2011

A probabilistic approach to early communication performance estimation for electronic system-level design

Ramon Alejandro Mercado
Iowa State University

A probabilistic approach to early communication performance estimation for electronic system-level design

by

Ramón A. Mercado Reyes

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Diane T. Rover, Major Professor
Akhilesh Tyagi
Arun Somani
Joseph Zambreno
Zhao Zhang
Ying Cai

Iowa State University
Ames, Iowa
2011

DEDICATION

This work is dedicated to my father, who showed me that I had the capacity to accomplish everything I set my mind into; to my mother, who gave me the conviction that drives everything I do; to my sister who taught me to be responsible above all; and to my dearest Zaida, whose constant support made the conclusion of this work a reality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This research would no exist without the support of many other people. Hence, I want to express my gratitude to my Major Professor Diane Rover for encouraging me to work towards a PhD and her continuous support on my thesis. Special thanks also go to Professor Ahmed Kamal for his valuable comments for the improvement of this research; and to Professors Zhao Zhang and Joseph Zambreno for their support throughout my student career. My sincere appreciation goes to my esteem colleague Lizandro Solano for the many fruitful discussions and the inspiring working environment we developed.

Finally, this work would not have been possible without the support from NSF grant No. 0431924 and a GAANN grant from the U.S. Dept. of Education to the Information Infrastructure Institute at Iowa State.

ABSTRACT

Today's embedded system designers face the challenges of ever increasing complexity and shorter time-to-market deadlines. System-level methodologies emerge to meet these challenges. Refinement-based methodologies, such as the SpecC methodology and Transaction Level Modeling, continue to gain popularity in the embedded system designers' community. However, as more communication-dominated applications and architectures appear in the market, designers find that the lack of models allowing system-level communication analysis is a major limiting factor in current system-level design methodologies. Thus, modeling for system-level communication analysis is key for a design methodology to thrive with today's embedded system designers. This work presents a new approach to system-level modeling that allows better communication analysis earlier in the design process. This approach defines a new model that utilizes random variables to include the communication details at higher abstraction levels. This work proposes a probabilistic model to include and evaluate the system communication features in the higher abstraction level. Guidelines to include the proposed model into a refinement-based methodology are presented, and methods for performance estimation are shown.

CHAPTER 1. Introduction

1.1 Motivation

System complexity continues to grow according to the well-known Moore's law [45]. In contrast, designer productivity grows at a much lower rate. The International Technology Roadmap (ITRS) [1] defines the *design productivity gap* as the growing disparity between the system complexity and designer productivity growths. Figure 1.1 depicts both the hardware design gap and the system design gap. Hardware design gap refers to the productivity gap considering only the design of the hardware components in the system. On the other hand, the system design gap includes the software demands, which double every 10 months, and the system productivity including the development of the necessary hardware-dependant software and application code for system design.

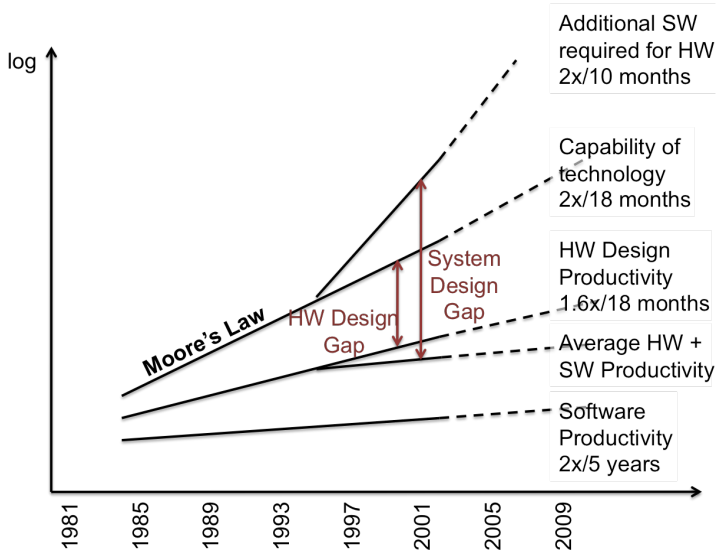


Figure 1.1: Design Productivity Gap

This growing productivity gap results in increasing non-recurrent engineering costs, and

large time-to-market cycles. To deal with this growing productivity gap the only solution is system-level design [28]. System-level design reduces the productivity gap by introducing new abstraction levels that allow designers to handle progressively more complex systems.

The higher abstraction levels in system-level design hide non-essential details from the designers and allow for a coarser design space exploration. A common representation of the design space is an orthogonal composition of the computation and communication design alternatives, shown in Figure 1.2. This orthogonal relation between computation and communication allows the designer to explore the computation architecture neglecting the communication effects on the system design. Starting at the highest abstraction levels, un-timed computation and communication, the system is refined by adding more details along the computation axis and evaluating the resulting model at the next level of abstraction.

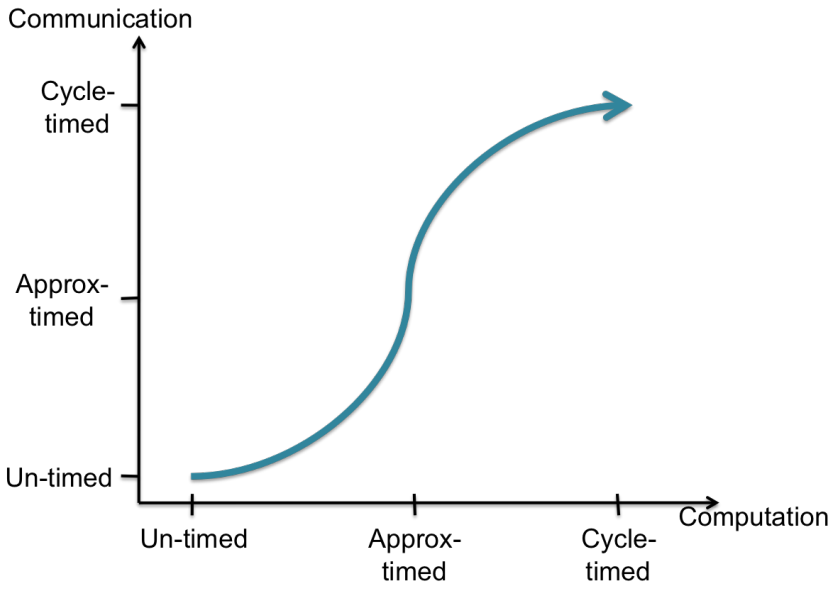


Figure 1.2: Computation/Communication System-Level Design Space

The manner in which the design space is explored, i.e. a broader computation exploration followed by a narrow communication exploration, reveals a computation-centric design methodology. Computation-centric design is reasonable where higher computation power translates to higher system performance. However, with the proliferation of embedded communication systems (e.g. smart-phones, GPS devices, etc) and multi-core systems, improvements in computation power are negated by communication issues. Today's proces-

sors are capable of processing large amounts of data much faster than the communication architecture can deliver this data. The system-level design community must reacts to this paradigm change, and introduce new tools and methodologies to better integrate communication analysis into the early stages of the design process. From the 2009 ITRS[1] report, *Design* chapter

> "Global synchronization becomes prohibitively costly due to process variability and power dissipation, and cross-chip signaling can no longer be achieved in a single clock cycle. This, system design must comprehend networking and distributed computation metaphors (for example, with communication structures design first, and functional blocks then integrated into the communication backbone), as well as interactions between functional and interconnect pipelining."

Some researchers in the field of system-level design acknowledge this paradigm change and start to move toward a communication-centric design process. Some examples include the works by Sgroi et al. [57] and Coppola et al. [13] which introduce SystemC[49] libraries to capture the characteristics of network-on-chip (NoC) architectures, as a collection of the services these architectures offer. On the other hand, Pasricha et al. [50] integrates the application behavior with the communication architecture details, generating a transaction based model that is used during architecture exploration.

Unlike the previous works, our approach develops higher abstraction models based on the impact that communication architecture features have on the system performance. Explicitly, we base our analysis on the routing latency of NoC architectures, and develop statistical models capable of capturing the effect that different communication features have on this latency. Further, this this research introduces a new performance estimate needed for the analysis of the communication impact on system performance, at the highest abstraction level.

1.2 Problem Statement

Recently the area of system-level design has seen a migration from *computation-centric* design, to *communication-centric*. This migration is caused by the realization that communication is becoming the performance bottleneck in today's complex systems. System-level design can no longer perform computation-centric architecture exploration neglecting the communication effects on the system performance, and pushing the communication architecture into the later stages of system design.

Communication analysis must be included at the higher abstraction levels. Nevertheless, performing communication analysis at higher abstraction levels is not trivial. The main challenge is the lack of timing information at these abstraction levels. Without this timing information, it is difficult to define communication performance metrics and acquire accurate performance estimates to guide the space exploration.

This research focuses on the methods, tools, and modeling guidelines needed to estimate communication performance at the abstraction levels where the required timing details are not available. The key issues are (1) the abstraction of the communication architecture features, and (2) performance estimation given the abstracted communication features.

Thesis Statement: *System-level communication characteristics provide meaningful information that in the past has only been used for interconnect optimization. Random variables and statistical methods may be used in a novel manner to estimate communication performance at higher levels of abstraction, where most communication details are not available. A new system design paradigm is defined that evaluates communication characteristics at higher levels of abstraction in the design methodology and performance analysis. New in this communication-centric design is the extraction of the application's communication behavior and the abstraction of the platform communication characteristics. The application communication behavior and platform communication characteristics may be introduced at higher levels of abstractions using random variables, and statistical methods can provide the tools to better estimate the system performance at these levels.*

1.3 Objectives and Contribution

1.3.1 Objectives

The primary objective of this work is to describe the methods by wich communication information may be introduce at earlier stages into the design process. To this end this research explores modeling of the communication architecture features at higher abstraction levels, and puts forward a new performance estimate to help guide the design process. The objectives of this research span across two major areas: *system modeling* and *performance estimation*.

1.3.1.1 System Modeling

- This work sets out to define new abstraction levels for communication components. To accomplish this, it is necessary to study and understand the communication information available at the different abstraction levels, and more importantly, the communication information necessary to perform design decisions. This research presents a new communication driven system-level design space, and evaluates the impact different communication features have on the system performance. In particular Chapter 3 organizes the communication design features in categories of broad and narrow impact. This organization aims to point out what communication features must be covered by the higher abstraction level models, if these models are to be useful for early system-level design exploration.

- This research aims to provide new models ready to use in system-level design, but more important are the methods used to develop these models. As presented in Chapter 4, the high level models introduced here are derived using statistical tools, and reflect the low level behavior of the communication features that have broader impact. Additionally, Chapter 4 outlines the step designers will follow to produce probability distributions derived from empirical data, and modeled by known density functions, to build their own models, and further improve the communication analysis.

1.3.1.2 Performance Estimation

- A new high-level communication estimate is introduce, that provides the necessary tools for communication-centric design exploration. Called the probabilistic metric, it combines the proposed statistical model with the traffic characteristics to produce a communication performance estimator at the higher abstraction levels, where the lack of communication details normally prohibit such estimation. Using the probabilistic metric, designers can evaluate the communication effects earlier on the design processes. Chapter 3 explains the necessity for early communication analysis on today's embedded systems, and Chapter 5 presents the details of the development and implementation of the probability metric.

- Models, metrics, and estimation tools are useless without the necessary framework in which to apply them. To provide such framework, SystemC is chosen as the platform for implementing the models and metrics presented in this work. This framework integrates the statistical models along with SystemC simulation engine, and produces the necessary data for the probabilistic metric computation. Within this simulation framework designers and researchers can use the provided models for system design, or any new models they derive.

1.3.2 Contributions

The contributions for this research are in three main areas. (1) Modeling to include communication details at higher abstractions levels, (2) performance estimation, and (3) system-level design space exploration.

1. *Communication Modeling at Higher Abstraction Levels*

   - **More complete communication model at higher abstraction levels.** Through the use of statistical tools, these models incorporate more communication details than previously possible.

- **Tools and guidelines for building models that capture the system communication characteristics at higher levels of abstraction.** More important than the models are the methods and guidelines that developed these models. System designers benefit more from a set of guidelines that provides them the knowhow to construct similar models for their systems. This research outlines the steps and necessary information to develop statistical models for systems other than the ones included in this work.

2. *Performance Estimation*

- **Communication performance estimation at higher abstraction levels.** For any model to be useful in system design, it must provide performance estimates of the details that it models. This research introduces a new performance estimate for the new communication models developed. Known as the probability metric and based on contention behavior, this research is the first to show how this probability may be used as an estimator.

3. *System-Level Design Space Exploration*

- **Design space exploration through the probabilistic metric.** Performance estimates give information about the current model, but to perform design space exploration it is necessary to compare the estimates from different models. Traditionally estimates can be directly compared, but this may not be the case for statistical models. This research demonstrates the behavior of the probabilistic metric across different design alternatives, and shows how this behavior may be use to guide the design exploration.

1.4 Overview of Dissertation

This chapter starts by pointing out the productivity gap of Figure 1.1, page 1. System-level design, with its abstraction levels and refinement methodologies, is a solution to the productivity gap. To further improve system-level design it is necessary to include communication

analysis at higher abstraction levels, but this is difficult due to the lack of communication details at these higher levels. This research addresses this challenge and proposes a statistical model to overcome the lack of details. The rest of this document is organized as follows.

Chapter 2 presents the current state of the art for communication exploration at the system-level. An outline of the communication features important for system-level design and an overview of the statistical model are presented in Chapter 3. Subsequently, Chapter 4 formally introduces the statistical model and shows, through an example, how to derive a statistical model from simulation data. Following this, Chapter 5 discusses how the statistical model fits on the overall system-level picture, and how the probability metric is developed to combine dynamic application behavior and the communication architecture features. Coming full circle a design case study is shown in Chapter 6, putting the statistical model and probability metric into the context of system-level design. Finally, Chapter 7 covers the concluding remarks.

CHAPTER 2. Related Work

System level design addresses the current productivity gap by introducing new abstraction levels, allowing designers to manage progressively more complex systems. Along with these new abstraction levels, system level design introduces new design methodologies that, together with performance estimation, guide the designer from one abstraction level to the next. This chapter presents the state of the research on system level design, and especially how communication details are included throughout the design flow.

2.1 System Level Design

In the literature there are two major approaches to system level design, refinement-based[18] and platform-based[52]. For both design approaches, refinement and platform, the initial step is to represent the application(s) to be implemented as a specification model. The specification model is the highest level of abstraction containing the application behavior and design constraints, but none of the implementation details. The difference between the two system level design approaches resides in the process by which the specification model is transformed into a system implementation that meets the design constraints.

The SpecC methodology [23] is probably the best well known example of a refinement-based design methodology. The SpecC design flow is shown in Figure 2.1. The design starts with a specification model of the application. The specification model represents the behavior of the application and includes the design constraints (e.g. power, performance, area, etc). In the SpecC methodology the specification model is written in the SpecC language[21], other methods for specifying the system specification include MS Excel sheets[26], XML[47], and even UML[46].

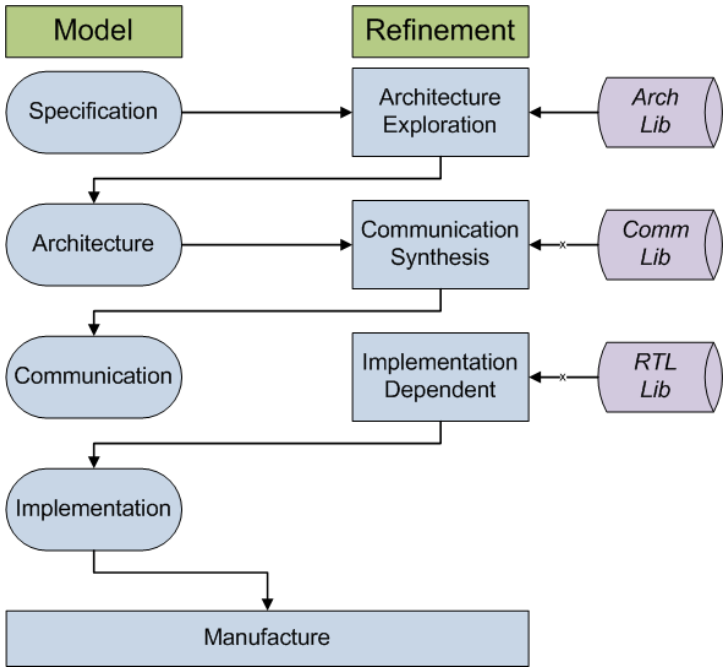


Figure 2.1: SpecC Design Flow

Through the architecture exploration the specification model is refined into an architecture model. This architecture model is the second abstraction level in the SpecC methodology. At this level the system architecture is modeled as a collection of processing elements (PE) that are approximate-timed models of the computation elements of the architecture.

The work in [11] show how the approximate-timed PE models are used for rapid design space exploration. This design space exploration depends on the performance estimates available at the architecture abstraction level. The research of [9] presents a method for performance estimation using weight-tables to represent the PE timing characteristics, and [54] shows how to improve the method in [9] by introducing more accurate low-level aware metrics. After the system architecture is decided, the next step is communication architecture exploration.

The communication model is the third abstraction level. In SpecC, the communication model is the result of the communication synthesis, a process by which the architecture model is refined into the communication model. The communication synthesis process refers to the narrow exploration of the communication design space, as shown in Figure 2.2. In the SpecC methodology the broader search in Figure 2.2 is, in fact, part of the architecture

exploration. Other works that do broader communication exploration, as shown in Figure 2.2, are introduced in later sections.

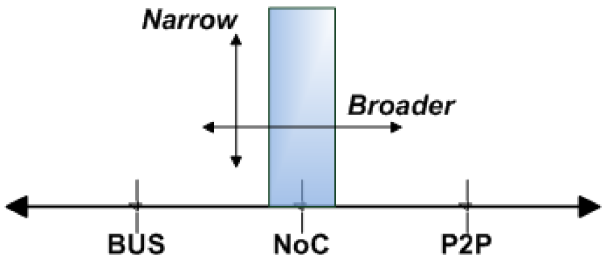


Figure 2.2: Communication Design Space

After the communication synthesis the resulting communication model reflects all the details of the final system implementation. With the resulting communication model designers can collect very accurate performance estimates to evaluate the final implementation before manufacturing. The final step is to transform the communication model to the physical l implementation, this is done through the manufacturing process.

As explained above, the SpecC design process flows from specification, to architecture, to communication, and finally to implementation. For communication intensive application, this refinement order may be a limiting factor. The problem lies in the architecture exploration. For communication intensive applications, the architecture exploration lacks the necessary communication architecture details to provide an accurate design space exploration. As communication intensive applications, and complex communication architectures start to dominate embedded system design, the limitation of computation-centric methodologies like SpecC becomes more apparent.

While SpecC is the most popular refinement methodology, it is not the only one. Another example of a refinement-based design approach is Transaction Level Modeling (TLM) [10]. While not a methodology, in the strictest sense, TLM borrows a lot of concepts from the SpecC methodology. Like the SpecC methodology, TLM defines different abstraction levels and the amount of detail that a model at each level contains. Unlike SpecC, TLM does not define the refinement processes by which one model is transformed into another.

Instead TLM lets the designer use any refinements he/she may see fit for the current

design. Several tools and methodologies became available to fill this gap in TLM. SystemC [2] is the tool most associated with TLM. SystemC is a C++ library that defines the constructs needed to build TLM models and includes a discrete event simulator used for executing SystemC models. Other SystemC-based TLM design tools and frameworks include Simics[41], CoWare[60], and SystemVerilog[56].

The second most common approach to system level design is the platform-based approach[52]. This approach differs from refinement-based in that the specification model is not recursively refined into an implementation. Instead, the specification model is directly mapped into a target platform that models all the implementation details. Performance estimates are gathered for the current mapping. If the design constraints are met the process stops, else another platform is considered, etc.

Metropolis [4] is an example of platform-based design framework. In Metropolis formal models are used to model each platform separately. The Metropolis methodology defines how a specification model is mapped into one of the platform models.

All of the tools mentioned above, regardless of the approach, initially trim the design space evaluating the effects of the PEs and neglecting the communication cost. While this order may seem intuitive, current research shows the pitfalls of not considering the communication cost early in the design process. The next sections show what researchers are currently doing to include communication behavior at higher abstraction levels.

