In this scheme, the whole link transmission capacity is partitioned into fixed time-slots, each of which represents a unit of time when a single traffic-flow can occupy a physical link exclusively for data transmission. A time slot is the minimum time assigned to each router of the network to read or write the data from/to the in/out ports. As a centralized scheduler, TDM monitor assigns time slots to individual applications in a exclusive way. This assures contention free time slot assignment. As a disadvantage, this scheme requires a global notion of time in the network. Besides that, the latency is coupled to bandwidth, preventing low latency from being provided to low rate requirements without over-allocating.

A circuit-switching technique is used in some works [30, 40]. A dedicated connection is constructed between source and destination nodes by reserving a sequence of network resources. The major problem of this scheme is that the resources that have been reserved for a flow can not be used by any other flow which results in under utilized links.

2.2.2 Real-Time Service

The resource reservation policy requires the network to be Multiple IP-cores based design using NoC architecture. This allows multiple tasks to run at the same time. These tasks undertake data processing and exchange information through the underlying communication infrastructure.

Some tasks have very stringent communication service requirements; the correctness relies on not only the communication result but also the completion time bound. A data packet received by a destination too late could be useless. These critical communication tasks are called real-time communications. For a packet transmitted over the network, the communication duration is denoted by the packet network latency. The maximum acceptable duration is defined to be the deadline of the packet. A 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 even under worst case scenarios [34].

Recently, the adaptation of NoCs for hard real-time systems has been cautious at best. This is because of the need to provide provable guarantees that the hard real-time software always meets its timing requirements. However, traditional NoCs focus on improving the average-case performance of the interconnect through dynamic routing algorithms, and flow control policies. This makes analysing the worst-case latencies of the communication difficult and pessimistic. Therefore, the deployment and worst-case latency analysis of hard real-time software on NoC-based CMPs remains a prominent challenge, and an impediment to adopting CMPs for hard real-time systems. However, the need to provide provable guarantees that the hard real-time software always meets its timing requirements is still driving engine for real-time NoCs research. The next section categorizes real-time communication for NoCs.

2.3 Real-Time Communication Categories

NoCs have the capability of allow several applications execute on different cores at the same time. According to this nature of parallelism, these application may want to communicate to each other and do handshaking. For real-time application, the correct functionality of system is not only related to correct communication but also the timeliness of the communication [33]. This type of communication with limited upper-bound is called real-time communication. This type of communication is common in critical applications such as robot control [33], multimedia applications, and system verification which require guaranteed timing delivery and a high degree of predictability.

For real-time systems, time is the most important characteristic that distinguishes them from other type of computing systems. For on-chip communication, the time we concern with is the packet transmission time which is defined as packet network latency. The packet network latency is taken into account from the time which the first bit of a packet is generated from the source until the last bit is received at the destination. The deadline is the boundary point in time which defines the latest time that a packet should arrive to its destination. In other words, assigning deadlines to each communication convert a non real-time communication to real-time communication for on-chip networks. This implies that real-time communication is about satisfying the timing constraints or meeting the deadlines.

According to the importance of real-time behaviour, the on-chip communication could be classified to different categories: hard real-time, soft real-time, and non real-time (best effort) communication [33].

2.3.1 Non Real-Time Communication

In non real-time communication, which is also known as best effort service, no guarantee is provided for communication deadline. In most NoC-related works, best effort communication refers to a kind of communication which just correctness of communication is guaranteed [3]. It is also very common that NoCs support a combination of real-time and non real-time communication at the same time. By doing this, we can have the timing guarantees for very important communication and also at the same time the entire system performance and complexity remain reasonable by integrating with best effort communication.

2.3.2 Soft Real-Time NoCs

Satisfying all of the deadlines, which is the case in hard real-time NoCs, is the matter of performance and enforces high system reliability. However, it may lead to an over-constrained NoC and very complex and expensive to implement. For being more practical, occasional missing deadline can be tolerated and would not cause serious harm and only the system overall performance gets hurt for some degree. This kind of service is defined as soft real-time communication. 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 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 [33]. A classical example for soft real-time communication is multimedia system [23].

2.3.3 Hard Real-Time NoCs

Hard real-time communications are used in applications where meeting the deadlines are absolutely mandatory; otherwise the entire system might fail. This means that the behaviour of hard real-time NoC should be totally predictable. It must be able to satisfy the hard real-time service and meet the deadlines in terms of both correctness of computation and also communication latency. Predictability implies that the implementation policies used in hard real-time NoCs must be analysable mathematically to ensure deadlines satisfaction. The verification of hard real-time model is reasoned by using precise mathematical models to assure each hard real-time service finished within the time limit in all possible scenarios.

2.4 Real-Time Communication Related Work

There are several related research efforts that enable real-time communication over a NoC [19, 24, 34, 4]. To address this challenge, researchers propose the use of resource reservation, and run-time arbitration as potential solutions. Resource reservation allocates resources before the start of the communication to ensure that there is no contention between any two packets for a resource. Run-time arbitration on the other hand, uses routers that arbitrate access to links at run-time. Hence, contention is expected, and the analysis accounts for these contentions to produce the worst-case latencies. Sections 2.5 and 2.6 explain these two schemes.

2.5 Resource Reservation

Bjerregaard and Sparso [4] present a clock-less NoC called message passing asynchronous network-on-chip (MANGO) for guaranteed services. They use an asynchronous latency guarantee arbiter (ALG), which consists of a set of virtual channels, and priority selection and arbitration modules (SPQ) to support real-time communication. MANGO combines wormhole switching with virtual circuits and provides guaranteed service in terms of bandwidth and latency [4]. Figure 2.3 shows the ALG and SPQ components of MANGO scheduling architecture.

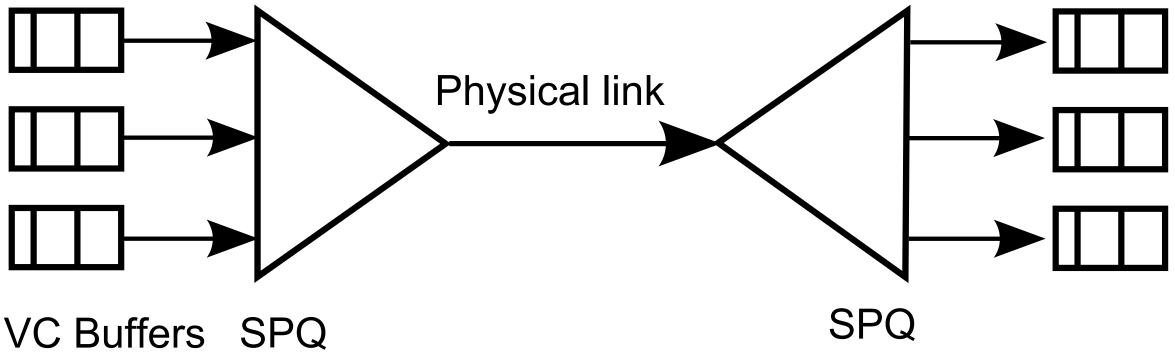

Figure 2.3: MANGO scheduling architecture.

Wiklund and Liu [39], and Wolkotte et al. [40] use circuit switching that requires establishing a connection between source and destination before sending data packets. A circuit with the fixed physical links is created between the source and the destination and all packets of the same communication follow the same path over the circuit.

Millberg et al. [25] and Lu et al.[24] use the TDM approach for communication. TDM divides the link access into equal time slots such that a traffic flow can use the slot time to transfer its own packets. Æthereal [19] also uses TDM to guarantee worst-case bounds on real-time flows. Each output port contains a slot table that multiplexes its access between different flows. The worst-case latency depends on the slot allocation [24], which computes the time for the last flit of a flow to reach its destination. During the connection establishment phase, a time slot per router will be reserved through the communication path from the source to the destination. Since of exclusive time slot assignment by the global monitor, no congestion can happen during the communication time. While this provides worst-case bounds, resources are statically allocated before starting communications. Consequently, a prominent criticism of the TDM approach is its inefficient use of network resources. To address this concern, Æthereal, like MANGO, combines guaranteed service with best-effort to increase its resource utilization [10]. MANGO and Æthereal only support one priority level for their traffic flows.

The problem of the worst-case latency computation on inter-process communication in real-time systems is addressed in [15, 28, 27]. Authors develop an upper bound on the delivery time of messages. The downsides of these methods are the overhead of the establishment and tear down of channels between source and destination pairs, as well as under utilization of the system's resources. They also store packets at intermediate nodes which leads to expensive buffer capacity for storing early arriving packets and queueing packets in order of arrival [15].

2.6 Runtime Arbitration

An alternative option is to use wormhole switching, which increases throughput, and decreases the required buffer capacity in the network. Shi and Burns [33, 34] propose the use of priority-based routers with wormhole switching to support real-time communication. Shi et. al. proposed flow level analysis (FLA), which considers a priority-aware NoC interconnect as its underlying hardware architecture. The analysis incorporates direct and indirect interferences caused by higher priority tasks transmitting data on the links. This approach supports multiple priorities, and their routers ensure that higher priority flows can preempt lower priority flows and computes tighter worst-case bounds compared to viewing the communication platform as a unique resource.

2.7 Summary

This chapter presented the required background of research in real-time communication analysis of NoCs which falls into either a time-division multiple access (TDMA) method or a priority based method. TDMA methods divide the link bandwidth into multiple time slots. Each of these time slots is then dedicated to a particular communication task for transmitting data. Depending on the requirements of the communication tasks, different number of slots can be allotted to the tasks. Examples of TDMA NoCs include those of AEthereal [19], and Nostrum [25]. While real-time analysis of TDMA is straightforward, resources are statically allocated resulting in an inefficient use of network resources.

The second alternative is priority aware communication analysis. Shi and Burns [34] present a priority-based method with the assumption that routers implement wormhole switching with fixed priority preemption. This method allows higher priority communication tasks to preempt the transmission of data of lower priority tasks for better utilization of the network resources. We refer to this approach as the flow-level analysis (FLA). The real-time analysis for FLA requires a detailed study of the interferences communication tasks may suffer during transmission. Our work in this paper extends FLA with considering the interferences in the link-level granularity.

In the next chapter, we explain the details of system model assumptions that FLA and LLA analyses consider from on-chip network to provide us the safe communication upper-bound. Later, we explain the FLA analysis and present an illustrative example that why FLA results into overestimation in communication latency.

Chapter 3

Description of System model

3.1 Network Model and Architecture

Our analysis pertains to NoCs that employ wormhole switching with priority-aware routers. The objective of priority-aware routers is to provide differentiated quality of service for flow with different priorities. Applying priority-aware routers makes the analysis capable to deterministically calculate the communication latency upper-bound. Priority-aware routers have multiple virtual channels (VCs) with each VC designated a distinct priority, and a deterministic routing algorithm. These priorities are used to preempt the routing of packets in lower priority VCs by packets in higher priority VCs. Each node in the NoC consists of a processing element (PE), and a router. Figure 3.1 presents the internals of a router. The router architecture we employ was originally proposed by Shi and Burns [33, 34], but for clarity we briefly describe its architecture.

