Performance Modelling: Acceleration and Optimisation of Networks at Different Scale

by

Shruti Yadav Narayana

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: 4/5/2024

The dissertation is approved by the following members of the Final Oral Committee:

Karthikeyan Sankaralingam, Professor, Computer Sciences and Electrical and Computer Engineering
Parameswaran Ramanathan, Professor, Electrical and Computer Engineering
Bhuvana Krishnaswamy, Assistant Professor, Electrical and Computer Engineering
Ming Liu, Assistant Professor, Computer Sciences Engineering

Dedicated to my parents and my sister.

ACKNOWLEDGMENTS

I express my heartfelt gratitude to my parents, Sarojini (my mother) and Narayana (my father), who instilled in me the belief that education transcends societal norms. Their unwavering emotional support and understanding have been pivotal in my journey to the U.S. and the completion of my dissertation. I consider myself exceptionally fortunate to be born to such amazing parents.

My gratitude extends to my extended family – Prabhat Uncle, Abha Aunty, Rini Akka, and Richie Anna – who, although not connected by blood, are an integral part of my life. From sponsoring my education to encouraging me to pursue a Ph.D., I am thankful for their invaluable advise, help and kindness, especially in a world where being kind and caring is rare.

I owe a special thanks to my sister, Shubha, who has been my guiding light. Her unwavering presence, patient listening, and comforting solace during challenging times have been indispensable. Her support has been instrumental in my dissertation journey.

I want to thank Sumit K Mandal for his invaluable mentorship and support during my transition into the new world of academia. His guidance not only equipped me with the necessary tools to navigate this unfamiliar terrain but also provided a source of comfort and encouragement during challenging times. I am particularly grateful for the late-night discus-

sions, assistance with code debugging, and his role as a trusted confidant throughout the rigors of pursuing a PhD. His unwavering belief in my abilities, coupled with his words of encouragement, bolstered my confidence when self-doubt crept in. Thank you for being not only a mentor but also a friend during this journey.

I want to thank Emily Shriver for constantly supporting, helping and checking up on me. Her guidance as my mentor during my internships has been indispensable. Her support has been instrumental in my dissertation journey.

Appreciation is also extended to my friends – Hemalatha, Nishanth, Anish, and Ganapati – who believed in my capabilities and shared their own challenges, motivating me to give my best. Hema and Nishanth for constantly believing in me and inspiring me to do better and to do more. Especially Hema who has supported me throughout my journey. Anish and Ganapati for always being there and helping me in case of need.

I am profoundly grateful for my partner, Levi Maxwell, who, though entering my life later, has been an immense blessing. His unwavering support, patience, and understanding have been my pillars. Without him, my journey would have felt lonely, and I credit his constant motivation for helping me complete my dissertation.

Finally, my sincere thanks to Professor Karu. His timely intervention and guidance were pivotal in my achievement of obtaining a PhD, a goal that seemed almost unattainable at one point. His unwavering assistance,

patience, and understanding were essential pillars throughout this process, and I sincerely acknowledge that I could not have succeeded without your help. His role as a mentor transcended mere academic guidance; he exemplified the qualities of compassion and kindness that are rare to find in today's world. I am deeply appreciative of his mentorship and the positive impact it has had on both my academic and personal development.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

As multicore processors continue to scale, Networks-on-Chips (NoCs) have emerged as the de facto on-chip interconnect solution. However, industrial NoCs are complex, comprising multiple physical layers with interdependencies that impact performance. Accurately understanding and modeling these interactions are paramount for effective design-space exploration. Pre-silicon design-space exploration and system-level simulations are essential in industrial design cycles to ensure specifications are met without costly post-tape-out iterations. Yet, cycle-accurate simulations are notoriously slow, particularly for processors with multi-layer NoCs, due to their vast design space.

To address these challenges, we introduce a performance analysis technique that models interactions and estimates destination layer blocking probability to compute finite queuing delays in the source layer. Experimental evaluations demonstrate our approach is consistently within 10% of, and five times faster than, cycle-accurate simulations for 6×6 mesh NoCs under both synthetic and real traffic scenarios.

In NoCs, congestion significantly impacts processor performance, leading to core stalls and wasted link bandwidth. Traditional approaches throttle cores post-congestion, reducing efficiency. In contrast, our lightweight machine learning-based technique predicts congestion, leveraging traffic-related features and a novel time reversal labeling approach. Experimental

evaluations on industrial 6×6 NoCs show up to 114% increase in fairness and memory read bandwidth compared to existing techniques, with less than 0.01% overhead.

The demand for low-latency, large-scale distributed systems continues to grow, necessitating performance-optimal distributed networks. Packet-level simulators provide accurate modeling but are slow. To address scalability and accuracy challenges, we propose a novel methodology combining queuing theory with the maximum entropy principle, augmented by regression tree learning. This ML-assisted technique achieves less than 3% modeling error on average compared to ns-3 simulations, with speedups ranging from 100× to 9000× on DCNs with 128 to 1024 nodes.

While analytical techniques offer speed advantages over packet-level simulators, their scalability may degrade with increasing network size. To mitigate this, we propose a similarity-based technique that enhances speed by up to 2000x with minimal accuracy impact, enabling performance estimates for large-scale DCNs.

1 INTRODUCTION

Analytical performance modeling of communication networks offers a rapid and lightweight approach to pinpoint congestion points, hotspots, and communication latency issues that impede performance. Regardless of the network scale, pre-silicon design space exploration and system-level simulations are crucial in industrial design cycles to meet specifications without costly iterations. Employing cycle-accurate simulators for overall performance evaluation is common, but detailed models, while accurate, can be time-prohibitive. Thus, accurate analytical models serve as promising alternatives, providing speed advantages. With confidence from analysis and simulation, designers can efficiently advance to circuit and backend design stages. In this thesis, we present analytical performance modelling for both small scale Networks-on-Chip (NoCs) and large scale data center networks (DCNs)

Small Scale Network: In Networks-on-Chip (NoCs), unlike early academic proposals that use virtual channels [61], industrial NoCs comprise multiple physical layers for requests, data, acknowledgment (ACK), and snoop traffic [62]. Multiple physical layers provide significantly higher interconnect bandwidth, making them better for many high-performance computing and networking applications [62]. While these NoCs have dedicated routers and links, they are still connected to the same endpoints, i.e., cores and cache controllers. Consequently, they interact with each other

by applying backpressure, hence affecting performance. Understanding and modeling the effects of these interactions is crucial for design-space exploration, but existing analytical performance modeling techniques fail to capture this critical interaction since they model a single-layer NoC.

As mentioned, performance modeling of networks (in this case, NoCs) aids in comprehending issues that impact performance of the system (in this case, multi-core processor). This is crucial as efficient communication among the various components on a multi-core chip is essential for achieving optimal performance. It has been shown that NoCs are affected more by networking problems such as congestion than by architectural problems [51]. NoCs (buffered or bufferless) implement backpressure mechanisms to prevent packet losses. As a result, the throughput (the number of packets processed per unit time) decreases, and the overall performance of the SoC deteriorates. Existing reactive source throttling techniques throttle the source after congestion happens and hence, they do not maintain the throughput of the NoC under heavy traffic. This emphasizes the necessity for a proactive and lightweight congestion control technique.

Large Scale Network: Data centers house close to 200,000 nodes aiding to the growing demand of emerging applications. Deploying DCs of such size requires significant capital and operational expense to architect the Data Center Network (DCN) that facilitates communication between the distributed components. Therefore, DCN architects spend substantial

effort designing the network topology, routing algorithms, and protocols. Notoriously slow simulation-based design space exploration is not practical. Therefore, there is need for accurate analytical models that speed up the analysis of DCNs that employ 1000's of nodes, while incorporating routing algorithms and protocols which affect the traffic pattern and the load respectively.

Because data centers employ close to 200,000 nodes, the number of queues represented by the analytical model is $>$ 200,000 which includes the queues in the switches. Although analytical models are fast and lightweight when compared to packet level simulator, their speed advantage can degrade as the network size increases. Furthermore, the routing algorithms used in DCNs enable multiple path from a single source to destination increasing the complexity of the analytical model. This necessitates a technique that speeds up the analytical model of DCNs when scaled up to 100,000 nodes.

In this dissertation, we tackle the diverse challenges posed by small-scale Network-on-Chip (NoC) architectures for multi-core processors and large-scale Data Center Networks (DCNs) by establishing a cohesive set of principles. These principles should revolve around the adept modeling of intricate interactions and congestion control mechanisms to ensure optimal communication performance in both scenarios. Concurrently, these principles should extend seamlessly to large-scale DCNs, where analytical models must accommodate the significant volume of nodes and the intri-

cacies introduced by routing algorithms that enable multiple paths. The overarching framework should prioritize scalability, facilitating the analysis of communication networks across various scales, from modest NoCs to extensive DCNs. Ultimately, these unifying principles should guide the formulation of analytical models that provide valuable insights into performance optimization, congestion mitigation, and efficient communication within diverse network architectures.

The contributions of each work aiming to tackle the aforementioned challenges are as follows:

1. Develop an analytical model for multiple Network-on-Chip (NoCs) considering back-pressure and finite buffers, coupled with a scalable end-to-end algorithm adaptable to various NoC sizes and buffer configurations.

2. Introduce a unique time reversal approach and supervised learning for constructing a decision tree tailored for proactive congestion control in industrial NoCs in order to optimise memory read bandwidth.

3. Develop an analytical model for large-scale network architectures with over 1000 nodes with the ability to provide detailed observability in to the network, complemented by an automated tool for generating executable performance models in Data Center Networks (DCNs).

4. Implement a similarity-based technique that reduces the algorithmic complexity of analytical model in DCN performance evaluation from quadratic to linear, with minimal accuracy trade-offs enabling the feasibility of DCN performance evaluation at scale.

In summary, this dissertation presents a series of works aimed at enhancing the modeling accuracy and performance optimization of Network-on-Chip (NoC) and Data Center Networks (DCNs). The first set of works achieves NoC models with less than 10% modeling error on average. The subsequent work focuses on optimizing NoC for memory read bandwidth, while ensuring favorable average bytes received per cycle and average transaction latency.

The third work in the dissertation addresses DCN modeling, achieving less than 3% modeling error on average and a significant speedup of at least 100$\times$ compared to simulation. Finally, the last work concentrates on optimizing the analytical model for DCNs, specifically targeting speed when testing on networks with over 1000 nodes. Overall, these contributions aim to advance the accuracy and efficiency of network modeling in the context of NoCs and DCNs.

The rest of the dissertation is organized as follows. Chapter 2 – Chapter 5 provide detailed description of the four works.

2 FAST ANALYSIS USING FINITE QUEUING MODEL FOR MULTI-LAYER NOCS

2.1 Overview

NoCs are now the standard on-chip interconnect solution for large many-core processors [62]. Unlike early academic proposals, industrial NoCs incorporate multiple physical layers for requests, data, acknowledgment (ACK), and snoop traffic [62], providing higher interconnect bandwidth for high-performance computing and networking applications. Despite dedicated routers and links, these NoCs still connect to the same endpoints (cores and cache controllers), leading to interactions and backpressure that impact performance. Existing techniques, designed for single-layer NoCs, fail to capture these crucial interactions.

In industrial design cycles, pre-silicon design-space exploration and system-level simulations are vital to meet specifications without costly post-tape-out iterations. Cycle-accurate simulators, commonly used, are slow [61], especially for processors with multi-layer NoCs, resulting in a massive design space that further slows simulations. Accurate analytical models offer a promising alternative, being lightweight and significantly faster. However, existing NoC performance analysis techniques assume a single physical layer with virtual channels, rendering them inadequate for modern processors with multi-layer NoCs.

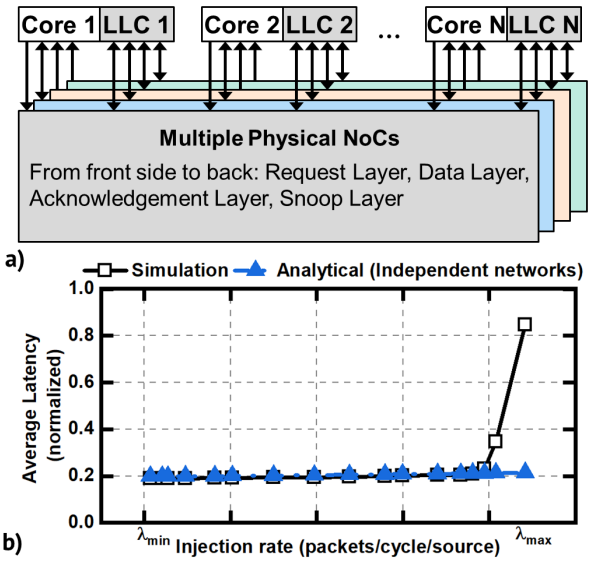

Figure 2.1: (a) Ilustration of multiple NoC layers with interfaces to all the cores, and last-level caches in the system. (b) Comparison between cycle-accurate simulation and performance analysis considering the interactions between different NoCs and treating them independently [42].

Different physical layers in NoCs have dedicated routers and links, but they connect at the cores and cache controller boundaries, as illustrated in Figure 2.1(a). Interconnected layers, including request, data, acknowledgment, and snoop, can apply back pressure due to finite buffers, leading to congestion. For instance, a core's request packet travels through the request layer, potentially causing congestion when data and acknowledgment layers apply back pressure. Traditional performance analysis techniques assuming independent layers or infinite buffers fail to capture such inter-dependencies, impacting accurate performance estimation. Figure 2.1(b) illustrates this, comparing the average packet latency of

an isolated layer from a recent analysis technique (blue triangle marker) against cycle-accurate simulations. The technique, lacking modeling for backpressure, inaccurately estimates latency under congestion.

This dissertation introduces a novel performance analysis technique for multi-layer NoCs, considering finite queuing models, priority-aware arbitration, and industry-standard deflection routing. Prior to this work, no attempt has been made to model the interdependence between multiple physical layers in modern NoCs comprehensively. The complexity of constructing such a technique arises from the intricate interactions between the networks. For instance, back pressure results from factors like source injection rate, request hit rate, cache controller and memory controller processing speeds, buffer sizes, and more. We address this complexity by estimating congestion probability at the response layers' egress queues (data and ACK layers). This probability informs the estimation of back-pressure on the request layer and the consequent deflection probability. Ultimately, the average packet latency in the NoC is estimated using this deflection probability.

The significant contributions of this work are as follows:

- Analytical model targeting multiple NoCs considering back-pressure and finite buffers,
- A scalable end-to-end algorithm that is generalized to NoC size and buffer sizes,

- Thorough experimental evaluation showing less than 10% modeling error on average for industrial NoC executing synthetic traffic as well as real applications.

2.2 Related Work

Several performance analysis techniques of NoCs using queuing theory have been proposed in the literature [26, 42, 53, 61]. One of the early NoC performance analysis techniques is proposed in [53], where authors assume exponentially distributed interarrival times, infinite buffers in the network, and finite buffers at the NoC input interfaces. Later, a similar analytical model developed in this work is used to develop a machine learning-based technique for NoC performance analysis, assuming a general distribution for the incoming traffic [61]. These studies assume round-robin arbitration for the NoC, while most NoCs used in state-of-the-art industrial SoCs are priority-aware [62].

Kiasari et al. [26] proposed an analytical performance model for priority-aware NoCs. However, the priority-aware NoCs used in the industry also incorporate deflection routing. Analytical performance analysis technique for industrial priority-aware NoCs with deflection routing is considered in [42]. The authors use the deflection probability as an input to the analytical model. Moreover, their technique assumes infinite buffers. In contrast, industrial NoCs consist of small buffers (4–16) and different layers for

different transactions.

The authors in [61] propose a new and accurate analytical model that uses the application communication graph, the NoC architecture, and the routing algorithm for queuing delay analysis. However, the framework does not consider deflection routing and small finite buffers. Kouvastos et al. [31] propose an open queueing network model for wormhole-routed hypercubes with finite buffers and deterministic routing. The authors in [5] propose a methodology that can be utilized to analyze different heterogeneous NoC architectures and traffic scenarios. Nikitin et al. [49] have modeled the NoC as a constant service time system using M/D/1 queue analysis which is unfeasible. However, none of the work above considers priority-aware deflection routing, which is generally used in industrial NoCs. They also do not model the interactions between the NoCs, which is crucial for accurate performance analysis, as shown in Figure 2.1.

In contrast to the prior work, this dissertation proposes a performance analysis technique for multi-layer NoCs with finite buffers used in modern industrial NoCs. This is challenging due to the nontrivial interactions between different NoC layers. To the best of our knowledge, this is the first analytical model for multi-layer NoCs considering finite queuing model.

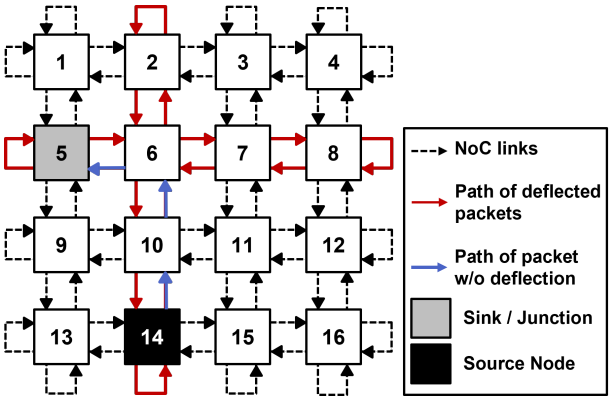

Figure 2.2: A 4x4 mesh with deflection routing

Background and Motivation

This work focuses on modeling multi-layer priority-aware NoCs with deflection routing and finite buffers, which applies to a wide range of industrial NoCs [3]. In priority-aware deflection routing, packets already in the network have higher priority than newly injected packets. Deflection occurs when the sink queue at the destination becomes full and cannot

Table 2.1: Comparison of prior research and our novel contribution.

Research	**Deflection Routing**	**Bursty Traffic**	**Discrete Time**	**Finite Buffer**	**Multiple NoCs**
Ben-Itzhak et al.[5]	No	No	No	No	No
Nikitin et al.[49]	No	No	No	No	No
Kouvatsos et al.[31]	No	Yes	No	Yes	No
Qian et al.[61]	No	Yes	No	Yes	No
Kiasari et al.[26]	No	Yes	No	No	No
Mandal et al.[42]	Yes	Yes	Yes	No	No
This work	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

accept more packets, causing the packets injected by the core to deflect from the sink queue into the network. For example, Figure 2.2 shows a 4x4 mesh NoC architecture, where Node 14 sends a packet to Node 5 following Y-X routing (highlighted by thick blue arrows). Deflection happens when the queue at the turning point (Node 6) or final destination (Node 5) becomes full. The probability of encountering a full queue, hence deflection, increases with the traffic load and smaller queue sizes (used to save area). Consequently, regular and deflected traffic can load the corresponding row and pressure the queue at the turning point (Node 6) and the sink. This, in turn, can lead to deflection on the column and row (shown by red arrows), propagating the congestion toward the source and wasting valuable NoC bandwidth.

Traffic congestion can also propagate from one layer (e.g., the Data layer) to another (e.g., request layer). For example, a cache controller injecting packets into the data and acknowledgment layers at a high rate (e.g., due to high LLC (Last Level Cache) hit rates) can cause a faster accumulation of packets in its egress queues. As the egress of the data and acknowledgment layer fills up, the rate at which the cache controller can serve new requests slows down. As a result, the sink buffers in the cache controllers fill up, applying backpressure to the request layer and causing deflection, leading to congestion. Indeed, Figure 2.1b shows that the average latency of packets in the layer increases significantly in the higher injection region due to the interdependence between layers. Existing

analytical models often fail to capture this interdependence, which is crucial for accurately analyzing NoC performance.

2.3 Proposed Multi-NoC Performance Analysis Technique

This section describes the proposed technique. First, we develop a model for each physical layer using a canonical system representing a single traffic class. Subsequently, we scale this model to accommodate multiple traffic classes within each layer. The output of one network layer serves as the input to the destination, capturing their interdependence. After constructing the models for all layers, we combine them to obtain the end-to-end latency.

