Packet-Level Static Timing Analysis for On Chip Networks

Evgeni Krimer

Packet-Level Static Timing Analysis for On Chip Networks

RESEARCH THESIS

In Partial Fulfillment of The Requirements for the Degree of Master of Science in Electrical Engineering

Evgeni Krimer

Submitted to the Senate of the Technion - Israel Institute of Technology
Sivan 5769 — Haifa — May 2009

The research thesis was done under the supervision of Dr. Isaac Keslassy and Dr. Avinoam Kolodny in the Faculty of Electrical Engineering.

Contents

List of Figures

Abstract

Networks-on-chip (NoCs) are used in a growing number of Systems-on-Chip (SoCs) and Chip Multi Processors (CMPs), increasing the need for accurate and efficient modeling to aid the design of integrated systems. Such modeling is required for many decisions that need to be made during the design flow, such as resource allocation, placement, and QoS assurance. With no such modeling available today, the choice is limited to simulations, which are very time-consuming and often decrease the efficiency of the design.

We present a methodology for packet-level Static Timing Analysis (STA) in NoCs. It enables quick and accurate gauging of the performance parameters of a virtual-channel (VC) wormhole NoC without using simulation techniques. The network model can handle any topology, link capacities, and buffer capacities. It provides per-flow analysis that is orders-of-magnitude faster than simulation while being both significantly more accurate and more complete than prior static modeling techniques. Our methodology is inspired by models of industrial flow-lines. Using a carefully derived and reduced Markov chain, the model can statically represent the dynamic network state and closely estimate the average latency of each flow. Simulations are used to validate and evaluate the model.

As an example application, we apply our model in a placement optimization scenario. We show that our model can accurately choose between placement options to minimize end-to-end delay as verified by detailed simulation. This example also demonstrates the disadvantages and inaccuracies of prior models, which choose a sub-optimal solution over a more optimal placement.

List of Acronyms

NoC	—	Network on Chip
SoC	—	System on Chip
CMP	—	Chip Multi Processor
QoS	—	Quality of Service
STA	—	Static Timing Analysis
$VLSI$	—	Very Large Scale Integration
MC	—	Markov Chain
VC	—	Virtual Channel
HDM	—	Heuristic Delay Model
CAD	—	Computer-Aided Design

List of Symbols

λ_α	—	packet arrival rate of flow α [packets per unit time]
M_α	—	packet length of flow α [flits]
$\phi^{(l)}$	—	capacity of link l [flits per unit time]
s_i	—	Markov chain state i
π_i	—	stationary distribution probability of state s_i
$\overline{\pi}$	—	stationary distribution probability (vector of π_i)
γ_i	—	fraction of packets of a measured flow (X) served while in state s_i
ρ_i	—	NoC throughput associated with state s_i for flow X [packets per unit time]
η_i	—	head-flit propagation delay of packets in flow X incurred while in state s_i [time units]
τ_α	—	expected time to fully transmit a packet of flow α [time units]
$\Delta^{(l)}$	—	buffer capacity on link l for flow X
$\delta^{(l)}$	—	buffer occupancy on link l for flow X
f_α	—	transmission completion probability for current packet of flow α
c_α	—	transmission incompletion probability for current packet of flow α
T_X	—	expected throughput observed by flow X [packets per unit time]
W_X	—	expected waiting time in the source node input buffer for flow X packets [time units]
H_X	—	expected propagation time of flow X packets head flit [time units]
L_X	—	expected delay of packets in flow X [time units]
S_X	—	average service time for packets in flow X $(\equiv \frac{1}{T_X})$ [time units]
$\sigma^2_{S_X}$	—	variance of the average service time for packets in flow X
$C^2_{S_X}$	—	coefficient of varation of the average service time for packets in flow X
ξ	—	delay error
F	—	total number of flows in the network
R	—	total number of nodes/routers
Ψ_α	—	number of interfering flows for flow α
P_α	—	number of hops between the source and the destination for flow α

Chapter 1

Introduction

1.1 Network on Chip

Networks-on-chip (NoCs) are increasingly used instead of buses and dedicated signal wires in large-scale processors and, even more so, in modern systems-on-chip (SoC) [5]. Predictions [1] quote tens and hundreds of interconnected processing units in several years.

Today there already are existing commercial chips with such properties. *Cisco Quantum-Flow Processor* with 40 cores, *Intel Teraflops research chip* with 80 cores, *NVidia Tesla C870* with 128 cores and *Cisco Silicon Packet Processor* with 188 cores. While there is no intention to claim that such numbers of cores are the mainstream today, their existence definitely points out the trends of tomorrow.

In NoC-based systems, data transmission takes the form of multi-packet flows routed through the NoC over multiple links and routers. Each module is connected to the network through a router and routers are connected among themselves. Different interconnection topologies are mentioned in the literature. Most common are mesh, torus and incomplete mesh.

1.2 Wormhole Switching

Wormhole Switching, which is also often referred to as *Wormhole Routing* [29], is a flow control technique widely used for decades in different areas from local computer clusters [27] to SoC, NoC [13] and SpaceWire [32,33] chips. That's since it minimizes the amount of buffering required in the routers across the network.

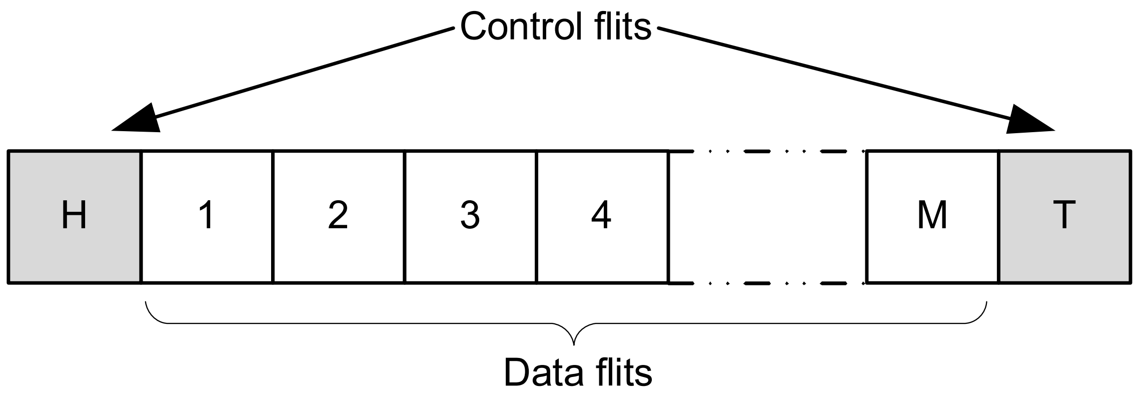


Figure 1.1: Wormhole packet structure

Large packets are broken into small (constant length) units called *flits* (flow control digits). Two control flits are added, a *head* flit at the beginning and a *tail* flit at the end of each packet as shown in Figure 1.1. Here the packet consists of M (data) flits while H and T represent head and tail (control) flits. The head flit contains all the routing relevant information and the subsequential flits follow its path. The tail flit is used to signal the end of the message and for various “bookkeeping” purposes. This behavior is depicted in Figure 1.2. The packet information propagates as a long worm of flits, hence the name of the wormhole routing.

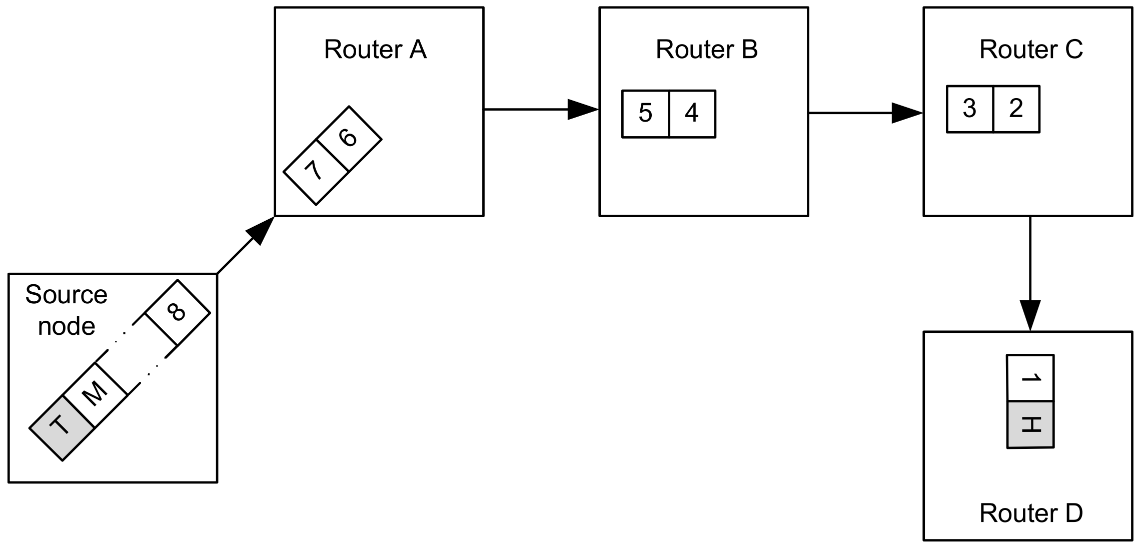


Figure 1.2: Example of wormhole packet propagation in network

1.3 Virtual Chanels

Using the *Wormhole Switching* technique has a few disadvantages. Once the head flit has been transmitted and until the tail flit is transmitted, the link is “allocated” to a specific packet and cannot be used for transmission of other packets. In case the head flit arrives to a router where a required link is allocated, the packet will be stalled until the link is available. During that time all the links that the head flit has already gone through but the tail flit has not, will not be used for any other packets. This has a strong performance impact.

The *Virtual Channels* (VC) [13] feature allows a router to serve a few packets simultaneously which logically can be interpreted as having multiple links. This is done by additional logic in the router. The logic is responsible for maintaining the status of several flows (the maximum number of maintainable flows is referred to as number of VCs). Upon arrival of a head flit, a VC is allocated. It is deallocated once the tail flit passes.

During normal operation the active (allocated) VCs are served. Different serving policies are mentioned in the literature [36].

1.4 Architecture

In the literature there are different approaches for the NoC architecture. This implies different network topologies and different router (node) structures. Moreover, there are different parameters in different NoC architectures. In this section we'll describe our architecture and assumptions.

1.4.1 Network

Unlike other previous NoC models which assume specific topologies and identical link parameters, we do not make any such assumptions. We do assume a deterministic routing (although the model can be probably adapted for a statistic routing as well as discussed in Chapter 5).

1.4.2 Routers

Each NoC node is a router. All the different endpoints (cores, caches, etc.) are connected to the NoC through a router. The basic router structure is presented on Figures 1.3a, 1.3b and 1.3c. In the same way, as there are no special requirements/assumptions on the network topology and transmission lines, there are no requirements on routers for being identical or symmetric. Different routers might have different numbers of ports and even the same router may have a different number of virtual channels per port.

However, in this work we do assume that there are "enough" VCs for all the flows which means that there is no blocking (due to lack of VCs) situation.

The scheduler block is responsible for selecting the flit to be sent to the out port among all the available input ports VCs. We assume that it acts in a round-robin manner. This behavior is chosen over the others mentioned in the literature [36], since it is easy for implementation in hardware and provides fairness. However, the approach used to develop the analytical model in this work can be used to develope models for other scheduling methods as well.

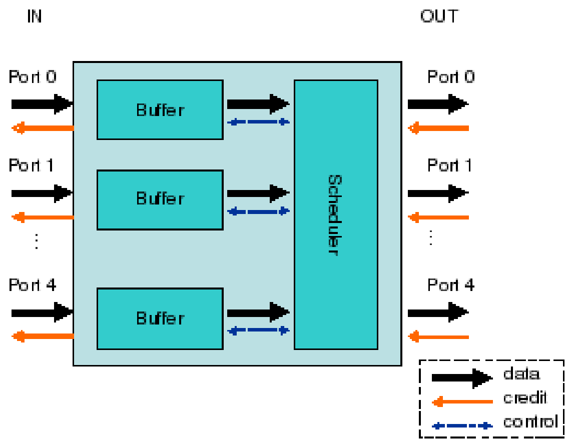
1.5 Related Work

Much of the prior work on analytical delay modeling in wormhole-enabled networks approximates the mean delay of packets in the entire system rather than estimating the delay of each source-destination flow separately [4, 21, 23, 26, 31]. Such gross approximations are often inadequate, and in such cases cannot be used in the NoC design process to efficiently optimize the allocation of resources.

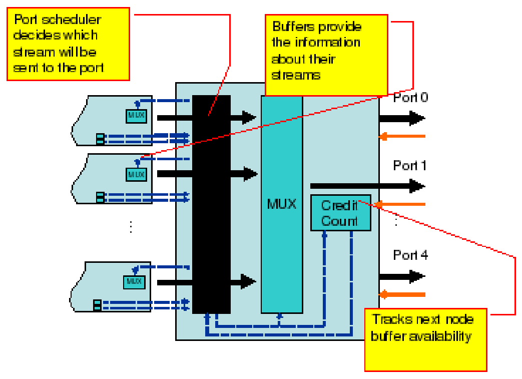
In [31], the authors develop an analytical model. Their approach relies on calculating the network delay and using the M/G/1 model to estimate the waiting time. A similar approach is used in all the later works well as in this one. However, the calculation of the network delay varies.

In [26], authors analyze the traffic behavior in the spidergon scheme with a deterministic routing and uniform traffic. Each physical link is assumed to share exactly *two* virtual channels. Regardless of the blocking, the arrival at each channel is approximated to be a Poisson process. The total traffic on each physical link is considered rather than the traffic on each individual virtual channel.