The router has a VC for every distinct communication flow with a unique priority that passes through the router. Each communication flow also has a unique priority and all the flits belonging to a flow have the same priority. Consequently, there exists a VC for each priority level in every router. This one-to-one correspondence assures that each VC can only be used for a unique communication flow. The VCs are designed as FIFO buffers at the input ports of the router. These FIFOs store the data to be routed from different VCs. The unit of stored data is in the form

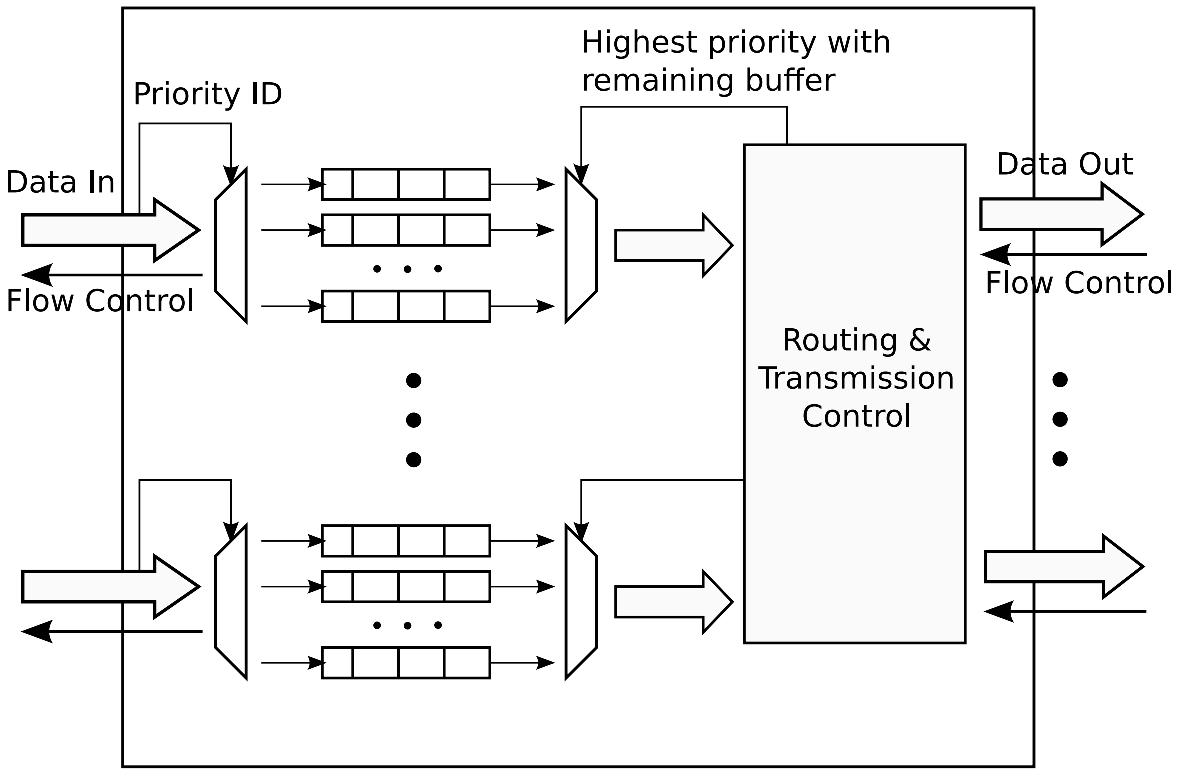
Highest priority with remaining buffer
Priority ID
Data In
Flow Control
Data Out
Flow Control
Routing & Transmission Control

Figure 3.1: Priority-aware router architecture.

of flits. Note that a flit is also the unit of data transmitted over the NoC. A packet on the other hand, is composed of multiple flits. The router selects the output port for a flit in the VCs based its desired destination. When there are multiple flits waiting to be routed to the same output port, the router selects and forwards the flit to the output port with the highest priority amongst all the waiting flits if the next corresponding router has enough buffer connected to that output port. This can be guaranteed by applying flow control mechanisms. This priority arbitration happens at the cycle level. This means that if the highest priority flit arrives at the router at time t, it will only be forwarded by the router at time $t + 1$ and will never preempt a lower priority flit being sent at time t. Flow control guarantees that the router only sends data to the neighbouring router if the neighbour has enough buffer space to store the data. If the highest priority flit is blocked in the network, the next highest priority flit can access the output link. This architecture allows the derivation of an upper bound on the latencies of all flows in the network due to the deterministic priority-aware arbitration.

3.2 System Model

We present the definitions, terminology and the model necessary for describing the flow-level and link-level analyses. These definitions extend and borrow definitions from previous work by Shi and Burns [33, 34] for flow-level analysis. In our analysis, we assume the assignment of distinct priorities to traffic flows, periodic flows, and that the deadline of a flow is less than or equal to its periodicity. We also assume that the flows mapping into the NoC is given and the flows paths are defined. It is possible to extend our analysis to priority sharing, like the scheme presented in [32].

Definition 1. *A real-time NoC is a 5-tuple $\langle G, \Gamma, R_{hop}, F, BW\rangle$ where G is the NoC's interconnection graph, Γ is a set of traffic flows deployed on the NoC, R_{hop} is the routing delay at each node in the NoC, F is the flit size, and BW is the link bandwidth.*

Definition 2. *The NoC's interconnection graph G is a directed graph $G := \langle V, E\rangle$, where V is the set of processing elements (including the router), and $E \subseteq V \times V$ is the set of edges describing the links between nodes. We use two directed edges to model a bidirectional link in the NoC.*

Definition 3. *The set of traffic flows $\Gamma := \{\tau_i : \forall i \in [1, n]\}$ has n traffic flows and each 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 and it is formally represented by the tuple [33]:*

$$\tau_i = (v_{s_i}, v_{d_i}, P_i, T_i, D_i, J_i^R, L_i)$$

This describes a traffic flow τ_i from source node $v_{s_i} \in V$ to destination node $v_{d_i} \in V$ with priority P_i, period T_i between successive packet transmissions, real-time deadline D_i, release jitter J_i^R, and the basic link latency L_i.

In the following we describe each of the flow characteristics in detail.

- Source v_s and Destination v_d. These present the source and the destination node of the flow in the NoC respectively. We assume the mapping of communication flows is given and each of the flows has a specific source and destination.

- Priority P. Each traffic flow has a distinct priority P compared to other flows and all packets of the same flow have the same priority. P value equal to 1 shows the highest priority flow and the greater the p value the lower the priority. In the prioritised flow switching techniques, lower priority flows might get blocked with higher priority ones.

- Period T. The length of time between the release of two continuous packets of the same traffic flow is a constant, which is called the period T_i for the traffic flow. For real-time applications, if the period is not constant, the minimum possible period should be considered to satisfy the requirements of hard real-time systems.

- Deadline D. Each real-time traffic flow has deadline *D*, which means that all the flits belonging to this traffic flow have the restriction to be delivered to the destination router within a certain delay less than or equal to *D* value, even in the worst-case scenario. The restriction for each flow is that each traffic flow's deadline must be less than or equal to its period, i.e. $D_i \leqslant T_i$ for all τ_i. This condition removes the self-blocking effect [32].

- Jitter J^R. The attribute J^R is the release jitter [1] of flow τ and denotes the maximum deviation of successive packets released from its period. A periodic traffic flow τ has to generate the communication instances within a fixed time which is its period, T, then it will be released as soon as it generated. However, when this periodic flow τ is subjected to release jitter, its generated time becomes under some circumstances different from its release time. So, τ is not become strictly periodic and a variation in its release times has arisen. Therefore, 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 is a, then the real release time may occur later than normal, by the time no more than $a + J^R$. Note that the value a is the worst-case release jitter that can accrue.

- Basic network latency L. The basic network latency occurs when no traffic flow contention exists. The basic network latency is determined by flow's source/destination routing distance, flow packet size, NoC links bandwidth and some additional protocol control overheads.

As the summary, a flow is schedulable if its worst-case latency is less than the deadline D_i. The release jitter J_i^R is the worst-case delay in a packet's release time. L_i is the latency that the flow experiences on a link when it does not suffer interferences from any other flows on that link. We compute L_i as follows: $L_i = \frac{Flits_i * F}{BW}$ where $Flits_i$ is the number of flits in one packet of the traffic flow τ_i. Now, we can compute the total basic latency of a flow with no interferences as $C_i = L_i + R_{hop} * hops$. Notice that $hops$ is the number of links the flow traverses. D_i is a time constraint representing a flow's deadline which is an upper bound on the flow's latency from the source node v_s to the destination node v_d. J_i^R represents an upper bound on the delay in a packet's release time originally designated by its period T_i. The route (or path) a flow τ_i traverses is a sequence of edges denoting the multiple links it crosses to reach from the source node to the destination node. For our work, we assume that these routes are fixed, and determined using off-line analysis (e.g. shortest path algorithm). We define a route, a subroute and a link in a network G as shown in Definitions 4 and 5, respectively.

Definition 4. *A route δ_i for flow τ_i is a sequence of edges $\langle(v_{s_i}, v_{s_i+1}), (v_{s_i+1}, v_{s_i+2}), \ldots, (v_{s_i+k}, v_{d_i})\rangle$ such that $k \in \mathbb{Z}$.*

Definition 5. *A subroute $\sigma_{i_{v_a}}$ for flow τ_i is a sequence of edges $\langle(v_{s_i}, v_{s_i+1}), \ldots, (v_{s_i+l}, v_a)\rangle$ such that $\sigma_{i_{v_a}}$ has the same source vertex v_{s_i} as route δ_i, and $\sigma_{i_{v_a}} \subset \delta_i$ with v_a being the destination node of the subroute.*

We use $|\delta_i|$ and $|\sigma_{i_{v_a}}|$ to denote the number of edges (or the length) in the route and subroute, respectively. We also use $e' = pre(\delta_i, e)$ to denote that an edge e' precedes edge e in the route δ_i where $e, e' \in \delta_i$. For $e, e' \in \delta_i$, we use tick (e') to denote an edge that precedes another edge e in the route δ_i.

3.2.1 Inter-Relationships between Traffic-Flows

To capture the relations between traffic-flows and the physical links of the network, the mesh network topology defined as a directed graph $G : V \times E$. V is a set, whose elements are called nodes, and each node v_i denotes a router combined with an IP core in the mesh network. E is a set of ordered pairs of vertices, called edges. Each edge represent a physical link in the NoC.

Based on whether flows share the same physical links or not, we categorize the competition relationship between 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. With the indirect competing relationship, the two traffic-flows do not share any physical link but there is at least one direct interference traffic-flow between the given two traffic-flows. For example, in Figure 3.2 flow τ_6 has direct interference with flows τ_5, τ_2, τ_3, and τ_4. It also has indirect interference with flow τ_1; since flow τ_2 has direct interference with both flows τ_6 and τ_1. We explain different type of interferences with an example in the following section.

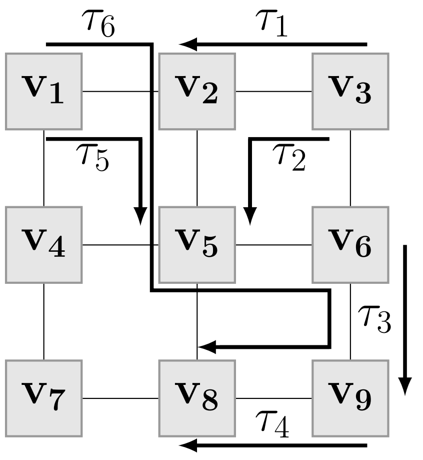

Figure 3.2: Interference example.

3.3 Illustrative Example

We present an illustrative example of a 3×3 NoC in Figure 3.2 to familiarize the reader with the terminology and interferences. Figure 3.2 maps 6 flows on the NoC: $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$ with the priorities $P_1 > P_2 > P_3 > P_4 > P_5 > P_6$. Note that the choice of our paths in Figure 3.2 are selected solely to illustrate, and explain the execution patterns of different flow with a link-by-link approach. Also note that the choice of a mesh topology is only for illustration purposes and the analysis we present is valid irrespective of the underlying network topology.

For experiencing the worst-case communication latency, the critical time instant should be defined. Critical time instant is a known term in the real-time systems theory that explains the conditions which results to the worst-case scenario [22]. We define the critical time instant as follows:

Definition 6. *Critical time instant on each link is defined as the time that all flows on that link attempt transmission at that time, which introduces the worst-case interference scenario.*

Table 3.1 presents the link latency, the period, the deadline, and the jitter for each of the flows in Figure 3.2. We use Figure 3.3 to show the transmission of flits for τ_6 on each of the links in route δ_6 for the example configuration in Figure 3.2. We assume a routing delay $R_{hop} = 1$.