2.2 System Level Communication Modeling

In the literature there are two distinct approaches to system level communication modeling. The first approach is to model the application communication behavior and use this information to guide the communication architecture design. The second approach is to abstract the communication features of the target platform and include these features in higher level models.

Representative to the first approach Deb et al. [16] evaluate the impact of control and data flow for DSP applications on system design. Tedesco et al. [59] explore the impact of

different traffic models for the same application, on the interconnect design, specifically on the quality of service (QoS). While this is not the first work to evaluate traffic models for interconnect design[32, 7, 24, 62], it is the first in evaluating the usefulness of the different models for a certain application class. Santi et al. [53] is another work on the impact of traffic on the QoS of the interconnect. Similar to [59], Santi et al. [53] characterize the traffic in terms of injection rates. What is new in this work is the use of traffic statistics to justify the need for QoS in the system implementation.

While the previous works used models to represent the applications, traces are also common on system design. Mahadevan et al. [42] presents a trace-based simulation environment. Unlike traditional trace-based design exploration, the traces used in this work are annotated relative timing. This relative timing information is used to map the trace to an architecture different than the reference design.

In system design, the application characteristics are also used for automated architecture generation. An example of this automated architecture generation is the two phase synthesis flow of $\times$pipes[5]. The first phase is where the system constraints are specified and the application characteristics are introduced. The second phase is the automated process of NoC architecture generation. The use of application characteristics for automated architecture generation is different from previous works, where the application characteristics were used to directly evaluate some performance metrics. Ho and Pinkston [30] present another work where the communication characteristics are used for automated on-chip interconnection network architecture generation. Different from [5], Ho and Pinkston [30] focus only on well-behaved communication patterns.

Chandraiah et al. [12] show yet another approach to the application communication analysis. Instead of focusing on how to better represent the application behavior, Chandraiah et al. [12] addresses the issue of how to build the specification model to better include the application communication features. Based on the SpecC methodology[23], this work presents an automated process to convert an specification model with non-explicit communication through global variables, into a model with explicit through abstract channels.

The second approach to system level communication modeling is to evaluate the com-

munication characteristics of the implementation platform and integrate these characteristics into the higher abstraction level models. Knudsen and Madsen [34] is one of the earlier attempts at integrating architecture details into the system design. As part of the LYCOS co-design framework [27], this work evaluates the timing information and implementation metrics (e.g. area, and power) for PCI and USB protocols and shows how to use these information to guide the partitioning step. Most recently, Pasricha et al. [50] takes a different approach, where instead of including the protocol timing information, the communication is evaluated in terms of transactions. Pasricha et al. [50] presents a model where (1) the communication behavior is characterized by the type of transactions, and (2) cycle accurate figures are know for each transaction type on the target platform.

In a more direct approach to communication architecture abstraction Kumar et al. [36] describes NoC architecture as a collection of communication resources and computation placeholders. The communication resources are further abstracted through the use of communication layers based on the OSI model. This layering approach to communication architecture abstraction has been adopted by others [57, 6, 25]. A similar approach is found in Coppola et al. [13], where a C++ library is introduced to facilitate the modeling of layered interconnection networks.

Other works look at the effects that different communication architectures have on an application or set of applications. For example, Lee et al. [39] evaluate different communication architectures for an implementation of the MPEG-2 video application. The application is implemented in three different communication architectures, bus, point-to-point (P2P), and NoC. This work is the first showing the true impact of these very different communication architectures in the system performance of one application. In a similar study, Bononi and Concer [8] evaluate several architectures and compares analytical versus quantitative results for a ring, 2D mesh, and the new spidergon mesh.

2.3 System Level Performance Estimation

While most work in the area of system level design tries to define new models or methods to include more information into the existing abstractions levels, all of this work aims at better performance estimation. Performance estimation is key to system level design space exploration. Simulation is the most common method for system level performance analysis.

Gajski et al. [23] is an example of simulation-based performance analysis. Simulation is used to introduce the application behavior, and the architecture details are included through back annotation of the timing details. In Gajski et al. [23] communication is included in the system performance only after the computation architecture has been explored and chosen. A similar approach is adopted by Baghdadi et al. [3]. However, Lahiri et al. [38] show the pitfalls of exploring the computation architecture without considering the communication cost.

In contrast Dey and Bommu [17] introduce a technique for estimating the communication performance of concurrent processes during the computation architecture exploration. Communication layers are defined as the relative times where the concurrent processes synchronize. Performance estimation is done in each layer separately. Another example of communication-centric performance analysis is Loghi et al. [40]. Loghi et al. [40] presents a SystemC on-chip communication simulation environment for multi-processors system-on-chip (MPSoC) architectures. Other examples of communication-centric performance analysis may be found in Kim et al. [33] and Fummi et al. [22].

In this research, a probabilistic approach is proposed for system level performance estimation. However, this research is not the first to propose such an approach. The works of Kumar et al. [35] and Sonntag et al. [58] are two good examples of probabilistic approaches to system level performance estimation.

Kumar et al. [35] evaluates the case where multiple application content for shared processing elements, and probability is used to estimate the system delay due to the contention for the shared computation resources. To use the probabilistic approach of [35] it is necessary to know (1) the application execution times on the processing elements, and (2) which

application fractions content for the shared computation resources. Through the approach described in [35] it is possible to compute better than worst case estimates in a fraction of the time required to simulate the cycle-accurate design. The drawback is the amount of information required for this approach, since it is necessary to know the application execution times on the shared resources. Also, this method does not account for the communication overhead between the processing elements, or shared communication resources.

Another common probabilistic approach to system level performance estimation is queuing modeling, as it is done in SystemQ[58]. SystemQ is a SystemC queuing-based simulation environment. The different abstraction levels are defined through queuing theory. Different to previous approaches SystemQ does incorporate communication effects into their design environment.

In SystemQ, separation of concerns[61] is defined along three orthogonal axes: function, structure, and communication. *Functional* refers to the algorithmic behavior, *structural* refers to the computational architecture, and *communication* refers to the communication architecture. A queuing model is built to represent the system at an abstraction level. Refinement steps are defined to transform the queuing model by adding *functional*, *structural*, and *communication* details.

A system is refined throughout four levels of abstractions, named setup 1 through 4. Each setup adds details across one or more of the orthogonal concerns defined above.

[Setup 1] This is the highest level, most abstracted, and the entire system is modeled as a queuing network of two queuing systems, a producer and a consumer. Average service delays, derived from expert knowledge, are used in this model.

[Setup 2] This setup is generated through structural refinement of Setup 1. In Setup 2 each queuing system of Setup 1 is replaced by queuing network that reflects the structural details of the implementation platform.

[Setup 3] Functional and communication refinements are applied to generate Setup 3. The service time in this setup is determine for variable size packets, instead of us-

ing average packet length as in Setup 2. Communication is refined to account for mean arbitration and contention delays in the target communication architecture.

[Setup 4] This last setup is the result of further structural refinement to Setup 3. Setup 4 includes shared components between the original producer and consumer.

This chapter showed the current state of the system level design research. In particular, it showed the trends on system level communication analysis, and the approaches for communication performance estimation. Subsequent chapters will show the details of the proposed approach.

CHAPTER 3. Probability as a System Metric

This research proposes the use of probability as a high level estimator for system performance. The use of probability as a metric stems from the lack of communication details at higher abstraction levels. This chapter introduces the concepts necessary to understand the development and use of the system probability metric.

3.1 Communication Architecture Design Alternatives

To discuss system performance estimation it is necessary to first understand the communication architecture design alternatives and their effects on the systems performance and cost. Intuitively, as the system model is refined from specification to implementation relations between the design alternatives, system performance, and implementation cost become more concrete. Performance estimation is more accurate at lower abstraction levels due to availability of implementation details. This section presents the relations between a subset of communication architecture design alternatives and the system performance. Whereas, the next section shows why it is impossible to directly measure the effects of these design alternatives on system performance at higher abstraction levels, justifying the need for the proposed probability estimator.

At higher levels of abstraction the interconnection network may be seen as a shapeless communication medium. Shown in Figure 3.1, is an abstract view of the interconnect network at the specification and architectural abstraction levels. From a communication point of view, the system is composed of nodes, PEs in the figure, and an interconnection network that is subdivided into links or communication channels. Each node in the system is capable of sourcing or sinking network messages. Each node, also, contributes to the interconnection

network by performing routing duties. That is, a node will route any message for which it is not the source or sink. Figure 3.1 is meant to show the lack of communication details found at these higher abstraction levels. The questions then are, what communication details are important at these abstraction levels, and what is the best way to include these communication details at higher levels? In other words, what is the best method for giving shape to the interconnect network model of Figure 3.1?

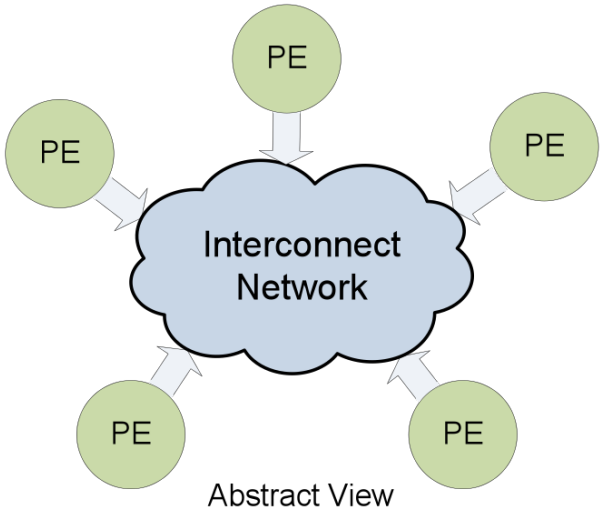


Figure 3.1: Abstract Interconnection Network

The first step is to assess the design alternatives available for the communication architecture. At higher abstraction levels, the communication architecture design space is a three-dimensional space along topology, routing, and flow control. All of these communication architecture features have direct influence on the system performance. The following is a list of the design alternatives and their broader effects on system performance and cost.

- **Topology** is the static arrangement of nodes and links. The topology has a direct effect on the throughput and latency of the interconnection network, as it determines the node degree and defines all possible paths on the network. The topology cost is reflected in the number and complexity of communication components, and in the density and length of the interconnection links.

- **Routing** defines the selection policy for choosing an specific path from those given by the network topology. While in a lesser degree than topology, routing still has a direct

impact on the interconnect throughput and latency. Routing costs are measured in node complexity.

- **Flow Control** represent the policies for resource allocation. Flow control is probably the feature that has the biggest dynamic impact on the communication performance, since it handles contention resolution. As for routing, the cost of flow control implementation is also reflected in the node complexity.

3.1.1 Communication Performance

Before continuing, it is necessary to define how the communication performance is measured and analyzed. Communication through the interconnection network is done exchanging messages. A message is the largest logical unit of data that is delivered from source to destination. To traverse the interconnect network, a message uses resources: links, buffers, control logic, etc. These messages may be arbitrarily long, depending on the communication needs of the participating nodes, therefore it is not convenient to allocate the network resources to the messages. Instead, messages are divided into fixed length packets.

A packet is the basic unit of routing. They have a fixed maximum length, and are subdivided into header and payload. The packet header is used to determine the route taken by the packet from source to destination. Similarly to how messages are split into packets, packets may be further divided into flow control digits, or flits. Figure 6.1 shows a network message and all of its subdividing units. Flits are the basic unit of bandwidth and storage allocation, and they carry no routing information. Thus, all flits in a packet must follow the same path from source to destination.

Depending on its position on the packet, a flit may be a head flit, body flit, or tail flit. A head flit is the first flit of a packet and carries the packet routing information. All bandwidth and storage allocation for the packet is performed by the head flit. The head flit is followed by zero or more body flits and one tail flit. The tail flit is the last flit of the packet and is most commonly used for resource deallocation. Finally, a flit may be divided into physical transfer digits, or phits. A phit is the unit of information that is transferred across a channel

in a single clock cycle.

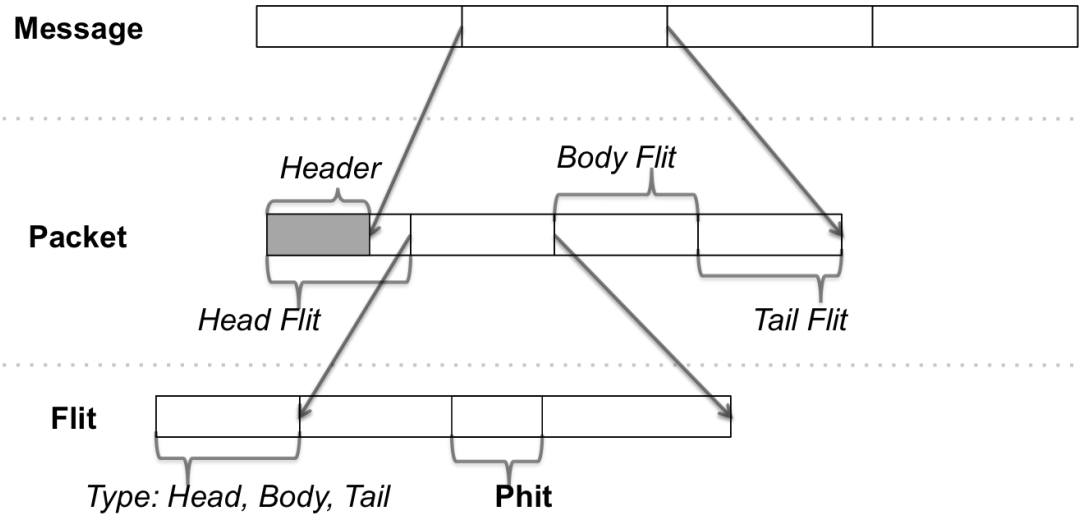


Figure 3.2: Network Message

Knowing how the network message is decomposed for traversing the network, it is possible to define the network performance. The most basic measurement of performance is latency. The latency of a network is the time required for a packet to traverse the network, from the time the head of the packet departs the source node to the time the tail of the packet arrives at the destination node. From this definition of latency, it is clear that the two major components of latency are: the path a messages takes from source to destination, and the size of the message. The message path is measured in hops, or routing nodes. These are the nodes in between the messages source and destination. The number of hops, or path length, is determined by the network topology and the routing protocol. As the message travel across the network, it allocates resources across its path. The amount of resources needed by a message is determined by the message length and the flow control policies.

The second most common network performance metric is its throughput. While latency is a measurement of the time needed by a message to travel the network, while throughput measures how quickly the interconnection network process a message. Formally defined as the data rate, in bits per cycle, that the network accepts per input port, throughput is a property of the entire network and depends on routing and flow control as much as on topology.

To finish the discussion on the communication performance it is necessary to include bandwidth. Bandwidth measures the capacity of the communication medium to move data, and it is a function of the width and speed of the medium. Width refers to the number of bits that may be transmitted in parallel. The width is always the same as the size of a phit. Speed is the maximum frequency at wich this bits may be switched. This switching speed is a property of the communication medium. Having introduced the communication performance metrics, the next step is to look at how the communication performance affects the overall system, and how these performance metrics may be investigated at higher abstraction levels.

3.1.2 System Performance

Intuitively the system performance is affected by the interconnection network throughput and latency. The rest of this chapter only deals with latency, but similar analysis may be done for throughput. In system design the important issue is the relation between the design alternatives, the system performance, and the implementation cost. In the case of latency, and for the sample system of Figure 3.3, a well known relation between the overall system performance and the lower level communication metrics is the path latency.

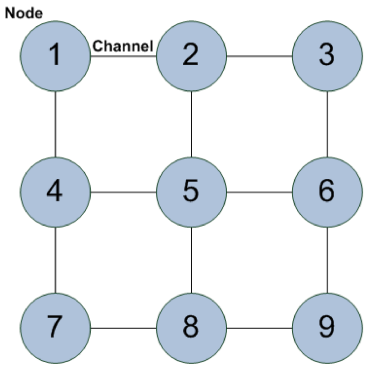


Figure 3.3: 3x3 Mesh Interconnection Network

Path latency, T_n, is defined in Dally and Towles [14] as

$$T_n = \sum_{i=0}^{n} \left(t_{ir} + \frac{L}{BW} \right) \tag{3.1}$$

Where n is the number of nodes between source and destination, and it is as much influenced by topology as routing. L represents the size of the packet in flits. L is determined by the flow control policy. BW is the bandwidth of the communication channels. The term L/BW is know as the travel time, t_t. Finally, t_{ir}, known as routing time, refers to the time a packet resides in an intermediate node in the network. Routing time is a function of the topology, routing, flow control, and even traffic. For the simple example of Figure 3.3, let's consider

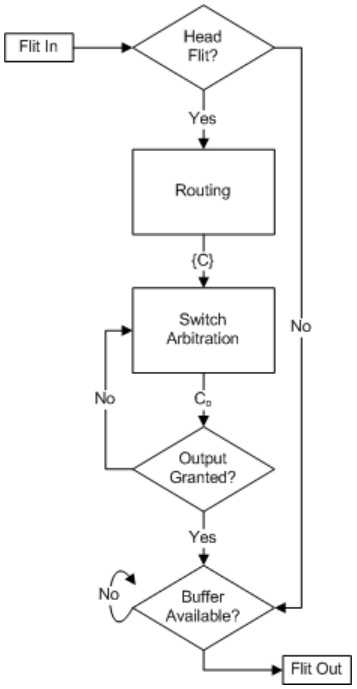


Figure 3.4: Typical Routing Flow

the typical routing flow shown in Figure 3.4. This routing flow allocates resources at the flit level. For a common implementation of the routing flow of Figure 3.4 routing time (t_r) may be between 40 to 50 cycles, depending on the number of available output channels and buffers[29].

From Figure 3.4 it is clear how the design alternatives on topology, routing, and flow control have a direct effect on t_{ir}. For example, topology and routing determine the amount of time a head flit spent in the *Routing* process. Further, the time required for switching arbitration is a function of flow control as well as topology, routing, and current network load. Finally, the time a flit waits for buffers to become available depends on flow control

and load.

3.2 Communication Modeling at Higher Abstraction Levels

This work focuses on performance estimation at the highest levels of abstraction. These abstractions levels are characterized by approximate-timed computation, and un-timed (or approximate-timed) communication. This section presents the key issues that make communication performance estimation an interesting and significant problem at higher abstraction levels, and provides an overview of how probability may be used as a performance estimator.

3.2.1 Performance Estimation for Un-timed Communication Models

As shown in section 3.1.2 the performance for the sample system of Figure 3.3 is determined by the path latency of equation 3.1. This section presents how the different components of equation 3.1 are found at higher abstraction levels where communication is untimed.

As defined by Cai and Gajski [10], a model with un-timed communication is characterized by concurrently executing processing elements and communication through abstract channels. These channels are message passing channels, which only represent data transfer or synchronization between processing elements. No timing information about the communication architecture is included in either the processing elements or the channels.

Using the previous definition, equation 3.1 is evaluated for the un-timed communication model of the sample system in Figure 3.3. On each node a packet is delayed by the routing time, t_{ir}, and the travel time, $t_t = \frac{L}{BW}$. This example focus on routing time, and it is assumed that L and BW are known.

Routing time is determined by the implementation of the routing flow (Figure 3.4, on page 23) in the routing nodes. For the case of un-timed computation/communiation models, only the behavior of the routing flow is included. This means that whenever a packet arrives at a routing node the routing decision is made instantaneous. Moreover since a packet is never held by the routing node there is never contention for the node resources by other

packages.

From the previous analysis, it is clear that, $t_{ir} = 0$ for the un-timed model. Unfortunately this reduces path delay to

$$\begin{aligned} T_n &= \sum_{i=0}^{n} \left(\frac{L}{BW} \right) \\ &= H \left(\frac{L}{BW} \right) \end{aligned}$$

which is clearly not a useful relation for accurate performance estimation. Further, the number of nodes in the path, H, may change depending on dynamic effects due to traffic.

Therefore, to proceed with the system design it is necessary to include timing details of the routing implementation to the higher abstraction levels. The proposed solution is to use a probabilistic timing model, or p-timed model. Contrary to the typical approximate-timed models, a p-timed model does not use back annotation of cycle time characteristics to include timing information. Instead a p-timed model relies on a probabilistic description of the target implementation.