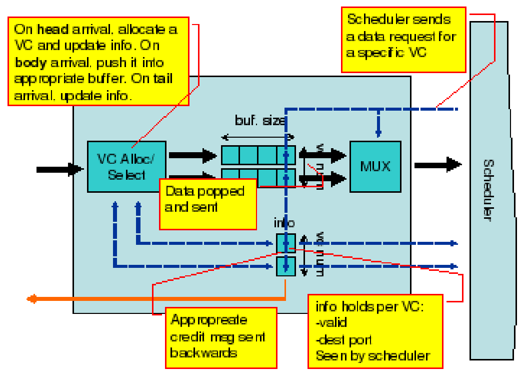
A stochastic model to predict the average message latency in k-ary n-cubes with deter-



(a) High-level router structure with input buffers


Port scheduler decides which stream will be sent to the port
Buffers provide the information about their streams
MUX
MUX
MUX
MUX
Port 0
Port 1
Port 4
Credit Count
Tracks next node buffer availability

(b) Router scheduler


On head arrival, allocate a VC and update info. On body arrival, push it into appropriate buffer. On tail arrival, update info.
Scheduler sends a data request for a specific VC
buf. size
VC Alloc/ Select
Data popped and sent
MUX
info
Scheduler
Appropreate credit msg sent backwards
info holds per VC:
-valid
-dest port
Seen by scheduler

(c) Input buffer structure

Figure 1.3: Internal router structures

ministic routing in the presence of hot-spot traffic is presented in [23]. Authors assume exactly *two* virtual channels, a constant message length and Poisson arrival process.

[21] analyzes a torus network topology with deterministic routing and wormhole switching under Poisson arrival process and uniform traffic pattern. Unlike the previous work, it assumes an arbitrary number of virtual channels. An average of all the flow delays across the network is modeled and it is used in the derivation of the waiting time using the M/G/1 model.

None of [21, 23, 26, 31] refer to intermediate buffer sizes. However, as shown in Section 3.4, they have a direct impact on the end-to-end latency. [4] claims to capture the effects of finite buffers on the performance. Its authors analyze the performance of wormhole switching on k-ary n-cubes assuming deterministic routing, fixed message length and Poisson arrival process.

In addition, while state-of-the-art NoC architectures multiplex multiple packets on the network links using virtual channels [6, 7, 25, 28, 37], most existing analytical models do not support virtual channels [10, 12, 15, 30, 39].

Further, in [19, 24, 38], the authors formulate worst-case latencies of flows in the NoC. While this approach is suitable for real-time flows with hard deadlines, the vast majority of communication flows in typical SoCs has a set of more relaxed timing requirements, which can be satisfied with statistical guarantees.

A heuristic approach to estimate the average delay of each flow was taken in [18]. The authors developed a *heuristic delay model* (HDM) that takes into account the capacities of all the links traversed by the modeled flow, as well as the bandwidth consumed by all other network flows which share some links with the modeled flow. Their heuristics attempt to estimate the serialization and head-of-line blocking, which add to the delays of a link because of congestion further downstream. This approach is useful, providing a closed-formed formula to estimate the delay for each flow in the network based on traffic parameters of all the flows. However, the model uses heuristics and its accuracy has not been confirmed in a rigorous fashion. In addition, [18] does not capture the effects of finite buffer sizes. In Chapter 3, we compare this model against ours and show how its heuristic approach can lead to a wrong optimization decision.

1.6 Contributions

The purpose of this work is to rigorously derive a delay model for packet-level *static timing analysis (STA)* for NoC-based SoCs. Static timing analysis in a shared network is a non-simulation-based technique to estimate the average delay of each flow in the network, given the network topology, link capacities, router architecture, and the bandwidth requirements and characteristics of all flows.

The motivation for a per-flow STA technique is to enable a range of design optimizations that can rely on accurate and fast network analysis. Methods such as module placement and resource allocation [2,35] require a large number of iterations, and thus the evaluation of network performance within each iteration must be very efficient. Until now, an accurate and complete modeling of advanced NoCs has only been possible with detailed and time-consuming simulations. The main reason is that network resources, including links, routers, buffers, and ports, are shared between several information flows. Thus, contention can arise inducing *statistical uncertainty* in the delay of each packet. Detailed simulation, however, is too slow to be effective within an optimization inner loop because all internal buffers and states must be modeled on a cycle-by-cycle basis.

We present a rigorous analytical model that relies on a carefully constructed and reduced Markov chain to represent network state, including the occupancy of all buffers. Our model is inspired by industrial work-flow modeling techniques and, to the best of our knowledge, is the first that can accurately account for arbitrary network topology, link capacities, and buffering, when using wormhole routing with virtual channels. We rely on the well-developed theory of stochastic processes and show that our technique faithfully predicts network queuing delay for both synthetic and real-world SoC traffic scenarios. In this work we limit the analysis to packets that have random arrival times according to a Poisson distribution. We present results and validate the model for the delay analysis of flows with fixed-length packets that are composed of a large number of flits. We discuss extensions to these assumptions as future work.

To summarize our contributions:

- We present the first rigorous NoC model that is based on stochastic theory and show how to represent and solve for the network state using a Markov chain.
- We show how to account for arbitrary and finite buffering, as well as support wormhole routing and virtual channels. We use network delay analysis as an illustrative example of the modeling technique

- We validate our model using synthetic and real-world scenarios, and discuss why it is more complete and more accurate than prior analytical models.
- We demonstrate that our model can serve at the core of a design optimization method by showing that it can faithfully choose between multiple placement options in a real-world SoC example, and do so while requiring *orders of magnitude* less time than simulation. We also show that the most advanced prior-art model fails to make the correct optimization decision.

Chapter 2

Analytical Model

Our model supports an arbitrary NoC topology with wormhole routing and virtual channels. The capacity of each link in the network may be set arbitrarily. Likewise, the capacities of the buffers in each virtual channel are arbitrary as well. We assume that all packets have a fixed length, and that the packet arrival times at the injection port of each node can be modeled by a Poisson random process. In Chapter 5 we discuss extensions to our model that relax these assumptions on packet length and distribution. We also assume that there is no blocking in the network due to a lack of virtual channels, and that the destination node can always eject packets from the network. Finally, we place no restriction on the routing algorithm except that it be deterministic.

Our technique follows three main steps:

1. We focus on the NoC service for a particular flow of interest, which we generically call *flow X*, and model it using a Markov chain (MC) [8]. The Markov chain represents the network state of the routers and buffers on the path of flow X, as well as the impact of *interfering flows*, i.e., those flows that share at least one link with flow X.

2. We derive the flit propagation characteristics by computing the stationary distribution of the Markov chain.

3. We use the derived properties and standard analysis of M/G/1 queues to calculate the expected packet delay and the throughput of flow X.

2.1 Constructing the Markov Chain

2.1.1 The Reduced Configuration

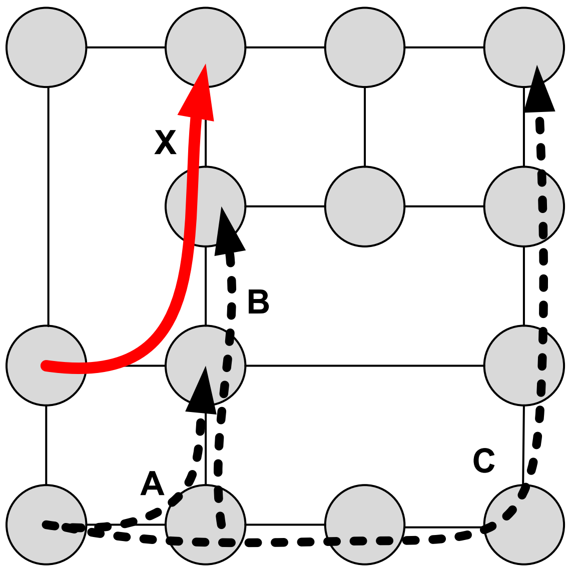
To fully represent the NoC as a Markov chain, the internal state of each router (and in particular the buffer occupancies) as well as the characteristics of all flows need to be expressed as states in the chain. Unfortunately, this naive approach would result in an enormous and intractable number of states.

As shown in the transition from Figure 2.1a to Figure 2.1b, to reduce this Markov chain to a manageable size, we generate a separate model for each ”isolated” flow, generically represented as flow X. We call this model the *reduced configuration*. In the reduced configuration, we limit the analysis to the routers on the path of X, and only consider those other flows that share network links with X, such as flows A and B, but not flow C (this particular reduction method was also used in [17, 18, 40]).

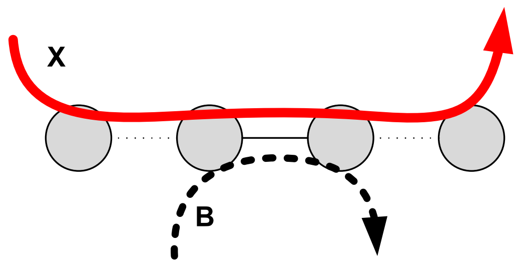
To further simplify the Markov chain, we restrict our analysis to epochs in which the flits of flow X are waiting to be served by the NoC. This last assumption eliminates the need to model the large buffers at the network injection points and permits to model them as infinite buffers. Without assuming that flow X is active, we would need to track the state of the network during periods of inactivity, which would complicate the MC representation. We discuss the implications of this simplification in Section 3.2.

The reduced configuration for flow X can be viewed as a Markov chain in which each state is defined by the buffer occupancies and the existence of interfering flows along the links, as shown in Figure 2.1c and explained in Section 2.1.2. We construct a specific Markov chain for flow X to model its reduced configuration. This Markov chain representation accounts for both the extra queuing delay caused by interfering flows, which share links and ports with flow X, as well as for the delay of serialization and back-pressure within flow X and between the flits of a single packet (Figure 2.1e).

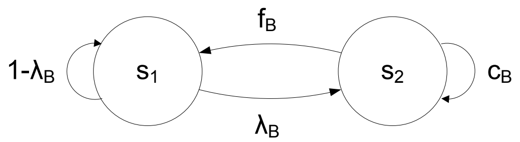
This system is equivalent to an open queuing network (Figure 2.1e) and to a manufacturing flowline (Figure 2.1d) with unreliable parallel machines [14]. By casting our problem in terms of unreliable machines, we can leverage a large body of works on stochastic theory and modeling methodology. We represent each hop taken by flow X as a production station consisting of a group of parallel machines. Likewise, in each cycle, the router is modeled as choosing a new machine in this group of parallel machines, in a round-robin fashion. A functioning machine processes flow X and contributes to its throughput, while a malfunctioning machine is equivalent to the link being used by an interfering flow.



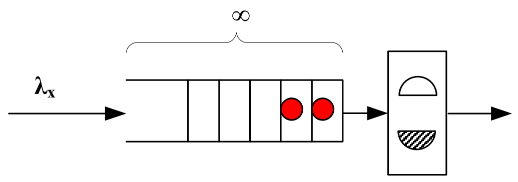
(a) Full configuration



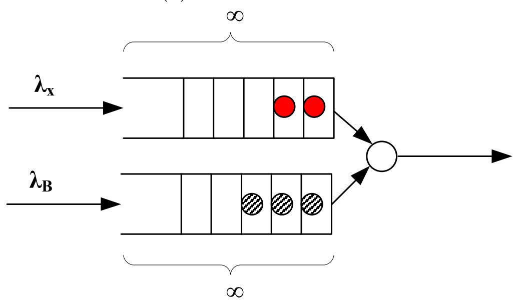
(b) Reduced configuration



(c) Associated MC



(d) Flowline representation



(e) Queue representation

Figure 2.1: Flow X sharing a single link with a single interfering flow.

In the following subsections we show how to construct the Markov chain for a number of representative interference patterns and conclude the analytical model section by deriving delay and throughput using the Markov chain (Section 2.2).

2.1.2 Sharing a Single Link with a Single Flow

In this scenario, measured flow X shares a single link with interfering flow B, as shown in Figure 2.1, which also depicts the flow-line and queue representations and the associated MC. The MC, which represents this single-interferer case, requires only two states (Figure 2.1c). State s_1 represents a “no interference” situation, where flow X can use the entire capacity of the shared link. If there are flits of both flows X and B waiting in the buffers, as illustrated in Figure 2.1e, the round robin arbitration mechanism allocates the shared link to each flow on every other cycle. As a result, each flow can utilize only half of the link capacity. State s_2 represents the situation where flow B interferes with the measured flow X. The MC does not need to represent a situation where only flow B is active, because it need only model the network as seen by flow X, and a separate MC is constructed to model the network properties of flow B.

The MC representation also expresses the probabilities of transitioning between the states, denoted in Figure 2.1c by the labels on the arrows connecting the two states. Starting from state s_1, where only flow X is active, the probability of transitioning to state s_2 is simply the probability that a new packet of flow B arrives. Assuming Poisson arrival times, this probability is simply (λ_B). Conversely, the probability of staying in s_1 is $(1 - \lambda_B)$. Starting from state s_2, where both flows are active, the probability of transitioning to s_1 is the probability that the current packet of flow B is fully transmitted (f_B) and that no new packet of flow B has arrived during this transmission time. The time required to transmit a packet is the length of the packet (M_B) divided by the available link capacity ϕ, which is only half of the total capacity because flow X is also active: $\left(\tau_B = \frac{M_B}{\frac{1}{2}\phi}\right)$. Thus, the probability of fully transmitting a packet at any time is $\frac{1}{\tau_B}$. The probability for another packet to appear during this time, which prevents transitioning back to s_1 is $(\lambda_B \tau_B)$. Therefore, the probability for transitioning from state s_2 to state s_1 is modeled by:

$$f_B = \frac{1}{\tau_B} \max\left(1 - \tau_B \lambda_B, 0\right) = \max\left(\frac{\phi}{2M_B} - \lambda_B, 0\right)$$

Finally, the probability for flow B to continue being active, and the MC to remain in state s_2 is modeled by:

$$c_B = 1 - f_B = \min\left((1 - \frac{\phi}{2M_B}) + \lambda_B, 1\right)$$

Section 3.2 shows how to use this MC to derive delay and throughput properties for flow X, validates the results with detailed simulation, and discusses the implications and comparison to HDM.