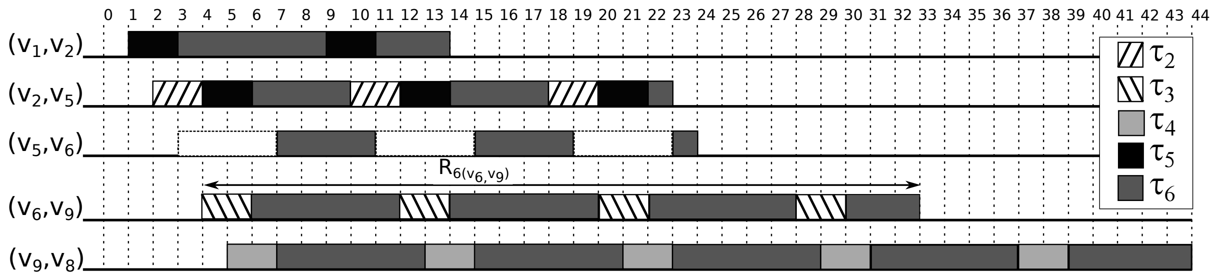

Figure 3.3: Timeline of direct interference on flow τ_6 in Figure 3.2.

For example, on link (v_1, v_2) the first flit transmits on at time 1 instead of 0 due to the routing delay. The number of flits from other flows that interfere with a flit of τ_6 on a particular link depends on the timespan of the τ_6 flit on that link. For experiencing the critical instant scenario, we assume the interfering flows start transmitting whenever τ_6 reaches that link.

Since τ_5 has a higher priority than τ_6, flits of τ_5 will preempt flits of τ_6. Observe that only two flits from τ_5 interfere with τ_6 on link (v_1, v_2). However, on link (v_2, v_5), flits from τ_5 and τ_2 cause interferences for τ_6. White spaces on link (v_5, v_6) represent gaps caused by interfering flows on the predecessor link (v_2, v_5). These flows no longer interfere with τ_6 on link (v_5, v_6), but the flits of τ_6 remain separated by the gaps shown due to interferences on previous links. Note that for facilitating the analysis on subsequent links, we assume that these gaps are part of the τ_6 flits as we show in more detail in Section 5.1.

Flow	**L**	**T**	**D**	**J**
τ_1	2	8	8	0
τ_2	2	8	8	0
τ_3	2	8	8	0
τ_4	2	8	8	0
τ_5	2	8	8	0
τ_6	9	50	50	0

Table 3.1: Motivating example data.

Chapter 4

Flow Level Analysis (FLA)

We present an overview of a method to compute the worst-case latencies of communication tasks, which we call the flow-level analysis (FLA). Shi and Burns [33, 32] proposed FLA, which considers a priority-aware NoC interconnect as its underlying hardware architecture, and they incorporate direct and indirect interferences caused by higher priority communication tasks. For additional details and proofs, we refer the reader to [33, 32]. The analysis incorporates direct and indirect interferences caused by higher priority tasks transmitting data on the links. FLA computes tighter worst-case bounds compared to viewing the communication platform as a unique resource.

4.1 Definitions

Let a communication task under analysis be τ_{ab}, and its worst-case latency be R_{ab}. Direct interference occurs when a higher priority task τ_{ij} preempts τ_{ab} along its path δ_{ab}. The set of higher priority communication tasks preempting τ_{ab} are identified as the direct interference set $S^D(\tau_{ab})$. Task τ_{ab} suffers indirect interference from task τ_{kl} when task τ_{ab} has direct interference with task τ_{ij} which has direct interference with task τ_{kl}; however, tasks τ_{ab} and τ_{kl} do not directly interfere with each other.

Definition 7. *A task τ_{ab} suffers direct interference from task τ_{ij} on its path δ_{ab} if and only if*

$$\delta_{ab} \cap \delta_{ij} \neq \emptyset, \textit{ and } P_{ab} < P_{ij}.$$

Definition 8. *The set of tasks directly interfering with τ_{ab} on its path δ_{ab} is:*

$$S^D(\tau_{ab}) = \{\tau_{ij} \mid \forall \tau_{ij} \in \Gamma_i^M, \delta_{ab} \cap \delta_{ij} \neq \emptyset \textit{ and } P_{ab} < P_{ij}\}.$$

Definition 9. *A task τ_{ab} suffers indirect interference from task τ_{kl} on its path δ_{ab} if and only if:*

$$(\delta_{ab} \cap \delta_{ij} \neq \emptyset) \wedge (\delta_{ij} \cap \delta_{kl} \neq \emptyset) \wedge (\delta_{ab} \cap \delta_{kl} = \emptyset), \textit{ and } P_{ab} < P_{ij} < P_{kl}.$$

Definition 10. *The set of tasks indirectly interfering with task τ_{ab} on its path δ_{ab} is:*

$$S^I(\tau_{ab}) = \{\tau_{kl} \mid \forall \tau_{ij} \in \Gamma_i^M, \forall \tau_{kl} \in \Gamma_k^M, (\delta_{ab} \cap \delta_{ij} \neq \emptyset) \textit{ and } (\delta_{ij} \cap \delta_{kl} \neq \emptyset) \\ \textit{and } (\delta_{ab} \cap \delta_{kl} = \emptyset) \textit{ and } P_{ab} < P_{ij} < P_{kl}\}.$$

4.2 Worst-case Latency

FLA assesses the interference on the communication path of τ_{ab} to compute its worst-case latency. It incorporates indirect interference as interference jitter that directly interfering flows suffer. Suppose that task τ_{ab} suffers interference from a higher priority task τ_{ij}. The time interval during which interference occurs is $R_{ab} + J_{ij}^R + J_{ij}^I$ where J_{ij}^R and J_{ij}^I are the release and interference jitters of task τ_{ij}, respectively. The interference jitter J_{ij}^I is the interference that task τ_{ij} suffers from tasks in the indirect interference set of task τ_{ab}. Hence, the maximum number of times τ_{ij} preempts τ_{ab} is computed by $\lceil \frac{R_{ab}+J_{ij}^R+J_{ij}^I}{T_i} \rceil$. The amount of time task τ_{ij} contributes to the worst-case latency of τ_{ab} is simply the transmission latency C_{ij} multiplied by the number of preemptions: $\lceil \frac{R_{ab}+J_{ij}^R+J_{ij}^I}{T_i} \rceil * C_{ij}$. For all higher priority communication tasks on the path δ_{ab}, FLA computes the worst-case latency of τ_{ab} as:

$$R_{ab} = \sum_{\forall i} \sum_{\forall j \in S_{ij}^D} \lceil \frac{R_{ab} + J_{ij}^R + J_{ij}^I}{T_i} \rceil * C_{ij} + C_{ab}$$

This only considers interference from higher priority tasks but not jobs from the same task under analysis τ_{ab}, which is necessary if the deadline is greater than the period. To incorporate this self-blocking, we define a busy period B_{ab} as the contiguous time interval during which the links on the path for communication task τ_{ab} suffer interferences from communications tasks of equal or higher priority. Thus, the busy period is given by:

$$B_{ab} = \sum_{\forall i} \sum_{\forall j \in S_{ij}^{D}} \lceil \frac{B_{ab} + J_{ij}^{R} + J_{ij}^{I}}{T_i} \rceil * C_{ij} + \lceil \frac{B_{ab} + J_{ab}^{R}}{T_a} \rceil * C_{ab}$$

The number of instances of task τ_{ab} in the busy period $p_{B,ab}$ is equal to $\lceil \frac{B_{ab}+J_{ab}^{R}}{T_a} \rceil$. The worst-case latency R_{ab} of task τ_{ab} is then the maximum response time of all instances and is given by:

$$R_{ab} = \max_{p=1 \ldots p_{B,ab}} (w_{ab}(p) - (p-1)T_a + J_{ab}^{R})$$

where p is the index of the job of task τ_{ab}. $w_{ab}(p)$ is the completion time of job p in the busy period B_{ab} and is given by:

$$w_{ab}(p) = p * C_{ab} + \sum_{\forall i} \sum_{\forall j \in S_{ij}^{D}} \lceil \frac{w_{ab}(p) + J_{ij}^{R} + J_{ij}^{I}}{T_i} \rceil * C_{ij}$$

4.3 An Illustrative Example

Figure 4.1 shows a simple example to demonstrate FLA.

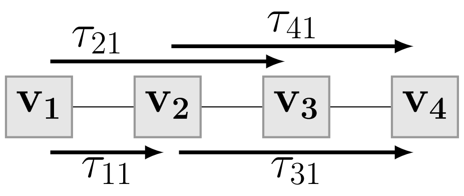

Figure 4.1: Simple example of FLA.

Table 4.1 characterizes tasks τ_{11}, τ_{12}, τ_{13}, and τ_{14} (task under analysis) in the figure. Assuming $P_{41} < P_{31} < P_{21} < P_{11}$, then $S^{D}(\tau_{41}) = \{\tau_{21}.\tau_{31}\}$ and $S^{I}(\tau_{41}) = \{\tau_{11}\}$.

Table 4.1: FLA example data

Task	**C**	**T**	J^R
τ_{11}	3	9	0
τ_{21}	2	9	0
τ_{31}	4	12	0
τ_{41}	3	8	0

To compute R_{41}, we first need to compute R_{21} to get the interference jitter of task τ_{21}.

$$\begin{aligned}
B_{21} &= \lceil\frac{B_{21}}{9}\rceil * 3 + \lceil\frac{B_{21}}{9}\rceil * 2 = 5 \\
p_{B,21} &= 1 \\
\therefore R_{21} &= 5 \text{ and } J_{21}^{I} = 5 - 2 = 3 \\
B_{41} &= \lceil\frac{B_{41} + J_{21}^{I}}{T_2}\rceil * C_{21} + \lceil\frac{B_{41}}{T_3}\rceil * C_{31} + \lceil\frac{B_{41}}{T_4}\rceil * C_{41} \\
&= \lceil\frac{B_{41} + 3}{9}\rceil * 2 + \lceil\frac{B_{41}}{12}\rceil * 4 + \lceil\frac{B_{41}}{8}\rceil * 3 = 23 \\
p_{B,41} &= \lceil\frac{23}{8}\rceil = 3 \\
w_{41}(1) &= 11 \text{ and } R_{41}(1) = 11 - 0 = 11 \\
w_{41}(2) &= 20 \text{ and } R_{41}(2) = 20 - 8 = 12 \\
w_{41}(3) &= 23 V R_{41}(3) = 23 - 16 = 7 \\
R_{41} &= max(R_{41}(1), R_{41}(2), R_{41}(3)) = 12
\end{aligned} \tag{4.1}$$

Chapter 5

Link Level Analysis (LLA)

5.1 Link-level Analysis (LLA)

Link-level analysis (LLA) incorporates direct and indirect interferences with other traffic flows. We describe the details of this analysis in this section, and we continue to use Figure 3.3 as a running example to provide the intuition behind the mathematical formulation. We first present the worst-case latency with direct interference of flits on a particular link. Then, we use this to compute the worst-case latency for the entire route. We later augment the worst-case latency with indirect interferences as well.

5.1.1 Worst-case Latency with Direct Interference

Direct interference on a link occurs when a higher priority flow τ_j preempts a lower priority flow τ_i on a shared link. We formalize direct interference at the link-level in Definition 11, and the set of flows resulting in direct interferences on a link in Definition 12. Notice that we use R_i and R_{i_e} to represent the worst-case latency of flow τ_i on its route δ_i, and on a particular link e, respectively. We use $R_{i_{e'}}$ to denote the worst-case latency of τ_i on a predecessor link e'. We also use $S^D_{i_e}$ and $S^I_{i_e}$ to denote the set of direct and indirect interfering flows with τ_i on link e, respectively.

Definition 11. *A flow τ_i suffers direct interference from flow τ_j on a link e if and only if $e \in \delta_i \cap \delta_j$, and $P_i < P_j$.*

Definition 12. *The set of flows directly interfering with τ_i on a link e is $S^D_{i_e} = \{\tau_j \mid \forall \tau_j \in \Gamma, e \in \delta_i \cap \delta_j$ and $P_i < P_j\}$.*

From Figure 3.2, we observe that τ_6 has direct interference with τ_2 and τ_5 on link (v_2, v_5) such that $S^D_{6_{(v_2,v_5)}} = \{\tau_2, \tau_5\}$.