For the current example a good probabilistic description of the routing flow of Figure 3.4 is a discrete uniform distribution. Routing time may be expressed as a random variable of the form

$$\begin{aligned} t_r &\sim U(45, 50), \text{or} \\ t_r &\sim U(0, 5), \end{aligned} \tag{3.2}$$

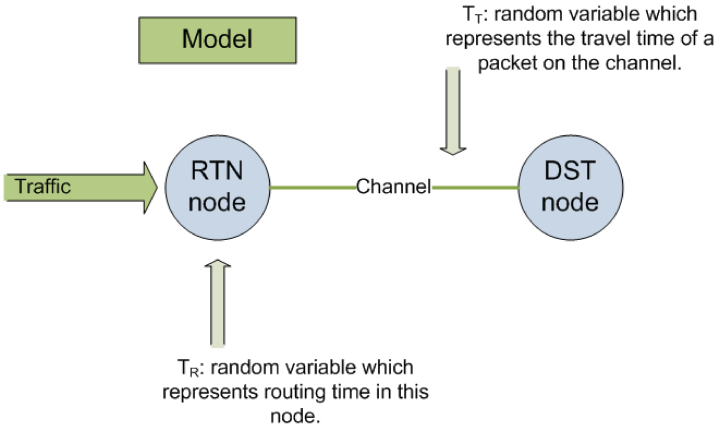


Figure 3.5: Node P-Model

since only the absolute difference is important. A refined node p-model is shown in Figure 3.5. With the definition 3.2 in hand the next step is to compute the path delay of equation 3.1.

Since $\frac{L}{BW}$ is assumed constant for the current analysis, the only part left to evaluate is path length H. For better system performance estimation, it is necessary to account for the dynamic behavior of H. That is, it is necessary to account for the effects of the network load and flow control policy on the path length.

Through simulation of the approximate-timed computation and p-timed communication model it is possible to determine a set of characteristic paths $\{H_0\}$ for a given traffic. Figure 3.6 shows the simulation model. The final step to estimate the system performance is to combine the p-timed model with the set $\{H_0\}$ to generate the probabilistic metric.

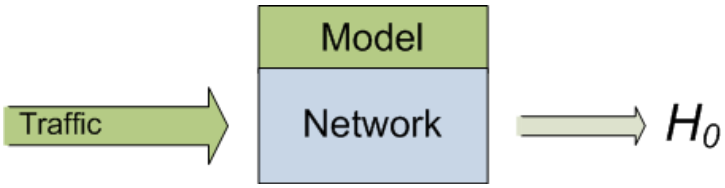


Figure 3.6: System Model

3.2.2 The Probabilistic Metric

Section 3.2 presented an overview of the p-model for a sample design, and how simulation may be used to include the dynamic effects of traffic. In this section everything comes together to generate the final performance estimate, i.e. probabilistic metric.

The first step is to partition the path set $\{H_0\}$ into smaller sets $\{h_0\}$. Where $\{h_{0i}\}$ is a set of paths, p_i, such that:

$$\begin{aligned} H_0 &= \{h_{01}h_{02}\ldots h_{0n}\}, \\ \bigcap^{n} h_{0i} &= \emptyset, \\ h_{0i} &= \{p_1 p_2 \ldots p_k\}, \\ \bigcap^{k} p_i &\neq \emptyset, \\ P_c(p_m, p_n) &> 0. \end{aligned}$$

Each path, p_i, is the characteristic path for a given packet, and $P_c(p_m, p_n)$ is the probability that the packet in path p_m collides with the packet in path p_n. Therefore, only paths carrying packets that have a probability of collision greater than zero ($P_c > 0$) belong to the same set (h_{0i}).

Given packets M and N with respective paths p_m and p_n, shown in Figure 3.7, collision will occur if both packets share a link, shown in green on the figure, and each packet requires the shared link at the same time T. Packet M uses the shared link during the time interval $[T_m, T_m + \tau]$, where T_m is given by

$$T_m = t_{m_{1r}} + t_{m_{2r}}, \tag{3.3}$$

τ is a constant, and $t_{m_{1r}}$ and $t_{m_{2r}}$ represent the routing times for nodes 1 and 2 respectively. From the analysis in section 3.2.1 the routing time follows a discrete uniform distribution, i.e.

$$t_{m_{1r}} = t_{m_{2r}} = t_r \sim U(0,5), \tag{3.4}$$

and with equations 3.3 and 3.4 it is possible to compute the probability of collision $P_c(p_n, p_m) = P(T_M = T_M) = 1/7$, for $\tau = 0$.

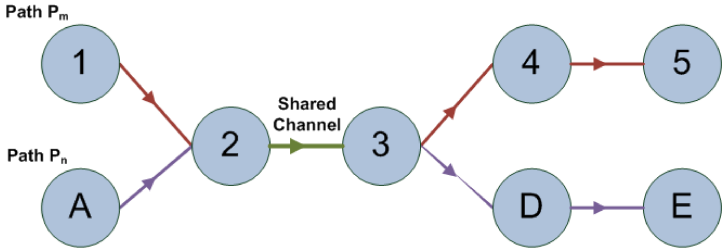


Figure 3.7: Paths p_m and p_n that share one channel.

The same analysis is done through the paths of $\{h_{0i}\}$ to compute the probability of collision $P_c(h_{0i})$. The process is then repeated for every the set, $\{h_{0i}\}$, in $\{H_0\}$. This produces a probability mass function which represents the system at the current level of abstraction, a probabilistic metric.

This section presented a sample system at the un-timed communication abstraction level, discussed the effects of routing on the system performance, and showed the limitations for

performance analysis at this high abstraction level. To overcome these limitations a probabilistic approach is proposed. A probabilistic model is presented to capture the architecture features, particularly routing, and a probabilistic metric is derived as a high level performance estimator. The probabilistic metric is derived using the probabilistic representation of the architecture features, and simulation to include the system dynamic behavior.

3.3 Probability Metric for System Level Design

Previous sections presented the probabilistic metric for an specific example. This section shows how this metric may be used for system level design.

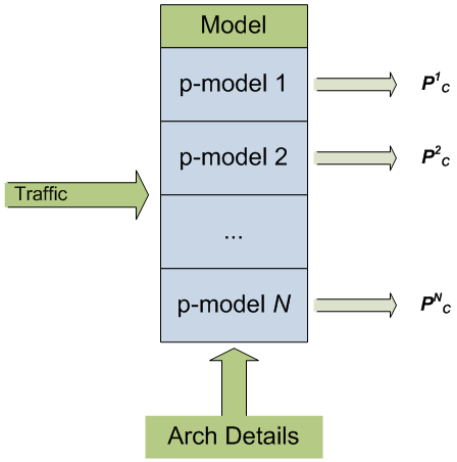


Figure 3.8: System Level Exploration

Recalling from section 3.2.2, it is possible to compute a probabilistic metric for a set of architecture options and a given traffic pattern. Figure 3.8 shows how different p-models representing certain architecture details yield different probabilistic metrics. The resulting probabilistic metric becomes a representation of the chosen architecture features and their interaction with the traffic pattern.

System design now continues by comparing the probabilistic metrics for different model-traffic combinations, P_c^1 through P_c^N, in Figure 3.8. The key to this probability driven system design is the relation between the high level probability of the model and the low level performance of the implementation, as show in Figure 3.9.

This relation between probability and performance is one of fidelity. That is, there is a

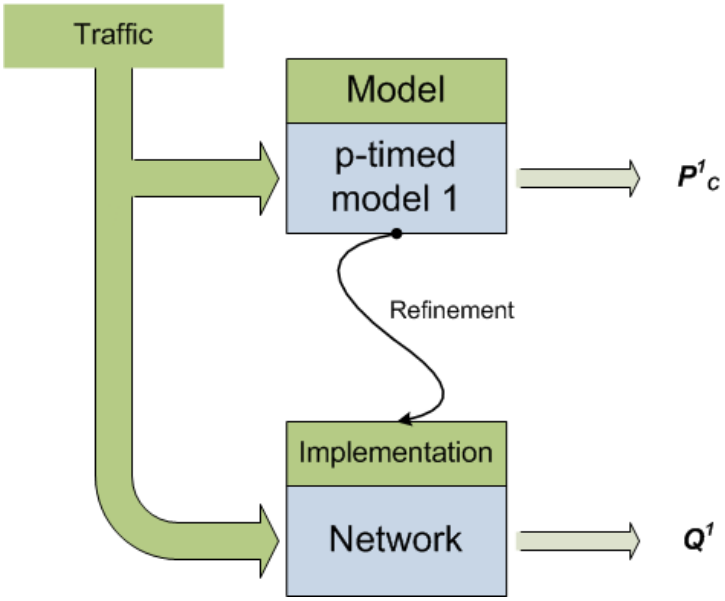


Figure 3.9: Model to Implementation

relation for comparing P_c^1, P_c^2, and P_c^3, such that

$$P_c^1 > P_c^2 > P_c^3 \Longrightarrow Q^1 > Q^2 > Q^3. \tag{3.5}$$

The derivation of this relation is not trivial and may be dependent on the architectural features under testing.

This chapter introduced the major limitations for performance estimation at the un-timed communication abstraction level. A probabilistic approach to system modeling is proposed to surmount these limitations, and through an example it was shown how the proposed approach may be used for performance estimation by generating a probabilistic metric. Lastly, it was shown how the probabilistic metric may be used to guide the system level design, and the importance of fidelity between the probability metric and the low level performance. The next chapters show the methods and tools by wich probability models were developed for several routing schemes found in typical embedded systems.

CHAPTER 4. Statistical Model

The goal of system-level design is to provide tools to measure the impact that low-level design decisions have on the system performance. In the case of the communication design alternatives, system performance is affected by the communication throughput and latency. The work presented in here shows how to develop good statistical models to capture the low-level latency relations and bring this information into the realm of system-level design.

4.1 Modeling Communication Components

The key modeling issue in system-level design is the relation between the design alternatives, performance metrics, and implementation cost. For latency a well known relation is the path latency[14]. For a path with n nodes and k channels, the path latency, T_n, is defined as:

$$T_n = \sum_{i=0}^{n} t_{ir} + \sum_{j=0}^{k} t_{jt} \tag{4.1}$$

Equation (4.1) has two components, routing time and traveling time. Routing time, t_r, refers to the time a packet resides in an intermediate routing node. Traveling time, t_t, is the time a package utilizes the communication channel between two nodes, and can be expressed in terms of the packet size and channel bandwidth.

Equation (4.1) shows how the design alternatives on topology, routing, and flow control have a direct effect on latency through their impact on routing and traveling time. For example, for a typical implementation of deterministic routing with wormhole flow control, topology and routing greatly affect routing time for the head flit at every routing node. It is shown in [29] that routing time may vary between 40 to 50 cycles, depending on the number of available output channels and buffers. In fact, most components of Equation (4.1) are

the result of one or more design alternatives. The path length, n, is as much influenced by topology as by the routing protocol. Routing time, t_r, is a function of the topology, routing, flow control, and even traffic. Traveling time, t_t, is heavily influenced by the packet size which is determined by the flow control.

Path latency provides a starting point for evaluating the relations between communication design alternatives and system performance. However, path latency, routing, and traveling time are still low level performance metrics. For these metrics to be meaningful at the system-level, it is necessary to provide models and methods to estimate the performance impact system-level design decisions have on these metrics.

Statistical models are widely used in the networking and communication research fields. For communication components, statistical models are most suitable for bridging the gat from the low-level implementation metrics to the system-level. Based on random variables, statistical models capture the behavior of communicating processes, and along with statistical methods, can provide the tools necessary to move the communication exploration into the higher levels of abstraction.

Using Equation (4.1) as the basis for this research, the challenge is how to properly model the path latency using random variables. The two candidates to model as random variables are, routing time (t_r) and traveling time (t_t). It is useful to see how these components fit in the system. Figure 4.1, on page 32, shows the relation between the routing and traveling times, and the communication components on the system model.

From Figure 4.1 it is clear that routing time captures the system design alternatives defined as part of the node architecture. These alternatives include number of IN/OUT ports, buffer sizes, routing scheme implementation, flow control implementation, among others. On the other hand traveling time encapsulates the design alternatives related to the communication medium, including , but not limited to, bandwidth and packet size. Also present in Figure 4.1 is the dynamic phenomena, or traffic.

While random variables may properly model the characteristics of the routing and traveling time; these are not sufficient for system-level design. Along with the new statistical models, there must be methods and tools to capture both the static design alternatives and

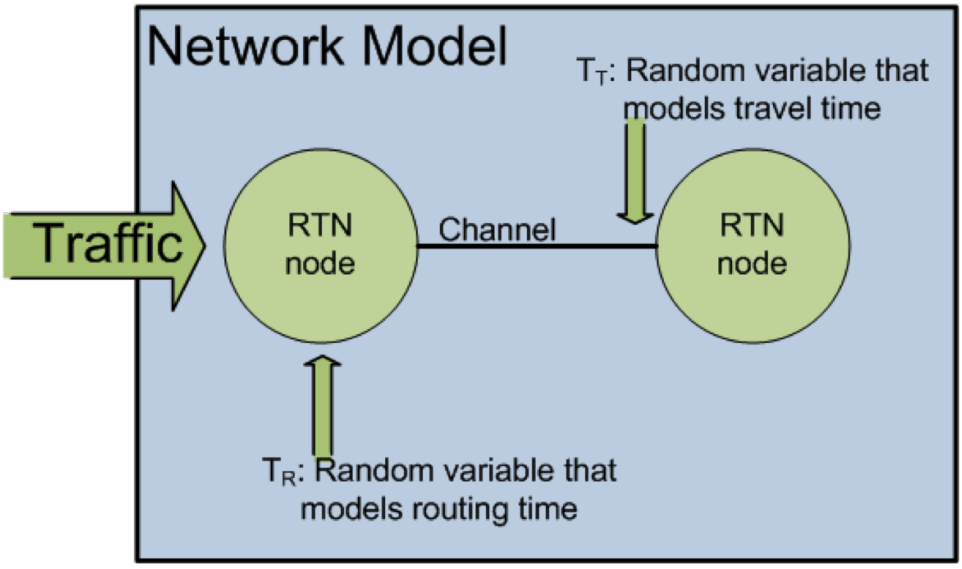


Figure 4.1: Communication Components in System

the dynamic phenomena due to load changes. The rest of this section presents the methods for developing the a statistical model for the particular case of routing time.

4.2 Developing a Statistical Model

Modeling the routing time using an statistical model can provide the abstraction necessary to bring the communication information into the higher abstraction levels of system design. However, the process for developing such statistical model is not trivial. The initial challenge is to determine the behavior of routing time. To this end, I've set up the following system:

- 64 nodes arranged in an 8x8 torus,
- XY deterministic routing, and
- wormhole flow control

This system was developed in SystemC using a modified version of the NOXIM[19] simulator.

The characteristics of these systems were carefully selected. Deterministic routing is still the most commonly routing type used [37, 43, 44, 31]. The advantages of deterministic

routing are many, simple design and analysis, ease of implementation, low latency for well behaved loads, etc. Likewise, wormhole flow control dominates the NoC architectures[51, 44, 15, 55]. Flit-based flow control schemes, such as wormhole, are popular because they provides a level of data management that is ideal for the kind of transfers and resource constrains commonly found in today's NoC.

Two traffic loads are considered: random, and hotspot. The random traffic uniformly distributed across all destinations, and is injected into the system with a poisson distribution at a given rate. Hotspot traffic is simulated using a hotspot that is 20% of the system node. Therefore, 80% of all nodes are sending to the other 20%. Just like in the case of random traffic, the packets during hotspot simulation are injected following a poisson distribution.

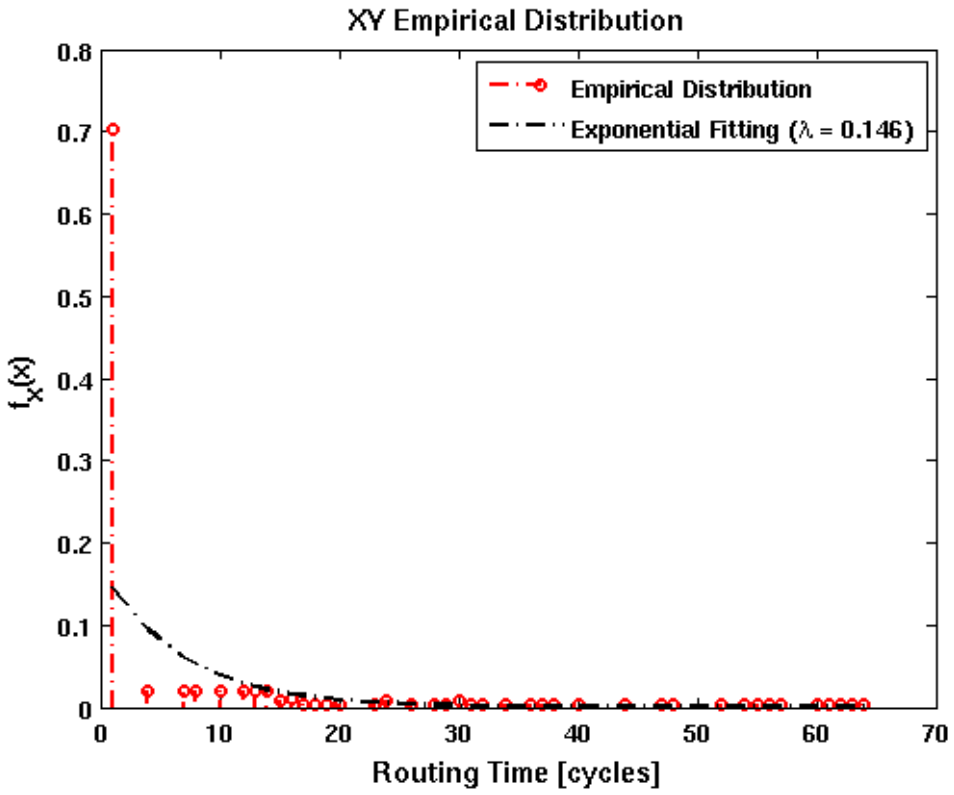


Figure 4.2: Routing Time Distribution and Fitting for Random Traffic

Figure 4.2 shows the empirical distribution of routing time for the random traffic case. This distribution is the basis for the random model. As seen in Figure 4.2, routing time in this system follows an exponential distribution. Several characteristics of this distribution

are worthy to point out:

1. One cycle is the most frequent latency.
2. About 80% of the area of Figure 4.2 lay within the first 6 cycles.
3. The average simulation path latency is 73.99 cycles/flit.
4. The average simulation path length is 5.3 nodes.

Path latency is computed as the end-2-end delay of the packet, as it travels from source to destination.

The most common method for developing random models from empirical data is to fit the data to a known distribution. Table 4.1 shows the average routing time from simulation, along with the mean, standard distribution, and mean square error (MSE) for an exponential fitting of the simulation data for random traffic. This table confirms what can be seen graphically in Figure 4.2, that an exponential fitting of the empirical distribution does not produce a good model.

Table 4.1: Exponential Fitting, XY Routing, Random Traffic

Traffic	$Routing_{ave}$	*Mean*	*Std. Dev.*	*MSE*
Random	7.85	6.85	46.93	0.2193

The problem with the exponential fitting is that it can't model the behavior of the routing time for the first data points. The routing time distribution decreases very quickly during the first three data points, however it still has a long and narrow tail. These properties are difficult to match with an exponential fitting alone. The solution is to use a mixed distribution.

4.3 A Mixture Model

A known good model for systems characterized by long-tail exponential, such as the systems presented here, is the mixture model. This section presents the steps I took to develop mixture models for our system under random and hotspot traffic. Mixture models[20] are of

the form

$$f_X(x) = \sum_{i=0}^{n} a_i f_{Y_i}(x) \tag{4.2}$$

Where the density functions $f_{Y_i}(x)$ are the *mixture components*, and a_i are the *mixture proportions*. With the constraints that $0 \leq a_i \leq 1$ and $a_1 + a_2 + \cdots + a_i = 1$.

To develop the mixture model the routing time distribution of Figure 4.2 is separated in two sets, or windows. For a simulation set of n data points, the first window contains the data points from 0 to w; and the second set has the rest of the data points, from $w+1$ to n. These two sets, $(0, w)$ and $(w+1, n)$, become our *mixture components*. Figure 4.3 shows the mixture components as they are derived from different windows. Based on this analysis our mixture model has two parameters:

1. the mixture proportion (a) from equation 4.2, and
2. window size (w).