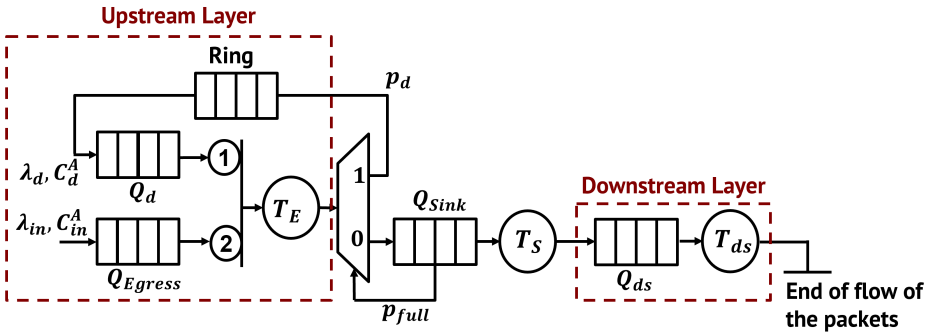

Figure 2.3: Canonical System with multiple NoC layers. The downstream NoC is abstracted as a single queue (Q_{ds}).

Description of Multi-NoC with Single Traffic Class

Simplified canonical system: The inputs to the canonical system are the traffic distributions of the new requests from the cores into the egress queues (Q_{Egress}). The traffic is modeled using Generalized Geometric (GGeo) distribution with two moments (mean injection rate (λ_{in}) and the coefficient of variation of inter-arrival time (C_{in}^{A}) of packets entering into Q_{Egress}) to capture burstiness, as shown in Figure 2.3.

Upstream Layer: The new requests traverse the request layer (marked as "Upstream Layer") to reach the Last Level Cache (LLC) controller. On reaching the destination, the request is either sunk into the sink queue (Q_{Sink}) or deflected back into the request layer if Q_{Sink} is full, indicated by p_{full} the probability that sink queue is full. This deflections require the request layer to handle both the new requests into Q_{Egress} and the deflected packets from Q_{Sink}. The deflected packets have higher priority (shown by ①) than the new requests (shown by ②), as seen in Figure 2.3. They keep circulating in the layer till Q_{Sink} consumes them. The latency of transaction of packets (the new request and the deflected packets) to reach the destination is used to model the service time ($\widehat{T}_E$), as shown in Figure 2.3.

Downstream Layer: After Q_{Sink} processes an incoming request packet from the upstream layer, the corresponding LLC controller generates a new packet and injects it into the downstream layer. When the request results in

an LLC hit, it produces a data and acknowledgement packet injected into the data layer and acknowledgement layer, respectively. Otherwise, the request is forwarded to the memory controller. Our approach can handle all interactions arising from the cache-coherency protocol. However, for simplicity, we abstract the downstream network as a single queue (Q_{ds}) with an effective service time (T_{ds}), as shown in Figure 2.3. The abstracted downstream layer is expanded and presented later in Section 2.3.B.

Dependencies between multiple NoCs: The upstream layer injects request packets into the downstream data and acknowledgement layers. As the number of injected packets into the downstream layer increases, the queuing system experiences backpressure due to accumulation of packets in queue. The accumulation of packets eventually causes Q_{Sink} in Figure 2.3 to become full. As a result, incoming requests from the Q_{Egress} in the upstream layer are deflected back into the upstream layer which in turn increases the waiting time of the packets in Q_{Egress}. That is, the backpressure from the downstream layer increases the traffic load and latency in the upstream layer. The rest of this section models this behaviour.

Analytical Modeling of Multi-NoC with Single Traffic Class

We first find the average waiting time ($\widehat{W}_E$) in Q_{Egress} using the following steps:

1. Understanding the accumulation of packets in the downstream layer. Therefore, we start by *modeling the blocking probability* ($\mathrm{p_b}$) *at the downstream layer.*

2. Understanding the accumulation of packets in $\mathrm{Q_{Sink}}$ caused by the blocking probability in the downstream layer. Therefore, we continue by *modeling the probability* ($\mathrm{p_{full}}$) *of* $\mathrm{Q_{Sink}}$ *being full.*

3. Understanding the impact of back-pressure from the downstream layer towards the upstream layer. Therefore, we finish by *modeling the waiting time* ($\widehat{W}_{\mathrm{E}}$) *in* $\mathrm{Q_{Egress}}$ *in the upstream layer.*

Figure 2.4 shows a pictorial representation of the steps for clarity, while Table 2.2 lists the symbols used in this dissertation. We want to point out that the maximum entropy (ME) method is employed due to its closed-form solution, to obtain a fast, lightweight and efficient analytical model. It provides the probability distribution of the system state (in this case, the queue occupancy) based on certain mean value constraints. The principle of maximum entropy states that the most unbiased distribution satisfying given constraints is the one that maximizes the system's entropy function. The maximization of the system's entropy function is carried out using Lagrange's method of undetermined multipliers. However, in a stochastic system, the implementation of a steady-state maximum entropy solution requires estimation of Lagrangian coefficients related to output parameters such as utilization and mean queue length. We make this estimation by

using a related closed network at equilibrium with infinite queue [29]. The details of the ME technique can be found in Appendix A.1 and [29]. Next, we elaborate each step.

1. Modeling the blocking probability (p_b) at the downstream layer

The downstream layer abstracted in Figure 2.3 as Q_{ds}, is viewed as another canonical system containing priority queues (Q_{hds}, Q_{lds}) which is marked as 'Step:1' in Figure 2.4. The probability of blocking at the downstream layer (p_b) is a function of the rate at which each queue is filled with packets. Hence, understanding the dependencies between each queue's operation in the downstream layer is crucial. 'Step:1' in the figure shows the parameters needed to be calculated before finding blocking probability p_b. The variables that blocking probability p_b is dependent on: We denote

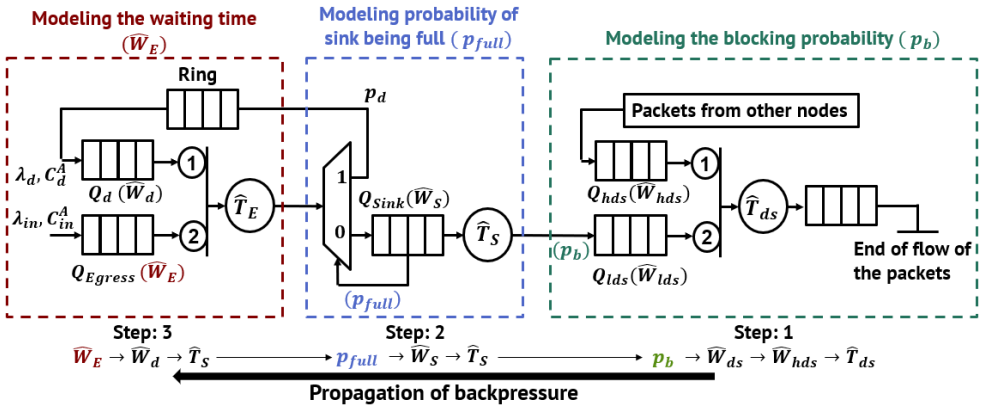

Figure 2.4: Steps involved in constructing the analytical model. The thick black arrow indicates the direction in which the back-pressure propagates.

Table 2.2: List of the symbols used in this work.

$\lambda_{in}, \lambda_{d}, \lambda_{sink}$	Injection rate to Q_{Egress}, Q_{d}, Q_{sink}
$\widehat{T}_{E}$,	Mean service time of Q_{Egress} and Q_{d}
$\widehat{T}_{ds}$, $\widehat{T}_{S}$	Mean service time of Q_{hds} and Q_{lds}, Q_{sink}
p_{full}	Probability of Q_{sink} being full
p_{d}	Probability of deflection of packet from Q_{sink}
p_{b}	Probability of blocking
C_{in}^{A}, C_{d}^{A}	Coeff. of variation of inter-arrival time of Q_{Egress}, Q_{d}
R_{sink}	Residual wait time in Q_{sink}
C_{S}	Coefficient of variation of service time of Q_{sink}
$\widehat{W}_{E}, \widehat{W}_{d}, \widehat{W}_{S}$	Average waiting time of the packets in Q_{Egress}, Q_{d}, Q_{sink}
$\widehat{W}_{hds}, \widehat{W}_{lds}$	Average waiting time of the packets in Q_{hds}, Q_{lds},
ρ_{ds}, ρ_{sink}	Utilisation of the server at Q_{hds} and Q_{lds}, Q_{sink}
$< n >$	Mean queue length of infinite queue
$< n >_{N}$	Mean queue length of finite queue
x,x_{N},y, g	Lagrangian coefficients
N	Queue size
p_{N}	Prob. dist. of occupancy in a queue of size N

the dependency of computing p_{b} in 'Step:1' through the following notation (as seen in the bottom of Figure 2.4):

Step 1: Blocking probability $(p_{b}) \rightarrow$ **Low priority queue waiting time** $(\widehat{W}_{lds}) \rightarrow$ **High priority queue waiting time** $(\widehat{W}_{hds}) \rightarrow$ **Modified service time** $(\widehat{T}_{ds})$**.**

The above notation denotes that computation of p_{b} needs $\widehat{W}_{lds}$; computation of $\widehat{W}_{lds}$ needs $\widehat{W}_{hds}$; $\widehat{W}_{hds}$ needs $\widehat{T}_{ds}$. Please note that we will follow

the same notation for each step for clarity. We begin by finding $\widehat{T}_{ds}$ and compute p_b at the end of 'Step:1'.

a. Obtaining the modified service time ($\widehat{T}_{ds}$) of the downstream layer

To obtain the mean service time ($\widehat{T}_{ds}$) of the egress queues (Q_{lds}) in the downstream layer, we first calculate the probability of no packets in Q_{lds} and in its corresponding server (i.e., $p_{Q_{lds}}(0)$) using maximum entropy (ME) method [29] as,

$$p_{Q_{lds}}(0) = 1 - \rho_{lds} - \rho_{hds}\frac{\bar{n}_{lds}}{\bar{n}_{lds} + \rho_{lds} + \rho_{hds}} \tag{2.1}$$

where ρ_{lds} denotes the utilization of Q_{lds} and ρ_{hds} denotes the utilization of Q_{hds} shown in the Figure 2.4, which is a queue with higher priority in the downstream layer. $\bar{n}_{lds}$ is the occupancy of Q_{lds}.

Using the resulting probability distribution of occupancy ($p_N(n)$) from the ME method, we apply Little's law to compute the first order moment of modified service time ($\widehat{T}_{ds}$) as:

$$\widehat{T}_{ds} = \frac{1 - p_{Q_{lds}}(0)}{\lambda_{lds}} \tag{2.2}$$

Using Equation 2.2, we obtain the waiting time of the packet in the egress queues ($\widehat{W}_{lds}$) in the downstream layer following the technique in [42].

b. Obtaining the probability of blocking (p_b)

As discussed earlier, p_b depends on the rate of packet accumulation in the downstream layer which is estimated using the ME method. However, the ME method is built on prior information of the system which is obtained either through measurements or through estimating the Lagrangian coefficients using an infinite queue system and extending the ME method to finite queues. In order to make the model purely analytical and steer clear of any simulations, we employ the latter technique.

We start the analytical modeling of blocking probability by first assuming infinite queue length to estimate the Lagrangian coefficients in the ME method [29], which provides the probability distribution of occupancy in Q_{ds}. Using Equation 2.2, we obtain the waiting time of the packet in the egress queues ($\widehat{W}_{lds}$) [42] which is used to find the mean infinite queue length $< n >_{ds}$ as follows:

$$< n >_{ds} = \lambda_{ds} \widehat{W}_{lds} + \rho_{ds} \tag{2.3}$$

where $\rho_{ds} = \lambda_{ds} \widehat{T}_{ds}$ and λ_{ds} is the injection rate to the egress queue of the downstream layer. Then, we compute the blocking probability p_b which equals to $p_N(N)$ in Equation A.9, using the technique described in the Appendix.

2. Modeling the probability (p_{full}) that Q_{sink} is full

The probability of blocking (p_b) obtained in 'Step:1' determines how long a packet will stay at the head of the Q_{sink} in 'Step:2' of Figure 2.4. This indicates that the T_S and C_S is a function of p_b. 'Step 2' in Figure 2.4, shows the parameters needed to calculate the probability (p_{full}) that the Q_{sink} is full. We denote the dependency of computing p_{full} in 'Step:2' through the following notation:

Step 2: Probability of sink queue being full (p_{full}) $\rightarrow$ **Waiting time in the sink queue** ($\widehat{W_S}$) $\rightarrow$ **Service time of the sink queue** ($\widehat{T_S}$) $\rightarrow$ **Blocking probability of the downstream network** (p_b)**.**

Therefore, we start by computing $\widehat{T_S}$ using p_b.

a. Obtain the service time ($\widehat{T_S}$) and coefficient of variation of service time of (C_S) of (Q_{sink}) using blocking probability of the downstream network(p_b) To compute C_S, we need to obtain the first order ($\widehat{T_S}$) and second order moment ($\widehat{T_S^2}$) of the service time of Q_{sink}. $\widehat{T_S}$ is obtained by finding the average number of cycles a packet will stay at the head of the Q_{sink} until the packet has been sent to the next layer. This takes into consideration the effect of backpressure applied by the downstream layers on the Q_{sink} which causes the packets to stay longer at the head of the Q_{sink}.

$$\begin{aligned}\widehat{T_S} &= 1 - p_b + 2p_b(1-p_b) + 3p_b^2(1-p_b) + \cdots \\ &= 1 - p_b + \sum_{n=2}^{\infty} n p_b^{(n-1)}(1-p_b)\end{aligned} \tag{2.4}$$

Therefore, $\widehat{T_S}$ can be expressed as:

$$\widehat{T_S} = \frac{1}{1 - p_b} \tag{2.5}$$

Similarly, we obtain $\widehat{T_S^2} = \frac{1+p_b}{(1-p_b)^2}$. Next, using $\widehat{T_S}$ and $\widehat{T_S^2}$, we calculate the coefficient of variation of service time.

$$C_S^2 = \frac{\widehat{T_S^2} - \widehat{T_S}^2}{\widehat{T_S}^2} \tag{2.6}$$

b. Obtaining the waiting time ($\widehat{W_S}$) of sink queue (Q_{sink})

We begin to calculate the waiting time ($\widehat{W_S}$) using the residual time R_{sink}, i.e, Equation 2.8. However, to find the residual time we must first compute ρ_{sink}.

$$\rho_{sink} = \lambda_{sink} \widehat{T_S} \tag{2.7}$$

where λ_{sink} is the injection rate to the sink queue of the current layer. Next, using Equation 2.5, 2.6 and 2.7, we compute the residual time, i.e, Equation 2.8. Using the residual time R_{sink}, we compute the waiting time $\widehat{W_S}$, i.e, Equation 2.9.

$$R_{sink} = \frac{\rho_{sink}(T_S - 1 + T_S C_S^2)}{2} \tag{2.8}$$

$$\widehat{W_S} = \frac{R_{sink}}{1 - \rho_{Sink}} \tag{2.9}$$

c. Obtaining the probability of sink queue being full (p_{full}) Similar to blocking probability, we start constructing the analytical model to find the probability of sink queue being full (p_{full}) by first assuming infinite queue length. Using equation 2.8 and 2.9, we find the mean infinite queue length $< n >_{sink}$:

$$< n >_{sink} = \lambda \widehat{W}_S + \rho_{sink} \tag{2.10}$$

Then, we find the Lagrangian coefficient x by using the mean infinite queue length $< n >$ from Equation 2.10, in Equation 2.11 [30].

$$x = \frac{< n >_{sink} - \rho_{sink}}{< n >_{sink}} \tag{2.11}$$

Using Equation 2.11 and following the complete derivation given in Appendix A, we get the probability distribution of queue occupancy as:

$$p_N(n) = \begin{cases} C_N \widehat{p}(n), & \text{for } n = 0, 1, \ldots, N-1 \\ 1 - \sum_{n=0}^{N-1} p_N(n), & \text{for } n = N \end{cases} \tag{2.12}$$

From the resulting probability distribution given by Equation 2.12, we acquire the probability of the sink queue being full p_{full} as $p_N(N)$, where N represents the queue size.

Algorithm 1: Iterative approach to obtain waiting time (W_E) of the packets in the egress queue.

Input: Deflection probability (p_d), Injection rate to the egress buffer (λ_{in}), Buffer size (N)
Output: Waiting time of the packets in the egress queue (W_E)
Initialization: $e = ke_t, k > 1; \widehat{\lambda}_{in} = \lambda_{in}$
while $e \geqslant e_t$ **do**
    Obtain the modified service time ($\widehat{T}_E$) of the egress queue using p_d and λ_{in} following [42].
    Obtain the probability that the egress queues are full (p^E_{full}) using λ_{in}, N following Equations A.2 - A.10 in Appendix A
    $\lambda_{temp} \leftarrow \widehat{\lambda}_{in}$
    $\widehat{\lambda}_{in} \leftarrow \widehat{\lambda}_{in}(1 - p^E_{full})$
    $e = \frac{|\lambda_{temp} - \widehat{\lambda}_{in}|}{\widehat{\lambda}_{in}}$
end
Obtain W_E from $\widehat{\lambda}_{in}$, $\widehat{T}_E$ [42].

3. Modeling the waiting time ($\widehat{W}_E$) of the packets in (Q_{Egress})

We denote the dependency of computing $\widehat{W}_E$ in 'Step:3' through the following notation:

Step 3: Waiting time of egress queue ($\widehat{W}_E$) $\rightarrow$ **High priority queue Waiting time** ($\widehat{W}_d$) $\rightarrow$ **Service time in the upstream network**($\widehat{T}_E$) $\rightarrow$ **Probability of sink queue being full** (p_{full})**.**

The p_{full} obtained in the Step:2 is used as probability of deflection (p_d) for the upstream layer. State-of-the-art analytical performance analysis technique consider layers with deflection routing [42] assuming infinite queues. In this work, we developed an iterative approach to model layers

with deflection routing while considering finite queues. Algorithm 1 shows the iterative approach. The algorithm takes deflection probability (p_d), injection rate to the egress buffer (λ_{in}) and the egress buffer size (N) as the input. The output of the algorithm is the waiting time ($\widehat{W_E}$) of the packets in the egress queue. We first obtain the modified service time of the egress queue using the technique described in [42]. Then, we compute the probability (p^E_{full}) that the egress queue is full using the technique described in Appendix A. We modify the injection rate using p^E_{full} as shown in line 8 of Algorithm 1. Then the relative error (e) between the modified injection obtained in the previous iteration and the current iteration is computed. The iteration is stopped when e is less than a pre-defined threshold (e_t). Finally, we compute $\widehat{W}_E$ using the modified injection rate ($\widehat{\lambda}_{in}$) and the modified service time ($\widehat{T}_E$).

End-to-End Analytical Model

Algorithm 2 describes the technique to obtain end-to-end latency for multi-layer NoCs. The input to the algorithm is NoC size, injection process of each traffic class and the microarchitectural details of the NoCs. The microarchitectural details of the NoCs include the service process of each queue and size of the queues. The output to the algorithm is the average packet latency ($\bar{L}$). In a mesh NoC with r rows and c columns, there are total $N_s = r \times c$ sinks as initialized in line 3 of the algorithm. Next, for each sink-i we compute the probability of blocking (p^i_b) and the proba-

bility of full (p^i_{full}) following the techniques described in Section 2.3 and Section 2.3 respectively. After that, we obtain the probability of deflection at sink-k (p^k_d) as shown in line 12 of Algorithm 2. Then, we obtain the waiting time for each source-j to sink-k (W_{jk}) using Algorithm 1. We add the zero-load-latency (z_{jk}) to W_{jk}. A cumulative latency (L) and cumulative injection rate (Λ) are computed for the latency (L_{jk}) and the injection rate (λ_{jk}) as depicted in line 15 and line 16 of the algorithm. Finally, L is divided by Λ to obtain the average packet latency ($\bar{L}$).