To compute the worst-case latency of a flow, we must incorporate the effect of direct interferences. However, a link's contribution to the latency due to direct interference depends on its predecessor link's direct interference as well. For example in Figure 3.3, the latency of flits on link (v_9, v_8) depend on the direct interference that they suffer on link (v_6, v_9). Hence, we compute the worst-case latency contribution of a flow τ_i on a link e as R_{i_e}. This depends on the direct interference the flits suffered on the predecessor link e'. For example, $R_{6_{(v_6,v_9)}}$ is the worst-case latency of flow τ_6 on link (v_6, v_9) and is equal to $33 - 4 = 29$ time units. Theorem 1 presents the worst-case latency on a link including direct interferences from higher priority flows.

Theorem 1. *The worst-case latency R_{i_e} of a flow τ_i suffering direct interferences from other flows in $S^D_{i_e}$ on link $e \in \delta_i$ is given by the following:*

$$R_{i_e} = R_{i_{e'}} + \sum_{\forall \tau_j \in S^D_{i_e}} \lceil \frac{R_{i_e} + J^R_j}{T_j} \rceil * L_j - \sum_{\forall \tau_j \in S^D_{i_e} \cap S^D_{i_{e'}}} \lceil \frac{R_{i_{e'}} + J^R_j}{T_j} \rceil * L_j$$

Proof. Assume flows $\tau_i, \tau_j \in \Gamma$ with $P_j > P_i$, and let the set of flows directly interfering with τ_i on links e and e' be $S^D_{i_e}$ and $S^D_{i_{e'}}$, respectively. The first term of the equation is $R_{i_{e'}}$ which is the worst-case latency on the predecessor link e'. The reason is that we need to take into account interferences from previous links as we compute latencies along the route δ_i. For the first link of the route δ_i, we replace $R_{i_{e'}}$ by L_i.

The second term of the equation accounts for interference from all higher priority flows on link e. The duration that flits of flow τ_i suffer direct interference from flow τ_j on link e is given by $R_{i_e} + J_j^R$, which sums the release jitter from τ_j with the worst-case latency of τ_i. The number of times τ_j preempts τ_i is $\lceil (R_{i_e} + J_j^R)/T_j \rceil$, and the latency of the preemptions by τ_j is $\lceil (R_{i_e} + J_j^R)/T_j \rceil * L_j$.

When a flow τ_j interferes with τ_i on several consecutive links, the interference has to be accounted for only once. If $S_{i_e}^D \cap S_{i_{e'}}^D \neq \emptyset$, the term $\sum_{\forall \tau_j \in S_{i_e}^D \cap S_{i_{e'}}^D} \lceil (R_{i_{e'}} + J_j^R)/T_j \rceil * L_j$ represents the latency of preemptions on e' from common flows between links e and e'. We subtract this term from the worst-case latency on link e because common flows are re-accounted for in the second term of the equation. The worst-case latency on link e, therefore, becomes

$$R_{i_e} = \sum_{\forall \tau_j \in S_{i_e}^D} \lceil \tfrac{R_{i_e} + J_j^R}{T_j} \rceil * L_j + R_{i_{e'}} - \sum_{\forall \tau_j \in S_{i_e}^D \cap S_{i_{e'}}^D} \lceil \tfrac{R_{i_{e'}} + J_j^R}{T_j} \rceil * L_j.$$

□

Illustrating LLA for Direct Interference:

We explain how the worst-case latency analysis works using different scenarios of interference and use Figure 3.3 for illustration.

Scenario 1: This case applies to the first link on the route δ_i. We illustrate this case on link (v_1, v_2). For this link, $R_{i_{e'}}$ is replaced by L_i; the basic link latency of flow τ_i. Since the previous link e' is undefined in this case, the third term evaluates to 0. Therefore, the analysis computes only the interference from τ_5 and adds it to L_6.

Scenario 2: We illustrate this case on links (v_2, v_5) and (v_5, v_6). In this case, $S_{i_{e'}}^D \supseteq S_{i_e}^D$. This occurs when there are some flows directly interfering on e' that are no longer causing any interference on e. Therefore, $S_{i_e}^D \cap S_{i_{e'}}^D = S_{i_e}^D$. Solving the equation yields $R_{i_e} = R_{i_{e'}}$. This is because of the interference on e', and the absence of new flows causing interference on e, the worst-case latency on e is the same as that on e'. On link (v_2, v_5), the flits of flow τ_6 take from time 2 to 23 to be transmitted. This includes delay from interferences from τ_2 and τ_5. On link

(v_5, v_6), τ_5 and τ_2 no longer interfere. Hence, the only delay visible on the flits of τ_6 on link (v_5, v_6) is the routing delay shown as the shift of one time unit. Also notice the white gaps that are a result of interferences on the previous link from flows τ_2 and τ_5. To analyze subsequent links, we assume that these gaps are part of the delay of the flits.

Scenario 3: We show this case using on links (v_1, v_2) and (v_2, v_5). In this case, $S^D_{i_{e'}} \subseteq S^D_{i_e}$. This arises when there are new flows directly interfering on e in addition to the flows that are directly interfering on e'. All the flits of the flow suffering interference and of the higher priority flows on link e' will continue in the same order to link e. However, on link e, the flits of the new flows (i.e. $S^D_{i_e} \backslash S^D_{i_{e'}}$) add to the direct interference, further preempting the flits of the flow suffering interference. Since $S^D_{i_e} \cap S^D_{i_{e'}} = S^D_{i_{e'}}$, then the term we subtract will include all interferences on the previous link e'. This is also intuitive because these common flows will be accounted for as interferences on link e. This indicates that the latency of flow τ_6 on link (v_2, v_5) is not affected by the latency on link (v_1, v_2) because the worst-case latency occurs when more flows interfere with τ_6 which occurs on link (v_2, v_5). So $R_{i_{e'}}$ is equal to L_6 plus the interference from flow τ_5 (which we subtract) for link (v_2, v_5) and the interference of both τ_5 and τ_2 is used to calculate the worst-case latency $R_{6_{(v_2,v_5)}}$.

Scenario 4: We elaborate this case on links (v_6, v_9) and (v_9, v_8). This case occurs when some flows interfering with τ_i on link e' cease to interfere on link e, but some new flows begin interfering on e, i.e., $S^D_{i_e} \cap S^D_{i_{e'}} = \emptyset$. However, the flows that cease to interfere already affect the stream of flits, which means that they still leave gaps in the transmission of flits. To account for this, we assume that the flits of τ_i consume the entire duration from the first to the last flit including the gaps. Therefore, $R_{i_{e'}}$ represents the latency of the flits before interference. This is an upper bound on the amount of time during which interference can occur. We then use $R_{i_{e'}}$ as the basic latency to calculate the total worst-case latency R_{i_e} from the set of flows $S^D_{i_e}$. By focusing on the interference that τ_6 suffers, we observe that the duration of time during which τ_6 suffers interference from τ_4 starts when the first flit of τ_6 can be sent. This occurs at time 5. Hence, the term $R_{i_{e'}}$ is equal to the link latency of τ_6 on link (v_6, v_9), $R_{6_{(v_6,v_9)}} = 33-4 = 29$. This time is assumed to be the basic latency of τ_6 before calculating $R_{6_{(v_9,v_8)}}$ after direct interference with τ_4.

5.1.2 Worst-case Latency of a Route

Recall that Theorem 1 presents the worst-case latency at a particular link. Theorem 2, on the other hand, computes the worst-case latency of a flow τ_i experiencing direct interferences along its entire route δ_i by aggregating the worst-case link latencies computed using Theorem 1.

Theorem 2. *The worst-case latency of flow τ_i for route δ_i denoted by R_i is*

$$R_i = R_{i_{(v_b,v_d)}} + R_{hop} * |\delta_i|$$

where $(v_b, v_{d_i}) \in \delta_i$ is the last edge on the route.

Proof. Recall that $R_{i_{(v_b,v_d)}}$ is the worst-case latency of flow τ_i on its last link connected to the destination node v_{d_i}. Notice that every flit suffers a routing delay R_{hop} when traversing each node. So, for $|\delta_i|$ nodes, we add the routing delay of $R_{hop} * |\delta_i|$ to obtain the total worst-case latency of flow τ_i as shown. □

Observe in Figure 3.3, the total worst-case latency for τ_6 is the sum of the routing delay $R_{hop} * |\delta_i| = 5$, and the worst-case latency on the last link $R_{6_{(v_9,v_8)}} = 44 - 5$ resulting in 44 time units.

5.1.3 Worst-case Latency with Indirect Interference

Flow τ_i suffers indirect interference on a link from flow τ_k when flow τ_i has direct interference with an intermediate flow τ_j, and τ_j has direct interference with flow τ_k; however, τ_i has no direct interference with τ_k. In addition, the interference between τ_j and τ_k must occur before τ_j interferes with τ_i. We formally describe this in Definition 13, and the set of indirectly interfering flows in Definition 14. Revisiting Figure 3.2, we point out that flow τ_6 has indirect interference with flow τ_1 through an intermediate flow τ_2 on link (v_2, v_5) such that $S^I_{6_{(v_2,v_5)}} = \{\tau_1\}$.

Definition 13. *A traffic flow τ_i suffers indirect interference from flow τ_k on a link $e(v_a, v_b)$ if and only if:*

$$(e \in \delta_i \cap \delta_j) \wedge ((v_c, v_d) \in \delta_j \cap \delta_k) \wedge (e \neq (v_c, v_d)) \wedge ((v_c, v_d) \notin \delta_i) \wedge ((v_c, v_d) \in \sigma_{j_{v_a}}),$$
and $P_i < P_j < P_k$.

Definition 14. *The set of flows indirectly interfering with flow τ_i on link $e(v_a, v_b)$ is:*

$$S^I_{i_e} = \{\tau_k \mid \forall \tau_j, \tau_k \in \Gamma, (e \in \delta_i \cap \delta_j) \wedge ((v_c, v_d) \in \delta_j \cap \delta_k) \wedge (e \neq (v_c, v_d)) \wedge ((v_c, v_d) \notin \delta_i) \wedge ((v_c, v_d) \in \sigma_{j_{v_a}}), \text{ and } P_i < P_j < P_k\}.$$

The worst-case latency of flow τ_i when it experiences both direct and indirect interferences is given by Theorem 3. $J^I_{j_e}$ represents the interference jitter on a higher priority flow due to indirect interference as described by Shi and Burns [33, 34]. Interference jitter is the delay that flits of flow τ_j suffers due to interference that is equal to the difference between its worst-case and basic latencies. We carry this definition forward such that the interference jitter of τ_j for its subroute $\sigma_{j_{v_a}}$ is equal to the difference between the total worst-case latency on the subroute and the link-level latency of τ_j.

Theorem 3. *The worst-case latency on a link of a flow τ_i, R_{i_e}, is when it suffers both direct and indirect interferences on link $e = (v_a, v_b)$ is given by:*

$$R_{i_e} = R_{i_{e'}} + \sum_{\forall \tau_j \in S^D_{i_e}} \lceil \frac{R_{i_e} + J^R_j + J^I_{j_e}}{T_j} \rceil * L_j - \sum_{\forall \tau_j \in S^D_{i_e} \cap S^D_{i_{e'}}} \lceil \frac{R_{i_{e'}} + J^R_j + J^I_{j_{e'}}}{T_j} \rceil * L_j$$

where $J^I_{j_e} = R_{j_{(v_c,v_a)}} - L_j$ and $(v_c, v_a) \in \sigma_{j_{v_a}}$ is the last edge on the subroute.