The mixture proportions are determined using equation 4.3. $T_{r_{ave}}$ is the average routing time from simulation, and μ_w, μ_{w+1} are the means of the mixture components $f_{Y_0}(x)$ and $f_{Y_1}(x)$,

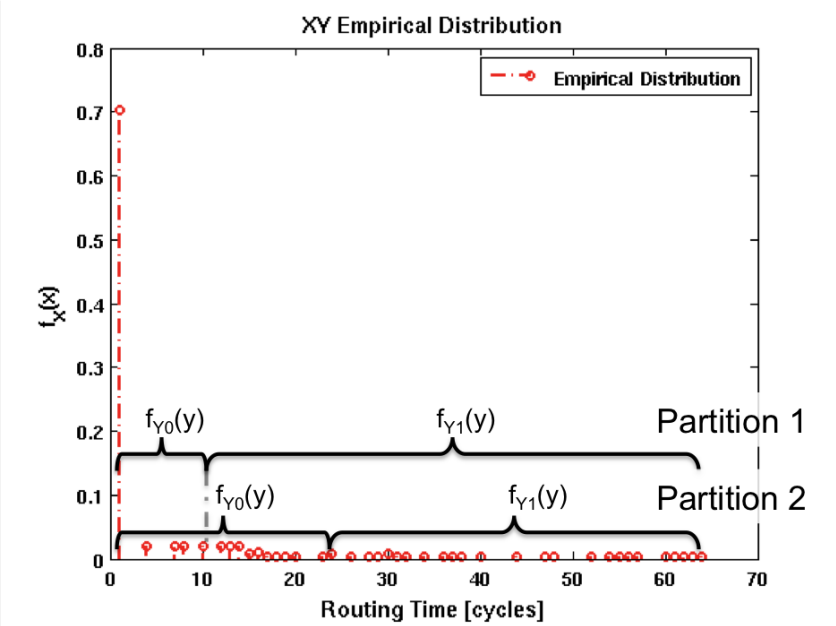


Figure 4.3: Communication Components in System

as shown in Figure 4.3.

$$T_{r_{ave}} = \mu_w * a + \mu_{w+1} * (1 - a). \tag{4.3}$$

Window sizes, on the other hand, are not as clearly defined as the mixture proportions. Different window sizes produce different mixture models. Therefore, it is necessary to determine the window size that produces the best mixture model. To determine the best window size we use the mean square error (MSE) and measured the error between a mixture model for a particular window to the empirical distribution from simulation.

Figure 4.4 shows the MSE as a function of the window size. The blue horizontal dotted line labeled *Exp. Fitting MSE* represents the MSE of the exponential fitting of Figure 4.2, and the red plot labeled *Mixture Proportion* show the different mixture proportions as the window size increases. There are three distinct sections in Figure 4.4: $w \leq 6$, $6 < w \leq 8$, and $w > 8$.

The first section, $w \leq 6$, is characterized by the largest MSE reduction rate. Knowing that about 80% of the routing time distribution is stored on the first 6 cycles, it is expected that the MSE would decrease significantly faster as these data points are integrated into the $f_{Y_0}(x)$ mixture component. After this point, $w = 6$, MSE decreases at a lower rate because any

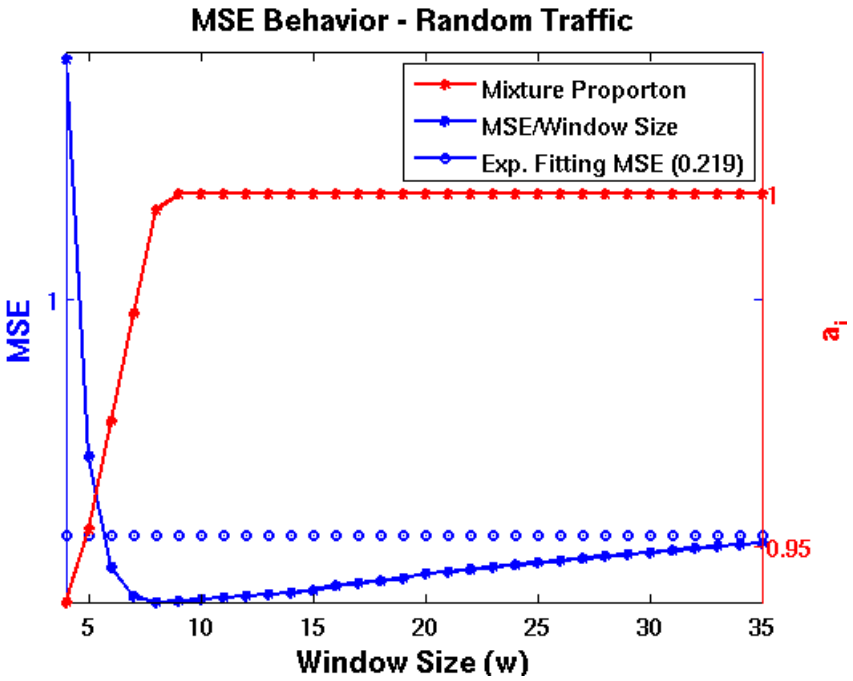


Figure 4.4: MSE Behavior Across Window Sizes (Random)

more data moved into $f_{Y_0}(x)$ adds significantly less information. It is important to note that already with 80% of the routing time distribution in $f_{Y_0}(x)$, the mixture model with $w = 6$ has an MSE lower than the MSE from the exponential fitting of Figure 4.2.

On the second section, $6 < w \leq 8$, the MSE continues to decrease, but at a lower rate. MSE reaches a minimum at $w = 8$. At $w = 8$, almost 85% of the routing distribution is now in $f_{Y_0}(x)$. The resulting mixture model at $w = 8$ is

$$\begin{aligned} f_{T_R}(x) &= a_0 f_{Y_0}(x) + a_1 f_{Y_1}(x), \text{ where} \\ f_{Y_0}(x) &\sim Exp(\mu = 1.4120), a_0 = 0.9961 \\ f_{Y_1}(x) &\sim Exp(\mu = 21.428), a_1 = 1 - a_0. \end{aligned}$$

Looking at the mixture proportions a_0 and a_1, it is clear why this mixture model produces a better MSE. It gives a higher weight (0.9961) to the mixture component representing almost 85% of the routing time distribution, $a_0 f_{Y_0}(x)$.

The last section of Figure 4.4, $w > 8$, is characterized by an increase in MSE. This last section represents when the information on the tail of the empirical distribution starts to impact the mixture model. Finally, as the window size increases, $w \to \infty$, our mixture model becomes the same as an exponential fitting over the entire simulation set.

In summary, the MSE analysis above shows that using mixture models it is possible to find a better fit than simply fitting the simulation data to an exponential distribution. Furthermore, Figure 4.4 shows that for all windows sizes $6 < w < 35$, the mixture model has a lower MSE than the exponential fit of Figure 4.2. Graphically the mixture models for windows $w_{[6\ 8\ 20]}$ are shown in Figure 4.5.

4.4 A Communication-Centric Modeling Methodology

As a matter of example, the previous section showed how to derive an statistical model from simulation data. This section outlines the steps taken to develop the statistical model. These steps are at the core of this communication-centric modeling methodology, and are fundamental to produce system level communication-based performance estimation at higher levels of abstraction.

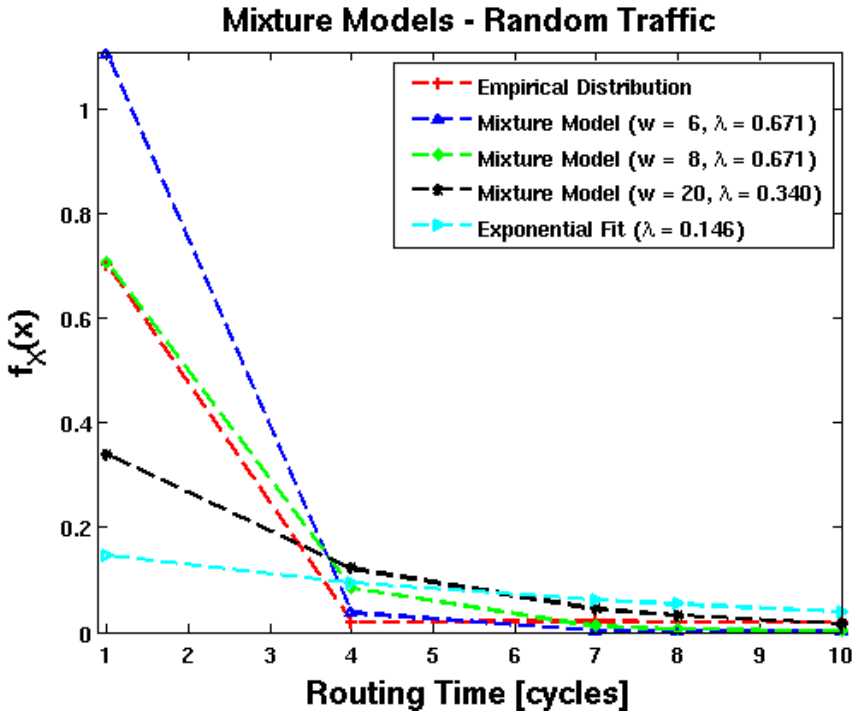


Figure 4.5: Mixture Models for Random Traffic

At the center of the statistical modeling presented in this chapter is the mixture distribution. Given by equation 4.2, shown here again

$$f_X(x) = \sum_{i=0}^{n} a_i f_{Y_i}(x),$$

the mixture distribution is described by its mixture components $\{f_{Y_i}(x)\}$ and its mixture proportions $\{a_i\}$. The modeling methodology are the steps necessary to produce the mixture components and proportions from the simulation data.

Given an empirical distribution the steps are as follow:

1. Partition the empirical distribution into two sets. For a distribution with n frequency points, the first set contains the frequencies from $[1, \ldots, i]$, and the second set from $[i+1, \ldots, n]$. Figure 4.3, on page 35, show two such partitions.

2. Independently fit each set.

   - In the example on Section 4.3, each partition was fitted to an exponential distribution.

3. Determine the mixture proportions $\{a_i\}$ for the given partition.

   - For the ongoing example, the proportions where determine by solving the set of equations:

$$\begin{aligned} \mu_{\text{mixture}} &= a_1 * \mu_{f_{Y_1}} + a_2 * \mu_{f_{Y_2}}, \\ a_1 &= 1 - a_2, \end{aligned}$$

     and setting μ_{mixture} to the average routing delay from simulation.

4. Given the mixture components and proportions, compose the mixture distribution for this partition $\{f_{X_i}(x)\}$ and compute the mean square error between the i^{th} mixture and the empirical distribution.

5. Repeat steps 1 to 4 for each $i = 2 \leq i \leq n$, and compute the mean square error for each resulting mixture model.

6. The best mixture is the one with components and proportions which yields the lower mean square error.

Resulting from this methodology is an statistical model based on the mixture distribution, that is completely described by its components and proportions. This model replaces the communication component, router for the case of the example of Section 4.3, for all subsequent simulations and exploration of the system at the higher abstraction levels. While section showed the derivation of the methodology, the next section shows how the methodology is apply to generate a new model.

4.5 Hotspot Traffic Model

Sections 4.2 and 4.3 evaluated the routing behavior for the system when loaded with random traffic. Random traffic is a good basic case that shows the behavior of the system for a load that is uniformly valanced across all node. However, random traffic may not capture the desire application behavior correctly. Another useful traffic model that may better represent the desire application is hotspot traffic.

For the case of hotspot traffic 20% of the system was chosen as the hotspot. That is, 80% of all the nodes on the system will chose the same 20% area or the system as their destination. The nodes inside the 20% hotspot are chosen at random, and are all equally likely. This hotspot behavior is capable of recreating the type of bottle necks commonly found in audio and video encoding/decoding applications.

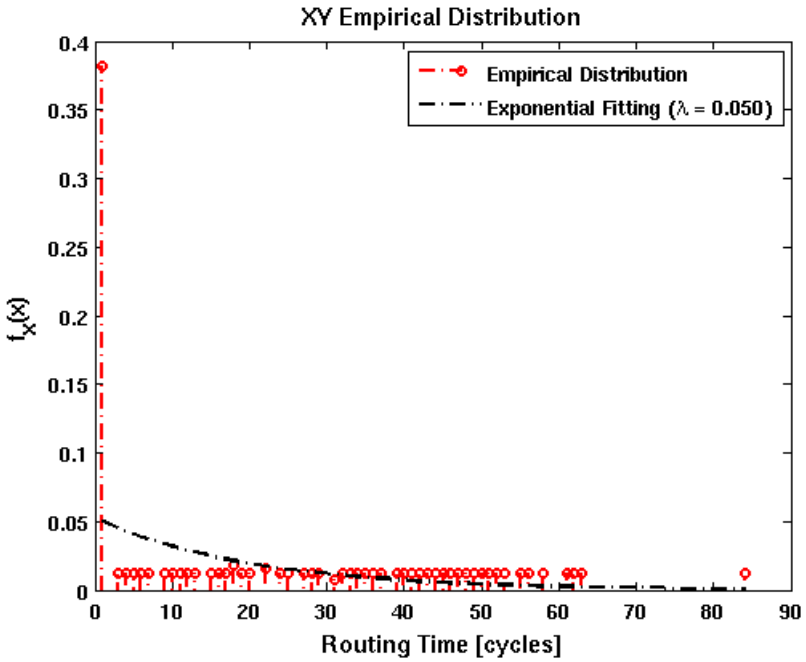


Figure 4.6: Routing Time Distribution and Fitting for Hotspot Traffic

Figure 4.6 shows the empirical distribution of the routing time for hotspot traffic. When using hotspot traffic the shape of the routing time distribution is very similar to the case with random traffic, with some key differences. (1) The tail of the routing distribution is longer for hotspot. (2) Along with a longer tail, the average routing time is also higher than that of the random traffic, about three times. Table 4.2 shows the routing average for random and hotspot traffic for comparison. Perhaps the most noticeable difference from random traffic is that for hotspot traffic the routing time distribution seams to be more spread out across all cycles for hotspot. While the first cycle still dominates the distribution its frequency reduced from 0.7, for the random traffic, to 0.4, shown in Figure 4.6.

Following the methodology outlined in Section 4.4, the mixture model for this traffic is found evaluating the MSE behavior. Figure 4.7 show the MSE curve for the different

Table 4.2: Exponential Fitting, XY Routing

Traffic	$Routing_{ave}$	*Mean*	*Std. Dev.*	*MSE*
Random	7.85	6.85	46.93	0.2193
Hotspot	20.94	19.94	397.47	0.0422

partitions on the empirical distribution of Figure 4.6. The mixture model for hotspot with minimum MSE has the parameters:

$$f_{Y_0}(x) \sim Exp(\mu = 2.5660), a_0 = 0.9360$$
$$f_{Y_1}(x) \sim Exp(\mu = 23.162), a_1 = 1 - a_0.$$

Graphically the mixture models for windows $w_{[10\ 14\ 30]}$ are shown in Figure 4.8. As expected, for hotspot traffic having more information in the tail of the empirical distribution the weight given to $f_{Y_0}(x)$ was reduced from 0.9961, for random traffic, to 0.9360.

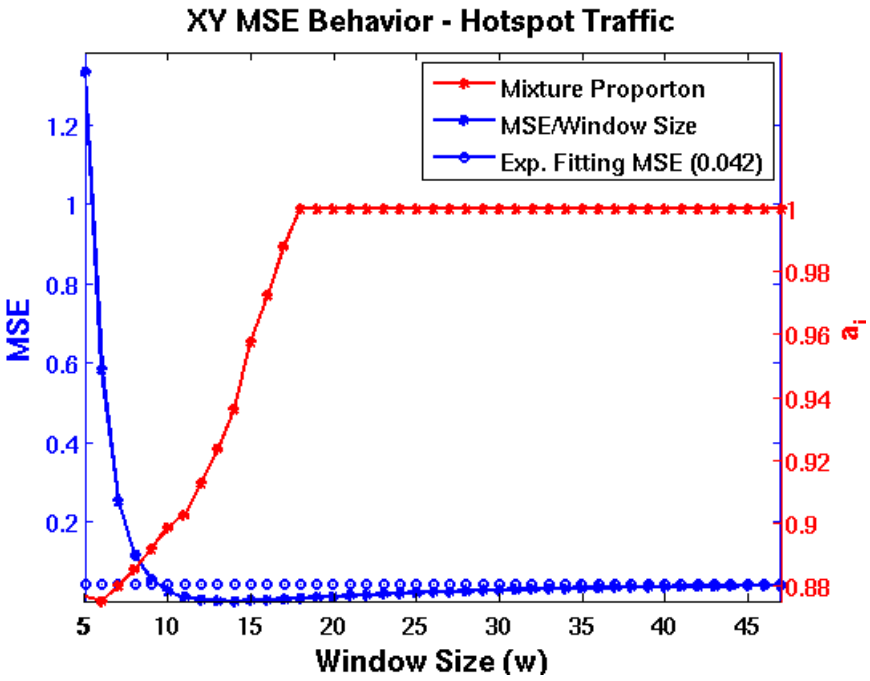


Figure 4.7: MSE Behavior Across Window Sizes (Random)

This chapter formally presents an statistical model which captures the communication features to raise them to higher levels of abstraction. The model is based on a mixture distribution and takes the shape of the empirical distribution extracted form simulation. It was shown how a mixture distribution better models the routing behavior for random and

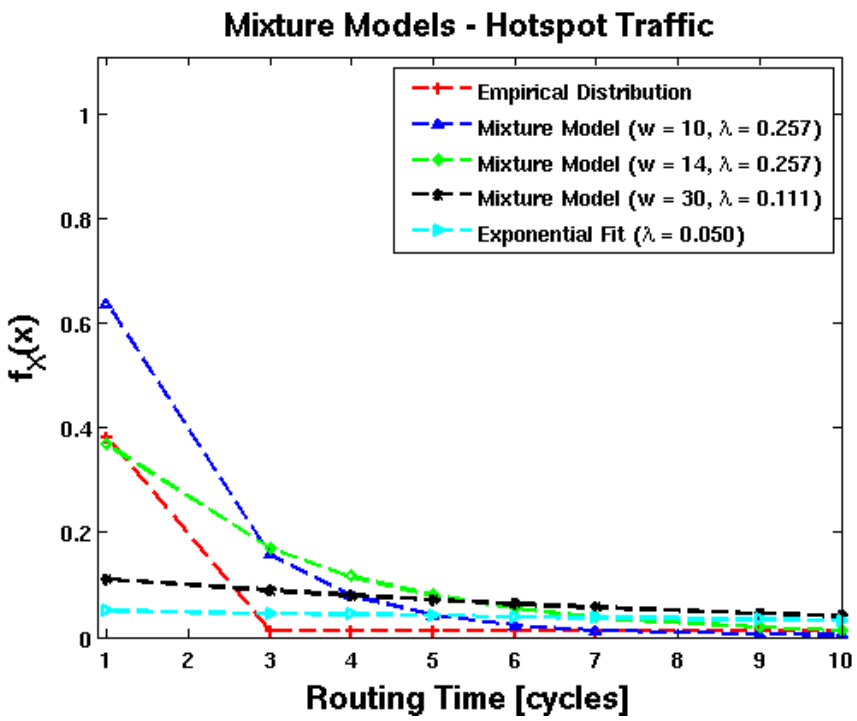


Figure 4.8: Mixture Models for Random Traffic

hotspot traffic, compared to a simple exponential fit. Further, and more importantly, this chapter outlines the steps used to developed these mixture models. System level designers may examine this chapter and follow the steps delineated here to develop similar models for the significant communication features on their designs.

CHAPTER 5. Probability Metric

Chapter 4 demonstrated how to derive a mixture model from simulation. These mixture models capture the communication architecture characteristics as they behave in a loaded system. For an example loaded under random traffic it was shown that a mixture model with two components better represents the routing element, than a simple exponential fit over the empirical distribution. In general, a mixture model with two components is of the form

$$f_{T_R}(x) = a_0 f_{Y_0}(x) + a_1 f_{Y_1}(x). \tag{5.1}$$

Furthermore, through MSE analysis it was found that a good mixture model for a system with XY routing, has mixture components given by

$$\begin{aligned} f_{Y_0}(x) &\sim Exp(\mu = 1.4120), a_0 = 0.9961 \\ f_{Y_1}(x) &\sim Exp(\mu = 21.428), a_1 = 1 - a_0. \end{aligned}$$

The mixture model $f_{T_R}(x)$ becomes the representation of the routing elements on any future high level models that include the same communication architecture. Different mixture models are derived to represent different architecture alternatives. Much useful information may be gathered from the mixture model directly; e.g. the mean routing time. However, the statistical mixture model becomes exceptionally useful when exposed to dynamic loads.