2.1.3 Sharing a Single Link with Multiple Flows

A configuration where the measured flow shares a link with two other flows is shown in Figure 2.2 along with its flowline/queue equivalence and associated MC. Here, state s_1 represents no interference, states s_2 and s_3 represent inference by only flow A or only flow B respectively (i.e., flits of a single flow other than flow X are being multiplexed on the same link), and s_4 represents the state in which flits of all three flows (A, B and X) are multiplexed on the same link.

With the additional states, the MC is more complex and has a larger number of possible transitions. The evaluation of the transition probabilities is similar to the derivation discussed above. For state s_1, the probability of staying in this state of no interference is the probability that no new packets arrive on flow A and no new packets arrive on flow B ($(1-\lambda_A)(1-\lambda_B)$). Conversely, a transition from s_1 to s_4 occurs when both flows A and B become active at the same time ($\lambda_A \lambda_B$). The probability of transitioning from s_1 to s_2 is that of a packet arriving on flow A and not arriving on flow B, while the opposite is true for a transition from s_1 to s_3.

We now discuss the transition probabilities from state s_2. When a packet of flow A is fully transmitted and no other packets of both flows A and B arrived during the transmission time, then the MC transitions back from s_2 to s_1. To model the probability of fully transmitting a packet, marked as f_A, we need to first derive the expected transmission time of a packet from flow A (τ_A). When a packet of flow A is being serviced, it can be done while at state s_2 at a rate of $\frac{\phi}{2}$, or in state s_4 with a rate of $\frac{\phi}{3}$ (because all three flows are active in s_4 but only two in s_2). Therefore, the expected transmission time is given by the expected fraction of time the packet is in states s_2 and s_4, which are related to the stationary distribution probabilities of these states (π_2 and π_4 respectively):

$$\begin{aligned}\tau_A &= \frac{M_A}{\frac{1}{2}\phi} \cdot \frac{\pi_2}{\pi_2+\pi_4} + \frac{M_A}{\frac{1}{3}\phi} \cdot \frac{\pi_4}{\pi_2+\pi_4} \\ &= \frac{2M_A}{\phi}\left(1 + \frac{1}{2}\frac{\pi_4}{\pi_2+\pi_4}\right)\end{aligned}$$

Given the expected transmission time, we can now model the probability of fully transmitting a packet as:

$$f_A = \frac{1}{\tau_A} \max\left(1 - \tau_A \lambda_A, 0\right) = \max\left(\frac{1}{\tau_A} - \lambda_A, 0\right)$$

Thus, the probability of transitioning from s_2 to s_1 is $(f_A(1 - \lambda_B))$. The probability for moving from s_2 to s_3 is that of fully transmitting the packet from flow A while a packet from flow B arrives at the same time, or $(f_A \lambda_B)$. The final two transitions are of remaining in s_2 or changing to s_4, which occur when flow A continues and either a packet from flow B does not arrive (stay in s_2), or does arrive (move to s_4). As before, we mark the probability of flow A continuing activity as:

$$c_A = 1 - f_A = \min\left((1 - \frac{1}{\tau_A}) + \lambda_A, 1\right)$$

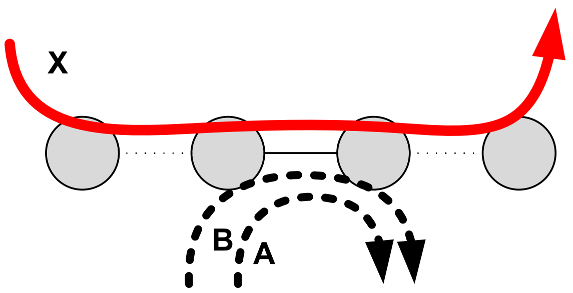
The transitions out of state s_3 are derived in the exact way as for state s_2, reversing the roles of flow A and flow B. With respect to state s_4, the probability of transitioning to s_1 is the probability that both flows A and B are fully transmitted, or $f_A f_B$. Transitioning from s_4 to s_2 occurs when flow A continues and flow B finishes and the probability is $c_A f_B$. In symmetric fashion, the probability of s_4 to s_3 is $c_B f_A$. Finally, the probability of staying in state s_4 is $c_A c_B$ signifying that all flows remain active.

2.1.4 Sharing Multiple Links

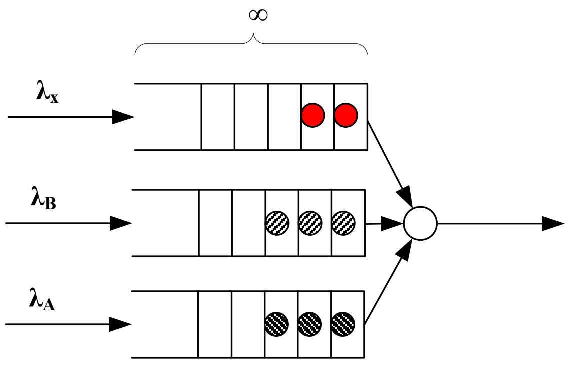
A configuration of interfering flows over multiple links, including the equivalent queuing and flowline representations, is shown in Figure 2.3. Unlike the previous cases, this scenario includes a finite buffer that has to be taken into account. Flow X passes through two routers, each with an interfering flow, and therefore can experience back-pressure from the intermediate node that does not have the infinite buffers assumed on the network injection and ejection packet queues. We assume that the intermediate flit queue has a depth of Δ flits, and will block transmission of an upstream node when it is full (Figure 2.3b).

We explicitly model the occupancy of the intermediate buffer in the MC, by dedicating states to each possible buffer occupancy level. We show this in Figure 2.4 for the case where all link capacities are equal. The figure has three parts: Figure 2.4a shows only those states of the MC that correspond to an occupancy level of δ; Figure 2.4b shows a schematic symbol that represents the partial MC of Figure 2.4a; and Figure 2.4c is the entire MC, using the schematic representation to simplify the figure.

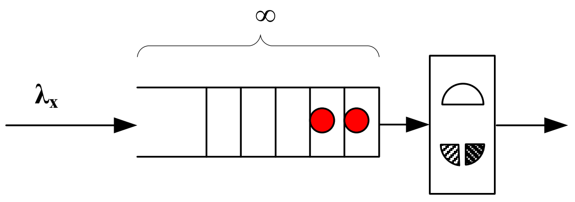
Focusing on Figure 2.4a, state $s_{1,\delta}$ represents the case where only flow X is active and the buffer has occupancy level δ. State $s_{2,\delta}$ represents the case where flows A and X



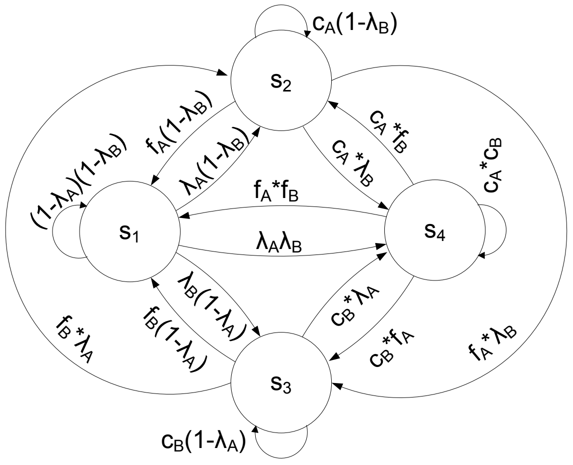
(a) Reduced configuration



(b) Queue representation



(c) Flowline representation



(d) Associated MC

Figure 2.2: Flow X sharing a single link with multiple interfering flows.

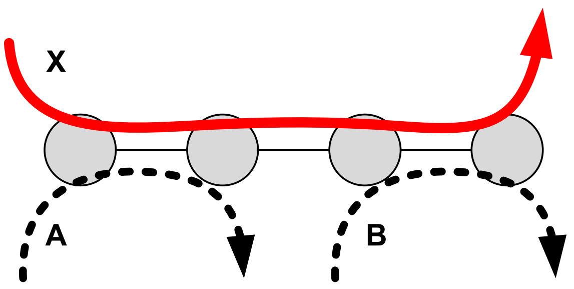
are active but flow B is inactive and $s_{3,\delta}$ is for when flows B and X are active but flow A is inactive. Finally, $s_{4,\delta}$ represents activity on all three flows and occupancy level δ. Starting from $s_{1,\delta}$ the buffer occupancy is not going to change in any scenario, because both routers are servicing flow X at the same rate and the buffer is emptied and filled at the same rate. The transition probabilities out of this state follow the same derivation descried in Section 2.1.3. Similarly, state $s_{4,\delta}$ cannot change the occupancy because the service rate for flow X is equal to $\frac{\phi}{2}$ at both routers. Again, the transition probabilities out of $s_{4,\delta}$ follow the reasoning presented in Section 2.1.3. Because each link has only two multiplexed flows, however, the probability for fully transmitting a packet of the flow interfering with flow X is derived in the same manner as in Section 2.1.2:

$$f_\alpha = \max\left(\frac{\phi}{2M_\alpha} - \lambda_\alpha, 0\right)$$
$$c_\alpha = 1 - f_\alpha = \min\left((1 - \frac{\phi}{2M_\alpha}) + \lambda_\alpha, 1\right)$$
$$\alpha \in \{A, B\}$$

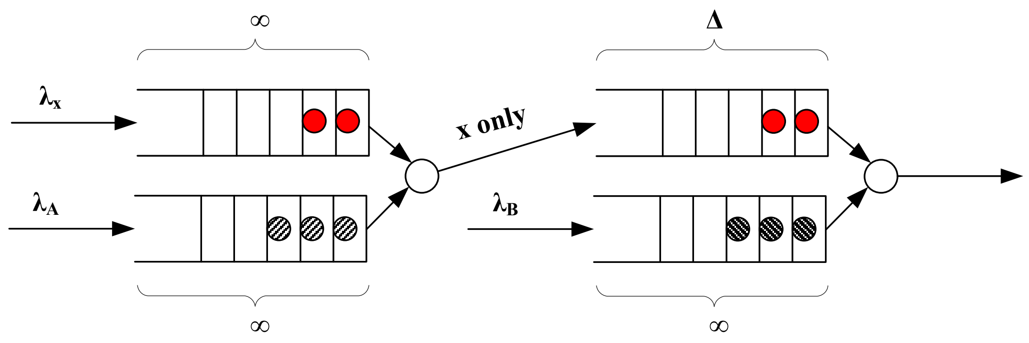
Note that these probabilities of fully transmitting, or continuing with, an interfering flow are independent of the buffer occupancy. This is true because we assume that flow X is always active and that the ejection port of the network is always available for each flow.

When flows A and X are active and flow B is inactive (state $s_{2,\delta}$), the service rate for flow X is higher in the downstream router, which is only servicing X, than in the upstream router that is servicing two flows. Therefore, the buffer occupancy decreases, which is represented by the transition from $s_{2,\delta}$ to $s_{2,\delta-1}$. This transition occurs with probability $(c_A(1-\lambda_B))$, which is the probability that flow A remains active and that flow B remains inactive. The other transition probabilities of $s_{2,\delta}$ represent transitions that do not change buffer occupancy, either because flow B becomes active or flow A is fully transmitted and the service rate for flow X becomes equal at both routers. A similar analysis of state $s_{3,\delta}$ shows that if flow B continues and flow A does not become active, then the buffer occupancy increases and the MC transitions to state $s_{3,\delta+1}$. This is because the downstream node is now servicing flow X at half the rate of the upstream node.

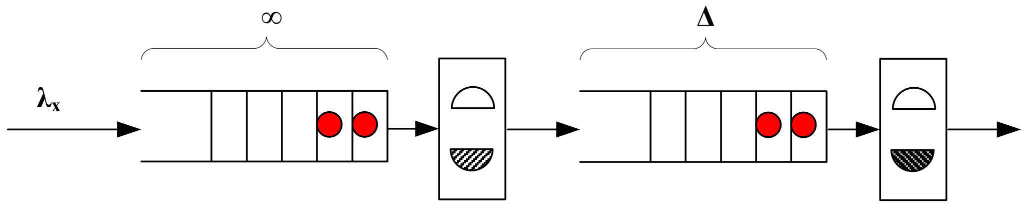
The edge states $s_{2,0}$ and $s_{3,\Delta}$ are connected to themselves as shown on Figure 2.4c. Essentially, the buffer can never have fewer than zero flits and cannot exceed Δ flits. When the buffer is full, flow X is still being serviced by the downstream buffer and is not stalled (remember that we assume ejection is always possible). A full buffer causes back-pressure, which reduces the transmission rate in the upstream node. Our model inherently accounts for this back-pressure through the stationary distribution of the states, which directly determines the throughput as explained in the following subsection.



(a) Simplified configuration



(b) Queue representation



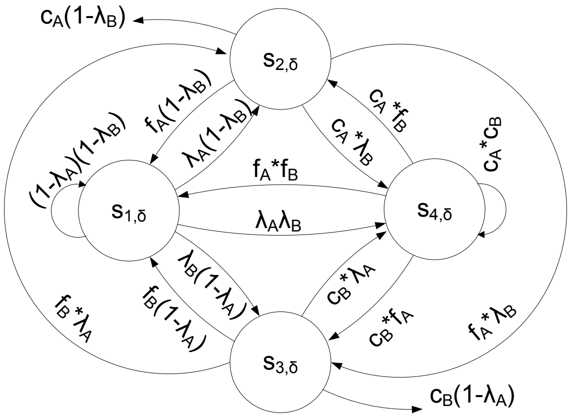
(c) Flowline representation

Figure 2.3: Flow X sharing multiple links with multiple interfering flows.