Proof. The first term of the equation represents the latency of direct and indirect interferences from higher priority flows with τ_i. The worst-case link latency of τ_i with indirect interferences is affected by the latency of τ_j due to interferences that it suffers. The increase in latency of τ_j due to interferences prior to link e is given by $J^I_{j_e}$, which is equal to the difference between the total worst-case latency on the subroute $\sigma_{j_{v_a}}$ and the basic link latency L_j. The worst-case scenario occurs when flits of flow τ_j suffers interference from τ_k and then the following flits follow with their normal periodicity [33, 34]. Since τ_j directly interferes with τ_i, $J^I_{j_e}$ increases the duration during which τ_i can suffer interference. Hence, this duration is equal to $R_{i_e} + J^R_j + J^I_{j_e}$. The number of times τ_j preempts τ_i is

$$\lceil (R_{i_e} + J_j^R + J_{j_e}^I)/T_j \rceil,$$

and the latency of the preemptions by τ_j is

$$\lceil (R_{i_e} + J_j^R + J_{j_e}^I)/T_j \rceil * L_j.$$

Therefore, the worst-case latency on e including indirect interferences from all higher priority flows is given by

$$\textstyle\sum_{\forall \tau_j \in S_{i_{e_1}}^D} \lceil \frac{(R_{i_e} + J_j^R + J_{j_e}^I)}{T_j} \rceil * L_j + R_{i_{e'}} - \sum_{\forall \tau_j \in S_{i_e}^D \cap S_{i_{e'}}^D} \lceil \frac{R_{i_{e'}} + J_j^R + J_{j_{e'}}^I}{T_j} \rceil * L_j.$$

The total worst-case latency of τ_i experiencing direct and indirect interferences along its route δ_i is given by applying Theorem 3 in Theorem 2. □

5.1.4 Tightness Analysis

We expect LLA to have tighter latency bounds compared to FLA. The reason is that FLA assumes that the interference that a flow suffers on any link occurs on the whole path of the flow while LLA restricts the interference only to the links on which they happen. In what follows, we formally prove that LLA provides tighter bounds than FLA.

Lemma 1. *Given a set of flows* Γ *and their paths,* $R_i^{LLA} \leq R_i^{FLA}$, $\forall \tau_i \in \Gamma$.

Proof. According to FLA [33], the worst-case latency of τ_i is given by:

$$R_i^{FLA} = \sum_{\forall \tau_j \in S_i^D} \lceil (R_i^{FLA} + J_j^R + J_j^I)/T_j \rceil * C_j + C_i \tag{5.1}$$

The worst-case interference occurs when the higher priority flows share all links with the path δ_i of τ_i. This means that the set of all interfering flows on δ_i, $S_i^D = S_{i_e}^D$. In that case, using Theorems 3 and 2, R_{i_e} on all links of δ_i are equal and $R_i^{LLA} = R_{i_e} + R_{hop} * hops$ where $R_{i_{e'}}$ equals L_i. Given that $C_i = L_i + R_{hop} * hops$, R_i^{LLA} can be given by:

$$\sum_{\forall \tau_j \in S_i^D} \lceil (R_i^{LLA} - R_{hop} * hops + J_j^R + J_{j_e}^I)/T_j \rceil * L_j + C_i \tag{5.2}$$

For FLA, $J_j^I = R_j - C_j$ and for LLA $J_{j_e}^I$ is given by Theorem 3. In the worst case, $J_{j_e}^I = R_{j_e} - L_j = R_j - R_{hop} * hops - L_j = R_j - C_j$. Therefore, $J_{j_e}^I = J_j^I$. Taking this into account and comparing Equations 5.1 and 5.2, the only difference between FLA and LLA are the terms L_j and C_j, and the routing delay in the summation. Since $C_j = L_j + R_{hop} * hops$, and $hops \geq 1$ then $L_j < C_j$, and since the routing delay has a negative sign in the numerator of the summation, therefore, $R_i^{LLA} < R_i^{FLA}$ and our analysis gives a tighter bound compared to FLA. For the case when there is no interference, $R_i^{FLA} = C_i$ and $R_i^{LLA} = L_i + R_{hop} * hops = R_i^{FLA}$. Since in the presence of interference, $R_i^{LLA} < R_i^{FLA}$, and in the absence of interference, $R_i^{LLA} = R_i^{FLA}$, then $R_i^{LLA} \leq R_i^{FLA}$. $\square$

5.2 An Illustrative Example

We use Figure 3.2 as an illustrative example to show the benefits of LLA versus the FLA [33, 34]. Recall that the flows in Figure 3.2 have the data in Table 3.1, and the following priorities: $P_1 > P_2 > P_3 > P_4 > P_5 > P_6$. Table 5.1 shows the worst-case latencies from FLA and LLA for all the flows. We observe that the results from LLA are less than or equal to the upper bounds computed by FLA. One of the interesting cases in Table 5.1 is for τ_6, which results in an unschedulable flow when using FLA. We elaborate the computation steps for this flow in more detail.

According to FLA, τ_6 has direct interference with the flows $S_6^D = \{\tau_2, \tau_3, \tau_4, \tau_5\}$, and indirect interference with $S_6^I = \{\tau_1\}$. Assuming $R_{hop} = 1$, R_6^{FLA} according to [33] is given by:

$$
\begin{aligned}
C_1 &= L_1 + R_{hop} * 1 = 3 \\
C_2 &= L_2 + R_{hop} * 2 = 4 \\
R_2^{FLA} &= \sum_{\forall \tau_j \in S_2^D} \lceil \frac{R_2^{FLA} + J_j^R}{T_j} \rceil * C_j + C_2 \\
&= \lceil \frac{R_2^{FLA}}{8} \rceil * 3 + 4 = 7 \\
C_3 &= C_4 = C_5 = L_3 + R_{hop} * 1 = 3 \\
C_6 &= L_6 + R_{hop} * 5 = 14 \\
R_6^{FLA} &= \sum_{\forall \tau_j \in S_6^D} \lceil \frac{R_6^{FLA} + J_j^R + J_j^I}{T_j} \rceil * C_j + C_6 \\
&= \lceil \frac{R_6^{FLA} + 3}{8} \rceil * 3 + \lceil \frac{R_6^{FLA}}{8} \rceil * 3 * 3 + 14
\end{aligned}
\tag{5.3}
$$

Notice that Equation 5.3 will never reach a fixpoint; hence, it has no solution. In every iteration of the equation, R_6^{FLA} will increase by more than 8, hence, never leading to a fixpoint. The reason is that 4 flows interfere with τ_6 each with a basic latency of 3 and period 8. This means that all 4 flows consume all bandwidth, i.e. the utilization of the links exceeds a 100%, and the flits of τ_6 can never be sent. Thus, τ_6 is not schedulable for any deadline. This occurs because FLA assumes that τ_6 suffers simultaneous interference from τ_2, τ_3, τ_4 and τ_5 on all links along its route. This is certainly not the case as shown in Figure 3.3. In fact, the worst-case latency of τ_6 can be computed, but it requires performing the analysis at the links.

Using LLA, the worst-case latency of τ_6 is given by:

$$\begin{aligned}
R_{2_{(v_3,v_2)}} &= \sum_{\forall \tau_j \in S^D_{2_{(3,2)}}} \lceil \frac{R_{2_{(v_3,v_2)}} + J^R_j}{T_j} \rceil * L_j + L_2 \\
&= \lceil \frac{R_{2_{(v_3,v_2)}}}{8} \rceil * 2 + 2 = 4 \\
R_{2_{(v_2,v_5)}} &= R_{2_{(v_3,v_2)}} = 4 \\
R^{LLA}_2 &= R_{2_{(v_2,v_5)}} + R_{hop} * 2 = 4 + 1 * 2 = 6 \\
R^{LLA}_6 &= 39 + 1 * 5 = 44
\end{aligned} \tag{5.4}$$

We can see from Equation 5.4 and Figure 3.3 that τ_6 indeed has a worst-case latency of 44 time

Flow	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
FLA	3	7	3	3	12	-
LLA	3	6	3	3	6	44

Table 5.1: Analysis results.

units, and it is schedulable. This provides a simple example where LLA performs better than FLA.

Chapter 6

Simulator

To evaluate the effectiveness of LLA, we implement a cycle-accurate simulator for a real-time NoC research, which we call UWNOC. The simulator uses SystemC, and it provides the ability to experiment with different real-time NoC designs that includes mesh size, buffer size, router designs, priority-aware routing protocols, and so on. It also allows deploying multiple applications with different priorities on the NoC. In this chapter, we describe the implementation of the UWNOC, and through a simple example, we illustrate how to use UWNOC.

6.1 UWNOC Features and Capabilities

UWNOC is a modular simulator implemented in SystemC. Parameters such as topology size, buffer sizes, number of deployed applications, and also their characteristics are configurable in the simulator. UWNOC assumes the mesh topology and the size of mesh is configurable by the user. The routing and switching policies are based on a prioritized flow model. The simulator outputs the latency of deployed flows on the NoC. Due to its modular design, the simulator may be extended to include new topologies or routing policies.

6.2 Configuration

The user can configure the following parameters:

- Topology size: the mesh size is configurable, for example size equal to 4 defines a 4$*$4 2D mesh NoC.
- Number of applications: configurable number of applications can be mapped to the NoC.
- Type of applications: applications can be sender or receiver. The sender application acts as a communication source and it sends a message to a receiver application which acts as the communication destination.
- Buffer size: the size of buffers inside the routers can be modified based on the application requirements.

6.3 Switching Policy

The simulator supports wormhole switching in which packets are divided into flits. Each packet consists of three different types of flits:

- Head flit: this is the first flit of packet and shows the start of the packet.
- Data flit: all flits following the header flit are data flits except the last one.
- Tail flit: this is the last flit and shows the end of the packet.

6.4 Routing Policy

Minimum random shortest path algorithm is applied for mesh topology to statically choose the flow route. Each router in the NoC has a queue per priority level. Routing starts from the highest priority communication flow and lower priority flows can get blocked with higher ones. Applying this method provides us the capability to observe and measure the effect of interference from higher priority flows on the lower priority ones.

6.5 Communication Time Measurement

The simulator can measure the communication time of the communication tasks. This is done by incorporating timestamps into flits. For periodic communications, the latency can be measured in different periods and the maximum value is reported as the worst-case communication latency. The result of the simulator measurement verifies a lower bound of latency for FLA and LLA analyses result.

6.6 Simulator Extension

For deploying applications with different properties and communication and computation modules, the user can tune several parameters such as sender and receive modules, and the routing policy.

6.6.1 Task

Tasks are computational modules mapped onto cores of the NoC. Tasks communicate with each other by sending communication messages through the NoC, which are characterized by the communication flow. Each task has two concurrent modules: the sender and the receiver. For our simulator *Task.cpp* represents the task module.

Sender Module

Sender module is a function that can be implemented within tasks and its application is to generate flows to be sent out to a particular destination. In other words, it works as the flow source node. Based on the specification of applications which comes from *application.config*, if there is a sender module in a particular node, the sender function inside the task is activated and a SystemC SC_THREAD is instantiated for this purpose. The following code snippet shows the way that is done.

```
SC_THREAD(send);
sensitive<<clock.pos();
```

Receiver Module

Receiver module (receiver function) works as a sink and it already exists in all NoC nodes. Whenever there is a flit targeted to that particular node to which the task is mapped, the receiver module accepts that flits and does the required processing.

6.6.2 Router

Router has connections to tasks and input/output ports. It also has a look-up table that stores the routing information for different communication flows.

6.7 Configuration Files

Simulator accepts two command line input files, the application configuration file and the router configuration file. In the following we explain each of these files.

6.7.1 Application Configuration

The following Figure 6.1 shows an example of application configuration file. This file contains the size of mesh in its first line. The second line explains that two sender applications are mapped to node zero. Line 3 expresses that the first application mapped to node 0. The numbers in line 3 are the priority, the period, the basic link latency, and the destination of the flow, respectively. Line 4 shows the characteristics of the second application mapped to node 0.

```
1 4
2 0 2
3 1 375 250 1
4 5 750 250 2
5 1 1
6 2 375 250 2
7 2 2
8 3 375 250 3
9 6 750 250 3
10 3 2
11 4 375 250 7
12 7 750 250 7
13 7 2
14 8 750 250 8
15 12 500 200 8
```

Figure 6.1: Application configuration file.