For instance, given the application communication characteristics, it is possible to use the mixture model to evaluate the collision behavior of a system at the highest abstraction level. Knowing that the routing time has a distribution $f_{T_R}(x)$, it is possible to combine the application communication characteristics with the mixture model to compute the probability of collision. The collision distribution represents how the entire system reacts to the dynamic

communication behavior of the application, and directly relates to the system performance. Dynamic analysis, such as the one for collision, are possible with mixture models because of the tools and methods available to statistical models.

5.1 Path Analysis

To find the probability of collision the first step is to look at the path a packet takes from source to destination. Figure 5.1 shows two packets and the path they take from source to destination.

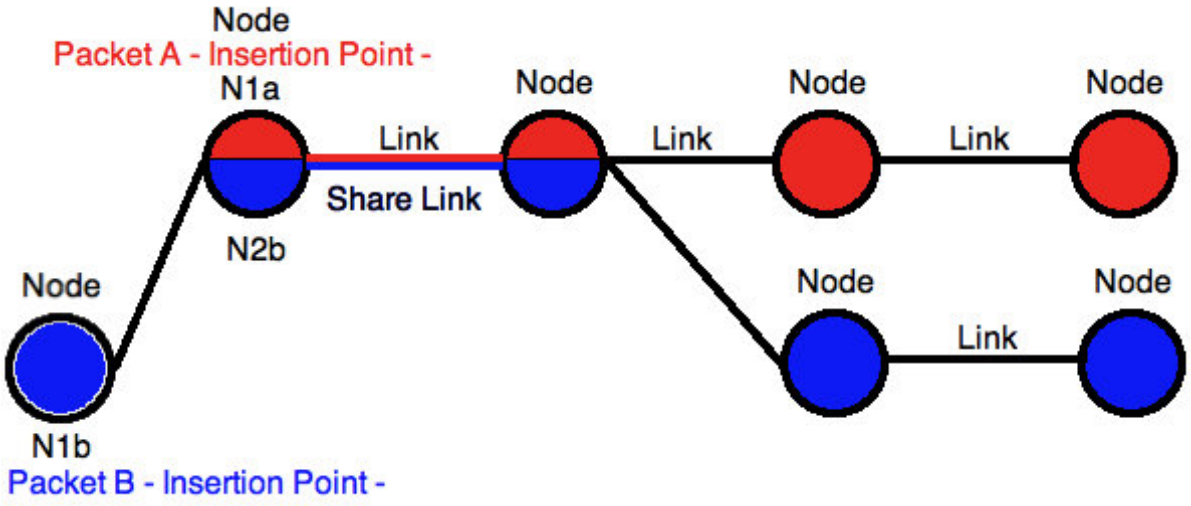


Figure 5.1: Path for Packet A (red) and B (blue)

Packet A, red in Figure 5.1, with its source at node $N1a$ has a path of four nodes. Packet B, blue in Figure 5.1, has five nodes on its path and its source at node $N1b$. As shown in the figure, both paths share two nodes and one link.

The proposed model only considers the shared links, and not the shared nodes. The assumption is that there are enough resources at the node to route the packets in parallel, while access to the communication medium is mutually exclusive. From the point of view of packet A, the shared link is the 1^{st} link, and from packet B's point of view, it is sharing the 2^{nd} link. Hence, the shared link is classified Type 1|2.

Table 5.1 show some sharing types, and their frequency, for an 8x8 XY/Wormhole system under random traffic. These sharing types are derived from a high level simulation using the

mixture model $f_{T_R}(x)$. Using the given classification and the definition of T_R from equation 5.1, it is possible to find the time when packet A and B require the shared link.

Table 5.1: Frequency of sharing types, from simulation.

Type	Frequency
1|2	4390
2|3	3571
2|2	2246
3|3	1879

Packet A starts using the shared link at time T_A, similarly, T_B is the time for packet B. Taking the travel time, T_T, from Figure 4.1 (page 32) as constant, packets A and B use the shared link during the time intervals

$$[T_A, T_A + T_T] \text{ and } [T_B, T_B + T_T]$$

respectively. For a shared link of Type 1|2 the start times are given by

$$T_A = T_{R_{1a}}, \text{ and} \tag{5.2}$$

$$T_B = T_{R_{1b}} + T_T + T_{R_{2b}}. \tag{5.3}$$

Where, $T_{R_{1a}}$, $T_{R_{1b}}$, and $T_{R_{2b}}$ are identically distributed independent random variables with the PDF of the form of equation 5.1.

5.2 Probability Computation

To compute the probability of collision it is necessary to define the events for which there would be collisions. There are three events that can produce a collision between packet A and B. Figure 5.2 shows all three cases where packets A and B may collide. The first event, Figure 5.2a, represents when packet B requests the shared link while it is being used by packet A. The second event, Figure 5.2b, is the complement of the first case. The final event, Figure 5.2a, illustrates when both packets A and B request the link at precisely the same time. From this discussion the probability of collision is defined as:

$$P_c = P(0 < |T_B - T_A| < T_T \cup T_A = T_B).$$

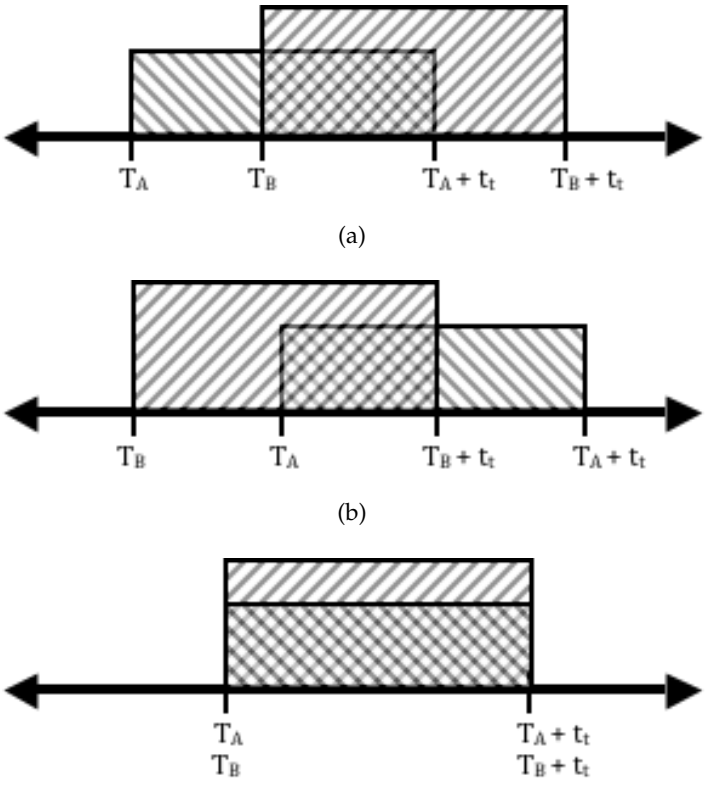


Figure 5.2: Collision Events

Since T_A and T_B are defined as continues random varaible

$$P(T_A = T_B) = 0.$$

Therefore, in general the probability of collision is given by

$$P_c = P(0 < |T_B - T_A| < T_T). \tag{5.4}$$

Finally, combining equations 5.2, 5.3, and 5.4, the probability of collision for sharing Type 1|2 of in Figure 5.1 is

$$\begin{aligned} P_c^{1|2} &= P(-T_T < (T_{R_{1b}} + T_{R_{2b}} + T_T) - T_{R_{1a}} < T_T) \\ &= P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T). \end{aligned} \tag{5.5}$$

Similarly, Table 5.2 shows the basic probability equations for the sharing types of Table 5.1.

Table 5.2: Frequency of sharing types, from simulation.

Type	P_c
1|2	$P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T)$
2|3	$P(0 < (T_{R_{1a}} + T_{R_{2a}}) - (T_{R_{1b}} + T_{R_{2b}} + T_{R_{3b}}) < 2T_T)$
2|2	$2 * P(0 < (T_{R_{1a}} + T_{R_{2a}}) - (T_{R_{1b}} + T_{R_{2b}}) < T_T)$
3|3	$2 * P(0 < (T_{R_{1a}} + T_{R_{2a}} + T_{R_{3a}}) - (T_{R_{1b}} + T_{R_{2b}} + T_{R_{3b}}) < T_T)$

5.2.1 Probability of the Difference

The probability of equation 5.5 is the difference of a combination of random variables with mixture distributions. This probability is easy to find by defining $Z = T_{R_{1a}} - Y$, and $Y = (T_{R_{1b}} + T_{R_{2b}})$. The first step is to find the PDF of Y. Given the distributions of $T_{R_{1b}}$ and $T_{R_{2b}}$,

$$\begin{aligned} f_{T_{R_{1b}}}(u) &= \frac{a_0}{\mu_0} e^{\frac{-u}{\mu_0}} + \frac{a_1}{\mu_1} e^{\frac{-u}{\mu_1}} \\ f_{T_{R_{2b}}}(v) &= \frac{a_0}{\mu_0} e^{\frac{-v}{\mu_0}} + \frac{a_1}{\mu_1} e^{\frac{-v}{\mu_1}} \end{aligned}$$

the new distribution of Y is given by the convolution

$$\begin{aligned} f_Y(y) &= \int_0^y f_{T_{R_{1b}}}(u) * f_{T_{R_{2b}}}(y-u)du \\ &= \frac{2a_0a_1}{\mu_0 - \mu_1}\left(e^{\frac{-y}{\mu_0}} - e^{\frac{-y}{\mu_1}}\right) + \left(\frac{a_0}{\mu_0}\right)^2 e^{\frac{-y}{\mu_0}} y + \left(\frac{a_1}{\mu_1}\right)^2 e^{\frac{-y}{\mu_1}} y. \end{aligned}$$

To test that $f_Y(y)$ is a propre PDF, note that

$$\int_0^\infty f_Y(y)dy = 1$$

Finally, the distribution of Z is found through a common transformation. Defining

$$\begin{aligned} W &= T_{R_{1a}} + Y \\ Z &= T_{R_{1a}} - Y, \end{aligned}$$

then the joint distribution $f_{(W,Z)}(w,z)$ is

$$\begin{aligned} f_{(W,Z)}(w,z) &= f_{T_{R_{1a}}}\left(\frac{w+z}{2}\right) f_Y\left(\frac{w-z}{2}\right) * abs\left(\begin{vmatrix} \frac{\partial T_{R_{1a}}+Y}{\partial T_{R_{1a}}} & \frac{\partial T_{R_{1a}}+Y}{\partial Y} \\ \frac{\partial T_{R_{1a}}-Y}{\partial T_{R_{1a}}} & \frac{\partial T_{R_{1a}}-Y}{\partial Y} \end{vmatrix}\right) \\ &= f_{T_{R_{1a}}}\left(\frac{w+z}{2}\right) f_Y\left(\frac{w-z}{2}\right) * \frac{1}{2} \end{aligned}$$

and the PDF of Z is found by rationalizing the joint distribution, $f_{(W,Z)}(w,z)$ over all the values of W.

$$f_Z(z) = \int_{|z|}^{\infty} f_{(W,Z)}(w,z)dw \tag{5.6}$$

$$f_Z(z) = \begin{cases} \frac{1}{4\mu_0^2\mu_1^2(\mu_0+\mu_1)^2(\mu_0-\mu_1)} \\ \Big[-a_0^3 e^{\frac{b}{\mu_0}}(2b-\mu_0)(\mu_0-\mu_1)\mu_1^2(\mu_0+\mu_1)^2 + \\ a_1^3 e^{\frac{b}{\mu_1}}\mu_0^2(2b-\mu_1)(-\mu_0+\mu_1)(\mu_0+\mu_1)^2 + \\ 4a_0a_1^2\mu_0^2\mu_1 * \\ \Big(be^{\frac{b}{\mu_1}}(-\mu_0^2+\mu_1^2) + \mu_1(2e^{\frac{b}{\mu_0}}\mu_0(\mu_0+\mu_1) - e^{\frac{b}{\mu_1}}\mu_1(3\mu_0+\mu_1))\Big) + \\ 4a_0^2a_1\mu_0\mu_1^2 * \\ \Big(-be^{\frac{b}{\mu_0}}(\mu_0^2-\mu_1^2) + \mu_0(-2e^{\frac{b}{\mu_1}}\mu_1(\mu_0+\mu_1) + e^{\frac{b}{\mu_0}}\mu_0(\mu_0+3\mu_1))\Big)\Big] & , z < 0 \\ \\ \frac{1}{4\mu_0\mu_1(\mu_0+\mu_1)^2}e^{-b(\frac{1}{\mu_0}+\frac{1}{\mu_1})} \\ \Big[a_1^3 e^{\frac{b}{\mu_0}}\mu_0(\mu_0+\mu_1)^2 + a_0^3 e^{\frac{b}{\mu_1}}\mu_1(\mu_0+\mu_1)^2 + \\ 4a_0a_1^2\mu_0\mu_1\left(e^{\frac{b}{\mu_1}}\mu_0 + e^{\frac{b}{\mu_0}}(\mu_0+\mu_1)\right) + \\ 4a_0^2a_1\mu_0\mu_1\left(e^{\frac{b}{\mu_0}}\mu_1 + e^{\frac{b}{\mu_1}}(\mu_0+\mu_1)\right)\Big] & , z \geq 0 \end{cases}$$

Lastly, the probability of collision for sharing Type 1|2 is fund using equation 5.6.

$$\begin{aligned} P_c^{1|2} &= P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T) \\ &= P(0 < Z < 2T_T) \\ &= \int_0^{2T_T} f_Z(z)\,dz \\ &= \int_0^{2T_T} \frac{1}{4\mu_0\mu_1(\mu_0+\mu_1)^2}e^{-b(\frac{1}{\mu_0}+\frac{1}{\mu_1})}\Big[a_1^3 e^{\frac{b}{\mu_0}}\mu_0(\mu_0+\mu_1)^2 + a_0^3 e^{\frac{b}{\mu_1}}\mu_1(\mu_0+\mu_1)^2 \\ &\quad + 4a_0a_1^2\mu_0\mu_1\left(e^{\frac{b}{\mu_1}}\mu_0 + e^{\frac{b}{\mu_0}}(\mu_0+\mu_1)\right) + 4a_0^2a_1\mu_0\mu_1\left(e^{\frac{b}{\mu_0}}\mu_1 + e^{\frac{b}{\mu_1}}(\mu_0+\mu_1)\right)\Big] \end{aligned} \tag{5.7}$$

Figure 5.3, on the next page, presents a graphical representation of the PDF of $f_Z(z)$ as derived in equation 5.6.

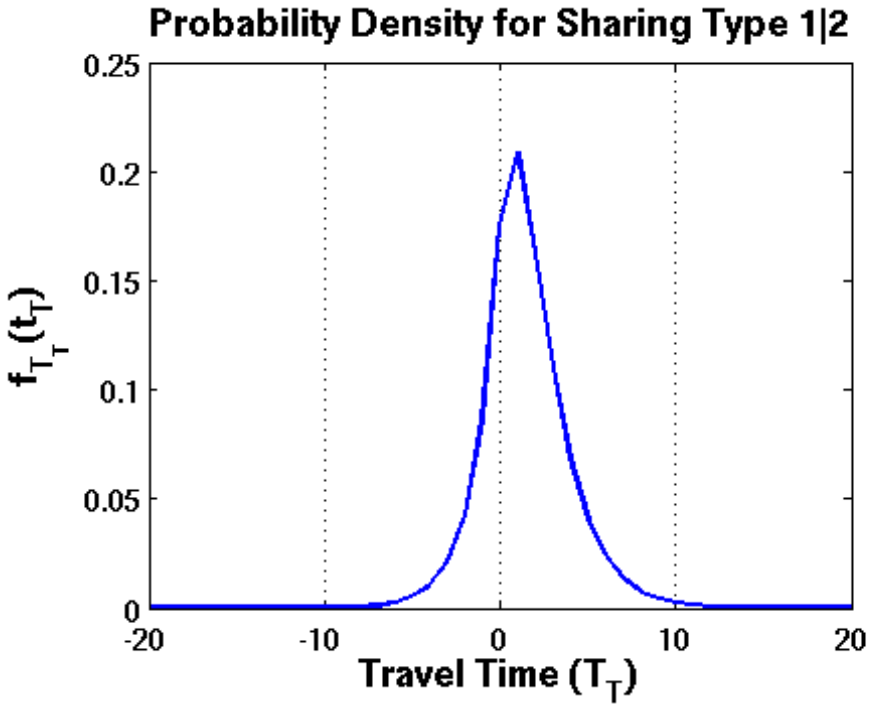


Figure 5.3: Probability Density Function for Sharing Type 1|2

5.3 Travel Time Analysis

Travel time, T_T, as defined in Section 5.1 refers to the time (in cycles) that a link is busy transmitting a packet. This time is a function of several parameters including, packet size, bandwidth, flow control, etc. This sections shows the relation between the probability of collision, bandwidth, and packet size.

5.3.1 Bandwidth

Bandwidth measures the capacity of the communication medium to move data, and it is a function of the width and speed of the medium. Width refers to the number of bits that may be transmitted in parallel. Speed is the maximum frequency at wich this bits may be switched. Table 5.3 lists the most common widths found in embedded systems, and shows the travel time, in cycles, that it takes to transmit a packet of 64B splited in 10 flits.

Table 5.3: Travel delay as a function of Bandwidth

Bandwidth (BITS/CYCLE)	T_T (CYCLES)
128	5
64	10
32	40
16	80
8	160

5.3.2 Packet Size

Packet size is traditionally measured in bytes, as a function of the amount of data carried in a packet plus any additional information required by the communication protocol. Because this research focuses on the impact of the communication protocols on system performance, packet size may also be measured as the number of flits required by the communication protocol to transmit a packet of a certain size in bytes. The flit size is fixed and defined by the communication protocol, in this research the flit is fixed at 10 bytes. Of these 10 bytes, 8 are allocated for the payload and 2 for the flit header. Therefore, a packet of 64 bytes is splited into 8 flits each of 10 bytes. Table 5.4 shows the travel time for packets of different size, for a bandwidth of 32 bits/cycle.

Table 5.4: Travel delay as a function of Packet Size

Packet Size (BYTES)	T_T (CYCLES)
16	5
32	10
64	20
128	40
256	80
512	160

5.3.3 Probability of Collision as a Function of T_T

Equation 5.7, on page 48, shows the probability of collision as a function of travel time, T_T, for sharing Type 1|2. As previously discussed, travel time is a function of both bandwidth and packet size. Therefore the probability of collision is also influenced by bandwidth and

packet size. Furthermore, each sharing type produces produces a separate probability of collision, Table 5.2. The probability of collision for the entire system is a linear combination of the probabilities of each sharing type, and is given by

$$P_c^s = \frac{\sum_i^n freq_i * P_c^i}{\sum_i^n freq_i}. \tag{5.8}$$

Where P_c^i represents the probability of the i^{th} sharing type, $freq_i$ is the associated frequency, and P_c^s is the total system probability of collision.

Table 5.5 summarizes the system probability of collision (P_c^s) for the sample system with XY routing and bandwidth of 32 bits/cycle. On this table the column labeled $P_c^s(T_T)$ shows the probability results from the model of equation 5.8, and the column labeled *Simulation* contains the actual package drop percents from simulation. Figure 5.4 is a graphical representation of the data in Table 5.5. As expected the system probability ($P_c^s(T_T)$) is a growing function of packet size. These probabilities are derived from the mixture model with the minimum MSE, and parameters $\mu_0 = 1.4120$, $\mu_1 = 21.428$, $a_0 = 0.9961$. It is necessary to address the accuracy of the system probability model, as depicted in Figure 5.4.