2.4 Experimental Evaluation

Experimental Setup

In this section, we evaluate the accuracy of the proposed analytical model by comparing it to a cycle-accurate industrial NoC simulator under various traffic scenarios. The simulator includes various key features of state-of-the-art industrial processors [15]. For example the simulator supports cache coherency protocol, multiple layers of NoCs and considers finite buffers. Specifically, the simulator models request NoC, data NoC and acknowledgement NoC separately which is representative of widely used industrial processors [15]. Existing cycle-accurate NoC simulators do not consist of these features [23]. The experiment scenarios include real applications and synthetic traffic that allow evaluations with varying injection

Algorithm 2: End-to-end latency of multiple NoC with finite buffers

Input: NoC size $(r \times c)$, Injection process of each class (λ, C_a), service process (T, C_s), buffer sizes (N)
Output: Average packet latency $(\bar{L})$
Initialization: $N_s = r \times c$, z_{jk} = zero load latency from source j to destination k, $L = 0, \Lambda = 0$
/* Obtain p_{full} at each sink*/
for $i = 1 : N_s$ **do**
    Obtain p_b^i through the technique described in Section 2.3
    Obtain p_{full}^i using p_b^i through the technique described in Section 2.3
end
/* Obtain waiting time in egress queue for each source-destination pair */
for $j = 1 : N_s$ **do**
    for $k = 1 : N_s$ **do**
        Obtain p_d^k using p_{full}^k.
        Obtain W_{jk} using p_d^k through Algorithm 1.
        $L_{jk} = W_{jk} + z_{jk}$
        $L = L + L_{jk}\lambda_{jk}$
        $\Lambda = \Lambda + \lambda_{jk}$
    end
end
$\bar{L} = \frac{L}{\Lambda}$

rates, hit rates (p_h), and probability of burstiness (p_{br}). The evaluations are performed on a 6×6 mesh configuration, which is representative of high-end server CPUs [62]. The traffic sources emulate high-end CPU cores. All the buffers are finite and the sizes are set based on the industrial NoC simulator buffer size. The list of requests in the LLC controller holds 40 requests, while the maximum number of outstanding requests is 16. All

cycle-accurate simulations run for 600K cycles for the experiments with synthetic traffic, with a warm-up period of 100K, allowing the NoC to reach the steady state.

Speed-up achieved with respect to cycle accurate simulation: The proposed analysis technique is implemented in C++ to perform a fair comparison in execution time with respect to the cycle-accurate simulator. Our proposed analysis technique achieves four orders of magnitude improvement in execution time compared to simulation for 6×6 NoC. Since multiple NoCs with finite buffers are being analyzed with an iterative approach in this work, the execution time of the proposed analytical model is slightly higher (less than 20%) than the state-of-the-art analytical model [42].

Evaluations under Geometric Traffic

This section evaluates the proposed model using input traffic distribution as geometric at the egress queue of the request network. The inputs to the rest of the NoCs are driven by the LLC controllers serving these requests.

Table 2.3: Summary of results with different probability of burstiness (p_{br}), LLC hit rate (p_h) and injection rate (λ).

p_{br}	0									0.1									0.3								
p_h	1			0.5			0.25			1			0.5			0.25			1			0.5			0.25		
λ	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H
Er- [42](%)	5.0	6.3	74	3.3	10	65	2.8	56	50	4.9	6.4	77	3.6	1.0	66	3.1	56	56.3	4.3	2.9	80	4.0	1.0	71	3.5	58	50.4
Er-ours(%)	2.7	1.9	1.7	3.4	4.9	7.8	2.9	9.6	10	2.7	1.4	8.8	3.6	4.8	8.3	3.2	3.3	7.6	8.1	11	13	4.0	6.7	10	3.6	8.1	11

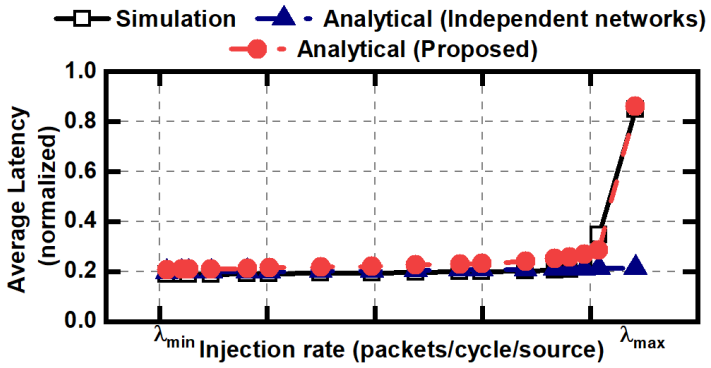

Figure 2.5: Comparison of average latency of packets between cycle-accurate simulation and performance analysis considering the interactions between different NoCs (proposed) and treating them independently [42], for varied injection rate in a 6×6 NoC in which the **probability of burst is 0** and the hit rate is 1.

Figure 2.5 shows the average latency of packets in the request network with a 100% LLC hit rate as the request injection rates scale from a light load until the request NoC saturates. The latency estimated with the proposed technique (● markers) matches well with cycle-accurate simulations (■ markers). It achieves less than 10% mean absolute percentage error (MAPE) compared to simulation, as summarized in Table 2.3. In contrast, prior work [42], which does not model the dependencies between different NoCs (▲ markers), fails to produce accurate latency at high traffic intensities. Its error under high traffic load explodes, leading to 30% MAPE on average.

In addition to the representative results in Figure 2.5, Table 2.3 lists a more comprehensive set of results for geometric traffic (burst probability $p_{br} = 0$) for LLC hits probabilities (p_h) of 1, 0.5, and 0.25 with 3

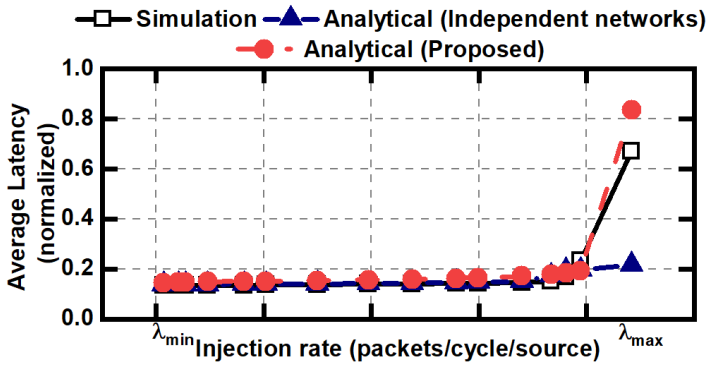

Figure 2.6: Comparison of average latency of packets between cycle-accurate simulation and performance analysis considering the interactions between different NoCs (proposed) and treating them independently [42], for varied injection rate in a 6×6 NoC in which the **probability of burst is 0.1** and the hit rate is 1.

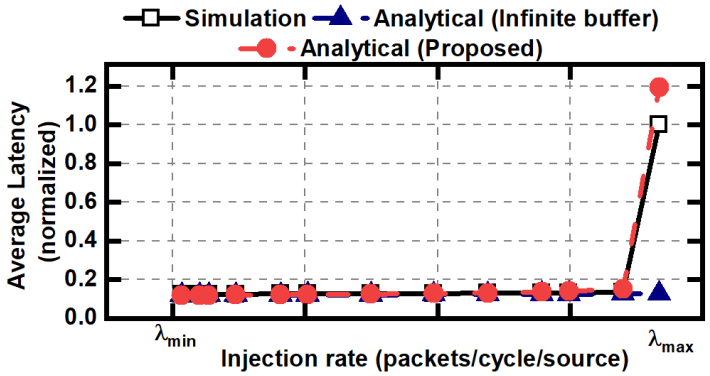

Figure 2.7: Comparison of analytical models without considering deflection at the sink.

different sets of injection rate. The table compares the error between [42], which does not model the dependencies between different NoCs and the proposed approach. The proposed approach which considers the inter-dependencies between the NoCs out performs the work [42].

Evaluations under Bursty Traffic

This section evaluates the proposed model under bursty input distribution at the egress queue of the request network. Figure 2.6 shows the average latency of packets in the request network under varying injection rates, with a hit rate of 1 and a burst probability of 0.1, which is representative of real application traces. Similar to the previous section, it compares our analytical results (● markers) against cycle-accurate simulations (markers) and prior work [42] (▲ markers) that models each NoC independently, i.e., without explicitly modeling their interactions. The proposed performance analysis technique achieves less than 10% modeling error compared to cycle-accurate simulations. In contrast, neglecting the interactions between different NoCs cause the error to explode during heavy load, similar to the geometric traffic results. As a result, the average MAPE grows to 30%.

Table 2.3 lists a more comprehensive set of results for bursty traffic ($p_{br} = 0.1$ and $p_{br} = 0.3$) for LLC hits probabilities (p_h) of 1, 0.5, 0.25 and 3 different sets of injection rate. The table compares the error between [42], which does not model the dependencies between different NoCs and the proposed approach. The proposed approach which considers the inter-dependencies between the NoCs consistently outperforms the work [42] in medium and high injection rates.

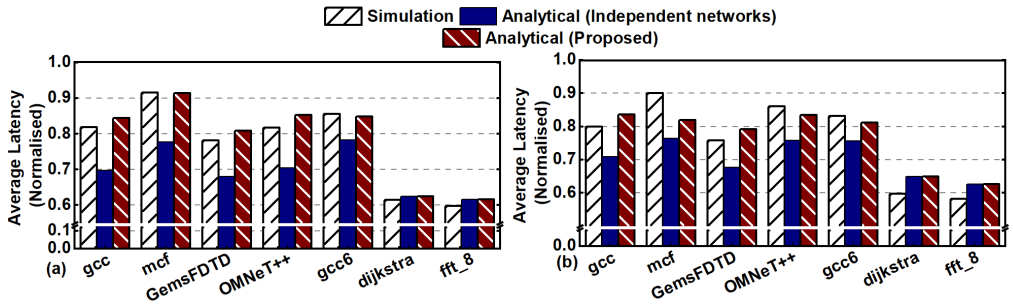

Figure 2.8: Comparison of average latency of packets between cycle-accurate simulation and performance analysis considering the interactions between different NoCs (proposed) for different real applications on (a) 6×6 and (b) 8×8 NoC.

Results with Real Applications

In addition to the synthetic traffic, the proposed analytical model is evaluated with real applications too. Real applications include SPEC CPU@2006 [18], SPEC CPU@2017 benchmark suites [11], and the SYSmark®2014 application [4]. Specifically, the evaluation includes SYSmark14, gcc, bwaves, mcf, GemsFDTD, OMNeT++, Xalan, and perlbench applications. These applications include a variety of different injection rate and different probability of burstiness.

Obtaining p_b of the real applications: We run the applications on gem5 [9] and collect traces with timestamps for each packet injection. Specifically, the traces consist of time of generation, source and destination of each packet to be injected in the NoC. Then, we use the traces to compute the injection rate (λ) and p_b. For each source, we feed traffic arrivals with timestamps over a 100K clock cycle window into a virtual queue with the

same service rate as the NoC to determine the queue occupancy. At the end of the window, we compute the average occupancy. Then, we employ the model described in [31] to find the occupancy and then p_b of each class.

The benchmark applications are executed on 6×6 and 8×8 mesh architectures. The 8×8 mesh-NoC experiments consists of 2 memory controllers and 62 co-located agents. The 6×6 mesh-NoC consists of 2 memory controllers and 34 co-located agents. Each co-located agent consists of a processing core and a cache similar to state-of-the-art server-scale computers [15]. In our experiments, we assume that each core is executing one copy of the real application and requesting data to each cache with uniform probability and 100% LLC hit traffic.

The comparison results are shown in Figure 2.8(a) and Figure 2.8(b). The state-of-the-art analytical model which does not consider dependencies between multiple layers of NoCs is inaccurate when the applications exhibit high injection rate and burstiness. The injection rate for Dijkstra and fft_8 applications are low. Hence, the analytical model considering independent networks is accurate enough for those applications. In contrast, the proposed model follows the simulation results very closely for all real applications. On average, the proposed analytical model achieves less than 10% modeling error. Therefore, our proposed analytical model considering the inter-dependencies between the NoCs are accurate both for synthetic and real applications.

Results without Considering Deflection in the Network

We evaluate our proposed analytical model for a 6×6 NoC without considering deflection at the sink. In this case we set the size of the ingress buffer to a high value (1000) so that the packets do not get deflected from the sink. However, we keep the sizes of egress and transgress buffers finite. Figure 2.7 shows the comparison in average latency between simulation, analytical model which does not consider finite buffer (and deflection routing) [41] and our proposed analytical model. At high injection rate, due to finite size of egress and transgress buffer, the NoC experiences backpressure. The backpressure increases congestion in the NoC which in turn increases the latency. We observe that the analytical model which considers infinite buffers results in more than 90% in latency estimation at λ_{max}. In contrast our proposed analytical model incurs only 9% error at the highest injection rate since we consider finite buffers in our proposed approach.

2.5 Conclusion

Modern processors consist of multiple physical networks which utilise finite buffers. The accurate performance evaluation of these Network-on-Chip (NoC) architectures is essential for efficient design space exploration, rapid system simulations, and optimization of architectural parameters. However, current state-of-the-art performance analysis models for NoCs

fails to capture the interactions between the networks. This dissertation introduces a novel performance analysis technique for multi-layer priority-aware NoCs with deflection routing with different traffic, hit rate and probability of burst scenarios. Experimental evaluations using industrial NoCs reveal that the proposed technique surpasses existing analytical models in accurately assessing real-world application workloads and synthetic traffic workloads spanning various scenarios.

3 A LIGHTWEIGHT CONGESTION CONTROL TECHNIQUE FOR NOCS WITH DEFLECTION ROUTING

3.1 Overview

Multi-core processors in systems-on-chip (SoCs) utilize networks-on-chip (NoCs) for efficient communication between processing elements, comprising routers, links, and queues. Buffered NoCs, employing wormhole routing, store flits in intermediate routers, while bufferless NoCs, common in industrial processors, store packets only at endpoints. Under heavy traffic, finite-sized queues apply backpressure, triggering backpressure mechanisms in NoCs to prevent packet losses. Buffered NoCs stall and propagate backpressure upstream, while bufferless NoCs deflect packets to available output ports [15].

To address congestion, researchers propose congestion control mechanisms for industrial NoCs, monitoring sink queue occupancy [69]. If it exceeds a threshold, a distress signal halts packet transmission until receiving a distress-off signal after queue occupancy drops. However, this reduces throughput under heavier workloads and lacks fairness for requests experiencing a miss in the shared last-level cache (LLC). Static techniques use predetermined thresholds, leading to unfairness with varying workloads. Reactive techniques act post-congestion, failing to maintain NoC throughput under heavy traffic.

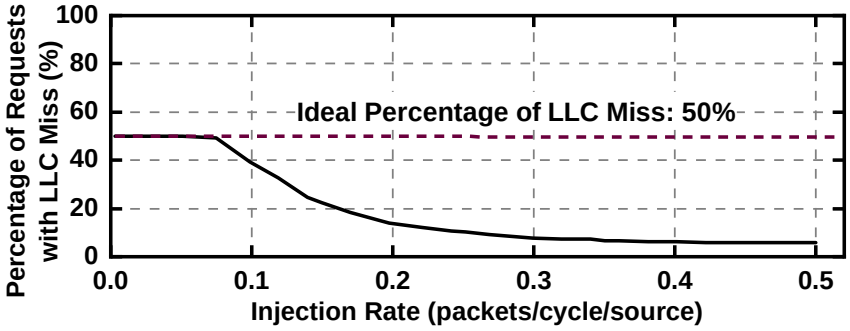

Figure 3.1: Percentage of miss packets with state-of-the-art congestion control technique. The traffic is generated with 50% LLC miss rate (shown in red dashed line). However, the percentage of miss packets decreases to 7% at the highest injection rate which is extremely unfair to miss traffic.

This approach aims to alleviate NoC congestion and enhance fairness, particularly for requests experiencing a miss in the shared last-level cache (LLC). For example, Figure 3.1 shows the percentage of traffic with LLC miss as a function of the traffic injection rate. The percentage of requests with LLC miss denotes the proportion of the requests fetched from the memory controller. In this experiment, we generate synthetic traffic that will result in 50% (the dotted line on the figure) LLC miss rate. At low traffic loads, the observed percentage of LLC miss rate is 50%, as expected. However, the percentage of completed transactions with LLC misses drops as the traffic intensity increases and becomes as low as 7% when the network becomes heavily congested. Similar unfairness happens also at other LLC miss rates. Existing congestion control mechanisms reduce congestion but lower throughput under heavy workloads and exhibit unfairness towards LLC miss traffic. To address this, our work pursues two objectives: maximize and sustain memory read/write bandwidth to

maintain core performance and enhance fairness between LLC hit and miss traffic.

We propose a proactive congestion control technique, employing supervised learning and a lightweight decision tree. The learning framework, based on a novel design of experiments and time reversal techniques, predicts congestion at sink nodes before queues are blocked. At runtime, the decision tree informs traffic source control, preventing new requests to likely congested sinks until the congestion signal is cleared. Experimental results with synthetic and realistic traces demonstrate a significant increase in memory read bandwidth (up to 114%) and a reduction in missed traffic (up to 3.1×) compared to a state-of-the-art congestion control technique.

The major contributions of the work are as follows:

- A novel time reversal approach and supervised learning to construct a decision tree for NoC congestion control,
- End-to-end congestion control algorithm for industrial NoCs,
- Thorough experimental evaluations showing up to 114% higher memory read bandwidth than a state-of-the-art technique with less than 0.01% of overhead.

3.2 Related Work

Existing NoC congestion control techniques can be broadly classified as 1) Global and 2) Local. The global congestion control techniques assess the congestion status of the whole network. Depending on the congestion status, the packet injection rates of all the sources are regulated [21, 44, 66]. In contrast, local congestion control techniques monitor the congestion at each node [1, 22, 69, 77]. State-of-the-art industrial NoCs monitor the ingress queue sizes of each node [22, 69]. If the size exceeds a certain threshold, then the injection of packets from all the sources is stopped. The packet injection resumes when the occupancies of all the queues drop below another predetermined threshold. With this technique, congestion at any of the ingress queues leads to throttling at all sources, leading to conservative behavior. Authors in [77] propose a fine-grained source throttling method for NoCs with mesh topology. In this work, the routers which are most affected by congestion are identified. Then, these routers are used to estimate the NoC congestion status. A heterogeneous congestion criterion for 2D mesh-NoC is proposed in [1]. When NoC congestion occurs in a node, the packets whose trajectory is through the congested node are stalled in the source. The authors in [52] present an an interval-based congestion control algorithm. But, the source is throttled after the congestion is set in the network which wastes useful bandwidth affecting performance. However, all the congestion control techniques described

above are reactive i.e., the congestion criteria kicks in only after the congestion physically occurs in the NoC.

Proactive congestion control techniques for NoCs are proposed in [54, 72]. The technique proposed in [54] estimates the availability of the neighboring router through analytical expression. When a traffic source observes that the input port connected to it does not have availability then it does not send the packets. Authors in [72] propose an artificial neural network (ANN)-based global admission controller for NoC. In this work, the admission controller slows down the injection rate from the sources by factor determined by the ANN. The aforementioned techniques are applicable to an NoC where the packets can wait at each router on its path. However, industrial NoCs are priority aware and incorporates deflection routing where the packets already injected in the NoC can never stop. Therefore, existing proactive congestion controls techniques are not applicable to industrial NoCs.

In contrast, we propose a proactive congestion control technique to increase memory read/write bandwidth and to improve the fairness between LLC hit and miss traffic for industrial NoCs with deflection routing. To the best of our knowledge, this is the first proactive congestion control technique proposed for industrial NoCs with deflection routing.

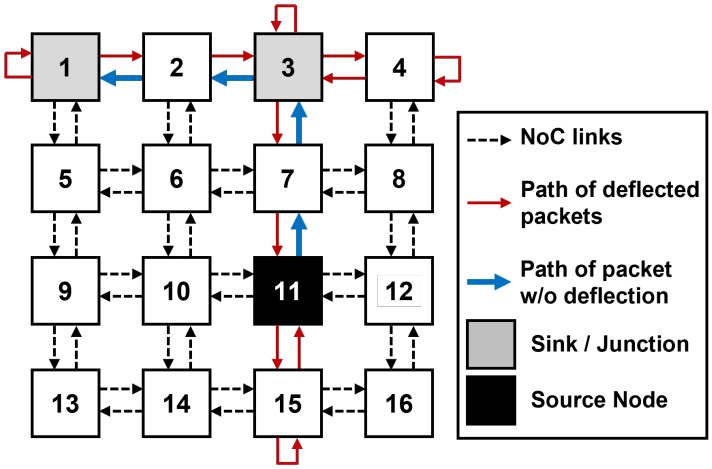

Figure 3.2: A representative 4×4 mesh-NoC with deflection routing.