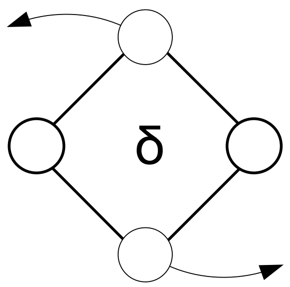
Observe that swapping the order of the interfering flows, i.e. swapping flows A and B in Figure 2.3a, would result in a symmetrical MC leading to exactly the same stationary distribution and estimated network properties. This is a significant improvement over the prior HDM technique, in which the order of the interfering flows affected the estimated delay. We further discuss this issue and show an example in Section 3.4.

2.2 Deriving Throughput and Delay

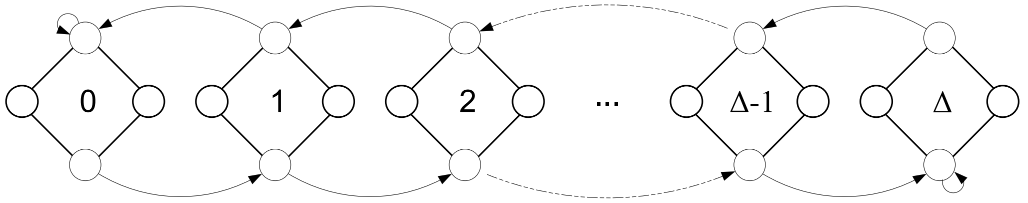
To derive the expected throughput of the NoC observed by flow X in the presence of interference we rely on the fact that the Markov chain we construct is positive recurrent



(a) MC given that the buffer occupancy is δ



(b) Schematic symbol representing Figure 2.4a



(c) Full MC

Figure 2.4: Markov chain representing flow X sharing multiple links with multiple interfering flows (Figure 2.3a).

and aperiodic. These properties imply that the random process corresponding to the state transitions, which represents the NoC, is ergodic. Using the ergodic theorem [8, 22] the expected throughput is given by:

$$T_X = \sum \pi_i \rho_i \tag{2.1}$$

$\overline{\pi}$ (a vector) is the stationary distribution of the MC, which can be computed by solving a system of linear equations.

Next, we express the expected delay of packets in flow X as:

$$L_X = W_X + H_X + \frac{1}{T_X} \tag{2.2}$$

W_X is the expected waiting time in the source node input buffer, which we derive in Equation 2.3. H_X, is the average propagation time of the head flit that we approximate in Equation 2.5. The final term, T_X, is the expected throughput observed by flow X as computed earlier and expressed in units of packets per unit time.

To express the input delay (W_X) we use classic results from the analysis of M/G/1 queues [22], which the NoC conforms to through the assumptions on Poisson packet arrival times and continuous service:

$$W_X = \frac{(1 + C^2_{S_X})\lambda_X}{2T_X(T_X - \lambda_X)} \tag{2.3}$$

The waiting time is dependent on the throughput (T_X), arrival rate (λ_X), and the coefficient of variation ($C^2_{S_X}$):

$$C^2_{S_X} = \frac{\sigma^2_{S_X}}{S^2_X} \tag{2.4}$$

To calculate the average service time (S_X) and its variance ($\sigma^2_{S_X}$) we first compute the expected number of packets serviced at each state of the Markov chain, $\overline{\gamma}$. $\overline{\gamma}$ is the fraction of packets at each state, rather than the fraction of time spent at each state, which is $\overline{\pi}$.

$$\frac{\gamma_i}{\pi_i \rho_i} = \frac{\gamma_j}{\pi_j \rho_j} \quad \forall i, j$$
$$\sum \gamma_i = 1$$

We can now compute the average service time and its variance, and use the result to compute the coefficient of variation (Equation 2.4) and finally the waiting time (Equation 2.3):

$$S_X = \sum \frac{\gamma_i}{\rho_i} \equiv \frac{1}{T_X}$$

$$\sigma^2_{S_X} = \sum \frac{\gamma_i}{\rho_i^2} - S_X^2$$

The last component of the expected packet delay (Equation 2.2) is the head-flit propagation delay:

$$H_X = \sum \gamma_i \eta_i \tag{2.5}$$

η_i is the head-flit propagation delay of packets in flow X incurred while in state s_i, measured in units of time.

Finally, for $S_X \gg H_X$ we can approximate the end-to-end packet delay as:

$$L_X \approx W_X + \frac{1}{T_X} \tag{2.6}$$

Chapter 3

Model Validation

This section illustrates how to apply the model to compute network properties such as delay and throughput. We use the scenarios described in Sec. 2.1.2–2.1.4 and, for each case, compare the results of our model with detailed simulations and with HDM [18]. Our cycle-accurate, discrete-event, NoC simulator uses the OMNET++ framework [41]. The simulator simulates wormhole switching with virtual channels [13], deterministic XY routing, and configurable network topology, buffers, and traffic parameters. The simulated NoC properties are summarized in Table 3.1.

Table 3.1: Simulated NoC properties.

Dimensions	4×4
Message length	256 flits
Flit length	32 bit
Virtual Channels	4
Buffer size	5 flits
Routing	wormhole XY
$Node \leftrightarrow Router$ capacity	400Gbps
$Router \leftrightarrow Router$ capacity	10Gbps
Router frequency	333MHz

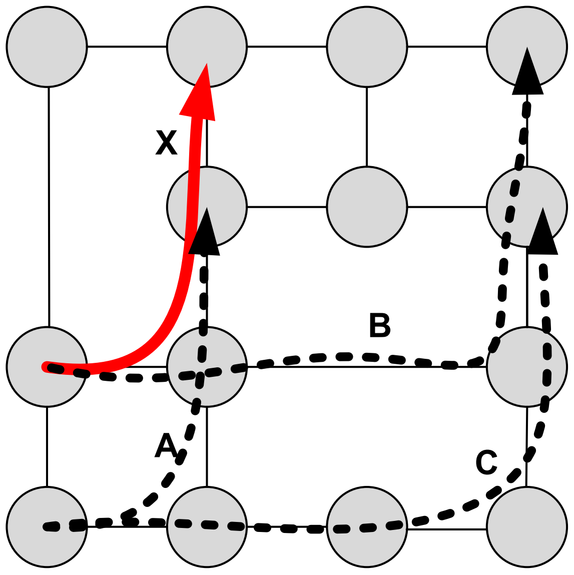


Figure 3.1: Example of possible neglected interaction between interfering flows.

3.1 Isolation

In Section 2.1.1 we explained how to isolate the investigated flow X from the rest of the flows. By doing this we neglect to account for possible indirect interactions between other flows and flow X in different parts of the NoC, as illustrated in Figure 3.1. In this figure, flow C does not interfere with flow X directly, but flows A and B, which do interfere with flow X, interact through flow C.

We investigate the potential impact of indirect interference using the configurations shown in Figure 3.2 and Figure 3.4, where flow X is interacting directly only with flow A, but flow A might interact with other flows (B,C,D). We only consider the case in which the network is stable. We simulated multiple configurations of indirect interference described in Table 3.2 and varied the arrival rate of flow X (λ_X), keeping other arrival rates fixed for simplicity ($\lambda_A = \lambda_B = \lambda_C = \lambda_D = 0.2$). The results (Figure 3.3 and Figure 3.5) show that regardless of whether flows B,C,D are present, the impact on the end-to-end delay for flow X packets is only affected by flow A, and the impact of flows B, C and D can be neglected. This observation holds since the examined system is stable, and hence, indirect interference does not change the average arrival rate of flow A as it interacts with flow X. Indirect interference is likely to impact the delay of flow X in unstable systems, but analysis of such systems is beyond the scope of this work.

Table 3.2: Different interference combinations

configuration	A	B	C	D
I	X	X	X	X
II	X		X	X
III	X	X	X	
IV	X	X		
V	X			X
VI	X			

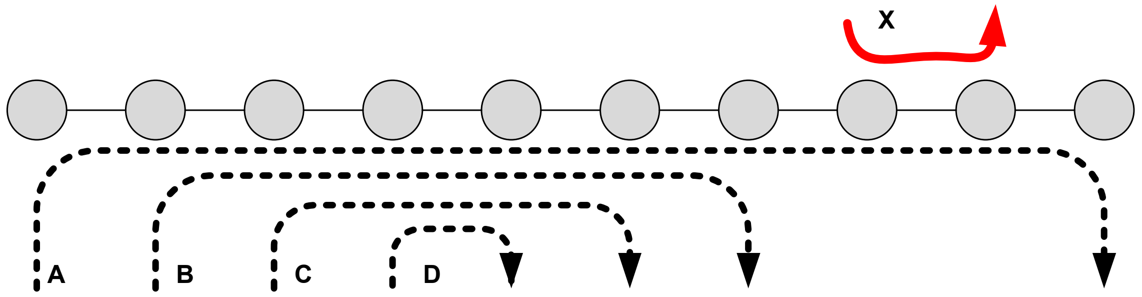


Figure 3.2: Configuration 1 showing the isolation method

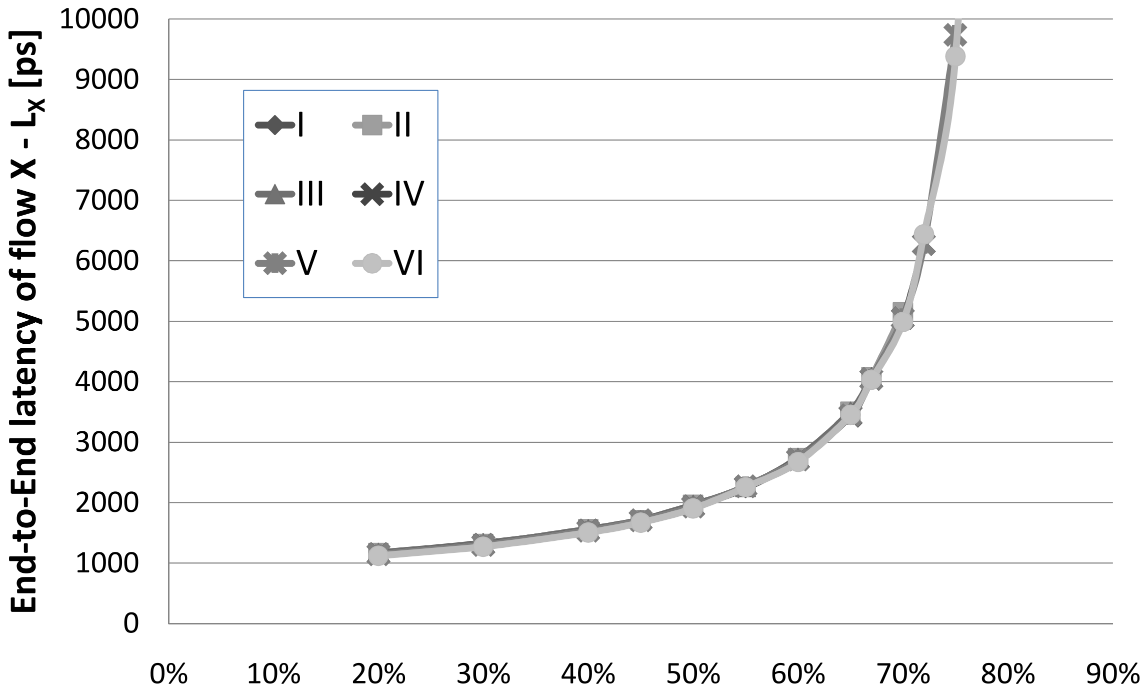


Figure 3.3: End-to-end latency for packets from flow X with different interference combinations according to Table 3.2 for configuration 1