6.7.2 Router Configuration

The router configuration file explains the contents of the routers look-up tables. Figure 6.2 shows an example of router configuration file. In this example, each line shows the configuration for a specific router. Here we have four routers, and five flows in the system. The first entry in the first line is equal to 1, which means that router 1 sends the flits of flow with priority 1 to its east output port. We assume this assignment of numbers to ports: North(0), East(1), South(2), West(3), Core(4), and 5 means undefined.

```
1 1 5 5 5 1
2 4 1 5 5 1
3 5 4 1 5 4
4 5 5 4 2 5
```

Figure 6.2: Router configuration file.

6.8 An example with simulator

We provide a simple example describing the configuration files for a sample set of flows. Figure 6.3

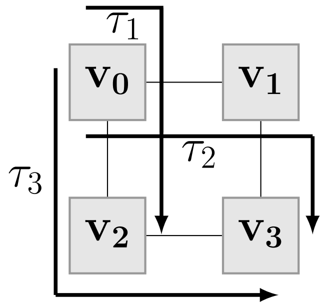

Figure 6.3: Simulator example.

shows a 2*2 NoC with three flows. As an example, τ_1's source is v_1 and its destination is v_3. The corresponding *Application.config* for the Figure 6.3 with arbitrary flow characteristics is as Figure 6.4.

Line 2 explains three flows are mapped to v_0. The characteristics of these three flows are being

```
1 2
2 0 3
3 1 375 250 2
4 2 750 250 3
5 3 750 250 3
```

Figure 6.4: Application configuration.

provided in the lines 3-5.

Figure 6.5 shows the router configurations. The first column is the line numbering. As an example, in line 2, which is for router v_1, the first 5 means that the decision of this router for flow τ_1 is undefined. The number 2 in the line 2 means that this router sends the flits of τ_2 to the south output port.

```
1 2 1 2
2 5 2 5
3 4 5 1
4 5 4 4
```

Figure 6.5: Router configuration.

After setting up the simulator with the proper input files, the following results are the latency measurements extracted from the simulator.

Flow	WC-L
τ_1	251
τ_2	252
τ_3	752

Table 6.1: Simulator result.

Chapter 7

Multimedia Application Case Study

7.1 Case study: Set-top Box

We experiment with a dual input channel set-top box application. This case study models common digital video recorders such as Cisco's Explorer 8300HD DVR that allow recording of an input video stream to the hard disk while simultaneously watching another independent input video stream. The second video stream can be a previously recorded video or one directly from the second input channel. For our case study, we implement the encoding of a video stream from one input channel, and the decoding of a video stream from the hard disk on a multicore with the real-time NoC. This allows both saving and viewing of the video streams with a higher priority given to the viewing. To observe the effect and impact of interferences on the latencies, we implement a cycle-accurate simulator of the NoC using SystemC.

This simulator models the cores, priority-aware routers, and the interconnect. Figure 7.1 shows the block diagram of this application. We annotate this diagram to show the encoder that writes to the hard disk, and the decoder that outputs to the display. Note that we use the MPEG2 standard in this experiment. The encoder consists of the tasks: Source, Motion Estimation, DCT Estimation, Transform, Quantize, VLE, iQuantize, iTransform, and Disk.

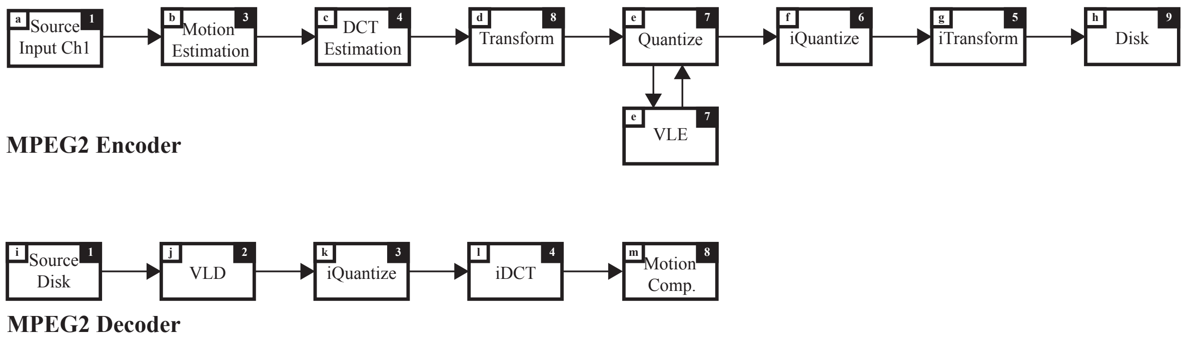

Figure 7.1: Set-top box block diagram.

Except for *Quantize* and *VLE* which are mapped onto the same core, only one task is mapped per core as shown in Figure 7.2.

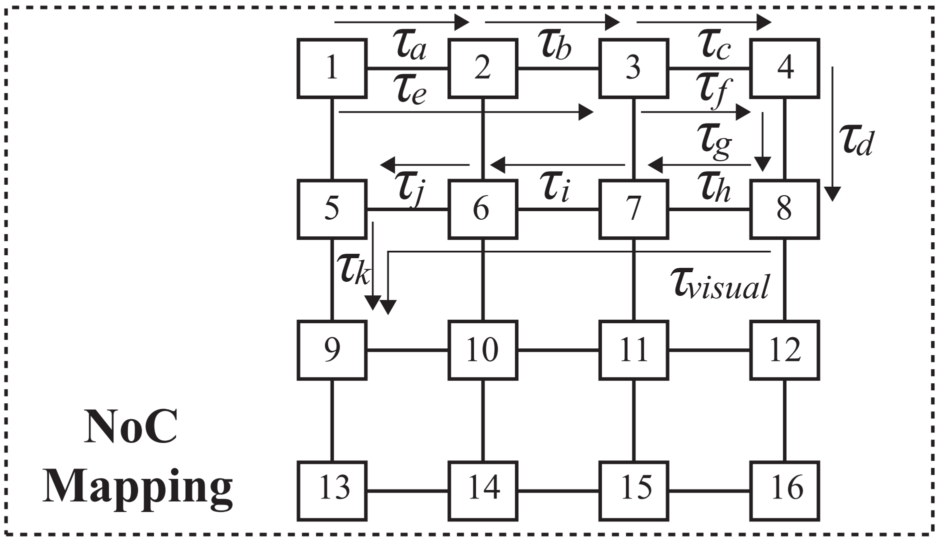

Figure 7.2: Set-top box mapping.

The decoder is composed of: Source, VLD, iQuantize, iDCT, and Motion Compensation. We map each decoder task on a separate core. We parallelize the encoding and the decoding, and map them to separate cores on the NoC. We also add source and destination tasks for a graphical user interface visualization, represented as τ_{visual} in Figure 7.1. We indicate the mapping on the block as the number, and use letters to indicate the flows on the adjacent mapped NoC. For example, the Motion Estimation block is mapped onto core 3, with a flow τ_e. Figure 7.2 describes the mapping and the flows on the NoC.

7.2 Case Study Result

Table 7.1 shows the flow characteristics of the different tasks.

Flow	**P**	**L**	**T**	**WC-L**	**LLA**	**FLA**
Source-VLD	1	250	375	251	251	251
VLD-iQuant	2	250	375	251	251	251
iQuant-iDCT	3	250	375	251	251	251
iDCT-MC	4	250	375	251	251	251
Source-ME	5	250	750	1002	2252	-
ME-DCT Estim.	6	250	750	751	751	1004
DCT Estim.-Trans.	7	250	750	751	751	1004
Trans.-Quant/VLE	8	250	750	251	251	251
Quant/VLE-iQuant	9	250	750	251	251	251
iQuant-iTrans.	10	250	750	251	251	251
iTrans.-Disk	11	250	750	251	251	251
Source-Screen	12	200	500	454	1954	-

Table 7.1: Case study data.

According to LLA, all flows are schedulable while two are unschedulable for FLA. For flows that do not suffer any interference, LLA and FLA have equal bounds, otherwise, LLA has tighter bounds. LLA performs as an upper bound to the measured worst-case latency (*WC-L*) for all flows.

Chapter 8

Experimentation Results

We present a quantitative comparison between the proposed LLA and FLA.

8.1 Experimentation

The experiment setup involves changing a number of factors as shown in Table 8.1 to assess their effect on the analysis. According to [29], approximately 60% of NoC designs use mesh and torus topology. Consequently, we experiment with a 4×4, and 8×8 mesh topologies. The basic link latency of a flow is randomly chosen from a uniform distribution in the range $[16, 1024]$. We define the link utilization of a flow τ_i as $U_i = L_i/T_i$. We vary the link utilization between $[0.4, 0.65]$ in steps of 0.05. The deadline D_i varies between $[0.7, 1.0]$ in steps of 0.1 as a ratio of the period T_i. The number of flows in the network range between $[10, 60]$ in steps of 10. Combining these factors, we have 336 different configurations of U, D, mesh size and number of flows. For each configuration, we generate 1000 test cases. Each test case has a random mapping of the source and destination nodes of the flows, and each flow is routed using a shortest path algorithm. If there exists multiple shortest paths for a flow, one of them is randomly chosen.

Our metrics for evaluation are the following:

Factor	**Variation**
Mesh size	$4 \times 4, 8 \times 8$
Basic link latency L_i	random from a uniform distribution $[16, 1024]$
Link utilization U_i	$[0.4, 0.65]$ in steps of 0.05
Deadline D_i	$[0.7, 1.0]$ in steps of 0.1 as a ratio of T_i
Number of flows	$[10, 60]$ in steps of 10

Table 8.1: Experiment setup.

- **Ratio of average worst-case latencies:** For each test case, we compute the worst-case latency of each flow for LLA and FLA. We compute the average of the worst-case latencies for both analyses in each test case. For each configuration, we have 1000 different values. The ratio of each pair of latencies shows the tightness of LLA's results when compared to FLA.

- **Ratio of unschedulable flows:** For each test case, we compute the number of unschedulable flows for LLA and FLA. A flow is unschedulable in one of two cases: $1)$ total worst-case latency of its route is larger than its deadline, or $2)$ no route is available that satisfies the bandwidth requirements of the flow i.e. any available route to the flow will require a link utilization of more than a 100%. The ratio of the number of unschedulable flows in each test case measures the ability of LLA to improve schedulability compared to FLA.

Figure 8.1 shows the ratio of average worst-case latencies for LLA and FLA against the number of flows for all 336 configurations. The lines in the graph show the mean value across all test cases of the same configuration. Although, all data points are in the range [0.0,1.0], meaning that LLA always provides a latency less than or equal to FLA, the mean lines provide insight to the general trend for any configuration as the number of flows increase. Initially, for 10 flows, the ratio of latencies is large in the range of 0.8 to 1.0. As the number of flows increase, the ratio decreases (larger difference between the LLA and FLA). The reason for this is that increasing the number of flows leads to more interferences, and as the interferences increase, the analytical results from LLA becomes tighter than FLA. Ratio decreases at a lower rate (lines closer to the top) for configurations with higher link utilization. In these configurations, the capability of the NoC to accommodate more flows decreases as the number of flows increase, thus resulting in a smaller difference between LLA and FLA.