System Architecture and Background

NoCs with Deflection Routing

This work targets NoCs used in high-end servers and state-of-the-art many core architectures [22]. Figure 3.2 shows a 4×4 mesh NoC architecture, each column of which is also used in client systems, such as Intel i7 processors [62]. Hence, the proposed congestion control technique is applicable to a wide range of priority-aware industrial NoCs, where the packets already in the network have higher priority than the packets waiting in the egress queues of the sources. Assume that Node 11 in Figure 3.2 sends a packet to Node 1 following Y-X routing (highlighted by thick blue arrows). Deflection in priority-aware NoCs happens when the queue at the turning point (Node 3) or final destination (Node 1) becomes full. This can happen if the receiving node, such as a cache controller, cannot process the packets fast enough. The probability of observing a full queue increases

with smaller queues (needed to save area) and heavy traffic load from the cores. If the packet is deflected at the destination node, it circulates within the same row (the red thin arrows), as shown in Figure 3.2. Consequently, a combination of regular and deflected traffic can load the corresponding row and pressure the queue at the turning point (Node 3). This, in turn, can lead to deflection on the column which propagates the congestion towards the source wasting useful NoC bandwidth.

Background on Cache Coherency Flow

This work assumes a local L1/L2 cache at each node, a distributed LLC, and non-inclusive MESI-like cache-coherency flow [56]. If a request from a core is not present in L1 or L2 cache, the request is sent to LLC. If the request is present in the LLC, then the corresponding data is returned from LLC to the requesting core. If the request is not present in the LLC then the request is forwarded to the memory controller. The corresponding data is fetched from the memory controller and returned to the core. The proposed congestion control technique is independent of the number of cores, LLC banks and on-chip memory controllers.

3.3 ML-Based Proactive Source Throttling

Overview of the Approach

When packets in the NoC are deflected at the sink, they continue to use the NoC bandwidth and aggravate congestion. Hence, the proposed runtime technique works as follows:

1. It monitors the congestion indicators, i.e., the features of our machine learning (ML) model, at each sink queue and determines whether they are likely to be blocked,

2. If a given queue is likely to be blocked, it sets a congestion signal at that sink. Otherwise, it clears the congestion signal.

3. The sources check the congestion signal at the destination before sending a new request. If the congestion signal is set, they throttle the corresponding request and move on to the next request. The requests to the sinks with congestion signal are delayed until the congestion signal is cleared.

We note that checking the congestion signal at the destination does not incur any additional overhead compared to existing techniques [15, 69], since they also incorporate similar mechanism. They also have a small (~ 10 cycles in our case) deterministic delay when the distress information is carried by a simple dedicated time-division-multiplexed channel.

The fundamental question is to determine when to throttle a source. Since the network traffic dynamics are fast, time-dependent, and nonlinear, we need to consider not only the current queue occupancies but also first and second order factors that can lead to congestion. For example, consider an ingress queue with depth 32. An occupancy of 16 packets may be safe if the current input traffic rate is lower than the service rate and the level of burstiness is low. Since the average occupancy is likely to be decreasing as time progresses, the sources do not need to be throttled. In contrast, an occupancy of 16 packets may be dangerous if the average occupancy is increasing. Therefore, a holistic approach must consider all relevant features summarized in Section 3.3. Moreover, determining the optimal criteria is non-trivial even when all the features are available. Hence, the second component of the proposed approach is to design a decision tree using an innovative data collection and labelling technique presented in Section 3.3. Finally, the last step is implementing the lightweight controller that uses the congestion signals and local criteria to throttle the source (Section 11).

Features used for Supervised Learning

To construct the machine learning-based model, we first collect the dataset required for training. The dataset consists of features ($\mathcal{F}$) listed in Table 3.1 with corresponding labels ($\mathcal{L}$). Here $\mathcal{F} = (f^1, f^2, ..., f^N)$, where N is the number of features, $f^j \in \mathbb{R}$, $1 \leqslant j \leqslant N$ and $\mathcal{L} \in \{0, 1\}$. The features ($\mathcal{F}$) are

sampled every time a packet arrives at the ingress queue at the sink. If the queue is not full, the packet is written to the queue. Otherwise, the packet bounces. Sampling the features in both conditions (sink or bounce) enables us to monitor congestion accurately at sink node.

To capture the features accurately, we compute exponentially weighted moving average (EWMA) of each feature as:

$$\vec{f}_i^j = \alpha f_i^j + (1-\alpha)\vec{f}_{i-1}^j,\ i > 0,\ 1 \leqslant j \leqslant N \tag{3.1}$$

In this equation, $\vec{f}_i^j$ denotes EWMA of the feature f^j for i^{th} packet, f_i^j denotes the original value of the feature f^j (e.g. injection rate) and α is the degree of mixing parameter ($0 \leqslant \alpha \leqslant 1$). The value of α is tuned to track the average accurate without a significant delay. The feature values are smoothened over time by computing EWMA. We implemented EWMA computation in a cycle-accurate industrial simulator. A five point derivative is computed for the features involving gradient. We track all the features in Table 3.1, which are potentially useful for congestion control. Since data collection is an offline process, the EWMA computation overheads are inconsequential. After deploying the machine learning model, EWMA of only the selected features are tracked at runtime.

Table 3.1: List of features collected at each sink.

Injection rate to the sink queue	Total injection rate (sunk + deflected)
Co-eff. of variation of the total traffic (sunk + deflected)	Co-eff. of variation of inter-arrival time of the traffic to the sink queue
Rate of deflected packets	Mean service time of the sink queue
Co-eff. of variation of deflected packet inter-arrival time	Co-eff. of variation of sink queue inter-departure time
Occupancy	Probability that the sink queue is full
Gradient of injection rate	Gradient of queue occupancy
Gradient of total (sunk + deflected) injection rate	Gradient of probability of sink being full

Training Data Collection and Decision Tree

Labeling the features: The collected features indicate the ingress queue and NoC congestion state at sampling time. For example, the features will capture if a sink queue is full and deflects a packet. However, one must throttle the source before the queue becomes full, i.e., before the onset of congestion. The main challenge is to know that a packet will bounce before it is even injected into the network. Having this knowledge at runtime is impossible, but we mitigate this challenge using a ***novel time reversal approach*** described next. The generation time stamps of all the deflected packets are recorded while sampling the features. If a packet is deflected at the sink, we know that the source must have been throttled at the generation time of this packet. This sense of time in our comprehensive simulation data enables us to go back to the generation time of the deflected packet and label the collected features around that time accordingly.

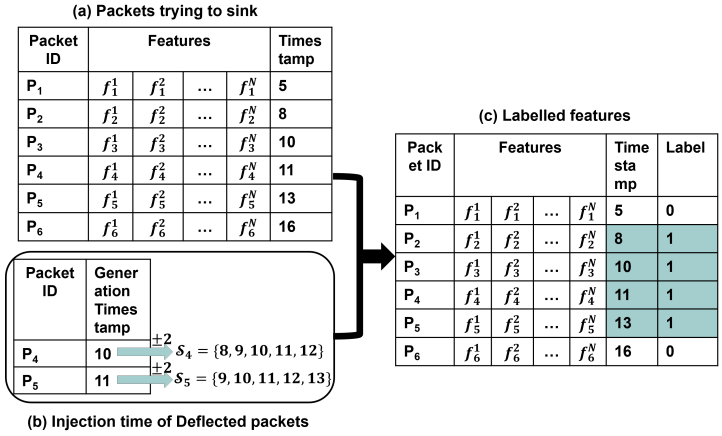

Figure 3.3: An illustrative example of our proposed time reversal approach to label the features.

Figure 3.3 shows an illustrative example of our proposed time reversal approach for labelling the features. Figure 3.3(a) shows the features sampled for six packets arriving at the ingress queue. Along with the features, the timestamps when the packets attempted to sink are also sampled (last column of the table). Apart from sampling the features of the packets arriving at the ingress queue, we also sample the generation timestamps of the deflected packets. As shown in the Figure 3.3(b), there are two deflected packets – P_4 and P_5. The generation timestamps (d_j in Equation 3.2) when they were injected from the source are 10 and 11 respectively. Next, we compute a set of timestamps for each deflected packets (two in this case) which are within 2 cycles of the injection timestamps. From P_4, we get $\mathcal{S}_4 = \{8, 9, 10, 11, 12\}$ and from P_5, we get $\mathcal{S}_5 = \{9, 10, 11, 12, 13\}$. If t_i is the timestamp of the packet P_i, where $1 \leqslant i \leqslant 5$, then we label P_i as 1 if

$t_i \in \mathcal{S}_4 \cup \mathcal{S}_5$. If d_j is the generation timestamp of when the j^{th} deflected packet and t_i is the timestamp of the i^{th} packet arriving at the sink, in general we label (l_i) the features of the i^{th} packet arriving at the sink as:

$$l_i = \begin{cases} 1, & \text{if } (d_j - \Delta) \leqslant t_i \leqslant (d_j + \Delta) \\ 0, & \text{Otherwise} \end{cases} \tag{3.2}$$

where $\Delta = 2$ in this example. A label of 0 denotes that if the source sends packet to that particular sink, then it *will not result in congestion*. A label of 1 denotes that if the source sends packet to that particular sink, then it *will result in congestion*. Therefore, all the features with timestamp within the range of Δ ($\Delta > 0$) of d_i are labelled as 1. In other words, features within a range of Δ timestamps from the same timestamp as the generation timestamp of the deflected packets are labelled as 1. The features of the packets with label of 1 are highlighted in Figure 3.3(c).

Supervised Learning: We can employ any supervised learning algorithm to create a model which can take congestion control decision. In this work, we choose binary decision tree since decision tree incurs low hardware overhead (detailed in Section 3.4). The output of the decision tree is either 0 or 1. An output of 0 denotes that cores can send packet without congesting the NoC. An output of 1 denotes that there is a possibility of congestion in the near future and cores should stop injecting packets in the NoC. We observe that the decision tree obtained through supervised

Algorithm 3: End-to-end congestion control algorithm

Input: Absolute value of the features, mixing parameter (α), size of the sink queue (N), target occupancy (N_T)
Output: To throttle (1) or not to throttle the source (0)
$L = LC(N_T, N, \lambda)$
if $L == 1$ **then**
    return 1
end
else
    $\bar{\mathcal{F}} \leftarrow$ EWMA of the features using Equation 3.1
    $D = DT(\bar{\mathcal{F}})$
    return D
end

learning supports our idea of proactive congestion control. For an example, the decision tree returns an output of 1 if both the occupancy of the sink is high and the gradient of injection rate to the sink is positive.

Local Source Control

Each source (e.g., the CPU cores) has controller in the NoC interface. The controller checks the congestion signal from the decision tree at each sink. Due to deflection routing, the sources which are located at the boundary of floorplan (e.g., Node-1, 5, 9, 13 in Figure 3.2) have highest priority. Therefore, packets sent from these sources do not compete with the packets waiting at other sources with lower priority. Hence, the sources with highest priority can inject freely and cause congestion. In addition to the sink nodes, a local condition at sources with highest priority can also proactively hint future congestion. Therefore, we also implement

a local condition for the sources with the highest priority. Let N be the current occupancy of the destination sink, and N_T be the target occupancy. According to Little's law, $\lambda \times t_{avg}$ more packet can be written to the queue, where λ is the injection rate to the ingress and t_{avg} is the average time between two source throttling decisions [36]. Hence, the traffic source is throttled if $N + \lambda t_{avg} > N_T$, i.e., the queue can become full.

Algorithm 3 shows the end-to-end algorithm for congestion control which combines the decision from decision tree model and the local condition. The input to the algorithm is the absolute value of the features, mixing parameter (α), occupancy of the sink queue (N), and target occupancy (N_T). First, the controller checks the local condition (LC). LC always returns false for the sources with lower priority. If the local condition's output (L) is true, then the algorithm returns true. Otherwise, EWMA of the features $(\bar{\mathcal{F}})$ are computed following Equation 3.1. Then, the algorithm returns the output of the decision tree (D).

3.4 Experimental Evaluations

Experimental Setup

We use a cycle-accurate industrial NoC simulator to evaluate our proposed approach on a 6×6 mesh NoC with two memory controllers. The NoC architecture is similar to the one used in recent industrial SoCs [15, 69]. Due to classified nature of the simulator and architecture, we present

normalized values. Each simulation is run for 600k cycles (with a warm-up period of 100k) to reach steady-state values. The experiments consider a non-inclusive MESI-like cache-coherency protocol [56] with varying traffic and last-level cache (LLC) hit rates.

Accuracy of Decision Tree

We perform simulations for LLC hit rate of 0.5 with different injection rates. The smoothing parameter (α in Equation 3.1) is set as $\frac{1}{16}$ and Δ (in Equation 3.2) is 5. The entire dataset is divided into 70% training data and 30% validation. Table 3.2 shows the accuracy of predicting label-0 and label-1 for the validation data with decision trees having different depths. As a reminder, label-0 denotes that if the sources inject packets, it will not lead to congestion and vice-versa. Therefore, if the labeled feature is 0 and the predicted label is 1, the decision tree will unnecessarily stop the cores from injecting packets. This scenario might be okay since it will not lead to congestion. However, if the labeled feature is 1 and the predicted label is 0, the core will still inject packets when it should have stopped. This misprediction will lead to congestion in NoC. Therefore, the accuracy of predicting label-1 is more important than the accuracy of predicting label-0. We observe that the decision tree with depth 4 has the highest accuracy in predicting label-1. The decision tree with a depth lower than 4 has lower prediction accuracy for label-0 and label-1, while a deeper decision tree has a lower accuracy for label-1 due to overfitting. Therefore,

Table 3.2: Accuracy(%) of decision trees with different depths. Decision tree of depth 4 is chosen based on the accuracy.

	Decision tree depth						
	2	3	4	5	6	7	8
Label-0	93.3	93.2	93.7	94.9	96.1	96.2	96.5
Label-1	95.9	97.4	97.6	97.5	96.3	95.6	94.8

we choose the decision tree with depth 4 for evaluation. We note that, the decision tree for each sink is trained offline (once) and the same decision tree is used for congestion control with any incoming workload.

Comparison of Average Transaction Latency

The primary goal of our congestion control technique is to reduce the number of deflected packets so that there is no wastage of NoC bandwidth. We observe that when no congestion control is enabled, the rate of deflected packets increases with increasing injection rate. For example, with an injection rate of 0.27 and LLC hit rate of 0.2, the average rate of deflected packets is 0.08. In this scenario, our proposed congestion control technique sees no deflected packets. A reduced number of deflected packets reduces NoC congestion, so packets experience lower wait time and average latency.

Figure 3.4 shows the comparison of average transaction latency for varying injection rates with LLC hit rate of 70%. The comparison is between the congestion control technique used in state-of-the-art industrial NoC [69] and our proposed approach. The average transaction latency denotes the round trip latency from the generation of a read/write request to its completion. In the state-of-the-art congestion control technique, if

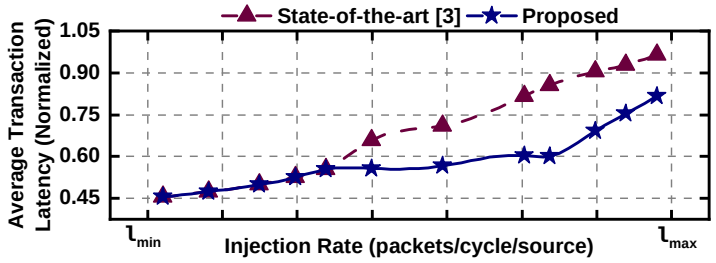

Figure 3.4: Comparison of average transaction latency for 70% hit rate. Lower transaction latency indicates less congestion.

the occupancy of the sink exceeds a predetermined value, the sources are throttled. The sources restart injecting packets in the NoC if the sink occupancy becomes lower than another predetermined value. Therefore, the state-of-the-art control technique is reactive. In contrast, our proposed congestion control technique predicts congestion and provides a proactive decision to throttle the sources. As a result, compared to the reactive state-of-the-art, our proposed proactive congestion control technique throttles at the onset of congestion in NoC, without wasting memory read bandwidth. From figure 3.4 it is observed that the proposed congestion control technique reduces the average transaction latency by up to 30% compared to the state-of-the-art approach. We also observe a similar improvement in average transaction latency for other LLC hit rates. For example, the proposed technique improves the average transaction latency by 7% for an LLC hit rate of 0.2 on average.

Comparison of Percentage of LLC Miss

This section compares the percentage of requests with LLC miss between the state-of-the-art congestion control technique and our proposed technique. The percentage of requests with LLC miss denotes the proportion of the requests fetched from the memory controller. Since our proposed technique reduces NoC congestion, more requests with LLC miss are allowed to be fetched from the memory controller. Therefore, our proposed technique consistently results in a higher percentage of requests with LLC miss, as shown in Figure 3.5 compared to the state-of-the-art congestion control technique. In this case, the synthetic traffic is generated with a 70% hit rate, i.e., ideally, 30% of the traffic should be miss traffic. We observe that at a lower injection rate, both techniques result in 30% of requests with LLC miss since there is no congestion in the NoC. With the increasing injection rate, the percentage of requests with LLC miss reduces. However, our proposed congestion control technique shows up to 3.1$\times$ improvement in the percentage of requests with LLC miss at the higher injection rate. We also observe similar LLC miss percentage improvement for other LLC hit rates. For example, the proposed technique improves the LLC miss percentage by 1.2$\times$ for LLC hit rate of 0.2. Therefore, the proposed technique is fairer towards the requests with LLC miss than the state-of-the-art technique.

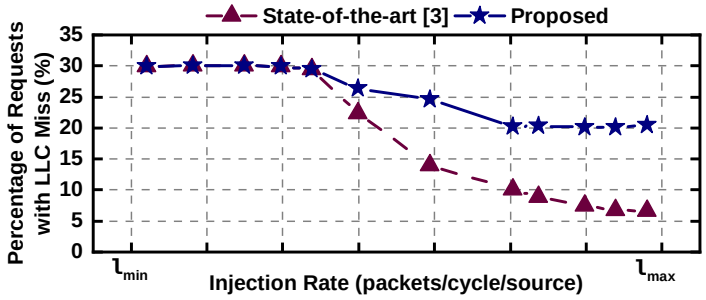

Figure 3.5: Comparison of percentage of LLC miss for 70% LLC hit rate (30% LLC miss). Higher percentage of LLC miss indicates that the congestion control technique is more fair towards the miss traffic.

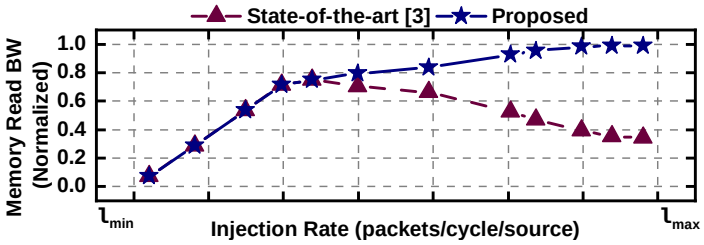

Figure 3.6: Comparison of memory read bandwidth for 20% LLC hit rate. Higher memory read bandwidth indicates less NoC congestion.

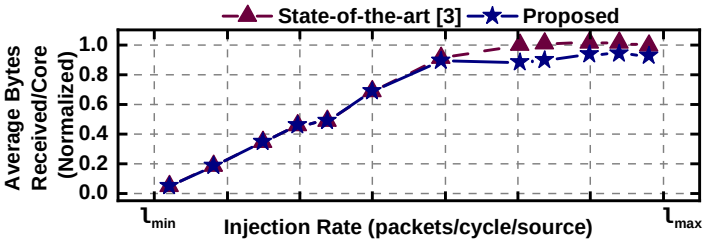

Figure 3.7: Comparison of average bytes received per core for 70% LLC hit rate.