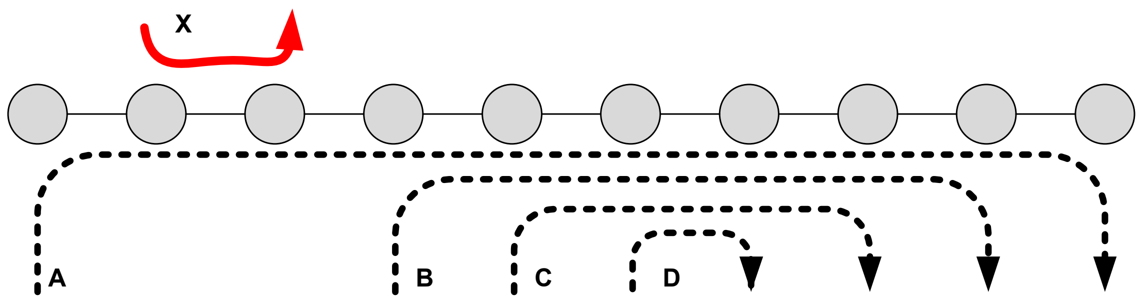


Figure 3.4: Configuration 2 showing the isolation method

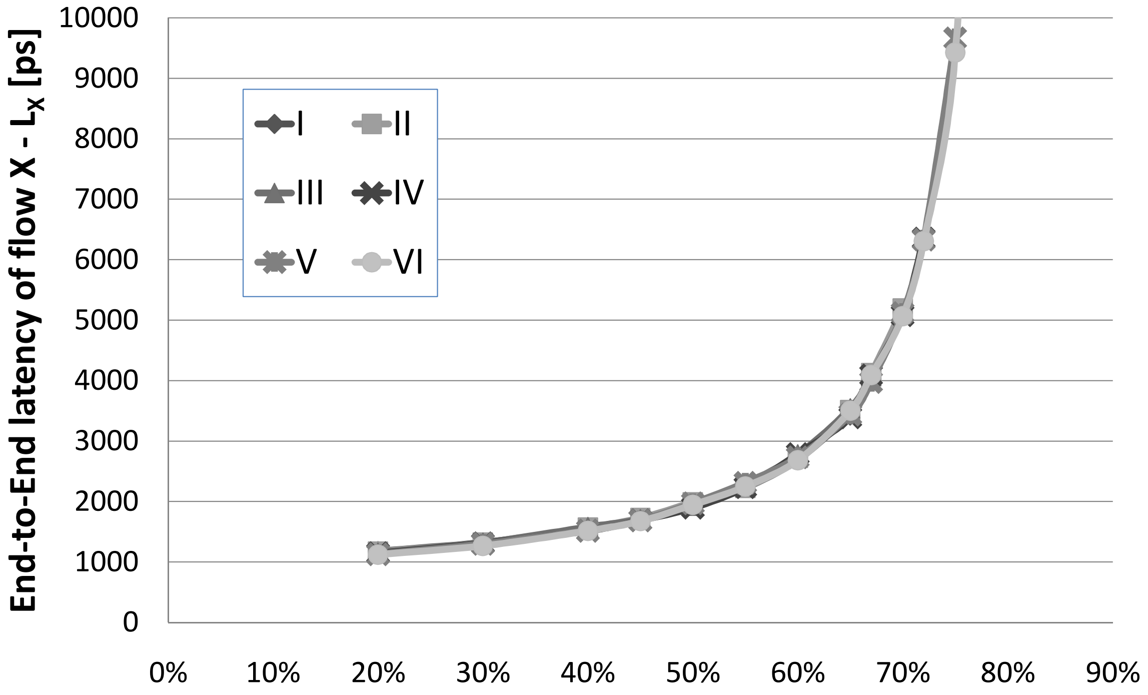


Figure 3.5: End-to-end latency for packets from flow X with different interference combinations according to Table 3.2 for configuration 2

3.2 Sharing a Single Link with a Single Flow

The simplest analyzed test case models a single link shared with a single flow, as shown in Figure 2.1b. The stationary distribution $\overline{\pi}$ of the Markov chain derived in Section 2.1.2 is the solution of the following set of linear equations:

$$\pi_1 \lambda_B = \pi_2 f_B$$
$$\pi_1 + \pi_2 = 1$$

Resulting in:

$$\pi_1 = \frac{f_B}{f_B + \lambda_B}$$
$$\pi_2 = \frac{\lambda_B}{f_B + \lambda_B}$$

Applying Equation 2.1 for the expected throughput provides:

$$\begin{aligned} T_X = \pi_1 \rho_1 + \pi_2 \rho_2 &= \frac{f_B}{f_B + \lambda_B} + \frac{\frac{1}{2}\lambda_B}{f_B + \lambda_B} = \\ &= \max\left(\frac{\phi}{M_X} - \lambda_B \frac{M_B}{M_X}, \frac{\phi}{2M_X}\right) \end{aligned}$$

T_B can be calculated in exactly the same way and following the steps in Section 2.2 L_X and L_B can be calculated (not shown). An important observation of this resulting throughput is that the worst-case throughput observed by flow X is $\frac{1}{2}$ of the link capacity, which corresponds to high contention with the interfering flow B.

Figure 3.6 shows the expected delay of packets in flow X as its throughput requirement, controlled by the arrival rate (λ_X) is increased, and where the arrival rate of the interfering flow B is constant ($\lambda_B = 0.4$). For these parameters, both our model and HDM match the simulation results well. As explained above, the minimum throughput observed by flow X is half of the link capacity, and this low throughput is reached when the interfering flow arrival rate exceeds 0.5. This phenomenon is not accounted for in HDM, because its heuristics were developed and tuned for low arrival rates. This error in delay estimation is apparent in Figure 3.7, which shows the case of fixed ($\lambda_X = 0.4$) and variable λ_B. The HDM allows the interfering flow B to consume more that half of the link capacity, resulting in a sharp increase of the estimated delay of flow X, which tends to infinity for ($\lambda_X \rightarrow 0.6$). Our model, which inherently accounts for the minimum observed throughput, on the other hand, matches simulations well.

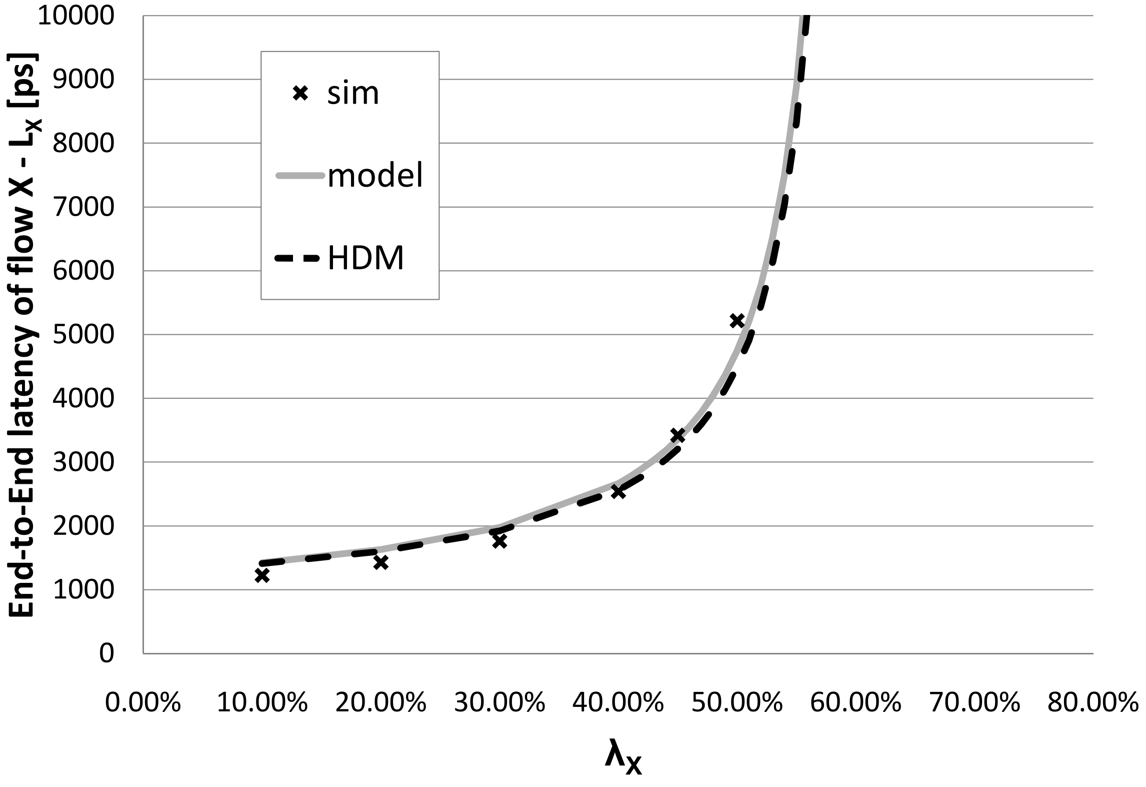


Figure 3.6: Flow X sharing a single link with a single flow - variable λ_X.

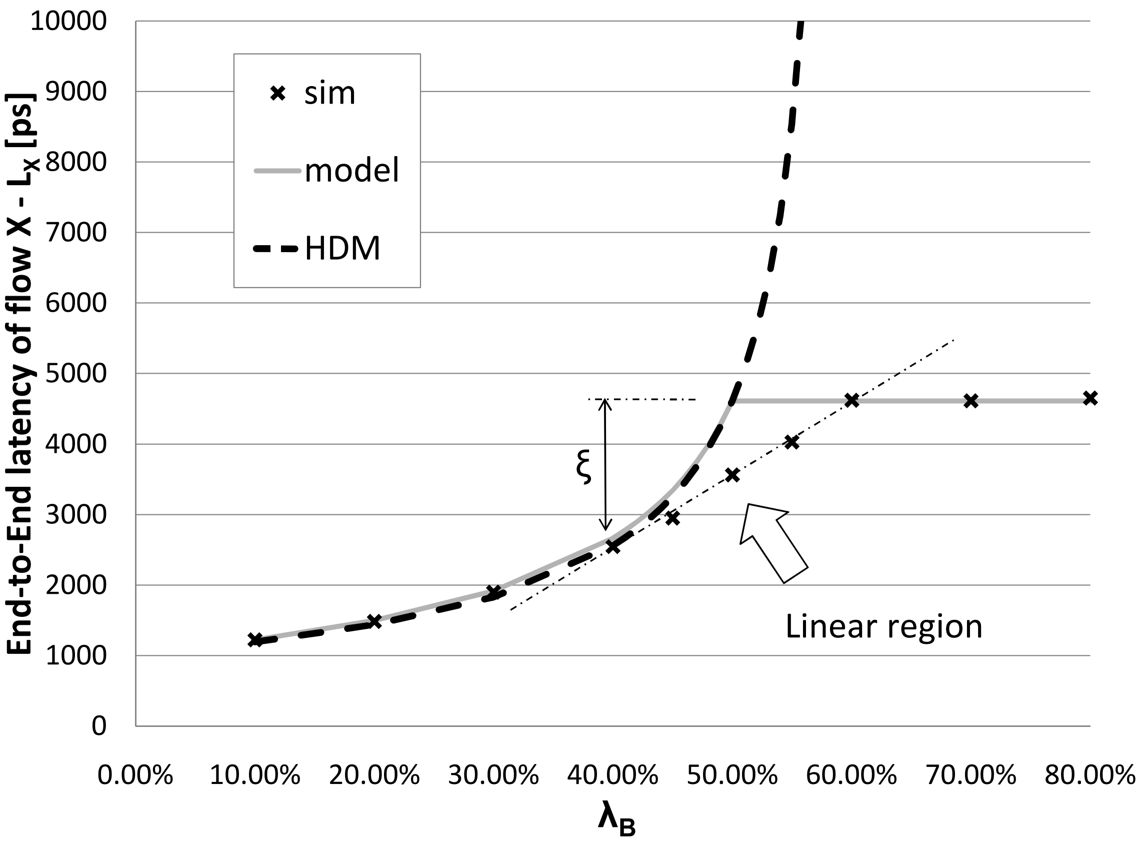


Figure 3.7: Flow X sharing a single link with a single flow - variable λ_B.

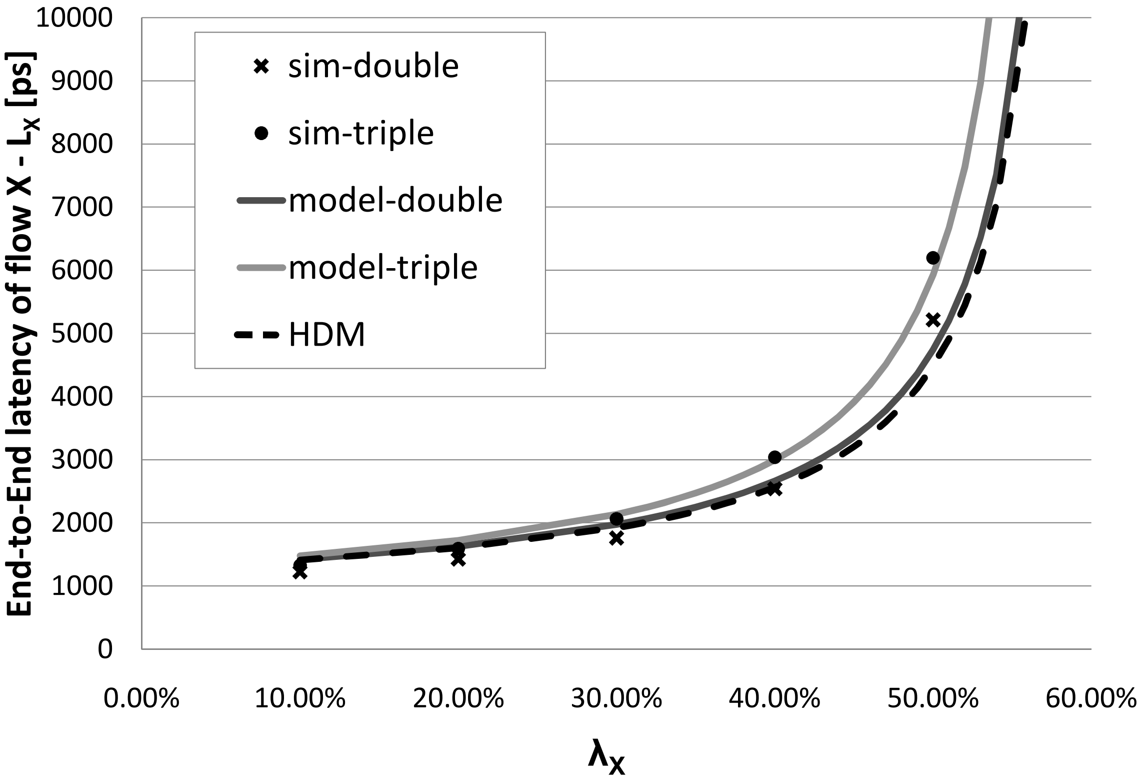


Figure 3.8: Comparison of sharing a single link with three total flows (two interfering flows with ($\lambda_A = \lambda_B = 0.2$) (sim-triple and model-triple) and sharing the link with one interfering flow of equivalent rate ($\lambda_A = 0.4$) (sim-double and model-double). HDM models both cases as a single interferer with ($\lambda = 0.4$).