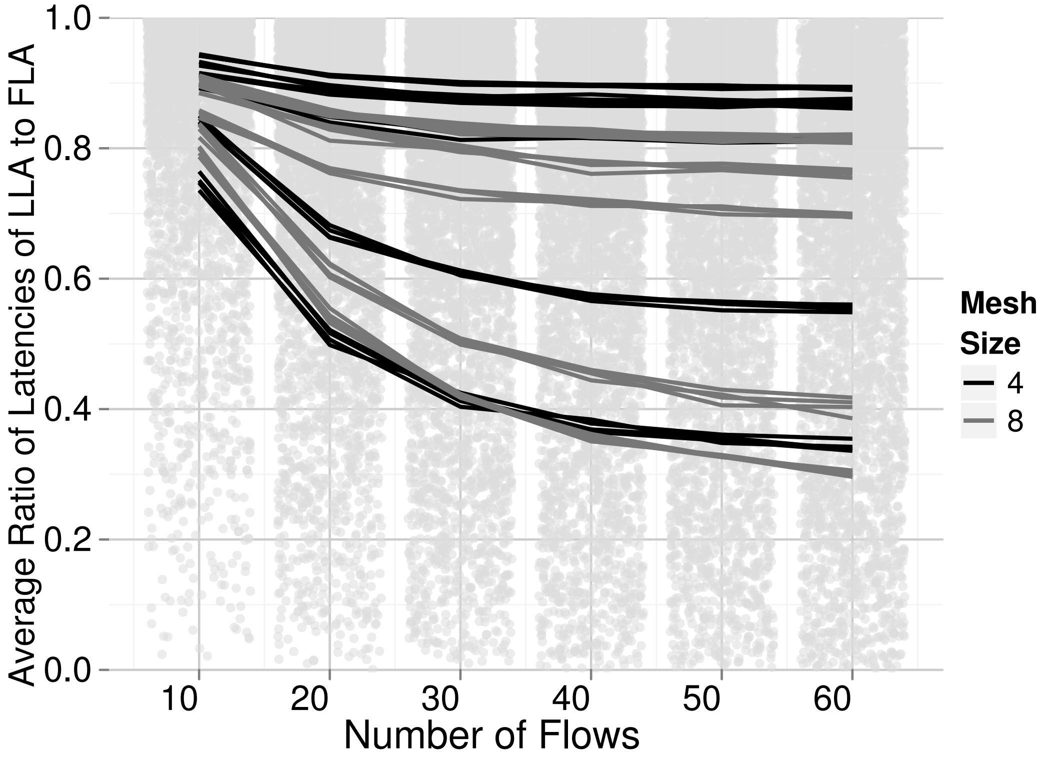

Figure 8.1: Ratio of latencies against the number of flows.

Figure 8.2 shows the ratio of average worst-case latencies of LLA to FLA against the link utilization. Increasing the link utilization beyond 0.5 decreases the ratio of the average worst-case latencies. This is because increasing the utilization means that the period of each flow decreases. When the period decreases, a larger number of higher priority flits preempt lower priority flits. This leads to increased interference, which in turn leads to a tighter analysis, and a smaller ratio. There is an interesting observation in this figure where the ratio of latencies decreases below a 0.5 link utilization. This happens because the period in these cases is greater than twice the basic link latency. Therefore, an interference between two flows of equal link latencies will only lead to one higher priority packet preempting the lower priority one. This decreases the time available for interference on a successor link, which minimizes the aggregated worst-case latency of LLA.

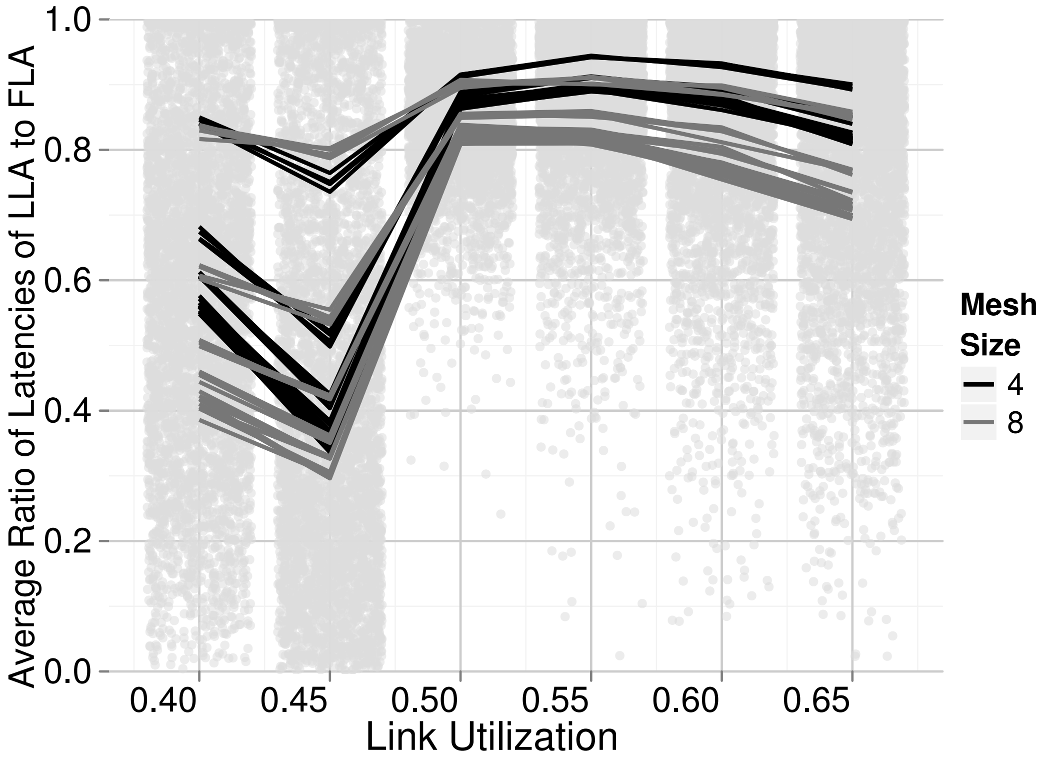

Figure 8.2: Ratio of latencies against the link utilization.

Figure 8.3 shows the ratio of unschedulable flows for LLA to FLA against the number of flows. Starting from 10 flows, increasing the number of flows up to 30 decreases the ratio of unschedulable flows because LLA accommodates more flows to meet their timing requirements. In general, beyond 30 flows, the difference between LLA and FLA is maintained, however, both increase leading to a slightly smaller ratio. 4×4 meshes with high link utilization, and lower deadlines because the number of unschedulable flows increases for both LLA and FLA leading to higher ratios.

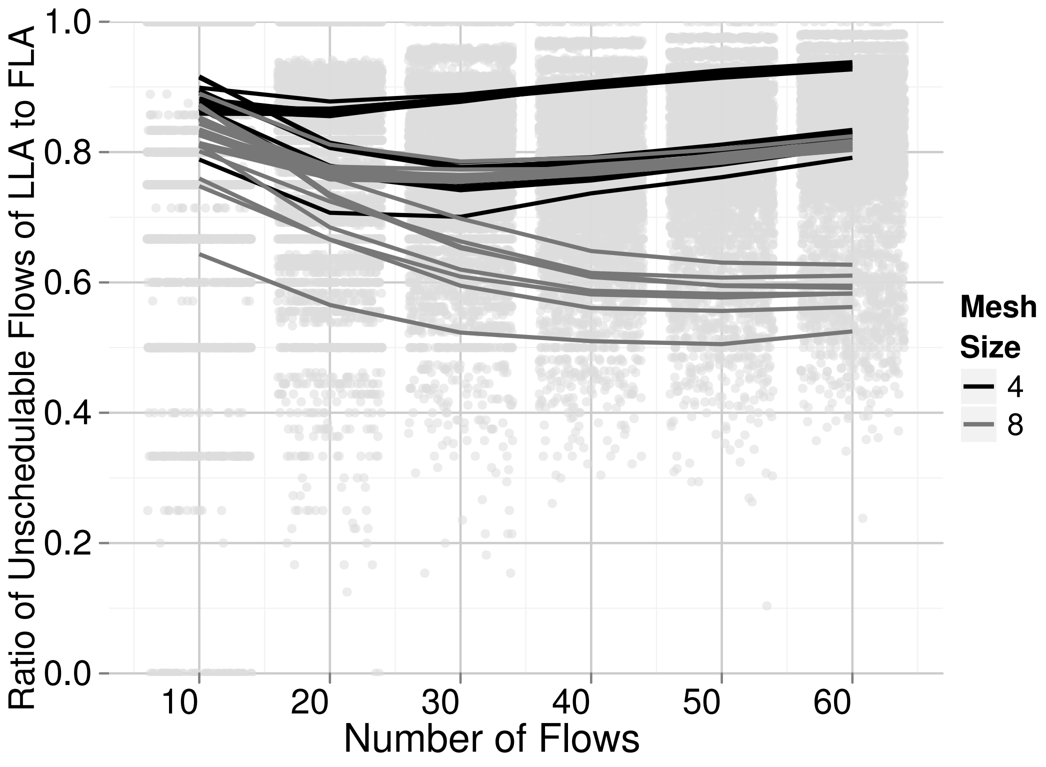

Figure 8.3: Ratio of unschedulable flows against the number of flows.

Figure 8.4 shows the ratio of unschedulable flows of LLA compared to FLA against link utilization. Increasing the link utilization beyond 0.5, maintains the ratio of unschedulable flows at nearly a constant ratio. This ratio is even less, below a 0.5 link utilization. The reason is the same as the one for the behavior in Figure 8.2. When the period is more than double the basic latency, interference between flows with equal link latencies leads to decreasing the probability of a lower priority packet suffering interference on a successor link.

This leads to the minimization of the overall worst-case latency of a flow and, hence, the schedulability of more flows. Configurations with a mesh size 8×8 have generally smaller ratios. In these configurations, the larger size of the NoC means longer paths for flows, which leads to a more prominent effect of the tightness of LLA compared to FLA.

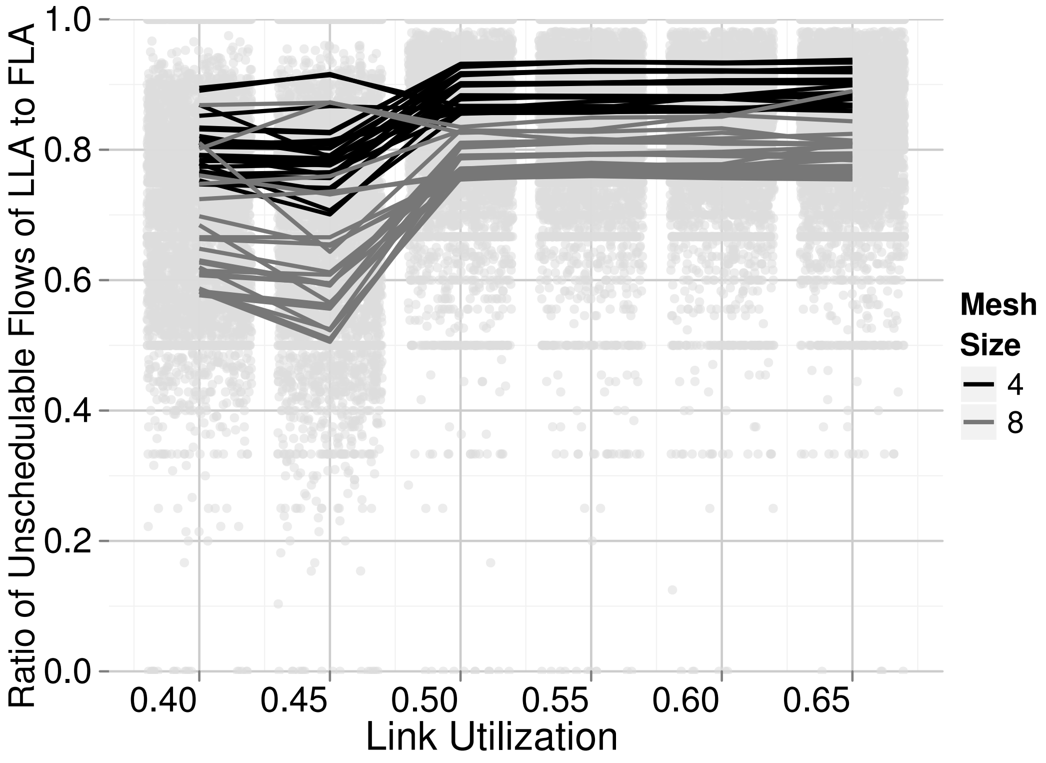

Figure 8.4: Ratio of unschedulable flows against the link utilization.