Comparison of Memory Read Bandwidth

This section compares the memory read bandwidth achieved by the state-of-the-art approach and our technique. Memory read bandwidth measures the average number of requests fetched from the memory controller in case of a cache miss. The percentage of missed packets with our proposed congestion control technique also significantly increases the memory read bandwidth. Figure 3.5 shows the 70% LLC hit rate comparison between state-of-the-art and proposed techniques. Both techniques result in equal memory read bandwidth at lower injection rates. However, with increasing injection rate, the memory read bandwidth decreases significantly with a state-of-the-art congestion control technique. Our proposed congestion control technique keeps the memory read bandwidth at a certain level, even at a higher injection rate. The highest improvement seen in memory read bandwidth is 190%. On average, the proposed technique achieves a 64% improvement in memory read bandwidth compared to state-of-the-art methods.

Transactions with an LLC miss take significantly longer than those with an LLC hit due to off-chip memory access. Therefore, the requests with LLC miss stay longer in the queue, reducing the total volume of data received per core (Bytes/core). However, our technique reduces congestion in the NoC and hence total volume of data received per core does not decrease drastically despite substantial increase in memory read

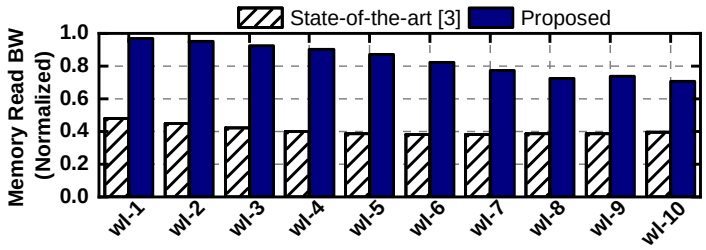

Figure 3.8: Memory read bandwidth comparison with realistic workloads.

bandwidth. Figure 3.7 shows that our proposed approach results in slight (4% on average) reduction in average bytes received per core although it completes 3.1× more transactions with LLC miss.

Results with varying injection rates: So far, we have shown the results when the cores inject with a fixed injection rate for the entire duration. However, real applications may have different phases, and in each phase, cores may inject at different injection rates. Therefore, we also perform experiments with synthetic workloads having different injection rates. Specifically, in each workload, we consider four different injection rates. Figure 3.8 shows the comparison of memory read bandwidth for ten such workloads executing with 70% hit rate. Our proposed congestion control technique achieves up to 114% improvement in memory read bandwidth compared to the state-of-the-art method. On average, the proposed congestion control technique shows a 106% improvement in memory read bandwidth for these realistic workloads.

Hardware Overhead Analysis

We implemented the RTL for the local condition at all sources and the feature computation as well as the decision tree at all ingress of the NoC. Then, we synthesized the RTL using Synopsys Design Compiler with 45 nm technology from TSMC. To have a fair comparison, we scaled the area and power values to 14 nm technology (using the technique described in [65]) since the state-of-the-art SkyLake SoC is fabricated with 14nm [69]. We observe that our proposed congestion control technique consumes only 0.01 mm^2 of area and 2.2 mW of power. The total area of SkyLake SoC is 694 mm^2 and it consumes power in the order of 10W [39, 69]. Hence, our proposed technique incurs negligible overhead (less than 0.01%) both in area and power. Therefore, the technique results in significant reduction of NoC congestion with negligible hardware overhead.

3.5 Conclusion

State-of-the-art NoC congestion control techniques are reactive, i.e., can detect NoC congestion only after it occurs. This dissertation proposes a supervised learning framework along with a time reversal technique to construct a lightweight decision tree. This decision tree proactively determines whether any given sink node will likely experience congestion or not (before the queue is blocked). Experimental evaluation shows that the proposed congestion control technique achieves up to 114% improvement

in memory read bandwidth for realistic workloads while incurring less than 0.01% of overhead.

4 MQL: ML-ASSISTED QUEUING LATENCY ANALYSIS FOR DATA CENTER NETWORKS

4.1 Overview

Data centers play a crucial role in meeting the computational needs of millions of users worldwide, offering shared access to data, applications, and compute resources. With increasing demand, data centers scale out and adopt various topologies like fat-tree. Designing efficient Data Center Networks (DCN) is critical for low latency and high bandwidth while adhering to cost constraints. Packet-level simulators like ns-3 and OMNet++ are commonly used for performance evaluations, providing flexibility but at the expense of slow simulation speeds.

Due to the impracticality of simulation-based design space exploration, analytical approaches have gained traction to estimate network performance. Queuing theory is a notable example, offering analytical models to approximate delay under different input traffic and service time distributions. While effective under matching assumptions, model accuracies decline with deviations from real-life behavior and complex interactions between tandem queues. Accurately knowing precise input traffic and service time distributions remains unrealistic.

Recent works have addressed the limitations of classical queuing theory by exploring deep learning techniques for Data Center Network (DCN)

performance analysis [16, 64, 75, 76]. For instance, MimicNet proposes a hybrid approach combining simulation with deep learning, where a deep neural network (DNN) is trained offline to model the DCN, abstracting clusters beyond the simulated one. Similarly, DeepQueueNet models each device in the network as a DNN, approximating packet delays and producing outgoing streams.

However, these approaches face challenges such as loss of network-level observability, substantial training data requirements, overfitting, and unproven composability of DNNs. In contrast, our proposed approach, ML-assisted queuing latency (MQL) analysis, leverages queuing theory rather than relying on black-box deep learning. This novel approach is built on insights: queuing theory can produce fast, accurate, and scalable models; existing methodologies provide a dataset to identify analytical model shortcomings, and lightweight ML techniques like regression analysis can learn and correct these errors.

Our proposed ML-assisted queuing latency (MQL) methodology leverages the insights mentioned earlier. It begins by developing analytical latency models based on a queuing network discipline suitable for the target scenario, utilizing the maximum entropy (ME) model. Mathematical expressions for each queue in the DCN are derived, and end-to-end flow latencies are determined based on input flows, topology, and routing algorithms. The ns-3 simulator serves as the benchmark for accuracy comparisons. The MQL methodology goes beyond accuracy measurement,

identifying regions where analytical models fall short. Systematic errors arising from network structure, such as tandem queues, are used to extract features highly correlated with underlying analytical model analysis. The final step involves modeling these systemic errors and incorporating them into the analytical estimates. While applicable to any machine learning (ML) technique, this work employs Regression Trees (RT).

In summary, this dissertation makes the following contributions:

- Demonstrates the first ML-assisted queuing theory-based technique that can handle a large scale (>1000 nodes) network of queues - for modern DCN protocols,
- An automated tool that generates an executable performance model (queuing analysis and RT) for a given DCN,
- The ability to provide detailed observability (e.g., individual queuing delay, occupancies, and tier-level visibility) without relying on any communication pattern and topology assumptions,
- Extensive simulation studies with synthetic traffic and network traces that demonstrate less than 3% error on average, 100× to 9000× speed up over ns-3, and scalability to 1024-node fat-tree.

4.2 Related Work

Packet-level simulations, such as ns-3 [50] and OMNet++ [55], provide high accuracy and fine-grain visibility and are versatile, handling different topologies, network protocols, queueing disciplines, and routing algorithms. However, they are too slow to scale to a large number of nodes (1k-10k+ nodes) required for data centers [75, 78]. Therefore, prior work focuses on speeding up network simulation and creating fast network performance models.

Part of the challenge in speeding up network simulators is the almost non-existent opportunities for parallelization [78]. Parsimon [78] observes that large-scale data centers are provisioned such that congestion events rarely occur, and when they do occur, they happen at different points along the path and at different times. Hence, the modeling of the interdependence between queues is a second-order effect. Breaking this dependency enabled the authors to speed up simulation by decomposing the problem into a large number of parallel independent single-link simulations. While their approach handles cases with limited congestion, design space exploration also requires identifying solutions that satisfy highly congested workloads, especially for deep learning workloads.

In addition to simulators, prior work creates fast network performance models via pure analytical, pure ML, and hybrid techniques. A well-established performance analysis approach is queuing-theoretic (QT) esti-

mators. They are fast, but their assumptions, including the Poisson arrival process, FIFO queueing discipline, and queue independence, can lead to unacceptable accuracy in some realistic use cases [7, 10, 29, 30, 32], which we address in this work. QT-based performance analysis appoaches have also been applied to networks-on-chip (NoC). However, these models are typically limited to few 100s of nodes, with 16x16 2D mesh being the largest [26, 40, 53], unlike our approach that scales to over 1000 nodes.

RouteNet-Erlang [16] observes that Graph Neural Networks (GNNs) capture the underlying graph structure of computer networks. It trains on two small networks (10s) of nodes with various traffic communication patterns and queuing disciplines. However, it requires tuning hyperparameters which is empirical. The tunable hyperparameters question scalability of the approach to larger networks. Finally, they do not present DCN performance evaluations with network protocols and congestion control algorithms, which are essential for DCNs.

MimicNet [76] uses deep learning to speed up simulation by combining deep learning with simulation. They simulate one cluster and use a deep neural network-trained model to estimate the remaining clusters. The technique relies on symmetry in both the topology and the traffic (e.g., symmetric bisection bandwidth), thereby limiting its applicability to many real-world traffic patterns and topologies.

DeepQueueNet [75] models each device in the network as a DNN that adds a delay to each packet in the incoming packet stream and pro-

duces an outgoing stream of packets. It composes these DNNs in one-to-one correspondence with the network structure, but there is no formal guarantee that the DNN can be composed to model the whole network. DeepQueueNet targets packet-level visibility, enabled by its detailed simulations.

QT-RouteNet [13] combines queueing theoretic model with a GNN in two steps. First, it runs a simplistic queueing model (M/M/1/B) and extracts features, such as predicted latency, to use in training. Then, it combines with path and link features to train the RouteNet GNN model [64]. Using RouteNet/RouteNet-Erlang at its core, it suffers from similar limitations: tuning the hyperparameters, which is empirical, generalizability, and long traning time. Moreover, it does not present DCN evaluations and learns non-interpretable black-box models, like other purely ML-based approaches.

In summary, existing models suffer from long training times, generalisability, and scalability. In contrast, our MQL approach needs no empirical hyperparameters, making the approach generalizable. Moroever, it leverages the data from simulations that are performed to validate the analytical performance models (as part of the regular design flows). To the best of our knowledge, it is the first approach that provides ML assistance to QT-based performance analysis.

Background and Motivation

Data centers are evolving rapidly with new paradigms and emerging innovations, such as resource disaggregation and low-diameter topologies, to meet the unprecedented growth of data center computing and enable new system-level architectures [70]. Similarly, application disaggregation approaches disintegrate monolithic applications into small microservices or functions with communication overheads [67]. CPU tasks are increasingly offloaded to special-purpose accelerators, often linked over the network, including FGPA resource pooling [80]. In addition, memory and storage are being disaggregated, opening up new system-level architectures to explore [34, 38].

From a DCN perspective, integrated silicon photonics [45] delivers breakthrough performance, enabling high bandwidth and low latency interconnection. It has generated renewed interest and innovations in low-diameter (shorter path-length) topologies [8, 27, 28, 33, 74]. These benefits enable exploring large-scale topologies (>10K end points) [43] and numerous permutations. Hence, there is a critical need for fast performance models to explore the vast design space of new architectures and data center topologies enabled by these innovations.

This work develops fast performance models that generalize to any topology [6]. While exploring new topologies is a crucial use case, we initially demonstrate the proposed technique on fat-tree topologies. Since

fat-trees are widely used in DCN and HPC systems [43, 73], they provide the baseline for new DCNs "to beat." Figure 4.1 shows an example of a three-tier fat-tree topology with 16 compute nodes and twenty k-port switches arranged hierarchically in three layers.

A well-established approach for fast network modeling is queuing theory [6, 16]. Each compute node and switch in the DCN is represented by one or more queues, while the interconnection of compute nodes and switches is modeled as a network of queues (shown in blue), as illustrated in Figure 4.1. The internal structure of switches is modeled via different techniques, such as output-queue (OQ), with queues only on the output ports, and combined input/output-queued (CIOQ) switch with queues on both the input and output ports [12]. OQ switches have a simple scheduling policy which is easier to model, but they are impractical due to the high-speed crossbar. CIOQ switches, on the other hand, are practical to implement but are harder to model due to their more complex scheduling policies. For modeling a switch's performance, [12] shows that the simpler

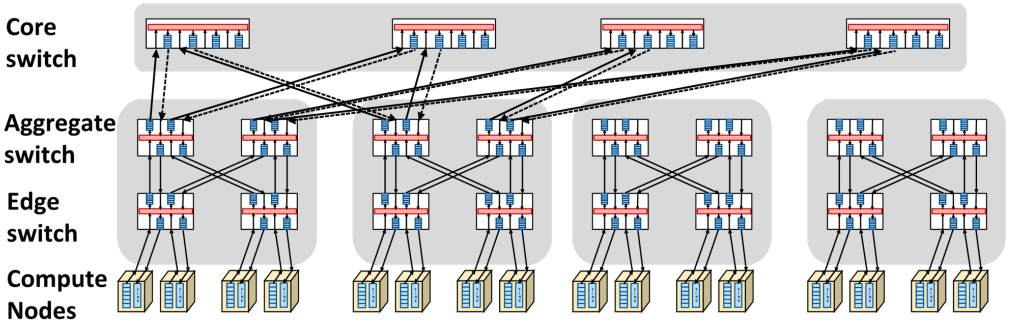

Figure 4.1: Queuing theory representation of 16 node fat-tree

OQ switch is equivalent to CIOQ+CCF (critical cell first IO scheduling) in its input/output timing characteristics. Thus, we use the OQ switch as the modeling abstraction.

There are different versions of queuing theory-based analytical models. This work uses the principle of maximum entropy (ME) because it handles bursty traffic and generalized service distributions quite well by generating the least biased distribution that matches the constraints. We first find the mean queue occupancies and then calculate the waiting time using Little's law [37]. One must also model the interconnection of different queues and flows going through the queues following the routing algorithm. We use the decomposition method to find the inputs to each queue [60], as detailed in Section 5.3. Finally, the accuracy of analytical models can degrade when the real traffic diverges from the assumed parameters. Instead of completely switching to a machine-learning (ML)-based approach, we harness the power of queuing theory and improve on it using a light-weight ML based correction technique, as described in Section 14.

4.3 MQL: ML-Assisted Queuing Latency Analysis

MQL Overview

We set the following goals while scaling to thousands of nodes and providing visibility of internal queue utilization:

- High accuracy in estimating end-to-end packet latency and round-trip delay,
- Fast and lightweight estimation, scaling to DCNs with thousands of nodes,
- Tier- and queue-level visibility (e.g., observing individual queue utilizations),
- Generality to support different workloads and protocols.

Offline Phase: The first two components of MQL leverage the current DCN performance evaluation practice, as illustrated in Figure 4.2. It develops analytical performance models for the topologies, workloads, and network protocols of interest. Then, the accuracy of these models is compared against simulations. The common practice does not offer any systematic technique to learn the accuracy mismatches and use them for correction. MQL elaborates on this step by systematically analyzing the modeling

error as a function of the simulation and analytical parameters. For example, the error may be a function of the queue within a particular switch, data rate, or any input used by analytical models. Therefore, the final component, which provides ML assistance, extracts the most prominent features. Then, it trains a regression model that estimates the error as a function of these features.

Online Use: MQL first runs the analytical performance model to determine the end-to-end latencies. Then, it adds the error estimate to its results as a correction factor.

The proposed MQL methodology can be implemented with any performance analysis and ML technique. The following section describes the specific methods used in this dissertation.

Modeling Assumptions and Target Illustrative DCN

The proposed MQL methodology can be applied to arbitrary DCN topologies and routing algorithms. This work demonstrates MQL on the three-level fat-tree topology (shown in Figure 4.1) with up to 1024 nodes (16 pods) due to its popularity. It is validated with fixed and equal-cost multi-path routing (ECMP) [19] due to its wide usage and complexity.

Since real-world traffic can be bursty, we model the arrival process at each queue based on the Generalized Exponential (GE) distribution [30, 32] which can also handle other distributions like Poisson. The queues accept packets one at a time. Hence, multiple flows (a stream of packets

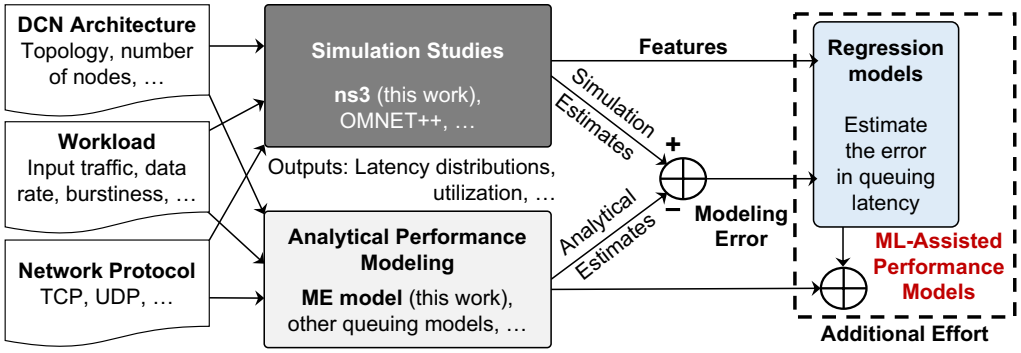

Figure 4.2: Overview of the proposed MQL methodology.

per source-destination pair) can merge and decompose while entering and leaving queues. This general behavior suggests that even if the individual flows follow a specific distribution at the input, the output stream of packets can follow an unknown distribution. Similarly, we do not make any assumptions about the packet size distributions and thus select the Generalized distribution to model the service time. The channel (link) between switches and nodes in Figure 4.1 is modeled as a server for the output queue on the corresponding port. Therefore, we start with a GE/G/1 model and extend it to GE/G/1/N, where N is the finite queue length.

Analytical Queuing Models

This work uses ME-based queuing models to illustrate the proposed MQL methodology due to its accuracy and scalability. Since we focus on the network latency, we use the traffic from each node entering the DCN as the primary input. The ME-based models with generalized exponential traffic employ the first two moments: the average arrival rate (λ_i) and

squared coefficient of variation ($C^2_{A_i}$) of each input flow, as summarized in Table 5.1. These inputs are propagated to the switches in the fat-tree using a decomposition method [60]. Then, the ME models are used to compute the delay in each queue in the target DCN, as described next.

Decomposition method

The flows entering the DCN from the node queues go through multiple merges and separations as they travel to their destination. Since we used a generalized exponential model, the sum of arrival rates alone is insufficient, unlike the Poisson distribution assumption. Figure 5.3 illustrates two flows entering the same queue. The packets from these flows are stored in their arrival order, which is arbitrary. Hence, we need to estimate the first and second order of the models, i.e., the arrival rate and squared coefficient of variation. Following the derivation in the decomposition model given in [60], we find the squared coefficient of variation of inter-arrival times of merged flow as the weighted average of the incoming squared coefficient of variations, as shown in Figure 5.3. Similarly, the squared coefficient of variation of inter-departure times (C^2_D) of the merged flows is found using the decomposition approach illustrated in Figure 5.3. The output flows are split to enter downstream queues. Therefore, we calculate the C^2_D of the split flow and the probability of splitting based on the number of downstream queues using the equation in Phase 3 of Figure 5.3. These split $C^2_{D_i}$ will be the $C^2_{A_i}$ to the downstream queues. Furthermore, the

Table 4.1: Summary of notations used in this dissertation.

λ_i	Injection rate of flow-i
ρ_i	Link utilization of flow-i
p_i	Probability of flow-i split when leaving the queue
$C^2_{A_i}$	Squared coefficient of variation of inter-arrival time for flow-i
$C^2_{D_i}$	Squared coefficient of variation of inter-departure time for flow-i
$C^2_{S_i}$	Squared coefficient of variation of service time for flow-i
λ	Injection rate for merged flow
ρ	Link utilization for merged flow
C^2_A	Squared coefficient of variation of inter-arrival time for merged flow
C^2_D	Squared coefficient of variation of inter-departure time for merged flow
C^2_S	Squared coefficient of variation of service time for merged flow
$\langle n_i \rangle$	Mean queue length of flow-i in an infinite-sized queue
$\langle n_i \rangle_N$	Mean queue length of flow-i in a finite-sized(N) queue
W_i	Average waiting time of flow-i

decomposition method is computed in one pass making our approach scalable.