In addition, an interesting observation can be made regarding the degree of estimation error of our model (see Figure 3.7). The end-to-end latency of flow X when ($\lambda_X = 0.4 < \lambda_B$) has linear behavior with respect to λ_B. As can be seen, our presented model overestimates the end-to-end delay for this case, albeit with bounded error. The reason is that we assume that flow X is always active and has flits available for transmission. This assumption, however, partially fails when the interfering flow B has a higher rate than the investigated flow X. When flow X is not active, flow B observes higher service rate and is transmitted more quickly. This reduces the probability that it actually interferes with flow X and leads to the overestimation error.

While leaving more in-depth investigation of this phenomenon to future work, both in terms of curbing it and with respect to providing a tight bound, we now characterize the error behavior. First, we notice that because of this error, the end-to-end latency can only be over estimated and not under estimated. Second, the error can be roughly bounded by:

$$\xi < L_B(\lambda_X = 0.5) - L_B(\lambda_X = \lambda_B)$$

3.3 Sharing a Single Link with Multiple Flows

We now analyze the test case of a single link with multiple interfering flows, which corresponds to Section 2.1.3. In this scenario, HDM only considers the sum of the arrival rates of all the competing flows; hence, it cannot distinguish between a single interfering flow with ($\lambda = 0.4$) and two interfering flows with ($\lambda = 0.2$) each, for example.

As shown in Figure 3.8, however, these two distinct cases result in very different throughput/delay characteristics. Our technique faithfully models the two cases, and closely follows simulation results.

3.4 Sharing Multiple Links

The last example is of sharing multiple links with different flows as shown in Figure 2.3a and discussed in Section 2.1.4. In this scenario, there is a finite buffer in an intermediate node that can back-pressure flow X.

Figure 3.9 demonstrates the impact of varying the buffer capacity for interfering flows of fixed rate ($\lambda_A = \lambda_B = 0.2$) and varying λ_X. Simulation results for buffer capacities of 5 and 300 flits are shown, along with estimates provided by the proposed model and by HDM. With lower buffer capacity, the peak throughput drops substantially, which our model accurately predicts. HDM [18], which is oblivious to the buffer capacity, does not match the simulation results.

Finally, another aspect of this multiple interferer scenario relates to the order in which the interfering flows appear along the path of flow X. Following the conclusions of the analysis described in Section 2.1.4, our proposed model estimates the same network performance properties (throughput) regardless of the order of the interfering flows. HDM, on the other hand, is sensitive to the order in which the interfering flows are applied, as is evident in Figure 3.10, which shows results for two different configurations: Configuration A, where ($\lambda_A = 0.3$) and ($\lambda_B = 0.1$); and Configuration B where ($\lambda_A = 0.1$) and ($\lambda_B = 0.3$).

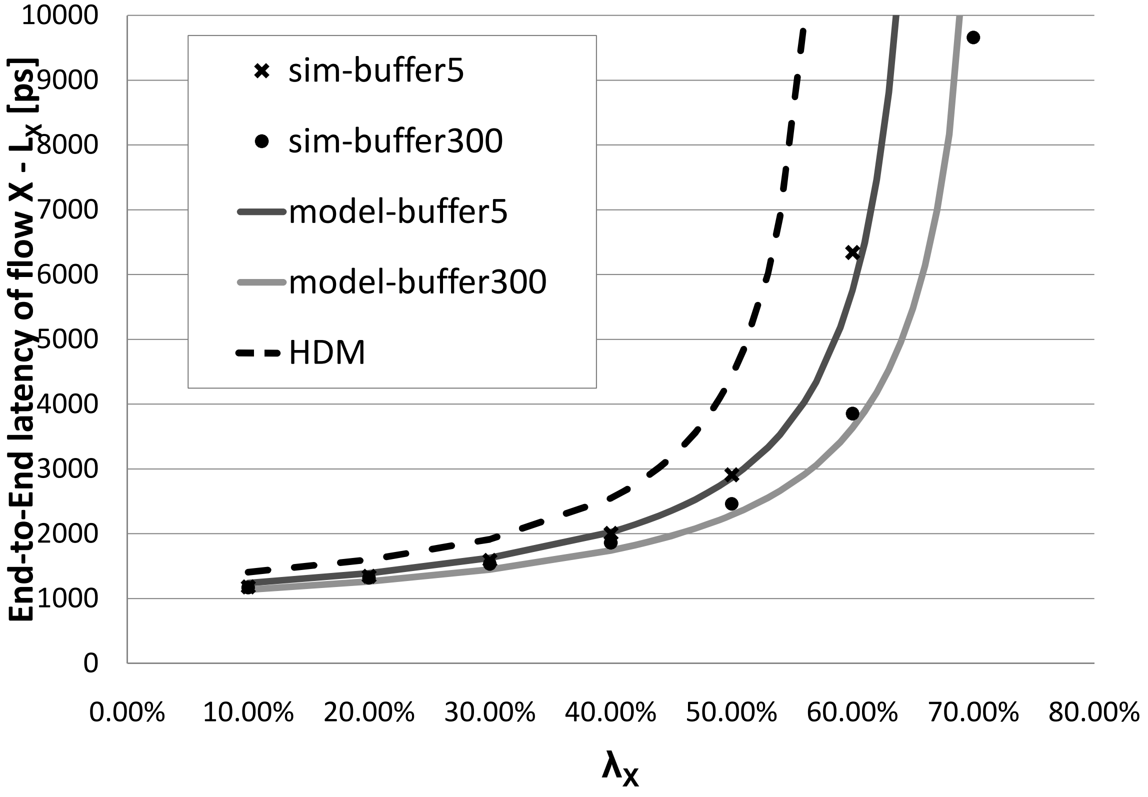


Figure 3.9: Flow X sharing multiple links with multiple flows – variable buffer capacity.

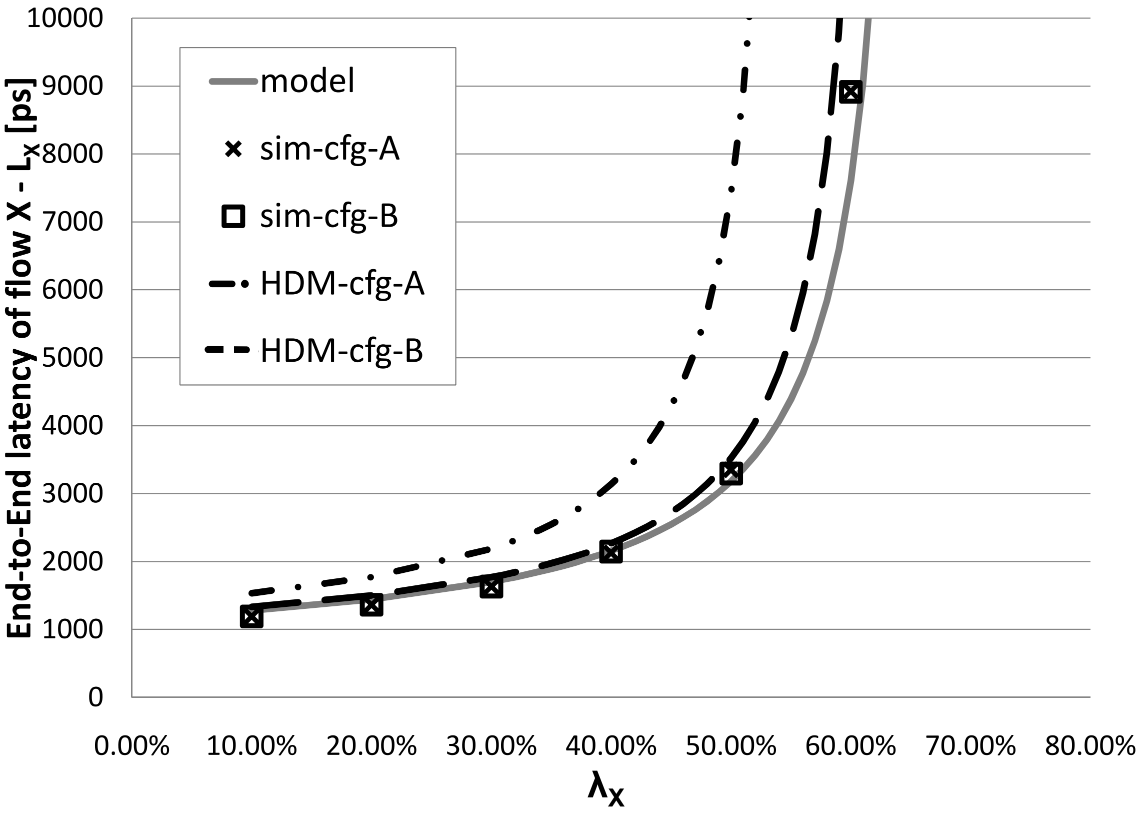


Figure 3.10: Flow X sharing multiple links with multiple flows – varying the order in which interfering flows interact along the path of flow X.

Chapter 4

Benchmark Delay Model and Placement Optimization

In this section we demonstrate how our analytical model can be used in the inner-loop of many optimization algorithms, such as module placement, buffer allocation, link capacity allocation, and network topology selection. These inner loops cannot solely rely on simulations, as simulations take too long to complete, making analytical models crucial for an efficient design process. Further, the correctness of the analytical models directly affects the correctness of the optimization algorithm. Therefore, we show that our analytical model is both fast and accurate in real-world scenarios.

We first analyze the delay of all flows in a SoC using the audio-video benchmark presented in [20], with the traffic requirements summarized in Table 4.1. Using detailed simulations, Section 4.1 compares the accuracy and computation time of our proposed model against HDM. We then illustrate in Section 4.2 how our analytical model can be used in a *module placement* algorithm that attempts to minimize overall flow delay, whereas HDM fails to make a correct optimization decision.

4.1 Benchmark Delay Model

We evaluate our analytical delay model for the benchmark system and traffic requirements shown above assuming Poisson arrival times. We use a 4×4 mesh topology with the parameters detailed in Table 3.1, and the module placement shown in Figure 4.1, denoted *placement A*, where arrows indicate the different flows. For the simulations, we use the

Table 4.1: Audio-video benchmark traffic requirements from [20].

flow	src	dst	rate [kB/s]		flow	src	dst	rate [kB/s]
F_1	MEM1	ASIC4	1168730		F_{16}	ASIC4	DSP1	338480
F_2	ASIC2	ASIC1	800		F_{17}	DSP1	DSP2	338480
F_3	MEM1	CPU	752050		F_{18}	DSP8	DSP7	282650
F_4	MEM3	CPU	755840		F_{19}	DSP6	ASIC2	282480
F_5	ASIC4	CPU	1970		F_{20}	DSP1	CPU	203630
F_6	DSP3	DSP6	70610		F_{21}	DSP2	DSP1	203630
F_7	ASIC1	ASIC2	250		F_{22}	DSP3	DSP5	70610
F_8	DSP3	ASIC4	380160		F_{23}	DSP7	MEM2	70650
F_9	DSP8	ASIC1	800		F_{24}	MEM2	ASIC3	77050
F_{10}	DSP5	DSP6	269240		F_{25}	ASIC3	DSP8	6410
F_{11}	CPU	MEM1	380160		F_{26}	DSP4	DSP1	36720
F_{12}	CPU	MEM3	380160		F_{27}	ASIC2	MEM2	6400
F_{13}	CPU	ASIC3	380160		F_{28}	ASIC2	ASIC3	7650
F_{14}	DSP2	ASIC2	338480		F_{29}	ASIC3	DSP4	1440
F_{15}	DSP4	CPU	1970		F_{30}	ASIC1	DSP8	250

simulator described in Chapter 3, and run the simulations long enough for all performance characteristics of the different flows to stabilize.

Figure 4.2 shows the average total queuing delay of each flow due to network contention, i.e. the average latency beyond the network propagation time. It compares the simulation results with our proposed analytical model, as well as the analytical model presented in [18]. Due to space limitations, we only present the eight flows with the greatest relative slowdown, as ranked by simulation results and presented in Table 4.1 as $(F_1, \ldots, F_8)$.

As shown in Figure 4.2, our model approximates the simulation results significantly more accurately than HDM. In particular, Figure 4.3 depicts the absolute error of the queuing delay for each flow for both analytical models. We can see that the error for each flow is under 15% for our model, while the error of HDM is often above 50% and can be greater than even a *factor* of 10 times.

We also note that the time required to compute the results is orders of magnitude faster using our model than with the detailed simulation, requiring only $33ms$ as opposed to over 7 hours of simulation time.

4.2 Placement Optimization

A possible use of the analytical delay model is to estimate network and flow properties within the inner-loop of a module placement optimization algorithm. As shown above, our model can quickly compute delay with high accuracy and in this subsection we demonstrate that it also reflects the change in delay as a result of varying the module placement. Hence, our model can be used to predict, and correctly and efficiently choose between multiple placement options. Without loss of generality, we assume that there is a need to choose

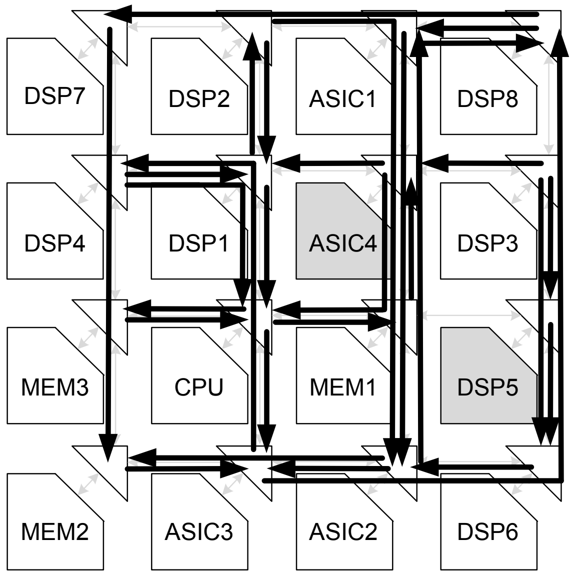


Figure 4.1: *Placement A* of the components and flows of the audio-video SoC of [20].

between two placements: *placement A*, illustrated in Figure 4.1, and *placement B*, shown in Figure 4.4, where modules *ASIC4* and *DSP5* have been swapped.