Table 5.5: System Probability of Collision: $P_c^s(T_T)$

Packet Size (BYTES)	$P_c^s(T_T)$	Simulation	T_T (CYCLES)
16	0.42764	0.49531	5
32	0.46284	0.54971	10
64	0.46698	0.55941	20
128	0.46874	0.56571	40
256	0.46960	0.58561	80
512	0.46965	0.61781	160

5.3.3.1 Accuracy Vs Fidelity

As can be seen in Figure 5.4 the probability model is of low accuracy. However, fidelity is the necessary attribute that makes the probability model useful for system-level design. Fidelity measures how well the model predict the behavior of the system. A good measurement of fidelity is the correlation factor between the model and the simulation. A correlation factor of 1 indicates the highes fidelity, while a factor of 0 suggest no fidelity. The model

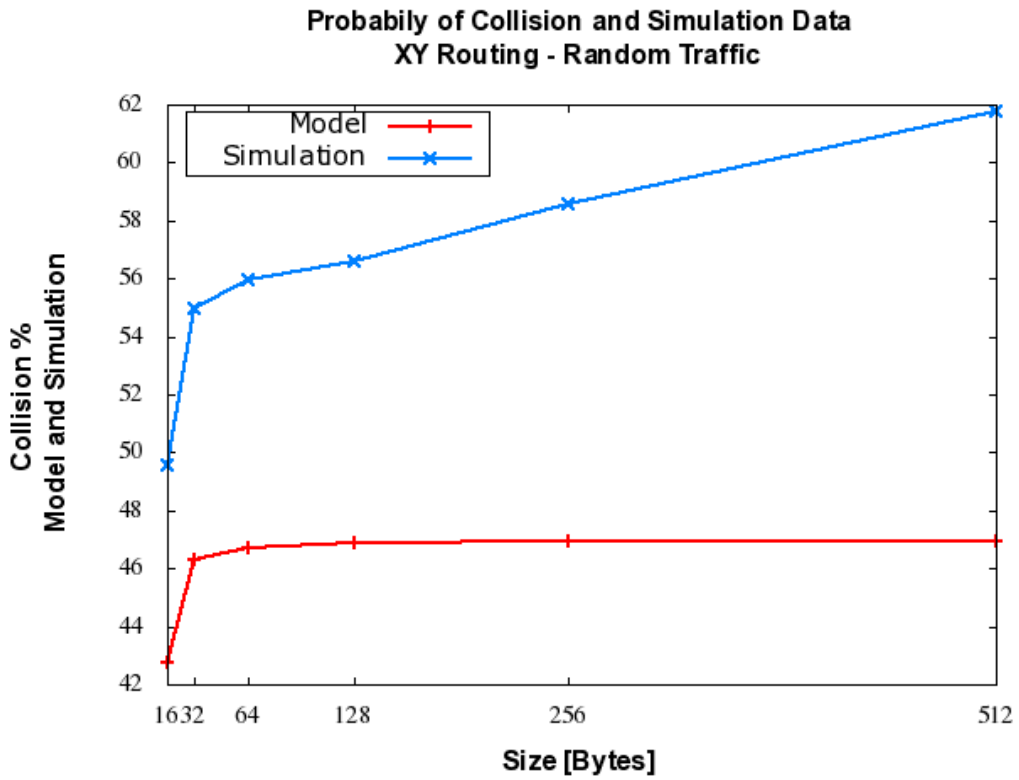


Figure 5.4: Probability of Collision for XY Routing - Random Traffic, 32[bits/cycle]

shown in Figure 5.4 has a correlation factor of 86, that shows that this model has very good fidelity to the actual simulation data.

Finally, the system probability model shown in Figure 5.4 depicts the initial relation between the system performance and the probability. Increasing the packet size results in system degradation, in the form of increased travel time and it is observed as an increased in packets dropped. At the same time, as the system performance degrades, the probability of collision increases. This is the expected behavior and the basis for proposed probabilistic approach. The next chapter shows how the system probability of collision may be used in the system-level framework to guided the design process.

CHAPTER 6. Case Study: System-Level Design with Mixture Models

Chapter 4 shows how to develop a mixture model for a given architecture, and Chapter 5 presents the system probability of collision as a performance metric. This chapter shows how the mixture model and probability of collision are used in a system-level framework. Particularly, this chapter presents mixture models for different architectures: adaptive and XY routing; and shows how the system probability of collision may be used as a performance estimator for comparing these architectures at higher levels of abstraction and making the design decisions.

6.1 System-Level Design

The purpose of developing the mixture models is to use these models to guide the design process at higher abstraction levels. For instance, given the application communication characteristics, it is possible to use the mixture model to evaluate the collision behavior of a system at higher abstraction levels. Moreover, knowing that the routing time has a distribution $f_{T_R}(x)$, it is possible to combine the application communication characteristics with the mixture model to compute the probability of collision.

The collision distribution represents how the entire system reacts to the dynamic communication behavior of the application, and directly relates to the system performance. Therefore, the probability of collision may be seen as a high level performance estimator. Dynamic analysis, such as the one for collision, are possible with mixture models because of the tools and methods available to statistical models, such as the mixture models.

The system probability of collision (P_c^s) was introduced in Section 5.3, and it is computed as a linear combination of the collision probabilities of each sharing type found on the system

level simulation. Defined as equation 5.8 it was shown to have a high correlation to the collision behavior found during the low level simulation.

6.1.1 Average Flit Delay

The most widely used performance metric on interconnection networks is delay. Several different delay metrics are available at the lower levels. This discussion focuses on the average flit delay. The average flit delay is defined as the average number of cycles a flit spends on a routing node.

Figure 6.1 shows a network message and how it is divided into packets, flits, and phits. A packet is the basic unit of routing. The packet header is used to determine the route taken by the packet from source to destination. Flow control digits, or flits, are the basic unit of bandwidth and storage allocation. Flits carry no routing information. Thus, all flits in a packet must follow the same path from source to destination.

Depending on its position on the packet, a flit may be a head flit, body flit, or tail flit. A head flit is the first flit of a packet and follows the route determined by the packet. All bandwidth and storage allocation for the packet is performed by the head flit. The head flit is followed by zero or more body flits and one tail flit. The tail flit is the last flit of the packet and is most commonly used for resource deallocation. Finally, a flit may be divided into physical transfer digits, or phits. A phit is the unit of information that is transferred across a channel in a single clock cycle.

The average flit delay measures the average number of cycles a flit spends on a routing node. This delay directly reflects the amount of time a packet is held at a node due to the node resource limits. That is, the average flit delay is a measurement of the flits that are queued waiting inside routing nodes to be serviced and forwarded along the packet's routing path.

6.1.2 Probability Metric

The new high-level estimator introduced in this research is the system probability of collision, P_c^s. The system probability of collision is a dynamic estimator, that extracts the

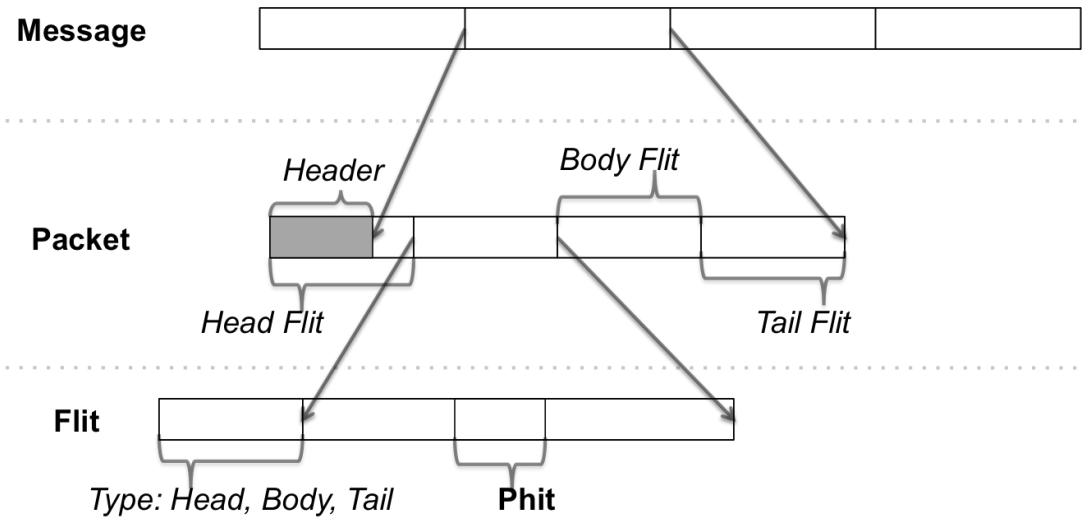


Figure 6.1: Anatomy of a Network Message

application communication behavior as the distribution of the shared links. This system probability estimator is defined as

$$P_c^s = \frac{\sum_i^n freq_i * P_c^i}{\sum_i^n freq_i}.$$

P_c^i, on the equation above, is the probability of collision for the i^{th} sharing type and it is a function of the mixture model, the relative location of the shared link on the path, and the travel time. For example, the probability of collision for sharing Types 1|2 and 2|2 are given by equations 6.1 and 6.2, respectively.

$$P_c^{1|2} = P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T), \text{ and} \quad (6.1)$$

$$P_c^{2|2} = 2 * P(0 < (T_{R_{1a}} + T_{R_{2a}}) - (T_{R_{1b}} + T_{R_{2b}}) < T_T). \quad (6.2)$$

For detail derivation of the probability of collision for the different sharing types and the system probability of collision see sections 5.2 and 5.3.3.

The goal of system-level design is to provide the designer with the tools to navigate the design space. The following sections will show how P_c^s is used in system-level design to rapidly and accurately navigate the design space. The rest of this chapter will particularly show the entire process of:

- defining the mixture models for XY and adaptive routing architectures,

- using these models to simulate the system at higher levels of abstraction to find the shared link distribution,
- computing sharing type probabilities P_c^i, and
- finding P_c^s.

Finally, to show the validity of P_c^s, the conclusions reached through P_c^s are shown to correlate to the theoretically expected behavior.

6.2 Design Space Exploration

Design Space Exploration refers to the process of investigating the various design options and their implementation cost. The goal of system-level design is to guide designers to explore the design space to find a design solution for a given set of parameters and constrains. This solution may not be optimal, since in the case of multiple design objectives like minimum communication delay or processing delay, minimal area, and lower power consumption; finding the optimal solution within realist time constraints is impossible.

System-level design defines several levels of abstraction. Using the tools and models available at the higher abstraction levels, it is possible to evaluate a larger area of the design space at a lower simulation/design cost. The first step to develop these abstracted models is to characterize the architecture features of interest.

6.2.1 Adaptive Routing

Adaptive routing is an important subset of the routing schemes available to interconnect networks. For this case study adaptive routing is implemented in a system with 64 nodes arranged in an 8x8 torus, and wormhole flow control. The particular adaptive scheme studied here makes the routing decision based on the available buffer space. That is the next node on the packet path, which is the one with most buffer memory available to receive the incoming packet.

To begin evaluating the adaptive routing the first step is to develop the mixture model for an adaptive routed system. To this end, adaptive routing is simulated over hotspot

traffic. Holding the buffer size at a nominal 16 filts, the packet size is varied from 16 to 512 flits. For each packet size a mixture model may be derived to represent the communication architecture behavior for the particular architecture options. Table 6.1 show the mixture models that captures the communication behavior of an adaptive routing architecture.

Table 6.1: Mixture Models for Adaptive Architecture

Buffer	Packet	Mixture Parameters		
		μ_0	μ_1	a_0
16	16	1.312	7.220	1.000
	32	1.520	93.604	0.949
	64	1.519	48.941	0.925
	128	1.705	108.064	0.912
	256	1.860	133.767	0.891
	512	2.132	174.422	0.843

While a single mixture model may be used to represent the different packet sizes, as seen on section 4.4, a set of models as shown in Table 6.1 more accurately captures the behavior of the particular architecture. High level models will alway benefit from the most detailed characterization possible. Figure 6.2 depicts the empirical distribution, exponential fit, and mixture model for a packet size of 64 flits. For the details outlining how each mixture model is developed see Section 4.3.

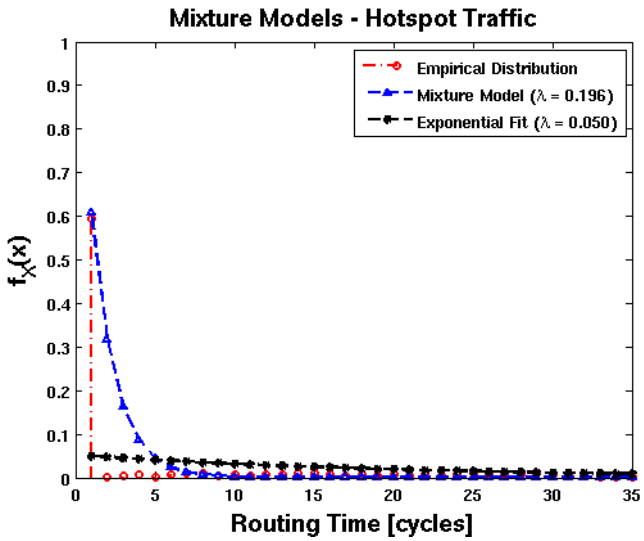


Figure 6.2: Mixture Models: Adaptive Routing, Hotspot Traffic, 64 flits

6.2.2 XY Routing

Having developed the mixture models for an adaptive routing architecture, the next research step is to evaluate is XY routing. Deterministic XY routing is a commonly used routing scheme. The advantages of deterministic routing are many, such as simple design and analysis, ease of implementation, low latency for well behaved loads, etc. Just like adaptive routing, the XY routing architecture was implemented as an 8x8 torus with 64 nodes, and wormhole flow control.

As with adaptive routing the XY routed system is simulated over hotspot traffic; and characterized with buffer size of 16 filts, and packet sizes varying from 16 to 512 flits. The set of mixtures models that capture the XY routing communication behavior are shown in Table 6.2. Figure 6.3 depicts the empirical distribution, exponential fit, and mixture model for a

Table 6.2: Mixture Models for XY Architecture

Buffer	Packet	Mixture Parameters		
		μ_0	μ_1	a_0
16	16	1.500	7.298	1.000
	32	1.764	56.812	0.935
	64	1.717	77.527	0.910
	128	1.849	152.999	0.895
	256	1.929	219.081	0.877
	512	2.545	263.668	0.854

packet size of 64 flits of the XY routed system.

Having developed mixture models capturing the communication architecture behavior for adaptive and XY routing, the next step is to use these models in a higher level simulation.

6.2.3 High Level Simulation

The mixture model $f_{T_R}(x)$ becomes the representation of the routing elements on any future high level models that include the same communication architecture. Different mixture models are derived to represent different architecture alternatives. Much useful information may be gathered from the mixture model directly; e.g. the mean routing time. However, the

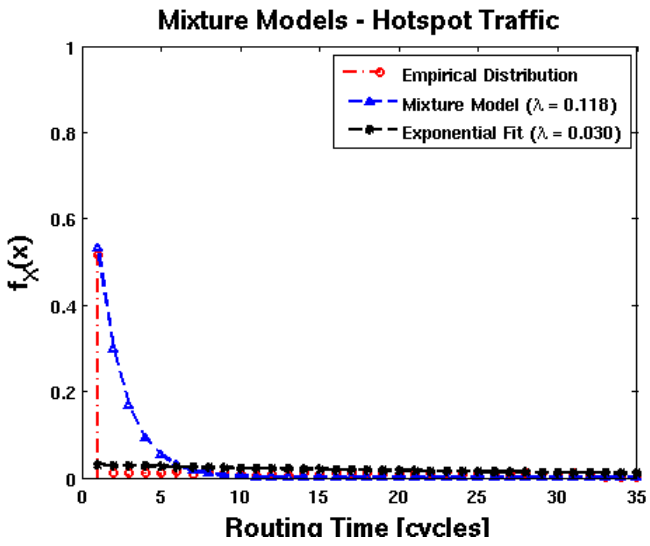


Figure 6.3: Mixture Models: XY Routing, Hotspot Traffic, 64 flits

statistical mixture model becomes exceptionally useful when exposed to dynamic loads.

For instance, given the application communication characteristics, it is possible to use the mixture model to evaluate the collision behavior of a system at the highest abstraction level. Given the routing distribution as the mixture distribution of $f_{T_R}(x)$, the application communication characteristics may be combined with $f_{T_R}(x)$ to compute the probability of collision. This probability of collision captures how the system behaves when interacting to the dynamic communication behavior of the application, and have a direct relation to the system performance. This kind of dynamic analysis, which combines the mixture model with the application communication behavior, is possible because of the tools and methods available to statistical models like the mixture models.

6.2.3.1 Simulation Path Analysis

The mixture models capture the communication behavior of the communication architecture. To produce a high level estimate it is necessary to combine the communication architecture characteristics, with the application communication pattern. In this case the application communication behavior is extracted as a particular traffic pattern model, hotspot traffic.

In this hotspot traffic model 20% of the system is designated as the hotspot. That is, 80% of

all the nodes on the system will choose the same 20% area or the system as their destination. The nodes inside the 20% hotspot are chosen at random, and are all equally likely, and all traffic is exponentially injected into the network. This hotspot behavior is capable of recreating the type of bottlenecks commonly found in audio and video encoding/decoding applications.

Simulating the mixture models at the higher abstraction levels, under a hotspot traffic, results in a set of link utilization figures. As disclosed in Section 5.1 the shared links are classified according to their position within the path. That is, given two packets A and B there is one link shared between the paths of two packets; a sharing Type 1|2 implies that the shared link is the first link on packet's A path and the second link on packet's B.

Table 6.3 shows the most frequent sharing types for both XY and adaptive mixture modes. Notice that, while for both XY and adaptive the most frequent types are 1|2 and 3|4, Type 1|2 is the most frequent for XY and Type 3|4 is for adaptive. This concurs with the behavior of the two routing schemes. Since XY is deterministic it will continue to favor the same paths regardless of the load on the network, as it is evident by the frequency magnitude of sharing types 1|2 and 3|4. On the other hand, the distribution of sharing types for adaptive is much more subtle. More over sharing Type 3|4 is the most frequent, showing that adaptive favors sharing links closer to the destination node, where there are less path options. The distribution of Table 6.3 captures the application communication characteristics, and shows the communication behavior when simulated at a higher abstraction level.

Table 6.3: Shared Link Frequencies, XY and Adaptive routing with 64 flit packet size.

Type	Routing	
	XY Routing	Adaptive
1|2	12341	9093
3|4	11368	9495
3|5	9591	8042
1|3	9494	8123
2|4	9062	8515
3|3	6631	5371

6.2.3.2 Performance Estimation

Having gathered the application communication characteristics, for a hotspot traffic on the different routing architectures, it is possible now to estimate the system performance. Particularly, the system performance is estimated through the system probability of collision. The system probability of collision, P_c^s, combines the dynamic application characteristics, from the share link distribution, with the communication architecture characteristics, and produces a performance estimate.

The system probability of collision is a linear combination of the probabilities of each sharing type, and is given by

$$P_c^s = \frac{\sum_i^n freq_i * P_c^i}{\sum_i^n freq_i}. \tag{6.3}$$

Where P_c^i represents the probability of the i^{th} sharing type, and $freq_i$ is the associated share link type frequency. For reference the probabilities of collision for sharing Types 1|2 and 2|2 are given below.

$$\begin{aligned} P_c^{1|2} &= P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T), \text{ and} \\ P_c^{3|4} &= P(0 < (T_{R_{1a}} + T_{R_{2a}} + T_{R_{3a}}) - (T_{R_{1b}} + T_{R_{2a}} + T_{R_{3a}} + T_{R_{4a}}) < 2T_T). \end{aligned}$$

Using equation 6.3 and the sharing frequencies from Table 6.3 a system probability is computed for each packet size. Figure 6.4 shows the system probability of collision for XY and adaptive routing using the different packet sizes. Figure 6.4 clearly shows that adaptive routing is a better choice for the particular system under hotspot traffic. However, more information may be gathered from P_c^s.