GE/G/1 Maximum Entropy model

The Maximum Entropy(ME) method approximates the networks when queues achieve equilibrium [10, 32]. For fat-tree topology, with the first-come-first-serve (FCFS) queuing discipline and a single server, we adopt the GE/G/1 ME model (generalized exponential arrival process, and generalized service process) proposed in [30, 32]. The proposed approach traverses the network from source to destination for each flow in the workload following the routing algorithm. During this process, it uses the decomposition process to find the mean arrival rate (λ_i), utilization (ρ_i),

squared coefficient of variation ($C^2_{A_i}$), and the coefficient of variation of the service time ($C^2_{S_i}$) at each queue. Then, it uses the GE/G/1 ME model to find the mean queue length of each flow i ($\langle n_i \rangle$) in an infinite-sized queue as:

$$\langle n_i \rangle = \frac{\rho_i}{2}(C^2_{A_i} - 1) + \frac{\sum_{k=1}^{N} \frac{\lambda_i}{\lambda_k} \rho_k^2 (C^2_{A_k} + C^2_{S_k})}{1 - \rho} \tag{4.1}$$

Finally, the waiting time (queuing delay) of flow-i becomes:

$$W_i = \frac{\langle n_i \rangle - \rho_i}{\lambda_i} \tag{4.2}$$

GE/G/1/N Maximum Entropy model

For finite-sized queue model, we adopt the treatment of queue occupancy (and delay) presented in [29]. We list the key steps here for completeness. Derivations can be found in [29]. We start constructing the analytical model for the finite-sized queue by first assuming infinite queues. With this assumption, we first find the mean queue length of each flow i ($\langle n_i \rangle$) in an infinite-sized queue using Equation 5.1. Then, we find the Lagrangian coefficient x by using the mean infinite queue length $\langle n_i \rangle$ from Equation 5.1 [29].

$$x = \frac{\langle n_i \rangle - \rho_i}{\langle n_i \rangle} \tag{4.3}$$

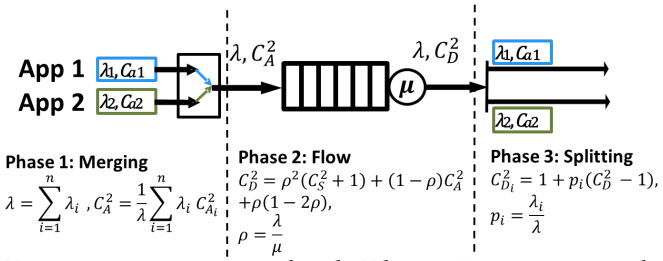

Figure 4.3: Decomposition method: Phase 1 merges multiple flows into single flow. Phase 2 computes the coefficient of variation of departure processes. Phase 3 splits the merged flow to derive the individual departure processes

The Lagrangian coefficient x is then used to find the the mean finite queue length given by Equation A.3, where N is the finite size of the queue:

$$\langle n_i \rangle_N = \frac{\rho_i}{1 - \rho_i^2 x^{N-1}} \left\{ \frac{1 - x^N}{1 - x} - N\rho_i x^{N-1} \right\} \tag{4.4}$$

Finally, it computes the finite mean occupancy ($\langle n_i \rangle_N$) and the waiting time (queuing delay) of flow-i as:

$$W_i = \frac{\langle n_i \rangle_N - \rho_i}{\lambda_i} \tag{4.5}$$

End-to-End and Round-Trip Latency Modeling

Using the λ_i, ρ_i, $C_{A_i}^2$ of the flows entering a queue, we find the queuing delay of that queue. Then, the squared coefficient of variation of inter-departure time ($C_{D_i}^2$) is calculated by the decomposition model, which uses the squared coefficient of variation of inter-arrival time ($C_{A_i}^2$) of the

Algorithm 4: End-to-end latency computation

```
1  Input: Fat-tree size, link bandwidth, flow metadata (flow ID, source,
     destination), characteristics for each flow (λ_i, C²_{D_i}, C²_{S_i}, mean packet size,
     queuing delay of host queues)
2  Output: Average end-to-end latency for each flow
3  foreach queue ready to be processed do
4      I = number of flows in the queue
5      for i = 1:I do
6          Compute W_i using GE/G/1 and GE/G/1/N ME model
7          Compute C²_{D_i} using decomposition model
8          Populate flow characteristics of the upstream queues
9      end
10 end
11 foreach flow in the traffic do
12     Traverse all the queues throughout flow's path
13     Aggregate queuing delay and link delay
14 end
```

upstream queues. Therefore, the algorithm iterates over every queue that has its inputs ready and computes the corresponding delay.

The next step is finding the end-to-end latency of each flow by summing up the queuing delays and the service times. The proposed approach achieves this objective using the fat-tree size, routing algorithm, and the characteristics of each flow$(\lambda_i, \rho_i, C^2_{D_i}, C^2_{S_i}$, mean packet size, queuing delay of host queues$)$. The output is the average end-to-end latency for each flow, as shown in the pseudo-code in Algorithm 5. Finally, we compute the average round-trip time (RTT) by adding the end-to-end latency of data and their corresponding acknowledgement packets.

ML-Based Correction Technique

While the ME model applies to any network that decomposes into a network of queues, it may fail to generalize to all scenarios. Therefore, MQL augments the ME model with a correction factor obtained by ML models. We first provide two examples when ME models perform poorly and then present the proposed ML-based correction technique.

Sample scenarios in which ME models fail to generalize

The packets pass through a sequence of queues as the packets traverse the DCN. For example, packets from the first node (far left) to the last node (far right) in Figure 4.1 pass through queues in the edge, aggregate, and core switches. This tandem arrangement of queues shapes the packet inter-arrival times as a function of their link service times. When packets with different sizes pass through the same link, the tandem nature causes the smaller packets to experience higher queuing latency than their larger counterparts. The ME models fail to capture this effect, resulting in poor latency estimations.

Similar to the packet size distribution, the communication protocol plays a crucial role in the manner the packets are transported through the network. For instance, the User Datagram Protocol (UDP) sends packets into the network irrespective of the size, while the Transmission Control Protocol (TCP) divides the packets into segments based on a

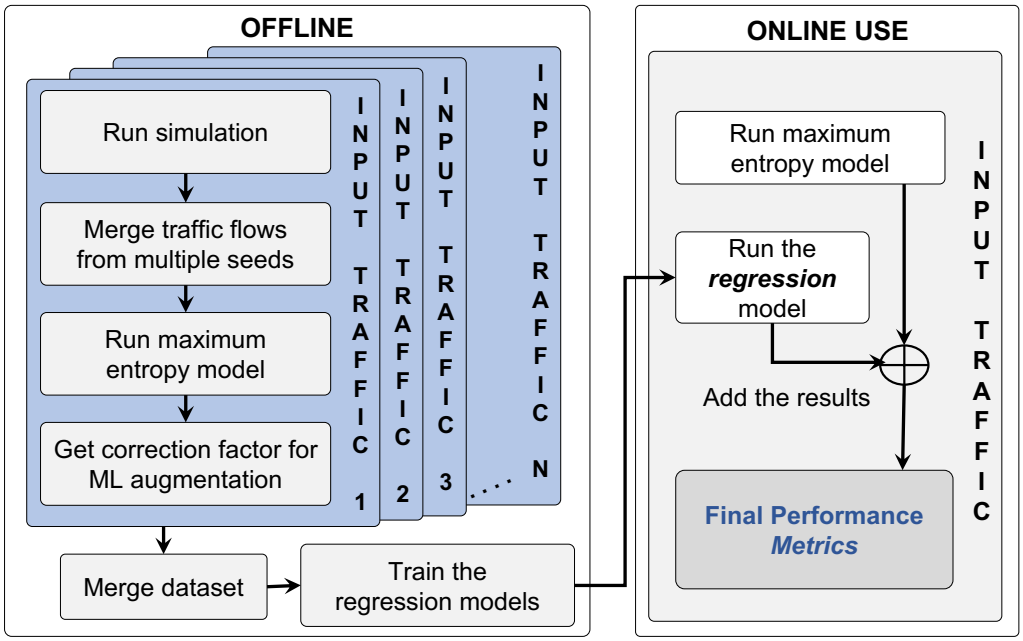

Figure 4.4: Workflow of the ML-assistance component in the MQL framework.

preset threshold size [59]. Similarly, the TCP sends an acknowledgement packet, which are typically significantly smaller than the data packets. The resulting bimodal packet size distribution combined with the tandem effect degrades the accuracy.

Proposed ML-based Assistance to Queuing Models

The ML-assistance component of the proposed MQL framework is presented in Figure 4.4. First, we run simulations with representative configurations (network sizes) and input traffic (packet arrival and size distributions, data rates and traffic types). Simulations with multiple random seed values help in eliminating randomness effects. Then, MQL obtains

the correction factors by comparing the expected latencies from simulation and the ME models. Since each queue type (e.g., edge-up, core-down, aggregate-up) in the fat-tree observes a different traffic pattern, we use a regression model for each queue type. The input features (described in Section 14) and the correction factors are aggregated to obtain a merged training dataset. Since the systematic errors are continuous quantities, any regression-based ML model can be used in this stage. We use regression trees (RT) due to their accuracy and explainable structure. To avoid over-fitting and minimize inference latency, we empirically set the maximum depth of RTs to 12. The RT uses 11 input features (shown in Table 4.2) and predicts one real-valued output. MQL employs one RT model for each type of queue (e.g., edge, aggregate, core) in the network regardless of the number of nodes, thereby providing excellent scalability across network sizes. Finally, we utilize the scikit-learn library to train RTs. Building a consolidated set of training samples allows MQL to generalize to different traffic patterns. Then, MQL trains generalizable regression models, which concludes the one-time offline process. Finally, the runtime step uses the pre-trained regression models to accurately estimate the latency.

Features for the ML-based Regression Model

End-to-end latencies are functions of DCN topology, the traffic arrival distribution, packet size distribution, data rates, and routing patterns. Since our goal is to estimate these delays accurately, we systematically

Table 4.2: List of the input features constructed in a particular queue for the regression model.

Input Feature	Mathematical Representation
Data rate of flow i	λ_i
Link utilization of flow i	ρ_i
Total utilization of a link leaving switch s	$\rho_{total,s}$
Co-efficient of inter-arrival time of flow i	$C^2_{A,i}$
Co-efficient of service time for flow i	$C^2_{S,i}$
Packet size of flow i	P_i
Link occupancy indicator of flow i	$1 \,/\, (1-\rho_i)$
Link occupancy indicator of switch s	$1 \,/\, (1-\rho_{total,s})$
Data rate/link occup. indicator of flow i	$\lambda_i \,/\, (1-\rho_i)$
Data rate/link occupancy indicator of switch s for flow i	$\frac{\lambda_i}{1-\rho_{total,s}}$
Queue occupancy indicator of flow i	$(C^2_{A,i}+C^2_{S,i})/(1-\rho_{total,s})$

construct its input features with the following attributes:

1. They must demonstrate a strong correlation with the target quantity to be estimated,

2. They cannot depend on any parameter or quantity we cannot obtain at runtime (e.g., information from simulation),

3. The overheads to compute them must be minimal.

To satisfy these requirements, we methodically architect the 11 input features to the regression model as shown in Table 4.2 for each queue type. The queuing models described in Section 5.3 use data rate (λ), link utilization (ρ, and ρ_{total}), and second-order moments of inter-arrival (C^2_A) and service times (C^2_S) in latency estimation. The proposed MQL framework

exploits the information to include these parameters as input features. In addition, we include input features, such as link occupancy and queue occupancy indicators, since they typically appear in analytical models (e.g., Equation 5.1). These features satisfy *Attribute 1* both intuitively (as described here) and empirically (demonstrated in Section 4.4). Basing the input features on quantities computed by the ME model is highly desirable since we reuse the information already computed, thereby simultaneously catering to *Attribute 2* and *Attribute 3*.

4.4 Experimental Evaluation

This section first describes the experimental setup. Section 4.4 and Section 4.4 evaluate the proposed MQL approach with synthetic traffic and real-world traces, respectively. Section 4.4 presents the execution time of the MQL models, its speedup w.r.t simulation and comparisons with approaches from literature. Finally, Section 4.4 compares the round-trip latency of MQL with state-of-the-art approaches.

Experimental Setup and Methodology

DCN Topology: While the MQL methodology is applicable to other topologies, this work focuses on the widely used fat-tree topology. Fat-tree topologies are represented by the number of layers and a parameter K, which determines the number of pods [2]. The number of nodes for a

K-ary fat-tree is $(K^3/4)$. In this work, we use three-layer fat-trees with $K \in \{4, 8, 12, 16\}$, leading to sizes listed in Table 5.2.

Workloads used for Evaluation: We utilize both synthetic and real-world traces to evaluate MQL. Synthetic traffic includes *three* different traffic types: all-to-all (each node in the DCN sends packets to every other node), broadcast (only one node (source) sends packets to all other nodes), and incast (all nodes send packets to one node (destination)), as summarized in Table 5.2 along with other parameters. The real-world trace is Anarchy [58].

Simulation Environment and Other Parameters: We performed simulations with ns-3, a discrete-event network simulator [50]. Ns-3 provides packet-level visibility. It also allows users to configure various parameters such as the number of source nodes and destination nodes in the DCN, routing patterns, mean flow sizes, simulation time, network protocol, and FIFO and queue sizes. We perform 30-second simulations with a warmup of an additional 10 seconds to ensure the inputs to the simulation are representative of the steady state. The queue sizes are set to 128 to represent the finite buffer scenario.

The simulations and analytical models are executed on an Intel® Xeon® Gold 6336Y CPU at 2.40GHz with 36 MB cache (OS: SUSE Linux Enterprise Server 12 SP5, compiler version: g++/gcc 11.1.0).

ML-based Assistance: The ML-based regression models must generalize to unseen scenarios for the application of the proposed framework at the

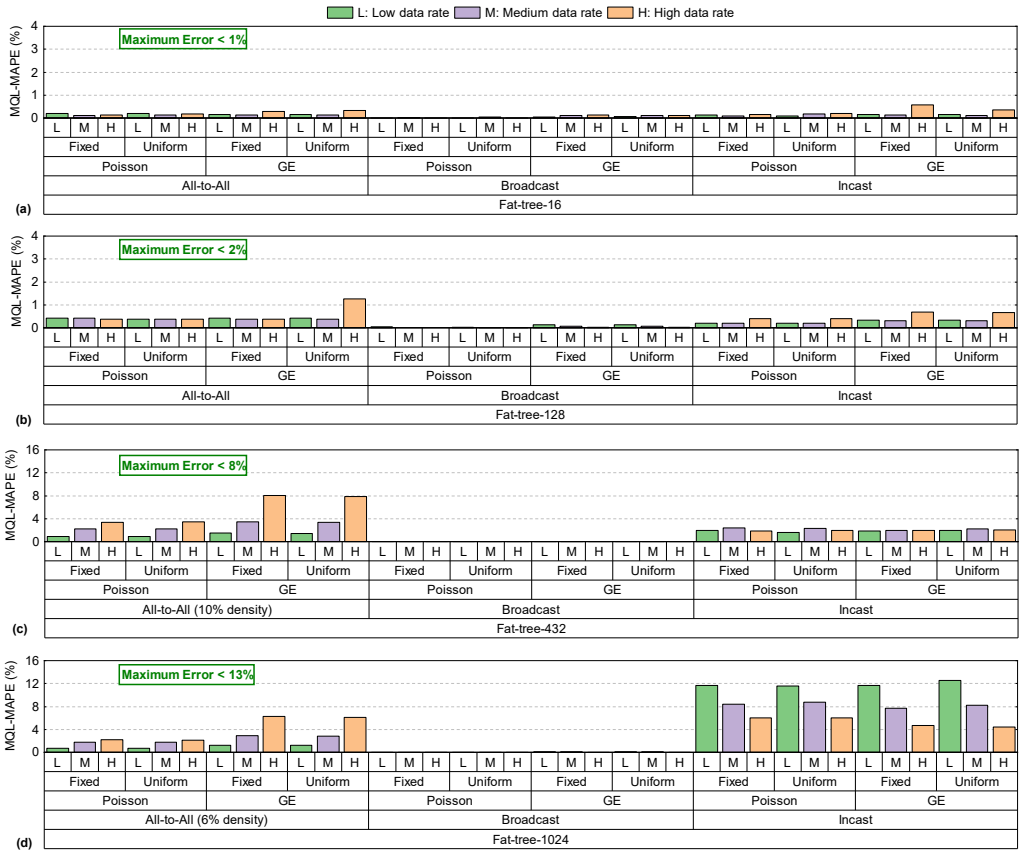

Figure 4.5: MAPE (%) of the round-trip latency achieved by MQL on all-to-all, incast and broadcast traffic for (a) Fat-Tree-16, (b) Fat-Tree-128, (c) Fat-Tree-432 and (d) Fat-Tree-1024 with different types of packet arrival distributions, packet size distributions, and data rates.

larger scale. Thus, we randomly pick 60% of the data to train the regression models, and then evaluate all configurations. In this particular work, we use the regression tree model with a maximum depth of 12 [57].

Comparison Metrics: We use the normalized Wasserstein distance [71] and mean absolute percentage error (MAPE) for accuracy evaluations. The Wasserstein distance compares probability distributions based on

Table 4.3: A summary of the experimental setup used for evaluations in this dissertation.

Parameter	Values Evaluated in this dissertation
3-Layer Fat-Tree Topology	Number of nodes: 16, 128, 432, 1024
Workloads	Synthetic and Real Traces
Synthetic Traffic Patterns	All-to-all, broadcast, incast
Traffic Arrival Distributions	Poisson and Generalized Exponential (GE)
Synthetic Packet Size Dist.	500 B; uniform (500B with 1% variation)
Synthetic Workload Data Rates	Low (link utilization of 25%) Medium (link utilization of 50%) High (link utilization of 75%)
Real Trace	Anarchy
Link Bandwidth	100 Mbps
Protocol	TCP, UDP
Queue	FIFO, 128 Packets Capacity

the theory of optimal mass transport, and is a measure of the distance between two distributions. MAPE is a measure of the average absolute error observed between simulation and MQL estimates.

State-of-the-Art Approaches Chosen for Comparison: We identify a combination of ML- and queuing theory-based approaches, namely DeepQueueNet [75], MimicNet [76], and RouteNet [13] for comparisons. Section 5.2 discusses the significance of these approaches. Comparisons with the state-of-the-art approaches are presented in Section 4.4.

Protocols: We evaluated the proposed MQL methodology using both UDP and TCP. UDP is a connectionless protocol with no congestion control mechanism. Thus, the flow distribution with UDP is less complicated than TCP. Since we obtain high accuracy (overall less than 10% MAPE) with MQL, the rest of this dissertation focuses on the TCP results.

Evaluations with Synthetic Traffic

This section presents extensive evaluations to compare the proposed MQL framework to ns-3 simulations using synthetic traffic. Figure 4.5(a), (b), (c), and (d) present the MAPE results for fat-tree-16, fat-tree-128, fat-tree-432, and fat-tree-1024, respectively. We sweep three data rates (low, medium, and high) with uniform distribution of packet size (fixed and uniform as defined in Table 5.2) in all cases. The entire all-to-all traffic simulations for fat-tree-432 and fat-tree-1024 take prohibitively long simulation times (over 5–9 days for one random seed only) since they simulate 373K flows and 2M flows, respectively. As a result, we reduced the number of flows by using a density parameter that uniformly selects a subset of active hosts participating in the all-to-all communication [25]. Considering the simulation time, we set the density parameters for fat-tree-432 and

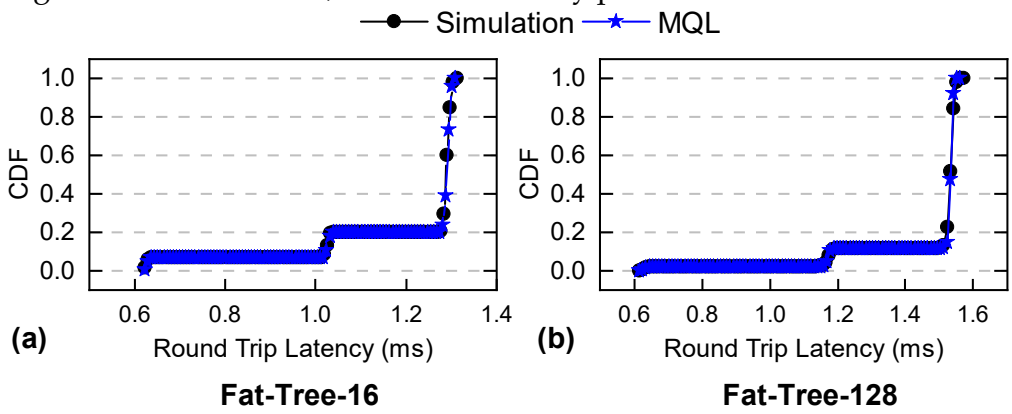

Figure 4.6: A comparison of the round-trip latency (RTT) (in milliseconds) cumulative distribution function (CDF) between simulation and MQL models for all-to-all traffic in (a) Fat-tree-16 and (b) Fat-tree-128 respectively.

fat-tree-1024 as 10% and 6%, respectively.