Figure 4.5 shows the average flow queuing delays for placements A (dark columns) and B (light gray columns). The simulation columns show that the average flow delay is lower in placement A. This is accurately reflected in our analytical model, which closely approximates the simulation delays to within 3%, and would also have pointed to placement A as having a smaller overall delay. HDM, on the other hand, can be quite inaccurate, and as a result leads to an incorrect placement decision, predicting that placement B has lower overall latency than placement A.

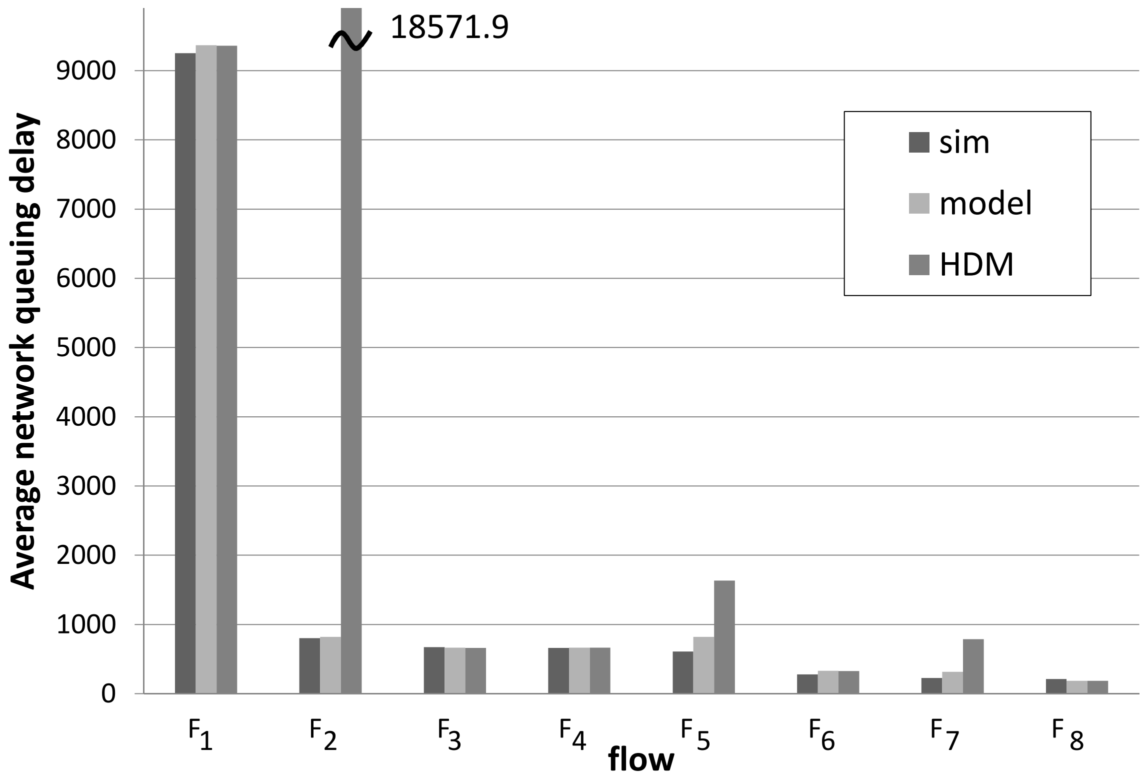


Figure 4.2: Total queuing delay predicted by detailed simulation, our model, and HDM for the 8 flows with highest latency corresponding to the system of Table 4.1 with the placement depicted in Figure 4.1.

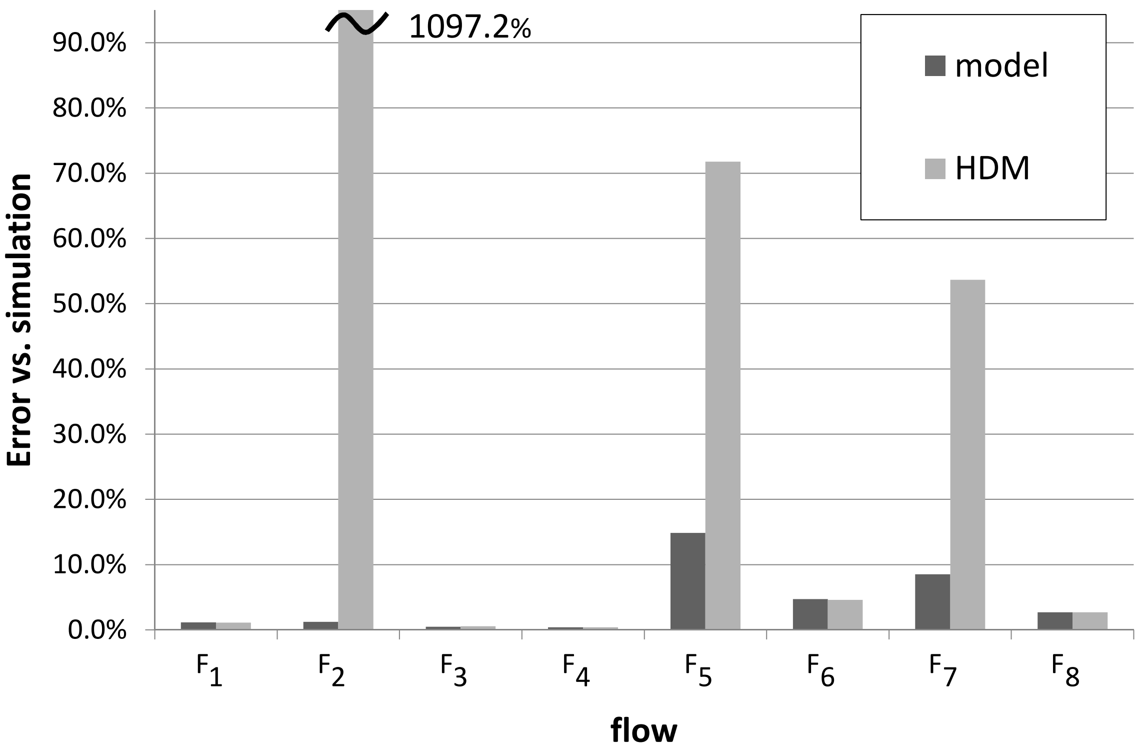


Figure 4.3: Relative error in latency estimation between our model and HDM relative to detailed simulation for the system requirements of Table 4.1 and Placement A (Figure 4.1).

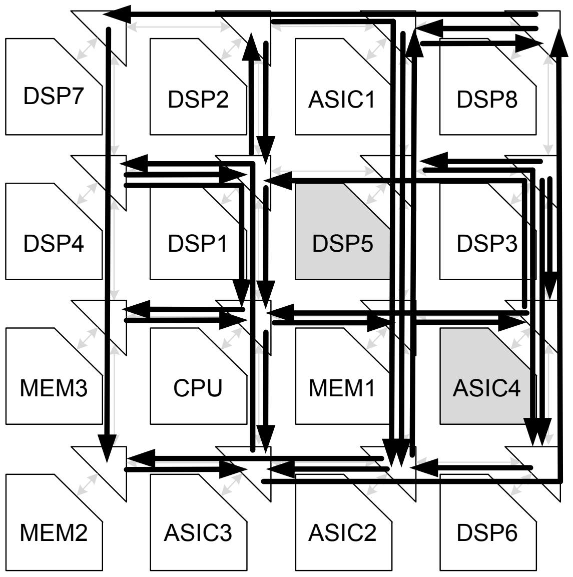


Figure 4.4: *Placement B* of the components and flows of the audio-video SoC of [20].

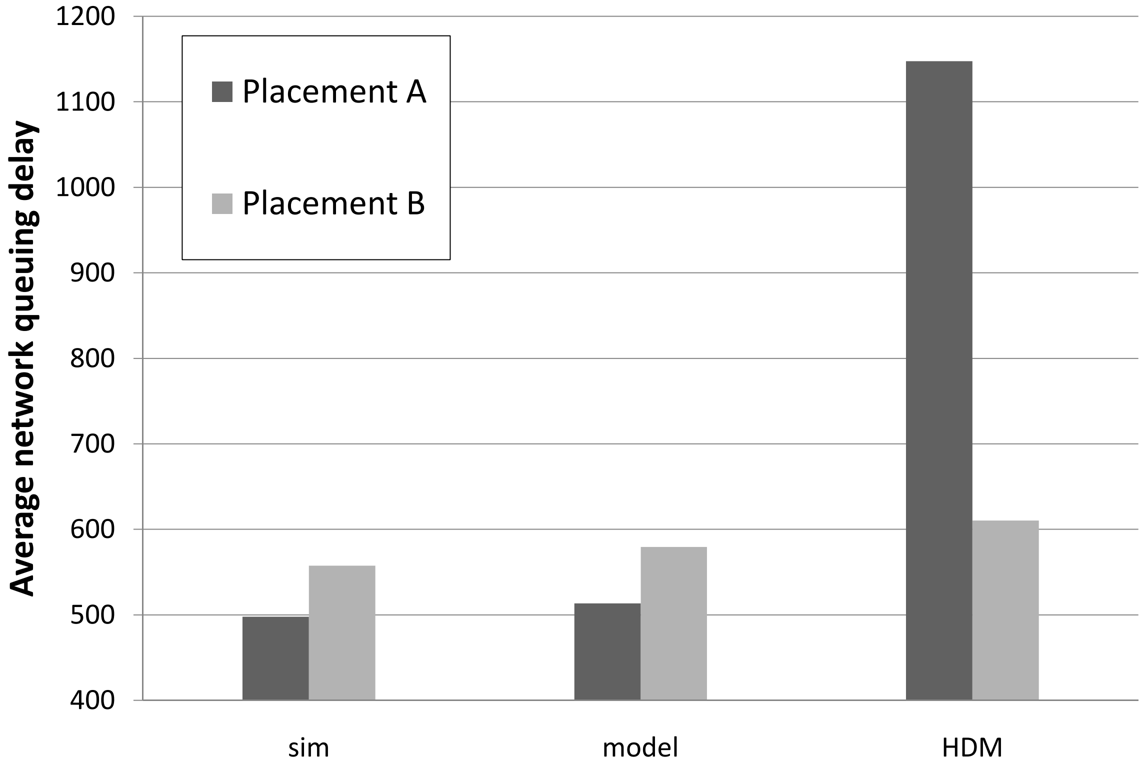


Figure 4.5: Comparison of estimated latency of placement A (Figure 4.1) and placement B (Figure 4.4) predicted by detailed simulation, the proposed model, and HDM.

Chapter 5

Discussion and Future Work

Our model is constructed based on several important assumptions, which impact its accuracy as discussed below. We assume that flow X is always active, and is thus always competing for link capacity with the interfering flows. We discuss the implications in Section 3.2 and show that the resulting error is both bounded and limited to scenarios where the interfering flows require more throughput than flow X.

In this work we assumed that packets are of constant length and arrive according to a Poisson process. The arrival process determines the expected delay through Equation 2.3, however, it is possible to compute the delay using a G/G/1 queuing model instead of the M/G/1 model we assumed to allow any stochastic arrival process. Finally, we approximated the end-to-end delay by neglecting the header flit propagation delay (Equation 2.5), which is only accurate for long packets.

Our final assumption is that virtual channels are always available for any flow. Extending the model to account for this type of head-of-line blocking is left for future work.

The list below summarizes all the assumptions in this work:

- Wormhole switching
- Number of VCs available always satisfies the demand
- Deterministic routing
- Round-robin arbitration amongst VCs
- Finite intermediate buffers

- Infinite network entrance queue
- Poisson packet injection process
- Constant packet length
- No backpressure from (final) destination — infinite network exit queue
- Network stability

In this work, an attempt to reduce the amount of MC states was taken using the isolation assumption and by not representing the state of the network entrance queues (Section 2.1.1). Trying to represent the occupancy of the infinite network entrance queues would lead to infinite amount of states. Avoiding the assumtption would require the MC to represent the states of all the available network flows (F) along with the occupancy of all intermediate buffers in all R routers. With total number of F flows, there are exactly 2^{F-1} possible combinations of interfering flows. Each intermediate buffer occupancy can be in the $[0..\Delta]$ range, requiring $(\Delta + 1)$ states for representation. Taking into account that the number of intermediate buffers is $O(R)$ leads to an upper bound of $O(2^F * \Delta^R)$ states. However, using the assumption, we reduce it to $O(2^{\Psi_\alpha} * \Delta^{P_\alpha})$ where Ψ_α is the number of interfering flows (for a specific flow of interest α) and P_α is the number of hops between the source and the destination for that flow. In the example discussed in Chapter 4, $F = 30$, $R = 16$, $\Delta = 5$, $max(\Psi_\alpha) = 2$, and $max(P_\alpha) = 6$.

As part of future work, should be investigated the possibility of significantly simplifying the Markov chain by relying on MC decomposition methods developed for industrial flowline analysis [3, 9, 11, 16, 34]. This line of work promises to expedite the solution of complex NoCs by curbing the exponential growth in the number of states required to model a large number of interfering flows.

While this work demonstrated our model for estimating throughput and end-to-end delay, the model inherently captures other network phenomena and parameters. For example, estimating buffer occupancy levels is a straight-forward extension, that uses the MC representation discussed in Section 2.1.4.