Figure 6.5 shows the normalized P_c^s Difference with respect to adaptive routing. The normalized difference shows the improvement, as a percentage, that adaptive routing is estimated to achieve over XY routing. Figure 6.5 clearly shows that the for smaller packets adaptive routing is a the better choice, with a nomalized difference between 2 and 3%. On the other hand, for larger packets adaptive is only marginally better with a difference of less than 1%. Thus figures 6.4 and 6.5 collectively show that adaptive routing outperforms XY

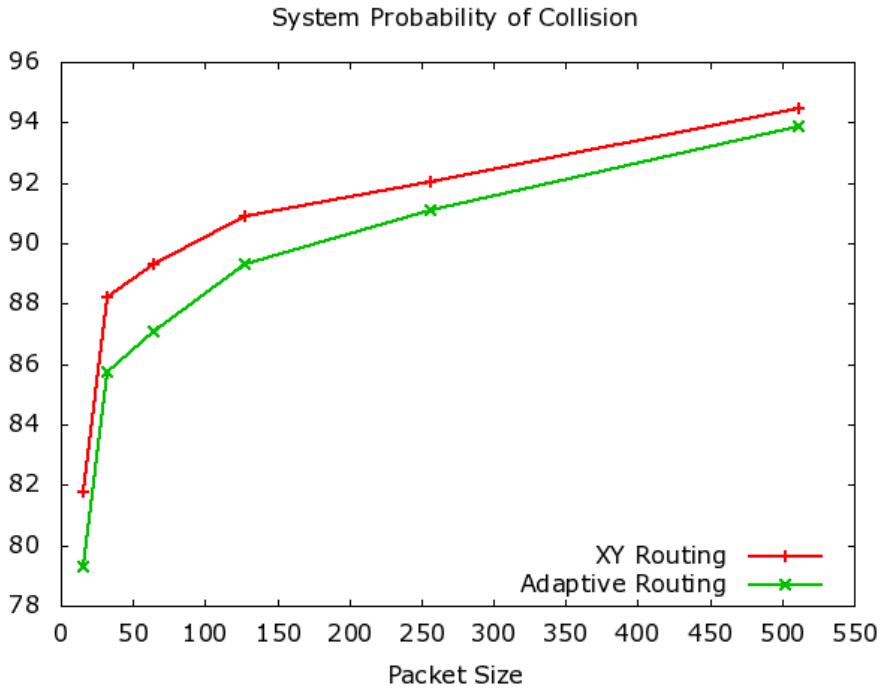


Figure 6.4: P_c^s for XY and Adaptive routing.

routing; and, for smaller packets in particular, the implementation cost of adaptive routing is well justified.

6.3 Transpose Traffic

The design space exploration of section 6.2 focused on hotspot traffic. However, hotspot is not the only standard synthetic traffic available, and other applications may be better modeled by other traffic patterns. One such traffic pattern is the transpose traffic.

In transpose traffic each node sends messages only to a destination with the upper and lower halves of its own address transpose. Just like in the case of hotspot, the transpose traffic is exponentially injected into the network. Both XY and adaptive routing are evaluated with this traffic.

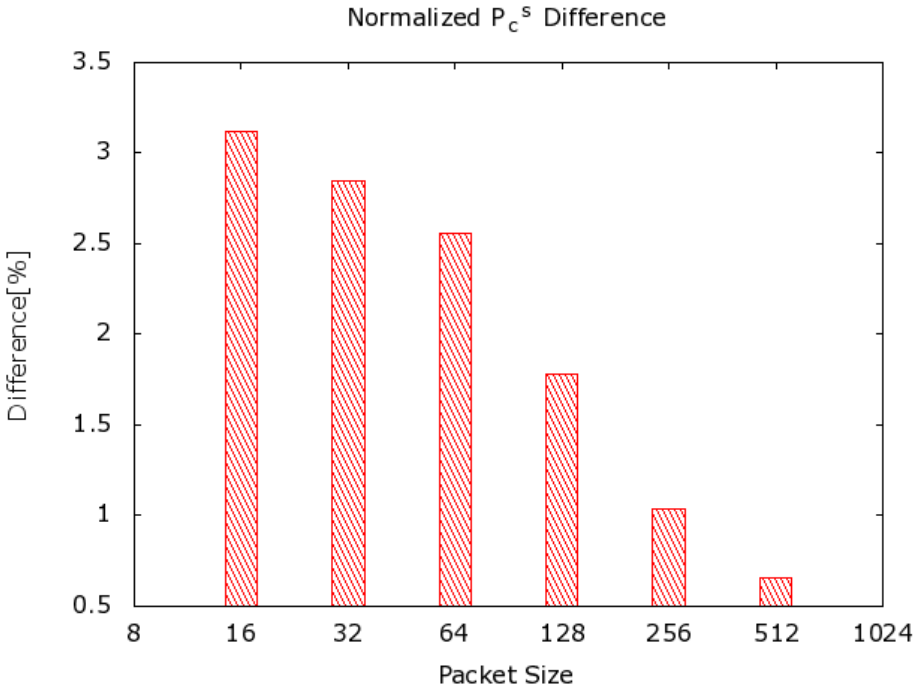


Figure 6.5: Normalized P_c^s Difference with respect to Adaptive routing.

6.3.1 Mixture Models for Transpose Traffic

It is known that the routing behavior is somewhat application dependent[48]. Therefore, different mixture models are used when evaluating different traffics. Following the methodology set forth in Chapter 4 statistical models for XY and adaptive routing under transpose traffic are developed. Table 6.4 shows the mixture parameters for these models.

Table 6.4: Mixture Models for Transpose Traffic

Buffer	Packet	Mixture Parameters					
		XY Routing			Adaptive Routing		
		μ_0	μ_1	a_0	μ_0	μ_1	a_0
16	16	1.298	8.936	0.784	1.118	3.595	1.000
	32	1.420	75.192	0.776	1.270	15.924	1.000
	64	1.468	89.328	0.663	1.273	31.640	0.957
	128	1.314	126.678	0.519	1.352	46.780	0.909
	256	1.120	305.605	0.472	1.510	66.812	0.867
	512	0.984	302.605	0.336	1.603	106.055	0.794

Using the models of Table 6.4 system designers can run high level simulations for appli-

cations with traffic patterns similar to the transpose pattern. The result from the high level simulations is the shared link distribution, shown in Table 6.5. Finally the system probability of collision, P_c^s, combines the mixture models on Table 6.4 and the shared link frequencies from Table 6.5 and generates the system performance estimate.

Table 6.5: Shared Link Frequencies, XY and Adaptive routing with 64 flit packet size and Transpose Traffic.

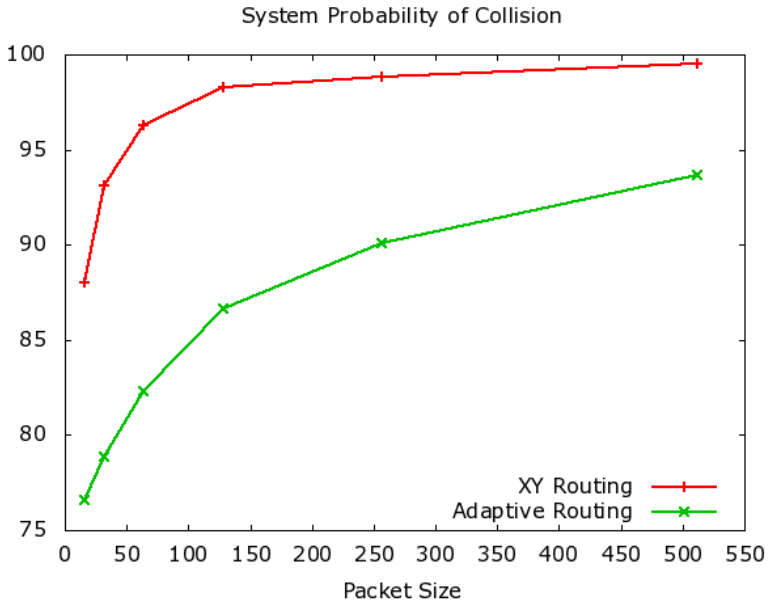
Type	Routing	
	XY Routing	Adaptive
1|2	16997	9837
3|4	12223	9196
3|5	8164	7518
1|3	12078	9427
2|4	12078	8408
3|3	8689	5359

Figure 6.6a shows the system probability of collision for XY and adaptive routing when loaded under transpose traffic. In this case, as it was for Hotspot, adaptive routing is shown to be the better design choice; with a lower system probability of collision for all packet sizes. However, the gap between the two routing schemes is wider for this traffic pattern. Figure 6.6b takes a closer look at the difference between the two routing schemes and shows the normalized difference between the two. Section 6.4 further compares the differences between the estimates for hotspot and transpose traffic.

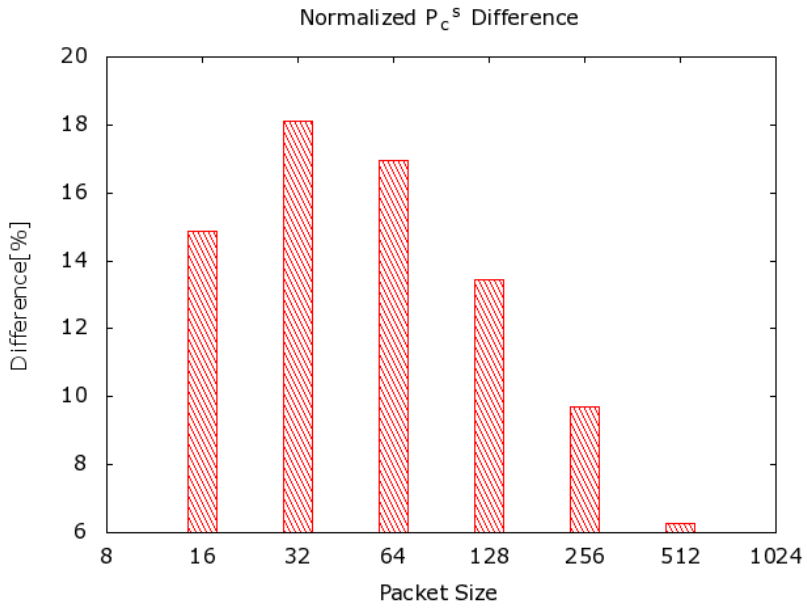
6.4 System Probability Estimate Evaluation

From the system provability of collision – P_c^s on figures 6.4, 6.5, and 6.6 – a system level designer would conclude that adaptive routing is the best choice for the particular application and given the set of design constraints. Thus the design space exploration continues moving to lower abstraction levels by adding more details to the models. However, the question for this research is whether P_c^s leads the designer to make the correct design decision.

To evaluate the validity of P_c^s as an estimator of system performance, it is necessary to correlate the conclusions from the estimate and what it is theoretically expected. Figure 6.7



(a) P_c^s for XY and adaptive routing, transpose traffic.

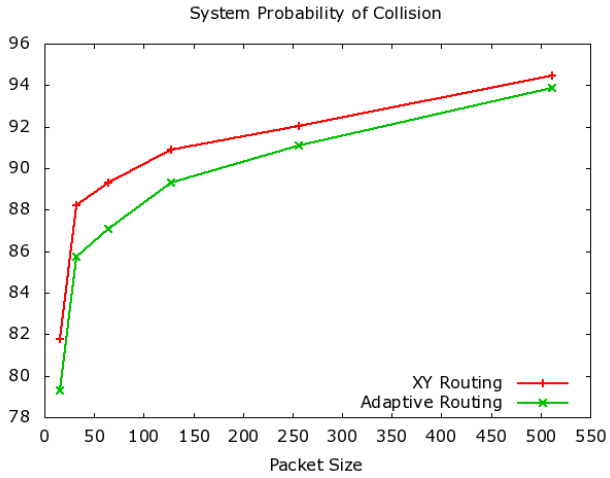


(b) Normalized P_c^s Difference with respect to adaptive routing, transpose traffic.

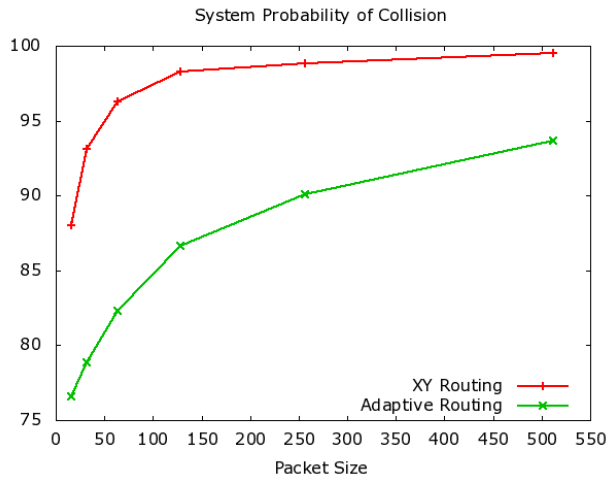
Figure 6.6: Performance estimate for XY and Adaptive routing when loaded under Transpose Traffic.

show the system probability of collision for XY and adaptive routing under hotspot, 6.7a, and transpose, 6.7b. As expected adaptive is shown to be the best choice for both cases.

While Figure 6.7 shows adaptive routing to be the better choice, it also shows that it is not



(a) P_c^s for hotspot traffic.



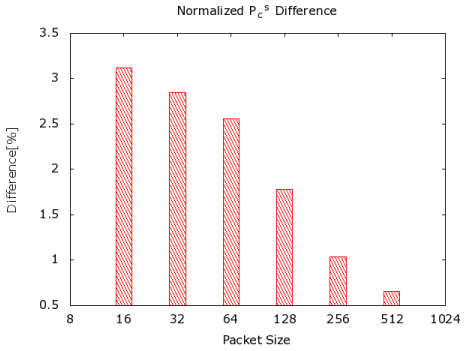
(b) P_c^s for transpose traffic.

Figure 6.7: P_c^s for XY and adaptive, hotspot *v.* transpose.

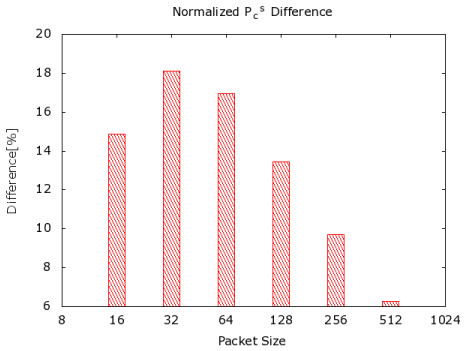
equally better for both cases. According the the system probability of collision estimation, adaptive is particularly better for a system with an application that having a transpose communication behavior. This comparison becomes more clear when looking at the normalized difference for hotspot versus transpose.

Figure 6.8 shows side-by-side the normalized P_c^s differences with respect to adaptive routing for hotspot and transpose traffic. Figures 6.8a and 6.8b shows the estimated performance improvement that adaptive routing may provide over XY routing. Figure 6.8a shows a maximum improvement of 3%, while for transpose traffic Figure 6.8b shows a maximum

improvement of 18%.



(a) Normalized P_c^s difference with respect to Adaptive routing, hotspot traffic.



(b) Normalized P_c^s difference with respect to adaptive routing, transpose traffic.

Figure 6.8: Normalized P_c^s differences with respect to adaptive routing, hotspot *v.* transpose.

The different improvement for the different traffic patterns is explained by the known behavior of the two routing schemes. The deterministic XY routing has no knowledge of the network conditions and obviously routes for the messages. As a result the collision performance of XY routing is is more susceptible to traffic. Figure 6.9 clearly shows this behavior.

On the other hand, adaptive considers the current state of the communication network when routing the messages. Therefore its collision performance is less susceptible to the traffic behavior. Figure 6.10 shows shows that for adaptive routing, unlike for XY routing, the estimated collision performance are closer together.

In conclusion this chapter showed how the mixture models may be used in the context of system level design. Two communication architectures were presented, adaptive and XY

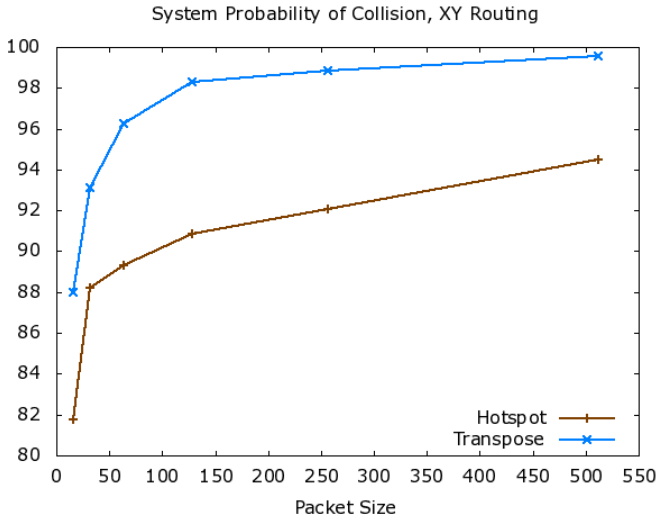


Figure 6.9: P_c^s for XY routing, hotspot *v.* transpose traffic.

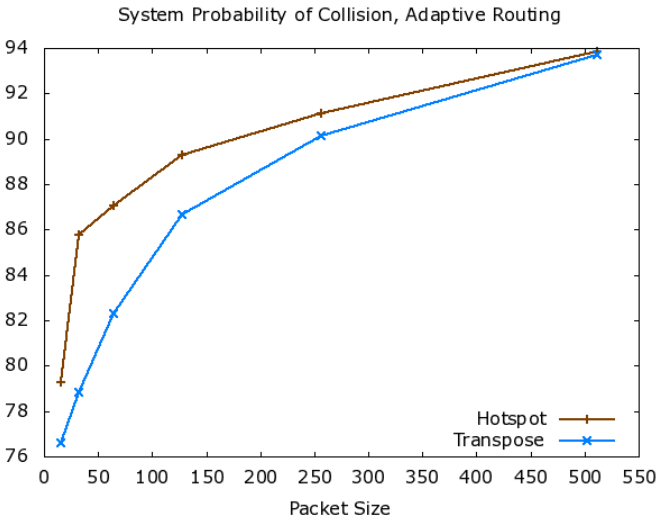


Figure 6.10: P_c^s for adaptive routing, hotspot *v.* transpose traffic.

routing, and compared using the high level estimator of system probability of collision for two different standard synthetic traffic patterns: hotspot and transpose traffic. In both cases the system probability of collision estimator indicated that adaptive routing was the better choice. Moreover, the system probability of collision estimator was shown to correlate with the theoretical behavior of both routing schemes across the different traffic patterns; thus, validating the system probability of collision as an estimator useful for making high level design decisions.

CHAPTER 7. Summary and Conclusion

This research asked the question of how to include the communication architecture features at the higher abstraction levels of the system level design. The proposed answer is a two part solution, (i) the models that capture the communication architecture features and move them into the higher abstraction levels, and (ii) the tools that allow designers to use these models to correctly navigate the design space at the higher levels of abstraction.

7.1 The Mixture Model

This research introduced a new methodology for modeling communication architecture features at higher abstraction levels. At the heart of this methodology is the mixture model. A mixture model is derived from the delay characteristics of the communication components, in particular the routing delay.

Mixture models developed for two different routing schemes, XY and adaptive. A mixture model is defined by

$$f_X(x) = \sum_{i=0}^{n} a_i f_{Y_i}(x),$$

where the density functions $f_{Y_i}(x)$ are the *mixture components*, and a_i are the *mixture proportions*; with the constraints that $0 \leq a_i \leq 1$ and $a_1 + a_2 + \cdots + a_i = 1$. Developing a mixture model to capture the communication behavior entails finding the combination of *mixture components* and *mixture proportions* that best describe the particular communication behavior.

To develop a mixture model for a particular routing scheme the steps are as follow.

1. Characterize the particular routing components and extract the delay distribution.

2. Split the characteristic distribution into two sets where, given a characteristic distribution of n data points, the first set contains the data points from 0 to w; and the second set has the rest of the data points from $w+1$ to n.

3. Then, iteratively:

   (a) perform separate exponential fits over each set to find the *mixture component* $f_{Y_1}(x)$ and $f_{Y_2}(x)$;

   (b) solve $T_{r_{ave}} = \mu_w * a + \mu_{w+1} * (1-a)$ to find the *mixture proportions* $a_1 = a$ and $a_1 = a - 1$, where $T_{r_{ave}}$ is the average routing time from the characterization;

   (c) combine the *mixture components* and *mixture proportions* into the mixture model $f_{T_R}(x) = a_0 f_{Y_0}(x) + a_1 f_{Y_1}(x)$;

   (d) compute the mean square error (MSE) between the characteristic distribution and the mixture model $f_{T_R}(x)$; and

   (e) change the size of the two initial sets by moving data points from the second set into the first set.

The mixture model with the minimum MSE is chosen to model the particular routing scheme at the higher abstraction levels.

This mixture model modeling methodology is the first contribution of this research. However, a higher level model is only useful if there are tools that allow the use of such model to estimate the system performance at the higher levels. For statistical models, like the mixture models, probability is such a tool. Given an event that properly captures the communication application behavior, then probability can measure the likelihood of this event happening in the system.

7.2 Probability of Collision

The collision distribution represents how the entire system reacts to the dynamic communication application behavior, and directly relates to the system performance. This research examined the probability that packets collide when competing for communication resources,

in particular the communication medium. Given two packets that share one link between their path, the collision event is defined as the time when both packets request this link.