Figure 8.5 shows the average analysis time for each configuration against the number of flows for both LLA and FLA. The trend of the analysis is the same for both LLA and FLA. This means that when the interference is high for a certain configuration, the analysis time increases for LLA and FLA. The graph shows that the analysis time for LLA is approximately double that of FLA. We find this to be reasonable for the quality of results delivered by LLA. The average analysis time over all test cases for LLA is 66.2 mS and 38.2 mS for FLA. The maximum analysis time recorded for FLA is 429 mS while it is 619 mS for LLA.

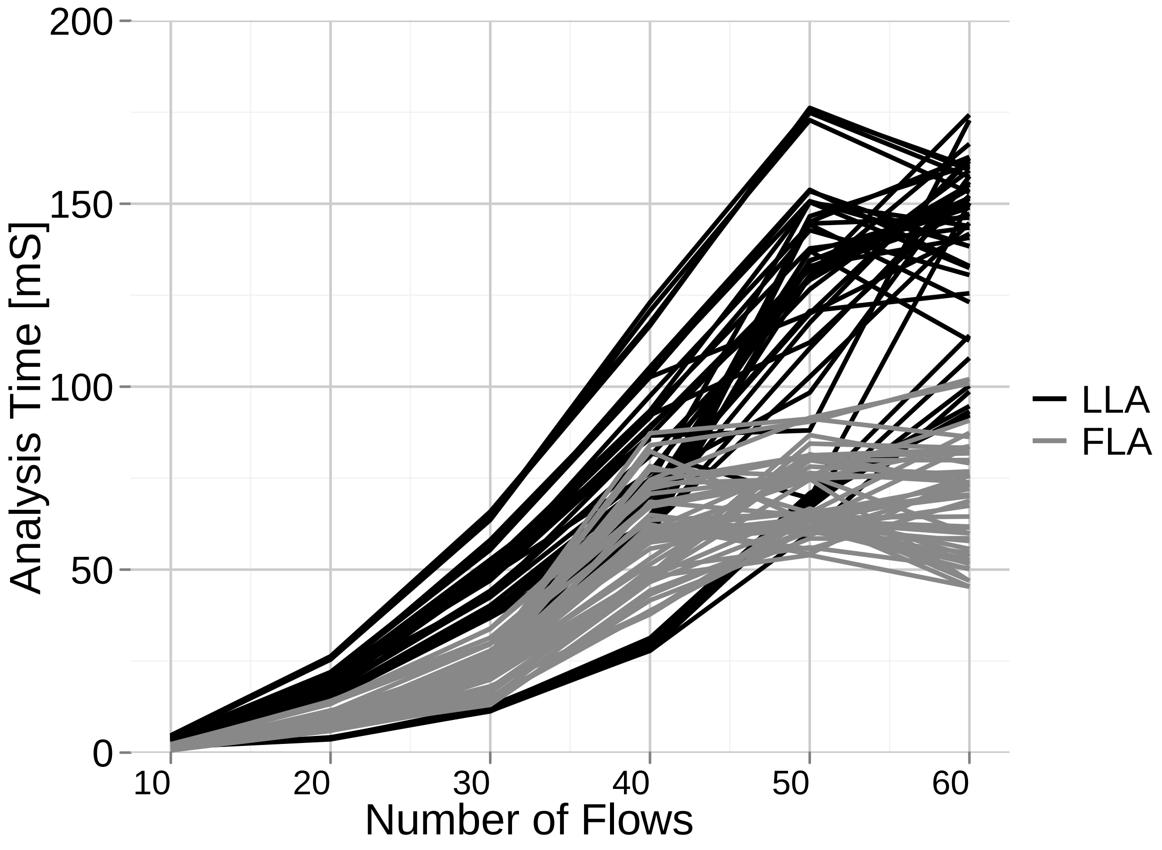

Figure 8.5: Analysis time against the number of flows.

In summary, for all $336,000$ test cases, the average worst-case latency is reduced by 31.7%, and the number of unschedulable flows is reduced by 13.7%. The ratio of unschedulable flow and ratio of worst case latencies never go beyond 1.0 for all test cases. This means that LLA is at worst the same as FLA, which verifies our tightness analysis. The analysis time of LLA is on average double that of FLA but still below one second. We use the Wilcoxon matched pairs test to reason about the significance of our results. The p-value for both the analysis, and the schedulability is less than 2.2×10^{-16}.

Chapter 9

Conclusion

This thesis presents an analysis that determines the worst-case latencies of communication across a PAR NoC. By performing this analysis at the link-level, we provide tighter upper-bounds than an existing technique that performs its analysis at the flow-level (FLA). Similar to FLA, LLA accounts for direct and indirect interferences from other communication flows; however, it also incorporates the effect of pipelined transmission across the NoC that FLA does not incorporate.

9.1 Contributions

We show that LLA provides results that are either tighter or equivalent to that of FLA. We illustrate this with an example that for a fixed topology, task mapping, and their respective paths, the number of schedulable flows when using LLA is higher than FLA. We experiment with synthetic benchmarks to show the strengths of LLA. Our results show an average improvement over FLA by approximately 31% for the worst-case latency estimate, and 13% in schedulability of flows. Another advantage of LLA is that it can be used for selecting paths that flows take in the NoC to improve schedulability. To further illustrate the application of LLA, we apply our analysis to a set-top box case study. In this case study, according to LLA, all flows are schedulable while two are unschedulable for FLA. For flows that do not suffer any interference, LLA and FLA give the same estimates, otherwise, LLA has tighter bounds.

9.2 Future Work

LLA assumes the communication flows are already mapped onto the NoC, and the flow priorities are known. We understand that a proper choice of the mapping and priority assignment has a considerable impact on the schedulability of the flows. In other words, improper mapping or priority assignment may result in low schedulability. Hence, our future work entails investigating mapping and priority assignment techniques that focus on maximizing schedulability of the flows.

Another area of future work observes that LLA only focuses on worst-case latencies for communications. However, applications consist of computation tasks that generate communication across the NoC. The communication is modelled as these flows, but LLA at the moment does not incorporate the latency of the computation tasks. Therefore, we plan to develop a thorough analysis that considers both communications and computations tasks, and provides the worst-case latency analysis for the entire application.

References

[1] Neil Audsley, Alan Burns, Thomas Richardson, Ken Tindell, and Andy Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284 –292, sep 1993.

[2] Luca Benini and Micheli De Micheli. *Networks on Chips: Technology And Tools*. The Morgan Kaufmann Series in Systems on Silicon. Elsevier Morgan Kaufmann Publishers, 2006.

[3] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38, June 2006.

[4] Tobias Bjerregaard and Jens Sparso. A Scheduling Discipline for Latency and Bandwidth Guarantees in Asynchronous Network-on-Chip. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 34–43, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[6] Bui D. Bach, Rodolfo Pellizzoni, and Marco Caccamo. A slot-based real-time scheduling algorithm for concurrent transactions in noc. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 329 –338, aug. 2011.

[7] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks, 2001.

[8] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[9] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[10] Edwin Rijpkema et. al. Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip. *Computers and Digital Techniques, IEEE Proceedings -*, 150(5):294–302, sept. 2003.

[11] Gustavo B. Figueiredo, Nelson L. S. da Fonseca, and José A. S. Monteiro. A Minimum Interference Routing Algorithm with Reduced Computational Complexity. *Computer Networks*, 50:1710–1732, August 2006.

[12] Crispín Gómez, María E. Gómez, Pedro López, and José Duato. Reducing packet dropping in a bufferless noc. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Euro-Par '08, pages 899–909, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] Roch A. Guerin, Ariel Orda, and Douglas Williams. QoS Routing Mechanisms and OSPF Extensions. In *Proceedings of IEEE GLOBECOM*, pages 1903–1908, 1996.

[14] Jingcao Hu and Radu Marculescu. Dyad - smart routing for networks-on-chip. In *ACM/IEEE Design Automation Conference*, pages 260–263, 2004.

[15] Dilip D. Kandlur, Kang G. Shin, and Domenico Ferrari. Real-Time Communication in Multihop Networks. *IEEE Trans. Parallel Distrib. Syst.*, pages 1044–1056, 1994.

[16] Koushik Kar, Murali Kodialam, and T. V. Lakshman. Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Application. *IEEE Journal on Selected Areas in Communications*, page 2579, 2000.

[17] Hany Kashif, Hiren D. Patel, and Sebastian Fischmeister. Using Link-level Latency Analysis for Path Selection for Real-time Communication on NoCs. pages 499–504, January 2012.

[18] Nikolay Kavaldjiev, Gerard J. M. Smit, Pierre G. Jansen, and Pascal T. Wolkotte. A Virtual Channel Network-on-Chip for GT and BE Traffic. In *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, pages 211–221, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Andrei Radulescu Kees Goossens, John Dielissen. Æthereal Network on Chip: Concepts, Architectures, and Implementations. *IEEE Design and Test*, 22(5):414 – 421, 2005.

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

[21] Insup Lee, Joseph Y-T. Leung, and Sang H. Son. *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC, 1st edition, 2007.

[22] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[23] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[24] Zhonghai Lu and Axel Jantsch. TDM Virtual-Circuit Configuration for Network-on-Chip. *IEEE Transactions on Very Large Scale Integrated Systems*, 16:1021–1034, August 2008.

[25] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch. Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, DATE'04, pages 20890–, Washington, DC, USA, 2004. IEEE Computer Society.

[26] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, February 1993.

[27] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case. *IEEE/ACM Trans. Netw.*, pages 344–357, 1993.

[28] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Trans. Netw.*, pages 137–150, 1994.

[29] Erno Salminen, Ari Kulmala, and Timo D. Hamalainen. Survey of Network-on-chip Proposals. White Paper, OCP-IP, 2008.

[30] Sumant Sathe, Daniel Wiklund, and Dake Liu. Design of a switching node (router) for on-chip networks. In *ASIC, 2003. Proceedings. 5th International Conference on*, volume 1, pages 75 – 78 Vol.1, oct. 2003.

[31] Julien Schmaltz and Dominique Borrione. A generic network on chip model. In *Theorem Proving in Higher Order Logics*, volume 3603 of *Lecture Notes in Computer Science*, pages 310–325. Springer Berlin, 2005.

[32] Zheng Shi. *Real-Time Communication Services for Networks on Chip*. PhD thesis, The University of York, UK, 2009.

[33] Zheng Shi and Alan Burns. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 161–170, Washington, DC, USA, 2008. IEEE Computer Society.

[34] Zheng Shi and Alan Burns. Schedulability Analysis and Task Mapping for Real-Time on-Chip Communication. *Real-Time Syst.*, 46:360–385, December 2010.

[35] Ganapati Srinivasa. Evolution of the server processor/platform architecture and the critical role of interconnect and future challenges. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, pages 1–1, New York, NY, USA, 2010. ACM.

[36] Subhash Suri, Marcel Waldvogel, and Priyank Ramesh Warkhede. Profile-Based Routing: A New Framework for MPLS Traffic Engineering. In *Proceedings of International Workshop on Quality of Future Internet Services*, COST 263, pages 138–157, London, UK, 2001. Springer-Verlag.

[37] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *Micro, IEEE*, 22(2):25 – 35, mar/apr 2002.

[38] Jens Spars Tobias Bjerregaard. Implementation of Guaranteed Services in the MANGO Clockless Network-on-Chip. *IEEE Proceedings of Computing and Digital Techniques*, 153:217–229, 2006.

[39] Daniel Wiklund and Dake Liu. SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 78.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[40] Pascal T. Wolkotte, Gerard J. M. Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. An energyefficient reconfigurable circuit switched network-on-chip. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS05) - 12th Reconfigurable Architecture Workshop (RAW 2005), p. 155a, ISBN*, pages 0–7695, 2005.