Chapter 6

Conclusions

In this work we introduced a packet-level *static timing analysis* (STA) for NoCs. We showed how it allows for a quick and precise evaluation of the performance parameters of a virtual-channel wormhole NoC without using any simulation techniques. It can handle any topology, link capacities, and buffer capacities — and unlike existing models, is able to evaluate the performance of a specific flow in a precise manner.

Our new model allows for a per-flow STA that is orders-of-magnitude faster than simulation. Ultimately, the objective is for this packet-level STA model to be used in the inner-loop of NoC optimization tools — and become the packet-level equivalent of gate-level critical path analysis utilized in CAD tools.

Appendix A

Simulator

In order to validate the model a cycle-accurate, discrete-event, NoC simulator was built. The simulator is based on an open source OMNET++ framework [41] widely used for network research. The simulator is written in C++ (tested both on Win XP and Linux platforms). Thanks to the OMNET++ infrastructure, it can be easily compiled in 2 configurations :

- interactive - debug oriented version allowing graphical user interface visualizing the behavior of the network with animation. Allows to click on an entity and to see it contens in real time (for example to track contents of a buffer) as shown on Figure A.1.
- command line - performance optimized version for long simulations run

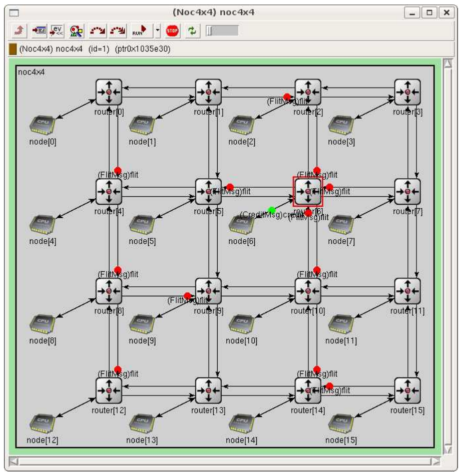


Figure A.1: Screenshot of 4x4 NoC simulation

A.1 Network Topology

Network topology is defined in a special format ned file. Bellow is an example of a 4×4 mesh network definition.

Listing A.1: noc4x4.ned

```
// include basic modules
import
  "nocnode.ned";
import
  "nocrouter.ned";

// define the links
// router to router
channel NoCxChannel
  datarate 10 //bit/psec = 1Gbps
endchannel

// module to router (and router to module)
channel NoCxLocalChannel
  datarate 400 //bit/psec = 400Gbps
endchannel

// define the network
module Noc4x4
  submodules:
  node: NocNode[16];
    parameters:
      display: "p=50,100,matrix,4,150,150;i=device/cpu";

  router: NocRouter[16]
    parameters:
      gatesizes:
        in[5], out[5], credit_recv[5], credit_send[5];
  display: "p=150,50,matrix,4,150,150;i=block/routing";

  connections nocheck:
  // nodes to routers
  for i=0..15 do
    node[i].out --> NoCxLocalChannel --> router[i].in[0];
    node[i].credit_send --> router[i].credit_recv[0];
    router[i].out[0] --> NoCxLocalChannel --> node[i].in;
    router[i].credit_send[0] --> node[i].credit_recv;
  endfor;

  // veritcal
  for i=0..11 do
    router[i].out[4] --> NoCxChannel --> router[i+4].in[2];
    router[i].in[4]  <-- NoCxChannel <-- router[i+4].out[2];
    router[i].credit_send[4] --> router[i+4].credit_recv[2];
    router[i].credit_recv[4] <-- router[i+4].credit_send[2];
  endfor;


  // horizontal
  for i=0..2, j=0..3 do
    router[j*4+i].out[3] --> NoCxChannel --> router[j*4+i+1].in[1];
    router[j*4+i].in[3]  <-- NoCxChannel <-- router[j*4+i+1].out[1];
    router[j*4+i].credit_send[3] --> router[j*4+i+1].credit_recv[1];
    router[j*4+i].credit_recv[3] <-- router[j*4+i+1].credit_send[1];
  endfor;

endmodule

network noc4x4 : Noc4x4;
endnetwork
```

Here in lines 7 – 16 the properties of the interconnecting link are defined and in lines 40 – 55 the actual topology is defined.

A.2 Experiments

The simulator makes possible to run multiple experiments in batch mode. For that an experiment file should be defined. Bellow is an example of an expirement file.

Listing A.2: experiment.ini

```
[General]
preload-ned-files=*.ned
network=noc7 # network topology to use (.ned file)
sim-time-limit = 8e7s # run time limit (in simulation time - not wallclock time)
rng-class="cLCG32"

[OutVectors]
**.enabled = no

[Cmdenv]
express-mode=yes
performance-display=no
event-banners=no
runs-to-execute = 1-3 # which runs to run
status-frequency=100000000

[Parameters]
noc7.node[0].nodecpu.*.address = 0
noc7.node[1].nodecpu.*.address = 1
noc7.node[2].nodecpu.*.address = 2
noc7.node[3].nodecpu.*.address = 3
noc7.node[4].nodecpu.*.address = 4
noc7.node[5].nodecpu.*.address = 5
noc7.node[6].nodecpu.*.address = 6

# limit number of messages for each active node to send
noc7.node[*].nodecpu.nodecpuout[*].numMessages = 100000000;

# packet properties
noc7.node[*].nodecpu.nodecpuout[*].messageLength = 256
noc7.node[*].nodecpu.nodecpuout[*].flitBytes = 4

noc7.node[*].nodecpu.nodecpuout[*].numStations = 7

# buffers sizes
noc7.node[*].*.bufSize = 5;
noc7.node[*].*.*.bufSize = 5;
noc7.*.bufSize = 5;

# scheduling policy : 1 - round robin, 2 - exhaustive
noc7.node[*].*.policy = 1;
noc7.node[*].*.*.policy = 1;
noc7.*.policy = 1;

# define here destination for each node; -1 stands for inactive
# note that each node has an array of 3 sources (cpuout 0:2)
noc7.node[0].nodecpu.nodecpuout[0].dest = 6
noc7.node[0].nodecpu.nodecpuout[1].dest = -1
noc7.node[0].nodecpu.nodecpuout[2].dest = -1
noc7.node[1].nodecpu.nodecpuout[*].dest = -1
noc7.node[2].nodecpu.nodecpuout[*].dest = -1
noc7.node[3].nodecpu.nodecpuout[0].dest = 4
noc7.node[3].nodecpu.nodecpuout[1].dest = -1
noc7.node[3].nodecpu.nodecpuout[2].dest = -1
```

```
55 noc7.node[4].nodecpu.nodecpuout[*].dest = -1
56 noc7.node[5].nodecpu.nodecpuout[*].dest = -1
57 noc7.node[6].nodecpu.nodecpuout[*].dest = -1
58
59 # for inactive sources, set interarrival time to 0
60 noc7.node[0].nodecpu.nodecpuout[1].interArrivalTime = 0;
61 noc7.node[0].nodecpu.nodecpuout[2].interArrivalTime = 0;
62 noc7.node[1].nodecpu.nodecpuout[*].interArrivalTime = 0;
63 noc7.node[2].nodecpu.nodecpuout[*].interArrivalTime = 0;
64 noc7.node[3].nodecpu.nodecpuout[1].interArrivalTime = 0;
65 noc7.node[3].nodecpu.nodecpuout[2].interArrivalTime = 0;
66 noc7.node[4].nodecpu.nodecpuout[*].interArrivalTime = 0;
67 noc7.node[5].nodecpu.nodecpuout[*].interArrivalTime = 0;
68 noc7.node[6].nodecpu.nodecpuout[*].interArrivalTime = 0;
69 noc7.node[7].nodecpu.nodecpuout[*].interArrivalTime = 0;
70
71 # for active sources, set different interarrival times for each run
72 # can use different random distributions as exponential, uniform etc.
73
74 [Run 1]
75 noc7.node[0].nodecpu.nodecpuout[0].interArrivalTime = exponential(8192)
76 noc7.node[3].nodecpu.nodecpuout[0].interArrivalTime = exponential(4096);
77
78 [Run 2]
79 noc7.node[0].nodecpu.nodecpuout[0].interArrivalTime = exponential(4096)
80 noc7.node[3].nodecpu.nodecpuout[0].interArrivalTime = exponential(4096);
81
82 [Run 3]
83 noc7.node[0].nodecpu.nodecpuout[0].interArrivalTime = exponential(2730.66667)
84 noc7.node[3].nodecpu.nodecpuout[0].interArrivalTime = exponential(4096);
85
86 ...
```

Here lines 47 – 57 define the flows (in this case $node0.src0 \rightarrow node6$ and $node3.src0 \rightarrow node4$). Different runs are defined starting from line 74 and which out of those to execute is specified in line 14.

To run the experiment, simply use 'noc_cmd -f experiment.ini'.

Bibliography

[1] International technology roadmap for semiconductors, 2008. `http://www.itrs.net`.

[2] T. Ahonen, D. Sigüenza-Tortosa, H. Bin, and J. Nurmi. Topology optimization for application-specific networks-on-chip. In *Proceedings of the 2004 international workshop on System level interconnect prediction*, pages 53–60, 2004.

[3] T. Altiok. *Performance Analysis of Manufacturing Systems*. Springer, Berlin, 1997.

[4] N. Alzeidi, M. Ould-Khaoua, and A. Khonsari. A new modelling approach of wormhole-switched networks with finite buffers. *International Journal of Parallel, Emergent and Distributed Systems*, 23(1):45–57, 2008.

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

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

[7] T. Bjerregaard and J. Sparso. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pages 1226–1231, 2005.

[8] P. Brémaud. *Markov Chains*. Springer, Berlin, 1999.

[9] J. Buzacott. *Stochastic Models of Manufacturing Systems*. Prentice Hall, Englewood Cliffs, 1993.

[10] B. Ciciani, M. Colajanni, and C. Paolucci. An accurate model for the performance analysis of deterministic wormhole routing. In *Proceedings of the 11th International Symposium on Parallel Processing*, page 353, 1997.

[11] Y. Dallery and Y. Frein. On decomposition methods for tandem queueing networks with blocking. *Oper. Res.*, 41(2):386–399, 1993.

[12] W. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[13] W. Dally. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, San Francisco, 2004.

[14] A. Diamantidis and C. Papadopoulos. Exact analysis of a two-workstation one-buffer flow line with parallel unreliable machines. *European Journal of Operational Research*, 2008.

[15] J. Draper and J. Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *Journal of Parallel and Distributed Computing*, 23(2):202–214, 1994.

[16] S. B. Gershwin. An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking. *Oper. Res.*, 35(2):291–305, 1987.

[17] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Efficient link capacity and qos design for network-on-chip. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 9–14, 2006.

[18] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Network delays and link capacities in application-specific wormhole nocs. *Special Issue of the Journal of VLSI Design*, 2007.

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

[20] J. Hu, U. Ogras, and R. Marculescu. System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2919, 2006.

[21] A. Kiasari, D. Rahmati, H. Sarbazi-Azad, and S. Hessabi. A Markovian Performance Model for Networks-on-Chip. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 157–164, 2008.

[22] L. Kleinrock. *Theory, Volume 1, Queueing Systems.* Wiley-Interscience, 1975.

[23] S. Loucif, M. Ould Khaoua, and G. Min. A queueing model for predicting message latency in uni-directional k-ary n-cubes with deterministic routing and non-uniform traffic. *Cluster Computing*, 10(2):229–239, 2007.

[24] Z. Lu, A. Jantsch, and I. Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 960–964, 2005.

[25] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual channels in networks on chip: implementation and evaluation on hermes NoC. In *Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 178–183, 2005.

[26] M. Moadeli, A. Shahrabi, W. Vanderbauwhede, and M. Ould-Khaoua. An analytical performance model for the Spidergon NoC. In *21st IEEE International Conference on Advanced Information Networking and Applications*, pages 1014–1021, 2007.

[27] P. Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Computing Surveys (CSUR)*, 30(3):374–410, 1998.

[28] R. Mullins, A. West, and S. Moore. The design and implementation of a low-latency on-chip network. In *Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 164–169, 2006.

[29] L. Ni and P. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, 1993.

[30] U. Y. Ogras and R. Marculescu. Analytical router modeling for networks-on-chip performance analysis. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1096–1101, 2007.

[31] M. Ould-Khaoua and H. Sarbazi-Azad. An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic. *IEEE Transactions on Parallel and Distributed Systems*, 12(3):283–292, 2001.

[32] S. Parkes and P. Armbruster. SpaceWire: a spacecraft onboard network for real-time communications. In *IEEE-NPSS Real time conf*, pages 6–10, 2005.

[33] S. Parkes and J. Rosello. SpaceWire- Links, nodes, routers and networks. In *DASIA 2001- DAta Systems in Aerospace; Proceedings of the Conference*, 2001.

[34] H. Perros. *Queueing Networks with Blocking.* Oxford University Press, Oxford Oxfordshire, 1994.

[35] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. DATE*, pages 764–769, 2004.

[36] M. Pirvu, L. Bhuyan, and N. Ni. The impact of link arbitration on switch performance. In *Proceedings of the Fifth Symposium on High-Performance Computer Architecture*, 1999.

[37] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar. An asynchronous router for multiple service levels networks on chip. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 44–53, 2005.

[38] Z. Shi and A. Burns. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 161–170, 2008.

[39] R. Thakur and A. Choudhary. All-to-all communication on meshes with wormhole routing. In *Parallel Processing Symposium, 1994. Proceedings., Eighth International*, pages 561–565, 1994.

[40] M. Uitert. *Generalized processor sharing queues.* Ponsen and Looijen BV, 2003.

[41] A. Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM 2001)*, 2001.