The broadcast traffic is the simplest pattern since packet injection is only from one source. Furthermore, the packets entering the network are serialized. Our MQL framework models this scenario very accurately, with average MAPE always less than 1%, as shown in Figure 4.5. Hence, in the following discussion we only cover all-to-all and incast patterns.

Fat-tree-16 Results: MQL achieves an MAPE of less than 1% for Poisson all-to-all traffic. Unlike Poisson, GE traffic can produce bursty traffic which is more complex to model. The maximum error with GE packet arrivals, even in the medium and high data rates, remains less than 2%, with an average MAPE of 0.9%. The incast traffic pattern is highly complex to model since several flows merge into one queue. A combination of the ME and ML models in the MQL framework effectively captures this behavior and achieves an average MAPE of 0.9%, with the highest being under 2%.

Fat-tree-128 Results: Similar to the analysis for Fat-tree-16, MQL achieves an average MAPE of 1% for all-to-all Poisson packet arrivals and $< 2\%$ MAPE for GE arrivals. MQL models incast for Fat-tree-128 accurately, with an average error less than 0.9%.

Fat-tree-432 Results: The number of flows and complexity of latency estimation grow with increasing fat-tree size. Besides scaling to these large sizes, the combination of the ME and ML models in the MQL framework perform well with an average error of less than 3% MAPE for the incast traffic and less than 8% MAPE for the all-to-all (10% density) traffic.

Table 4.4: Evaluations with the Anarchy [58] trace.

Size	MAPE(%)	avgRTT(w_1)	p99RTT(w_1)
Fat-tree-16	0.23	0.007	0.009
Fat-tree-128	0.46	0.029	0.034
Fat-tree-432	7.86	0.052	0.058
Fat-tree-1024	0.37	0.040	0.047

Fat-tree-1024 Results: The large number of flows in a 1024 node fat-tree, especially merging into a single queue in incast traffic, severely complicates the modeling. The proposed MQL models result in higher error compared to lower network sizes, with an average MAPE of 8% for incast patterns. However, we note that MQL still enables rapid design space exploration when compared to ns-3 which takes over a week to simulate a reasonable workload duration.

Cumulative Distribution Function of Round-Trip Latency: We must ensure that the RTT throughput distribution is close to the ground truth, as opposed to an averaged value such as the normalized Wasserstein distance or MAPE. Figure 4.6 presents the cumulative distribution function (CDF) of the RTT for all-to-all traffic in fat-tree-16 and fat-tree-128. We observe that MQL achieves high fidelity with the simulation ground truth in the RTT spectrum for both fat-tree-16 and fat-tree-128.

Error Reduction: As anticipated, the MQL demonstrates a significant improvement in error reduction compared to the ME model alone. Upon evaluating all of the synthetic workloads, we calculated the difference of MAPE between the ME model and MQL. The results indicate that the average error reduction is 7.1%, with a variance of 1.2%.

Results of 2-tier fat-tree: In addition to the 3-tier fat-tree, we also evaluated MQL on a proprietary 2-tier fat-tree. The 2-tier version offers a significantly larger level of parallelism by using additional pairs of parallel links between the same edge-core switch pairs. Hence, it is structurally different than the conventional 3-tier fat-tree. Our results indicate that we achieved a comparable accuracy (overall less than 9%) on a 128-node fat-tree when simulating all-to-all, incast, and broadcast synthetic traffic under UDP.

Evaluations with Real-World Traces

Synthetic traffic may often over-constrain the system with traffic that does not represent realistic scenarios. Therefore, we also evaluate a real-world public trace, Anarchy [58]. This trace provides time stamps of the packets injected into the DCN from 16 hosts, including the packet sizes and destinations. We mapped the source and destinations to a 16-node fat-tree (i.e., four pods). The round-trip times match almost perfectly with ns-3 simulations with 0.09% MAPE even without any ML assistance, as shown in the

Table 4.5: Execution time of the MQL models, speedup w.r.t simulations
A2A: all-to-all, IC: incast, BC: broadcast

Size	Traffic	Exec. Time	Speedup w.r.t Sim	Size	Traffic	Exec. Time	Speedup w.r.t Sim
16	BC	0.002s	22092	432	BC	2.420s	471
16	IC	0.002s	19198	432	IC	2.440s	532
16	A2A	0.028s	29111	432	A2A	13m17s	400
16	Anarchy	0.008s	6375	432	Anarchy	1.046s	9730
128	BC	0.083s	2246	1024	BC	9.159s	220
128	IC	0.090s	2429	1024	IC	36.71s	89
128	A2A	47.14s	1367	1024	A2A	1h49m	115
128	Anarchy	0.634s	456	1024	Anarchy	8.293s	5317

first row of Table 4.4. Consequently, the RT adds a negligible correction factor, maintaining the accuracy.

To analyze scalability, we also extended this trace into 128 nodes (i.e., eight pods) by replicating each flow and randomly reassigning its source and destination to different nodes. We repeat this flow replication and reassigning process until all 128 nodes are assigned a source or destination. Similarly, we expanded the original trace to 432- and 1024-node fat-trees. Table 4.4 shows the MAPE and RTT normalized Wasserstein distances in fat-tree-128, fat-tree-432, and fat-tree-1024. All of them achieve less than 8% MAPE and very small Wasserstein distances. Furthermore, Figure 4.7 displays the CDF of RTT for real-world traces on fat-tree-16, fat-tree-128, fat-tree-432, and fat-tree-1024. These plots demonstrates that MQL achieves good traffic generality and accurate results.

Scalability and MQL Execution Time Analysis

This section compares the execution time speedup of the proposed MQL analytical models to corresponding ns-3 simulations (40-second simulation including a 10-second warmup).

Since the fat-tree-1024 all-to-all simulations take extremely long time to complete, we compare the runtime based on a 10-second long simulation, whose results are not used for accuracy analysis due to small number of packets. The speedup is highest for fat-tree-16 at over four orders of magnitude in the best case and over three on average, as shown in

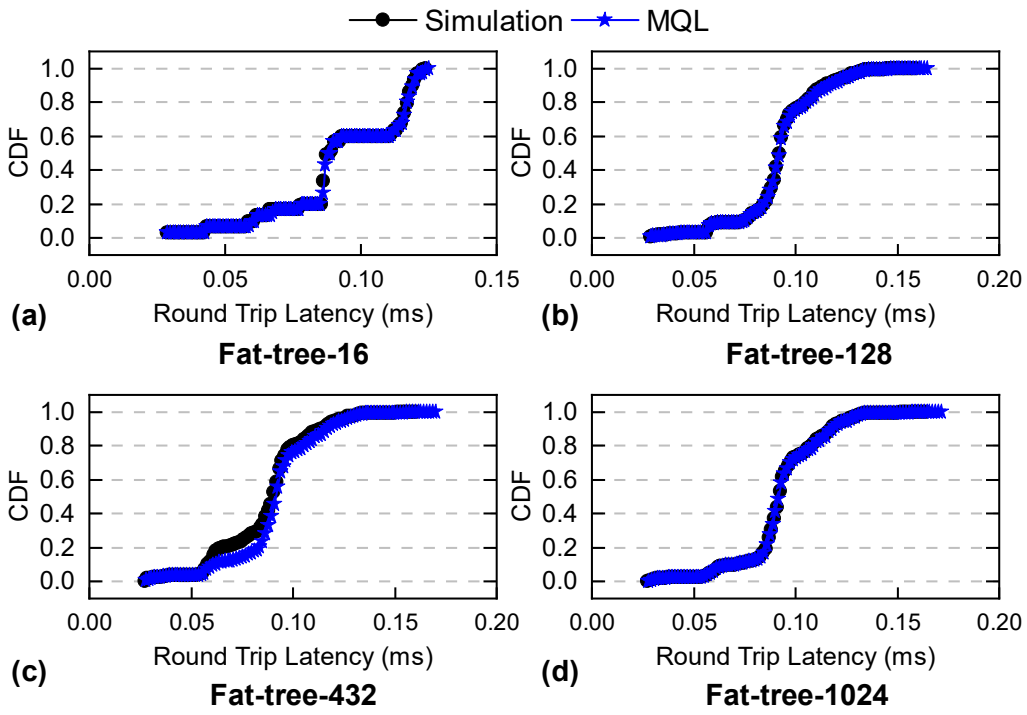

Figure 4.7: A comparison of the cumulative distribution function (CDF) of the round-trip time (RTT) (or latency) in milliseconds between simulation and MQL for the real-world trace Anarchy on (a) fat-tree-16, (b) fat-tree-128, (c) fat-tree-432, and fat-tree-1024 respectively.

Figure 4.8. Since the number of flows increases with the increase in the tree, the ME model component of MQL takes longer, while the ML component takes a constant amount of time. We emphasize that MQL uses the same regression models across sizes and configurations and achieves similar execution times across workloads. Even for fat-tree-1024, MQL achieves a speedup of 89× or higher. The benefits during rapid DCN design space exploration multiply since simulations need to be repeated for multiple random seeds.

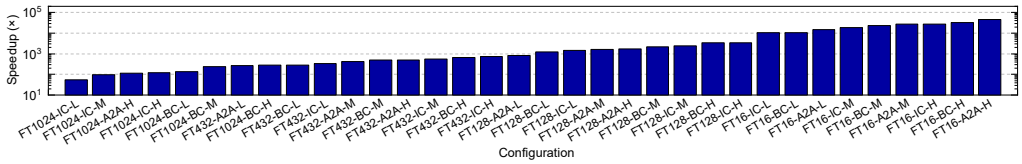

Figure 4.8: Speedup of the proposed MQL framework when compared to ns-3 simulations for different configurations of tree sizes, traffic type and data rates represented by FT{size}-{traffic type}-{data rate}. Sizes vary between 16, 128, 432 and 1024. Traffic types vary between all-to-all (A2A), incast (IC) and broadcast (BC). Data rates vary between low (L), medium (M), and high (H).

Comparison with State-of-the-Art Approaches

This section compares the proposed MQL approach to three state-of-the-art approaches: DeepQueueNet [75], RouteNet [64], and MimicNet [76]. Table 4.6 lists the normalized Wasserstein distances [71] (lower is better) between the RTT of these approaches and simulations for synthetic traffic in fat-tree-16 and fat-tree-128. We compare the w_1 distance of average RTT indicated by avgRTT(w_1), and the 99th percentile RTT w_1 distances indicated by p99RTT(w_1).

For fat-tree-16 using synthetic traffic with Poisson distribution arrivals, MQL achieves an avgRTT(w_1) better than competitive approaches. The proposed MQL approach also achieves lower 99th percentile w_1 distance (i.e., higher accuracy) for all configurations. Similarly, MQL outperforms the state-of-the-art approaches in terms of the avgRTT(w_1) and p99RT(w_1) for fat-tree-128, providing the best-in-class performance estimation models. We could not include comparison with larger fat-trees since the other

Table 4.6: A comparison of normalized Wasserstein distances of RTT (avgRTT(w_1)) and 99th percentile RTT (p99RTT(w_1)) between DeepQueueNet [75], MimicNet [76], RouteNet [13] and our proposed MQL framework for synthetic traffic.

avgRTT (w_1)				
Size	**DeepQueueNet**	**RouteNet**	**MimicNet**	**MQL (Ours)**
Fat-tree-16	0.0086	0.6737	0.0090	**0.0025**
Fat-tree-128	0.0133	0.9824	0.0172	**0.0077**
p99RTT (w_1)				
Size	**DeepQueueNet**	**RouteNet**	**MimicNet**	**MQL (Ours)**
Fat-tree-16	0.0145	0.9723	0.0135	**0.0021**
Fat-tree-128	0.0532	0.6397	0.0194	**0.0109**

approaches limit their evaluations to networks with 128 nodes. In contrast, we report evaluations with substantially larger network sizes, demonstrating the proposed MQL approach's scalability.

4.5 Conclusion

Data centers provide shared access to computing, storage, and memory resources for large organizations that serve millions of users. Efficient, accurate, and scalable performance analysis techniques are critical for rapid design exploration efforts enabling architectural optimizations. To address these challenges, we proposed MQL, a novel and scalable performance analysis methodology that combines a queuing theory-based maximum entropy principle and an ML-based assistance technique to correct systematic errors. MQL achieves a minimum of $\sim 80\times$ and up to four orders of magnitude speedup compared to simulations using the discrete-event ns-3

framework. With the evaluations used in this dissertation, MQL estimates the latencies with less than 3% error on average for DCNs with 16 to 1024 nodes. Future directions include demonstrating the approach on topologies other than fat-tree and validating MQL's modeling accuracy to ensure it is scalable when the queue buffer resources are highly constrained.

5 SIMILARITY-BASED FAST ANALYSIS OF DATA CENTER NETWORKS

5.1 Overview

Data Centers (DCs) are essential for numerous cloud computing and big data applications, with emerging demands necessitating larger distributed computing systems. Designing and deploying a Data Center Network (DCN) involves significant costs, leading to the use of low-cost network elements like commodity switches [2]. Modeling DCNs is crucial for various design aspects, including protocol design, performance evaluation, and network planning. While simulators provide detailed observations, they are slow compared to analytical approaches, and recent works have explored leveraging deep learning techniques for DCN performance analysis.

MimicNet and DeepQueueNet are examples of deep learning-based approaches that combine simulation with deep neural networks (DNNs) to model DCNs. Analytical techniques, such as queueing theory, offer speed and scalability without requiring extensive training data, providing models based on first principles. However, the speed advantage of analytical techniques can degrade as the network size increases. DCNs use topology structures and routing schemes to maintain low latency and high bandwidth, presenting challenges for analytical approaches in

representing multiple paths through the network efficiently.

Inspired by MimicNet's assumptions and high accuracy, we hypothesize that the high amount of symmetry and load balancing in DCN architecture results in paths with similar performance characteristics. We propose an approach to identify these similar performance characteristics and thus reduce the "redundant" analytical calculations. We make the following observations to be the sources of redundancy:

Key observation 1: DCN topologies and routing algorithms have a high degree of symmetry. We observe this symmetry at each level of the fat-tree topology. This results in similar queueing-theoretic model parameters and similar communication latency for a set of possible paths.

Key observation 2: In highly congested networks, congestion control algorithms will throttle the injection rate of flows, regularizing the data rates across flows, that cause an increased likelihood of similarity in each path.

Key observation 3: Many distributed applications have distributed and regular communication across the nodes, including Machine Learning (ML) / Artificial Intelligence (AI) and High Performance Computing (HPC) applications. For example, workloads that leverage neural networks must update their weights, which requires communicating those weights to all participating nodes. Our technique can significantly decrease the time to evaluate DCN performance with minimal impact on accuracy.

Using these observations, we propose a novel methodology that uses similarity analysis to cluster paths based on observing traffic statistics. First, we choose a single representative path from the cluster. Then, all the queueing-theory performance metrics (delay, occupancy) are calculated and reused for the remaining flow-splits in the cluster. This approach minimizes the calculations by one over cluster size with minimal impact on accuracy.

In summary, this dissertation makes the following contributions:

- Demonstrates a similarity-based technique that significantly reduces algorithmic complexity from quadratic to linear with a minimal reduction in accuracy for a commonly used family of DC topologies and routing algorithms, enabling the feasibility of DCN performance evaluation at scale.

- Provides a high degree of visibility (e.g., individual queueing delay per flow and queue occupancies), hence explainable results. Furthermore, a user-tunable similarity threshold enables users to make informed speed versus accuracy trade-offs, potentially increasing the scope of design exploration.

- Presents extensive simulation studies with synthetic traffic that show a similarity threshold as low as 5% yields up to a 2000x speed-up over the current state of the art and 400-40000x over ns-3 [50] with

a minimal 1% degradation in accuracy and scalability to a 2000 node fat-tree.

5.2 Related Work

Simulators such as ns-3 and OMNet++ enable high visibility for analysis and include various network features. However, their execution time for large-scale networks increases drastically because of the fine-grained packet-level simulations. Several methods have been proposed to speed-up the performance analysis of large-scale networks, like parallelism in simulations, integrating ML to model the network, and modularizing ML models instead of creating a monolithic model for the entire network.

Szymanski et al. [68] propose a method that partitions the network into domains and simulation time into intervals and simulates them independently. Domains iterate until convergence and then move on to simulate the next interval. However, this method achieves only a maximum speed-up of 20× in a 256-node network with non-feedback-based protocols and 3× in feedback-based protocols. Navaridas et al. [48] propose a flow-level simulation framework called INRFlow for modeling large-scale networks and computing systems. However, it does not consider the temporal and causal relationships between flows, that represent a more realistic simulation.

Increase in simulation speed is limited by dependencies of the network

flows and the protocol. Therefore, several studies introduce ML for network performance analysis. Parsimon [79] suggests congestion events in large-scale DCs are rare and occur at different points and times along the path, enabling parallelism of simulation. The authors in [35] propose modularized learning by dynamically combining "learning assignment" into an inference graph, allowing for on-the-fly modeling. However, this methodology provides only a 1.57× speed-up. The authors in [14] developed a Network Traffic Transformer (NTT), a transformer model designed to learn network dynamics from packet traces. Hagos et al. [17] propose a deep learning model to predict the Round Trip Time (RTT) of the Transmission Control Protocol (TCP) protocol using Recurrent Neural Network (RNN). Similarly, the authors in [20] model the TCP behavior using Gated Graph Neural Networks (GGNN). However, neural networks are complex models that are hard to explain, and all ML requires training.

Several analytical models have been proposed for network performance modeling [24, 46]. The authors in [46] present a new direction in Markov chain analysis of TCP by examining the cumulative distribution function of transfer time under various models. However, they tested their experiments for a maximum transfer of 1024 packets which is unrealistic, as shown by the authors in [63]. Another notable challenge is the computational complexity that increases with the network size. This complexity explodes due to the Equal-Cost-Multiple-Path (ECMP) routing, where each packet within a connection can navigate multiple routes to reach its

destination. This method leads to a significant increase in the number of available paths.

In summary, simulations take exorbitant time to run as the network scales, while ML models require training and often lack explainability. Closed-form analytical models provide runtime improvements; however, as the network size increases, the model's runtime significantly suffers. To address, we propose a novel similarity-based performance analysis technique, a tunable knob that allows users to trade-off accuracy versus speed-up. The runtime improvements of this technique show zero to negligible impact on accuracy, even with skewed traffic.

Background and Motivation

DCs are vital for executing computationally intensive distributed applications like MapReduce. As the DC compute capacity grows to meet applications demands, so does the size and cost of the network interconnect - this greatly affects design decisions.

Al-Fares et al. [2] propose a scalable architecture that leverages low-cost commodity ethernet switches to deliver full bandwidth from any node to another. The bandwidth comes from multiple paths enabled by a recursive "multi-rooted" tree (e.g., multiple core switches) that employs various routing schemes to leverage these paths.

Noting that DCs can be designed with any topology or routing scheme, we motivate our methodology using the 3-tier fat-tree with ECMP routing

as proposed by [2]. Figure 5.1 demonstrates a 54 node fat-tree with 3-tiers, each with corresponding switches: 1) the core tier, 2) the aggregation tier, and 3) the edge tier. The count of nodes, switches, and pods in topology can all be described as a function of the number of ports at each switch k and the number of levels l in the tree. Each successive tier from edge to core constitutes an additional fan out of possible paths from the source node to the core switch; $k/2$ from each edge switch, $k/2$ from each aggregate switch, and k paths from each core switch [2].