In general, the probability of two packets colliding is given by

$$P_c = P(0 < |T_B - T_A| < T_T).$$

Where T_A and T_B are the times when packets A and B require the communication medium, respectively, and T_T is the amount of time that each packet requires exclusive access to the communication medium.

On the one hand, T_A and T_B are random variables whose values depend on the relative position of the shared link on the paths of packets A and B. On the other hand, T_T is the travel time and refers to the time that the shared link is busy transmitting a packet. Travel time is a function of several system parameters including packet size, bandwidth, flow control, etc.

Because T_A and T_B vary depending on the location of the shared link, it is necessary to define the probability of collision for the different possible shared link locations. A naming convention is defined, where a shared link of type $1|2$ indicates that the shared link is the first link on the path of packet A and the second link on packet B's path. Using this naming convention the probability of collision for the different sharing types may be found.

For example for a Type $1|2$ shared link

$$\begin{aligned} T_A &= T_{R_{1a}}, \text{ and} \\ T_B &= T_{R_{1b}} + T_T + T_{R_{2b}}. \end{aligned}$$

Where, $T_{R_{1a}}$, $T_{R_{1b}}$, and $T_{R_{2b}}$ are identically distributed independent random variables with PDF given by the mixture model. Therefore, the probability of collision for Type $1|2$ is

$$\begin{aligned} P_c^{1|2} &= P(0 < |T_B - T_A| < T_T). \\ &= P(-T_T < (T_{R_{1b}} + T_{R_{2b}} + T_T) - T_{R_{1a}} < T_T) \\ &= P(0 < T_{R_{1a}} - (T_{R_{1b}} + T_{R_{2b}}) < 2T_T). \end{aligned}$$

Combining the mixture models and the probability of collision provides a proper framework where the communication architecture features can be combined with the application com-

munication behavior, to produce a system level metric that can help the designer to navigate the system level design space at the higher abstraction levels.

7.3 System Level Design

The main goal of this research is to provide a framework that allows system level designers to evaluate the impact communication design choices have at higher abstraction levels. Until today, at the higher abstraction levels, designers are only capable of measuring the impact that the computation architecture features have on the system performance. However, this research shows that using mixture models in combination with the probability of collision, it is possible to estimate the impact of the communication architecture features at this higher abstraction level.

To estimate the communication architecture impact on the system performance, it is necessary to combine all the individual probabilities for the different sharing types into one System Probability of Collision. The System Probability of Collision combines each sharing type probability, and it is defined as

$$P_c^s = \frac{\sum_i^n freq_i * P_c^i}{\sum_i^n freq_i}.$$

Where P_c^i represents the probability of the i^{th} sharing type, and $freq_i$ is the associated share link type frequency.

The sharing type frequency, $freq_i$, measures the number of times a particular sharing type is repeated on a high level simulation of the system under a given application. That is, $freq_i$ captures the application communication behavior and allows the system probability of collision to combine the application communication behavior and the architecture communication characteristics into one dynamic estimator.

The system probability of collision, P_c^s, is the main contribution of this research. This probability serves as a high level system performance estimator. This estimator allows the designer to evaluate the system performance due to communication architecture, and aids the designer during the design space exploration at the higher abstraction levels.

7.4 Future Work

This research pioneered a methodology that enables design space exploration to include the communication architecture features at the higher abstraction levels. A mixture model is presented that includes the communication features at the higher abstraction levels, and a new system performance estimator is introduced to guide the design space exploration. However, some important questions remain unanswered.

7.4.1 Further Exploration of the Mixture Models

The mixture models of Chapter 4 are of the form

$$f_X(x) = \sum_{i=0}^{n} a_i f_{Y_i}(x).$$

However, only models with two mixture components, $f_{Y_1}(x)$ and $f_{Y_2}(x)$, are considered in this research. More complex models are able to capture more details. Thus, it would be beneficial to study if and how the methodology would benefit from more complex mixtures with three or more components.

The question of the complexity of the mixture model is a complicated one. While an initial two component mixture is intuitive, this is not the case for mixtures of three or more components; and complicated issues arise from the added complexity. Furthermore, using more mixture components further complicate the relation between the mixture proportions, a_i.

In this research the mixture proportions are found through $T_{r_{ave}} = \mu_w * a + \mu_{w+1} * (1 - a)$, however this relation only holds for the particular exponential mixture components used. If the mixture components are changed so that the given relation no longer holds, then a new relation must be developed to properly assign the mixture proportions.

7.4.2 Further Path Analysis

Chapter 5 showed the path analysis necessary to derive the system probability of collision. This path analysis concentrated on paths sharing only one link. Nevertheless, packets may

share more that one link. A question remains, how is the system probability of collision affected when considering multiple shared links.

This research showed how to derived the probability of collision of one shared link, and showed that this probability depends on the relative location of the shared link. To find the probability of collision for paths with multiple shared links would require derivations similar to those in Section 5.2. Once these new probabilities are found they may be combined into the system probability of collision, given the correct sharing frequencies.

The system probability of collision is a function of the shared link probability of collision and the shared link frequency at the higher abstraction levels. This research concentrated solely on the most frequent sharing link types. However, the high level simulation generates many more sharing types. To include each of these sharing types into the system probability of collision it is necessary to derive the probability of collision for the particular type.

The derivation of probability of collision of each type is complex and time consuming, and it is not particularly clear how much more accuracy may be achieved from adding these other less frequent types, before entering the point of diminishing returns. However, it is an important question that is left unanswered.

7.4.3 Further Assessment of Communication Architecture Features

Adaptive and deterministic routing schemes are very popular in embedded systems, but they are not the only ones available to system designers. Other routing schemes remain, and combination thereof, remain to investigate. The methodology introduced in this research applies to all communication components, but different communication architectures have different features.

This research is further limited to wormhole workflow only. Wormhole is by far the most widely used workflow scheme in embedded systems, but similar to routing, other options remain unstudied. However, unlike the routing schemes that are solely characterized by the mixture models, the workflow selection has further implication, particularly regarding the probability of collision.

Given wormhole workflow when a collision occurs between two packets, one packet

continues to its destination while the losing packet simply disappears as if gone through a wormhole. However, other workflow schemes with different behavior may produce different collision distributions.

Thus, it is imperative to continue to investigate how the different communication architecture features may be modeled as mixture distributions, without losing sight of the implication this features may have on the collision distribution. This entails finding the best mixture components and proportions for the particular architecture feature, and reevaluating the conditions that define the probabilities of collision.

7.4.4 Further Validation of P_c^s

In Chapter 6 the system probability of collision predicted adaptive as the better routing scheme for both hotspot and transpose traffic. This was the expected result and adaptive is known to outperform XY for this two cases. However, further validation of P_c^s would certainly be beneficial. In particular validating P_c^s against different low-level metrics such as packet dropped rate and flit delay. Showing the correlation between the high level estimate and the low-level metrics would present a more complete picture and further show the utility of P_c^s as a guide for system-level design exploration.

7.5 Final Remarks

In conclusion, the research presented here breaks new ground in system level design by allowing communication architecture features to be considered at the higher levels of abstraction. A completely new modeling methodology is developed from which communication architecture features, like routing, are abstracted and raised higher on the system level design hierarchy. At the heart of this methodology is the statistical mixture model.

The mixture models enable the decomposition of the empirical data into its components, and enable designers to determine how these components may be judged. Intuitively, and stemming from the law of diminishing returns, not all of the empirical data is necessary to make correct design decisions. The innovated methodology introduced in this research

provides an algorithmic process that result in a high level model that correctly captures the particular communication architecture feature.

Along with the innovative methodology, this research further pioneered a high level system performance estimator. Based on the probability of collision, this cutting edge estimator provides the means by which the mixture models are included into the system level design exploration. The system probability of collision combines the dynamic aspects of the application communication behavior and, through the mixture models, the communication architecture features providing a reliable performance estimate that system level designers can use to correctly explore the design space.

BIBLIOGRAPHY

[1] International technology roadmap for semiconductors, 2009. URL `http://www.itrs.net/Links/2009ITRS/Home2009.htm`.

[2] Guido Arnout. The SystemC Standard. In *ASP-DAC '00: Proceedings of the 2000 conference on Asia South Pacific design automation*, pages 573–578, New York, NY, USA, 2000. ACM. ISBN 0-7803-5974-7. doi: http://doi.acm.org/10.1145/368434.368808.

[3] Amer Baghdadi, Nacer-Eddine Zergainoh, Wander O. Cesário, and Ahmed Amine Jerraya. Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems. *IEEE Trans. Softw. Eng.*, 28(9):822–831, 2002. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2002.1033223.

[4] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2003.1193228.

[5] Luca Benini. Application specific noc design. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 491–495, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. ISBN 3-9810801-0-6.

[6] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *Computer*, 35(1):70–78, 2002. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.976921.

[7] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc: Qos architecture and design process for network on chip. *J. Syst. Archit.*, 50(2-3):105–128, 2004. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2003.07.004.

[8] Luciano Bononi and Nicola Concer. Simulation and analysis of network on chip architectures: ring, spidergon and 2d mesh. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 154–159, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. ISBN 3-9810801-0-6.

[9] Lukai Cai. *Estimation and exploration automation of system level design*. PhD thesis, 2004. Chair-Gajski,, Daniel.

[10] Lukai Cai and Daniel Gajski. Transaction level modeling: an overview. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 19–24, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-742-7. doi: http://doi.acm.org/10.1145/944645.944651.

[11] Lukai Cai, Andreas Gerstlauer, and Daniel Gajski. Retargetable profiling for rapid, early system-level design space exploration. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 281–286, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-8. doi: http://doi.acm.org/10.1145/996566.996651.

[12] P. Chandraiah, Junyu Peng, and R. Domer. Creating explicit communication in soc models using interactive re-coding. pages 50–55, Jan. 2007. doi: 10.1109/ASP-DAC.2007.357791.

[13] Marcello Coppola, Stephane Curaba, Miltos D. Grammatikakis, Giuseppe Maruccia, and Francesco Papariello. Occn: A network-on-chip modeling and simulation framework. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 30174, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2085-5-3.

[14] Williams James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[15] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das. Application-aware prioritization mechanisms for on-chip networks. In *Micro-42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages

280–291, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-798-1. doi: http://doi.acm.org/10.1145/1669112.1669150.

[16] Abhijit K. Deb, Johnny Öberg, and Axel Jantsch. Control and communication performance analysis of embedded dsp systems in the masic methodology. In *ISSS '01: Proceedings of the 14th international symposium on Systems synthesis*, pages 274–273, New York, NY, USA, 2001. ACM. ISBN 1-58113-418-5. doi: http://doi.acm.org/10.1145/500001.500064.

[17] Sujit Dey and Surendra Bommu. Performance analysis of a system of communicating processes. In *ICCAD '97: Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 590–597, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8200-0.

[18] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, 85(3):366–390, Mar 1997. ISSN 0018-9219. doi: 10.1109/5.558710.

[19] Fabrizio Fazzino, Maurizio Palesi, and Davide Patti. Noxim: Network-on-chip simulator, 2008. URL `http://noxim.sourceforge.net/`.

[20] Anja Feldmann and Ward Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 1096, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7780-5.

[21] Masahiro Fujita and Hiroshi Nakamura. The standard specc language. In *ISSS '01: Proceedings of the 14th international symposium on Systems synthesis*, pages 81–86, New York, NY, USA, 2001. ACM. ISBN 1-58113-418-5. doi: http://doi.acm.org/10.1145/500001.500019.

[22] F. Fummi, D. Quaglia, and F. Stefanni. A systemc-based framework for modeling and simulation of networked embedded systems. pages 49–54, Sept. 2008. doi: 10.1109/FDL.2008.4641420.

[23] Daniel D. Gajski, Rainer Domer, Junyu Peng, and Andreas Gerstlauer. *System Design: A Practical Guide with Specc*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 0792373871.

[24] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 246–251, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2288-2. doi: http://dx.doi.org/10.1109/DATE.2005.5.

[25] Andreas Gerstlauer, Dongwan Shin, Rainer Dömer, and Daniel D. Gajski. System-level communication modeling for network-on-chip synthesis. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 45–48, New York, NY, USA, 2005. ACM. ISBN 0-7803-8737-6. doi: http://doi.acm.org/10.1145/1120725.1120740.

[26] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago Gonzalez Pestana, Andrei Radulescu, and Edwin Rijpkema. A design flow for application-specific networks on chip with guaranteed performance to accelerate soc design and verification. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1182–1187, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2288-2. doi: http://dx.doi.org/10.1109/DATE.2005.11.

[27] J. Grode, P. V. Knudsen, and J. Madsen. Hardware resource allocation for hardware/software partitioning in the lycos system. In *DATE '98: Proceedings of the conference on Design, automation and test in Europe*, pages 22–27, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8359-7.

[28] Omar Hammami and Muhammad Omer Cheema. Graduate education to fight system level design productivity gap in soc design. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 45–46,

Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2849-X. doi: http://dx.doi.org/10.1109/MSE.2007.46.

[29] H. Hashemi-Najafabadi, H. Sarbazi-Azad, and P. Rajabzadeh. An accurate performance model of fully adaptive routing in wormhole-switched two-dimensional mesh multicomputers. *Microprocess. Microsyst.*, 31(7):445–455, 2007. ISSN 0141-9331. doi: http://dx.doi.org/10.1016/j.micpro.2006.12.006.

[30] Wai Hong Ho and Timothy Mark Pinkston. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, page 377, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1871-0.

[31] Jingcao Hu and Radu Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10688, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1870-2.

[32] Jingcao Hu and Radu Marculescu. Dyad: smart routing for networks-on-chip. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 260–263, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-8. doi: http://doi.acm.org/10.1145/996566.996638.

[33] Sungchan Kim, Chaeseok Im, and Soonhoi Ha. Efficient exploration of on-chip bus architectures and memory allocation. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 248–253, New York, NY, USA, 2004. ACM Press. ISBN 1-58113- 937-3. doi: http://doi.acm.org/10.1145/1016720.1016779.

[34] Peter Voigt Knudsen and Jan Madsen. Integrating communication protocol selection with partitioning in hardware/software codesign. In *ISSS '98: Proceedings of the 11th international symposium on System synthesis*, pages 111–116, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8623-5.

[35] Akash Kumar, Bart Mesman, Henk Corporaal, Bart Theelen, and Yajun Ha. A probabilistic approach to model resource contention for performance estimation of multi-featured media devices. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 726–731, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-627-1. doi: http://doi.acm.org/10.1145/1278480.1278662.

[36] Shashi Kumar, Acel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johny Öberg, Kari Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *ISVLSI '02: Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, page 117, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1486-3.

[37] Santanu Kundu and Santanu Chattopadhyay. Mesh-of-tree deterministic routing for network-on-chip architecture. In *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 343–346, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-999-9. doi: http://doi.acm.org/10.1145/1366110.1366191.

[38] Kanishka Lahiri, Anand Raghunathan, and Sujit Dey. Efficient exploration of the soc communication architecture design space. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 424–430, Piscataway, NJ, USA, 2000. IEEE Press. ISBN 0-7803-6448-1.

[39] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, and Radu Marculescu. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):23, 2007. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1255456.1255460.

[40] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, and Roberto Zafalon. Analyzing on-chip communication in a mpsoc environment. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 20752, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2085-5-2.

[41] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.982916.

[42] Shankar Mahadevan, Federico Angiolini, Michael Storgaard, Rasmus Grondahl Olsen, Jens Sparso, and Jan Madsen. A network traffic generator model for fast network-on-chip simulation. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 780–785, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2288-2. doi: http://dx.doi.org/10.1109/DATE.2005.22.

[43] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram. An empirical investigation of mesh and torus noc topologies under different routing algorithms and traffic models. In *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 19–26, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2978-X. doi: http://dx.doi.org/10.1109/DSD.2007.28.

[44] Asit K. Mishra, Ravi Iyer, Reetuparna Das, N. Vijaykrishnan, Soumya Eachempati, and Chita R. Das. A case for dynamic frequency tuning in on-chip networks. In *Micro-42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 292–303, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-798-1. doi: http://doi.acm.org/10.1145/1669112.1669151.

[45] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1998. ISSN 0018-9219. doi: 10.1109/JPROC.1998.658762.

[46] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren. Uml for esl design: basic principles, tools, and applications. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 73–80, New York, NY, USA, 2006. ACM. ISBN 1-59593-389-1. doi: http://doi.acm.org/10.1145/1233501.1233518.

[47] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 146–151, Piscataway, NJ, USA, 2006. IEEE Press. ISBN 0-7803-9451-8. doi: http://doi.acm.org/10.1145/1118299.1118344.

[48] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. Application specific routing algorithms for networks on chip. *Parallel and Distributed Systems, IEEE Transactions on*, 20(3): 316 –330, march 2009. ISSN 1045-9219. doi: 10.1109/TPDS.2008.106.

[49] Preeti Ranjan Panda. Systemc: a modeling platform supporting multiple design abstractions. In *ISSS '01: Proceedings of the 14th international symposium on Systems synthesis*, pages 75–80, New York, NY, USA, 2001. ACM. ISBN 1-58113-418-5. doi: http://doi.acm.org/10.1145/500001.500018.

[50] Sudeep Pasricha, Nikil Dutt, and Mohamed Ben-Romdhane. Fast exploration of bus-based communication architectures at the ccatb abstraction. *Trans. on Embedded Computing Sys.*, 7(2):1–32, 2008. ISSN 1539-9087. doi: http://doi.acm.org/10.1145/1331331.1331346.

[51] Faizal A. Samman, Thomas Hollstein, and Manfred Glesner. Multicast parallel pipeline router architecture for network-on-chip. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 1396–1401, New York, NY, USA, 2008. ACM. ISBN 978-3-9810801-3-1. doi: http://doi.acm.org/10.1145/1403375.1403714.

[52] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Des. Test*, 18(6):23–33, 2001. ISSN 0740-7475. doi: http://dx.doi.org/10.1109/54.970421.

[53] S. Santi, B. Lin, L. Kocarev, G.M. Maggio, R. Rovatti, and G. Setti. On the impact of traffic statistics on quality of service for networks on chip. pages 2349–2352 Vol. 3, May 2005. doi: 10.1109/ISCAS.2005.1465096.

[54] Joseph P. Schneider. Low-level estimation at high-levels of abstraction in system-level design. MS in Computer Engineering, Iowa State University, Ames, IA, USA, 2007.

[55] Timo Schonwald, Jochen Zimmermann, Oliver Bringmann, and Wolfgang Rosenstiel. Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. In *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 527–534, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2978-X. doi: http://dx.doi.org/10.1109/DSD.2007.62.

[56] Donatella Sciuto, Grant Martin, Wolfgang Rosenstiel, Stuart Swan, Frank Ghenassia, Peter Flake, and Johny Srouji. Systemc and systemverilog: Where do they fit? where are they going? In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10122, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2085-5-1.

[57] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 667–672, New York, NY, USA, 2001. ACM. ISBN 1-58113-297-2. doi: http://doi.acm.org/10.1145/378239.379045.

[58] Sören Sonntag, Matthias Gries, and Christian Sauer. SystemQ: Bridging the gap between queuing-based performance evaluation and systemc. *Design Automation for Embedded Systems*, 11(2):91–117, September 2007. URL `http://dx.doi.org/10.1007/s10617-006-9002-3`.

[59] Leonel Tedesco, Aline Mello, Leonardo Giacomet, Ney Calazans, and Fernando Moraes. Application driven traffic modeling for NoCs. In *SBCCI '06: Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pages 62–67, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-479-0. doi: http://doi.acm.org/10.1145/1150343.1150364.

[60] K. Van Rompaey, I. Bolsens, H. De Man, and D. Verkest. Coware—a design environment for heterogenous hardware/software systems. In *EURO-DAC '96/EURO-VHDL '96: Proceedings of the conference on European design automation*, pages 252–257, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press. ISBN 0-8186-7573-X.

[61] Guang Yang, Alberto Sangiovanni-Vincentelli, Yosinori Watanabe, and Felice Balarin. Separation of concerns: overhead in modeling and efficient simulation techniques. In *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*, pages 44–53, New York, NY, USA, 2004. ACM. ISBN 1-58113-860-1. doi: http://doi.acm.org/10.1145/1017753.1017765.

[62] Ki Hwan Yum, A. Vaidya, C.R. Das, and A. Sivasubramaniam. Investigating qos support for traffic mixes with the mediaworm router. pages 97–106, 2000. doi: 10.1109/HPCA.2000.824342.