Routing schemes like ECMP evenly divide the traffic flowing from a source to destination pair (a flow) into each possible path (flow-splits). This typically results in highly symmetric and similar traffic at parallel network locations. Multiple paths impact the execution time of simulation and analytical models differently. In the simulation, only the time to access the routing table grows. In the analytical model, each flow-split requires calculating the queue occupancy (and latency), a more expensive set of operations (e.g., multiplications) than a routing table access.

Analytical models' algorithmic complexity can scale disfavorably con-

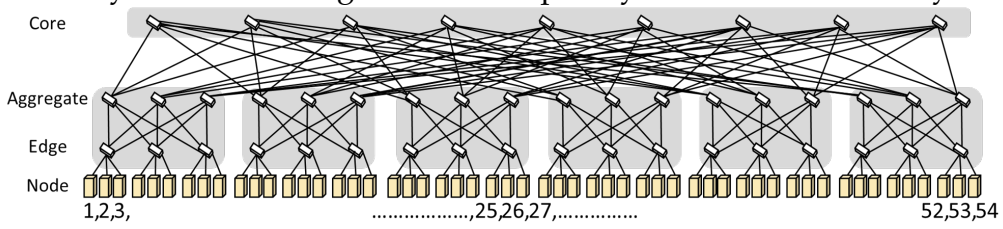

Figure 5.1: A 3-tier 54 node fat-tree with core, aggregate, edge switches. The left most 3 nodes and the right most 3 nodes are numbered.

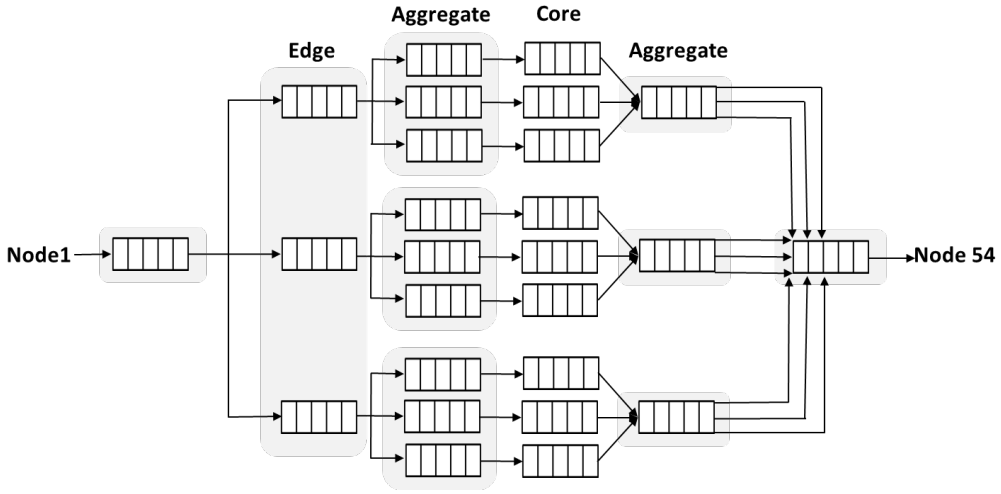

Figure 5.2: Unfolded representation of fat-tree topology with all possible paths from a source (node1) to destination (node54).

cerning simulation. Each packet, flow, and queue must incur a simulation penalty in simulation. Not considering packets, we see that this results in complexity $\mathcal{O}(f \times q)$, where f is the number of flows, and q is the number of queues in the flow. Analytical models extend this by adding possible routes to the complexity analysis resulting in $\mathcal{O}(f \times q \times s)$. We use s to denote network size, which dictates possible routes at each flow-split boundary, growing at a rate of $(k/2)^2$, a significant increase. For example, Figure 5.2 depicts one flow from a single source node (1) to a destination node (54). As shown in Figure 5.1, the switch radix is $k = 6$, making the number of flow-splits entering the final node 9, $((6/2)^2)$.

This work leverages the Maximum Entropy (ME) technique of queueing theoretic analytical modeling as it is robust to bursty traffic and generalized service distribution [47]. As we show in Section 5.4, when there is

a negligible difference in input parameters to the ME model, the resulting occupancies and latencies are close in value. To mitigate the additional computational complexity, we use a lightweight clustering approach to group the flow-splits into clusters based on similar model parameters. At each point, we choose one representative flow-split from each cluster, perform the analytical calculation, and substitute those values for all other flow-splits in the cluster. Thus reducing the additional complexity in the best case from multiple paths (quadratic) to a constant (1).

5.3 Proposed Technique

Overview

This section presents the proposed similarity-based technique in detail. The methodology uses a caching technique that works by clustering flows with similar characteristics to reduce the algorithmic complexity of the analytical model. To set the proper context of where and how similarity plays a crucial role in speeding up performance analysis, we describe how DCN is modeled using queuing theory. First, we describe how a set of queues represent each switch and node in DCN. Next, we elaborate on the constructed analytical model, which consists of the Decomposition method and the ME method. The decomposition method enables modeling the interconnection of queues. Consequently, it models the

input characteristics listed in Table 5.1 (e.g., the arrival rate (λ_i) and the squared coefficient of variation ($C^2_{A_i}$)) for each input flow entering the queues. Given the input characteristics, we use the ME method to find the communication latency of the network. We then explain where the algorithmic complexity lies and the reason behind it. Finally, we describe the proposed novel technique to reduce the algorithmic complexity of the model, providing a speed-up of 2000x while maintaining a Mean Absolute Percentage Error (MAPE) of less than 1%.

While our approach generalizes to any topology and routing scheme, we initially demonstrate the proposed approach using ECMP routing on a 3-tier and a modified 2-tier fat-tree topology that increases the switch radix to improve throughput and latency via increasing the network parallelism.

Model Assumptions

We begin the analytical model by using Output Queues (OQ) as modeling abstraction since they are easier to model [47]. Next, we adopt Generalized Exponential (GE) distribution as the arrival process to model bursty real-world traffic. This distribution is capable of handling various other distributions, including Poisson. Since the queues accept packets individually, multiple flow-splits belonging to a class (a stream of packets per source-destination pair) can merge and decompose while entering and leaving the queues. As a result, the output stream of packets may follow an unknown distribution, even if the individual flow-split adheres to a specific

Table 5.1: Summary of notations used in this dissertation.

λ_i	Injection rate of flow-i
ρ_i	Link utilization of flow-i
p_i	Probability of flow-i split when leaving the queue
$C^2_{A_i}$	Squared coefficient of variation of inter-arrival time for flow-i
$C^2_{D_i}$	Squared coefficient of variation of inter-departure time for flow-i
$C^2_{S_i}$	Squared coefficient of variation of service time for flow-i
λ	Injection rate for merged flow
ρ	Link utilization for merged flow
C^2_A	Squared coefficient of variation of inter-arrival time for merged flow
C^2_D	Squared coefficient of variation of inter-departure time for merged flow
C^2_S	Squared coefficient of variation of service time for merged flow
$\langle n_i \rangle$	Mean queue length of flow-i in an infinite-sized queue
$\langle n_i \rangle_N$	Mean queue length of flow-i in a finite-sized (N) queue
W_i	Average waiting time of flow-i

distribution at the input. Additionally, we do not make any assumptions regarding packet size distributions and choose Generalized distribution to model service time. In Figure **??**, the link between switches and nodes is considered a server for the output queue on the corresponding port. Thus, we begin with a GE/G/1 model and expand it to GE/G/1/N, where N represents the finite queue length.

Analytical Queuing Models

The proposed methodology utilizes ME-based queuing models due to their accuracy and scalability. As such, traffic originating from each node entering DCN serves as the primary input to the model. The ME-based

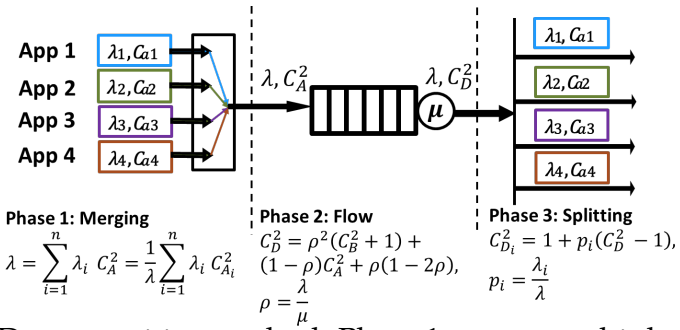

Figure 5.3: Decomposition method: Phase 1 merges multiple flows into single flow. Phase 2 computes coefficient of variation of departure processes. Phase 3 splits merged flow to derive individual departure processes.

models, employing generalized exponential traffic, consider average arrival rate (λ_i) and squared coefficient of variation ($C_{A_i}^2$) as the first two moments for each input flow, as Table 5.1 summarizes.

Decomposition Method

As the flows traverse DCN, they undergo merging and separation processes. Figure 5.3 illustrates merging four flows into the same queue, where packets are stored in arbitrary arrival order. Estimating the arrival rate and squared coefficient of variation becomes crucial in this scenario. Based on the decomposition model presented in [60], we calculate squared coefficient of variation of merged flow's inter-arrival times as the weighted average of the squared coefficient of variations of incoming flows (Figure 5.3). Similarly, we use the same decomposition approach to determine the squared coefficient of inter-departure times (C_D^2) variation for the merged flows (Figure 5.3). The output flows are split and enter their

desired downstream queues. To handle splitting, we compute C_D^2 and the probability of splitting for each flow-split based on the number of downstream queues, as defined in Phase 3 of Figure 5.3. These split $C_{D_i}^2$ values are used as $C_{A_i}^2$ values for the downstream queues. Importantly, our decomposition method can be computed in a single pass, ensuring the scalability of our approach.

GE/G/1 Maximum Entropy Model

The ME method approximates networks when queues achieve equilibrium [29]. Following the routing algorithm, the proposed approach traverses the network from source to destination for each flow in the workload. During this process, it uses decomposition process to find the mean arrival rate (λ_i), utilization (ρ_i), squared coefficient of variation ($C_{A_i}^2$), and coefficient of variation of service time ($C_{S_i}^2$) at each queue. Then, it uses the GE/G/1 ME model to find the mean queue length of each flow i ($\langle n_i \rangle$) in an infinite-sized queue as:

$$\langle n_i \rangle = \frac{\rho_i}{2}(C_{A_i}^2 - 1) + \frac{\sum_{k=1}^{N} \frac{\lambda_i}{\lambda_k} \rho_k^2 (C_{A_k}^2 + C_{S_k}^2)}{1 - \rho} \tag{5.1}$$

Finally, the waiting time (queuing delay) of flow-i becomes:

$$W_i = \frac{\langle n_i \rangle - \rho_i}{\lambda_i} \tag{5.2}$$

Algorithm 5: Similarity-based end-to-end latency calculation.

```
1  Input: Fat-tree size, link bandwidth, flow metadata (flow ID, source,
     destination), characteristics for each flow (λ_i, C²_{D_i}, C²_{S_i}, mean packet size,
     queuing delay of host queues)
2  Output: Average end-to-end latency for each flow
3  foreach queue ready to be processed do
4      Initial empty vector, cacheFlows
5      I = number of flows in queue
6      for i = 1:I do
7          if flow characteristics of i is similar to flow characteristics of an existing flow in
             cacheFlows then
8              W_i = W_matchedFlow
9              C²_{D_i} = C²_{D_matchedFlows}
10         end
11         else
12             Compute W_i using GE/G/1 and GE/G/1/N ME model
13             Compute C²_{D_i} using decomposition model
14             Populate flow characteristics of upstream queues
15             Add flow i in to vector cacheFlows
16         end
17     end
18     clear vector, cacheFlows
19 end
20 foreach flow in traffic do
21     Traverse all queues throughout flow's path
22     Aggregate queuing delay and link delay
23 end
```

For the finite-sized queue model (e.g., GE/G/1/N), we adopt the approach taken in [29]. The finite-size queue model performs additional compute-intensive operations that the citation expounds upon.

Similarity-Based Technique to Reduce the Algorithmic Complexity

For clustering, we represent each flow-split, f_i, as a tuple of the parameters in the ME-model. We use Absolute Percent Error (APE) as the distance

Table 5.2: A summary of the experimental setup used for evaluations in this dissertation.

Parameter	Values Evaluated in this dissertation
3-tier Fat-Tree Topology	Number of nodes: 16, 128, 432, 1024, 2000
2-tier Fat-Tree Topology	Number of nodes: 16, 128
Similarity threshold	5%, 15%
Workloads	Synthetic Traces
Synthetic Traffic Patterns	All-to-all, Broadcast, Incast, Asymmetric
Traffic Arrival Distributions	Poisson and Generalized Exponential (GE)
Synthetic Packet Size Dist.	500 B; uniform (500B with 1% variation)
Synthetic Workload Data Rates	Low (link utilization of 25%) Medium (link utilization of 50%) High (link utilization of 75%)
Link Bandwidth	100 Mbps
Protocol	TCP, UDP
Queue	FIFO, 128 Packets Capacity

metric between two corresponding flow-split parameters (f_{ip} and f_{jp}) for each parameter p in P (λ_2, $C^2_{A_2}$, ρ_2, $\langle n_2 \rangle$, $\langle n_2 \rangle_N$, W_2). If the APE for any parameter is greater than the similarity threshold, then the two flow-splits are not similar. This technique is lighter weight (fewer computations in the distance calculation) than traditional distance measures (e.g. Euclidian or Manhattan). We reduce the algorithmic complexity of the analytical model by only evaluating the analytical model once per cluster. Algorithm 5 details using similarity when estimating end-to-end latency.

5.4 Experimental Evaluation

We evaluated the approach on a rich set of workloads and topologies. Section 5.4 details the experimental setup. Section 5.4 evaluates the proposed technique using synthetic traffic on a 3-tier and a customized 2-tier fat-tree. For different similarity thresholds, we present the speed-up, the number of calls to the analytical model, the impact on MAPE, and the absolute runtimes of the proposed technique.

Experimental Setup and Methodology

DCN Topologies: While similarity-based technique applies to many topologies, our experiments focus on popular fat-tree topology with two variations: 3-tier and 2-tier fat-tree. The main distinction between 3-tier and 2-tier fat-tree is the number of layers and ports per switch. The 3-tier fat-tree follows a classical fat-tree with three layers where the parameter k determines the number of pods [2]. The 2-tier uses two layers of switches

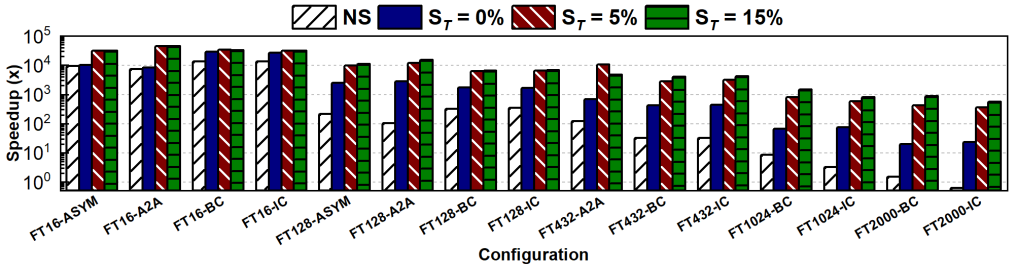

Figure 5.4: Performance analysis speed-up of 3-tier fat-tree with three different similarity thresholds (S_T) as compared to ns-3.

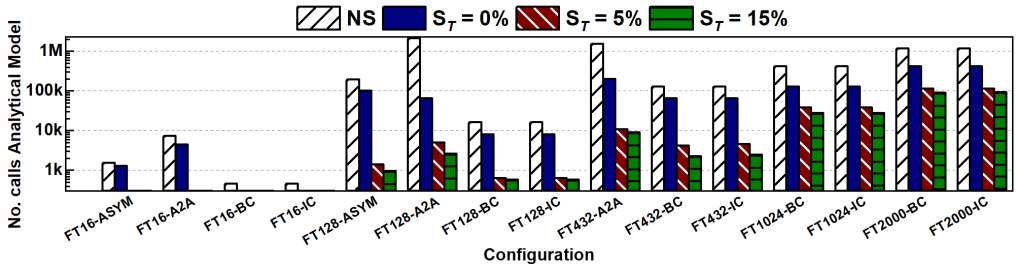

Figure 5.5: Number of times the analytical model is called for different similarity thresholds (S_T) in a 3-tier fat-tree.

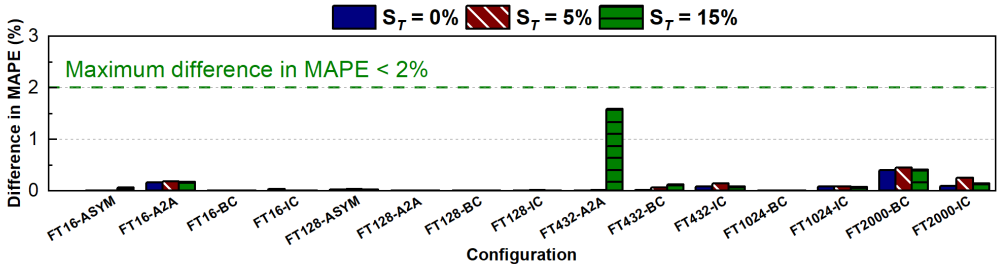

Figure 5.6: Comparison of baseline model MAPE with the MAPE obtained through different similarity thresholds (S_T) in a 3-tier fat-tree.

customized to have more ports than a classical 2-tier fat-tree. This customization of ports increases the number of paths between a source and destination by using larger switch radixes (8 and 64 ports for 16-nodes and 128-nodes, respectively) to add duplicate parallel links, p, between every pair of edge and core switches (2 and 16 parallel links for 16-nodes and 128-nodes, respectively). The parallel links add a factor of p^2 to the algorithmic complexity of customized 2-tier. In this work, we use 3-tier fat-trees with $k \in \{4, 8, 12, 16, 20\}$, leading to node sizes 16-2000 as listed in Table 5.2, whereas in the customized 2-tier fat-tree we perform the technique on 16 and 128 node sized network.

Workloads Used for Evaluation:

We evaluate the technique using synthetic traces, which include *four* different communication types: all-to-all (each node in DCN sends packets to every other node), broadcast (only one node (source) sends packets to all other nodes), incast (all nodes send packets to one node (destination)), and asymmetric as summarized in Table 5.2 along with other parameters. The asymmetric (ASYM) traffic pattern is a combination of all-to-all (A2A), broadcast (BC), and incast (IC), generated by dividing the entire network into three sets of nodes. For example, in Figure 5.1, we divide the network into 3 sets of 18 nodes. The first set of 18 nodes sends an incast traffic to node 1, and the third set of 18 nodes receives broadcast traffic from node 1. For the all-to-all traffic, two random set of 18 nodes are chosen. We run the traces for three different data rates, which correspond to a link utilization of 25% (Low), 50% (Medium), and 75% (High). However, since ns-3 simulation time increases drastically with network size, number of flows, and data rate, we limit the flows and data rate in networks with $>$ 250 nodes. To reduce the number of flows we use a density parameter (6%) which only injects 6% of the possible set of flow-splits supported by the topology and uses only one injection rate (medium). The flows are uniformly selected from a subset of active hosts participating in all-to-all communication. Furthermore, each simulation configuration was first run with each flow-split having the same data rate, and the same configuration was repeated with each flow-split having a different data

rate. The different data rate was selected uniformly between 0 and 75% (High) link utilization.

Similarity Thresholds: To determine threshold values, we first run all the configurations (network sizes up to 128 nodes) with no similarity (NS) as the baseline, then repeat the runs with two different similarity thresholds (5%, 15%). Analysis of simulation results showed a median difference between flow-split characteristics of around 5% among all the parameters indicated in Table 5.2. Therefore, we selected a threshold of 5% and added 15% for comparison.

Simulation Environment and Other Parameters: We performed simulations with ns-3, a discrete-event network simulator [50]. Ns-3 provides packet-level visibility and allows users to configure various parameters such as the number of source and destination nodes, routing patterns, mean flow sizes, simulation time, network protocol, and queue sizes. We perform 30-second simulations with a warm-up of an additional 10 seconds to ensure inputs to the simulation are representative of a steady state. The queue sizes are set to 128 to represent a finite buffer scenario. The runtime grows prohibitively high (weeks or more) as the number of nodes increases. Therefore, for network sizes greater than 250 nodes, we constricted the simulation length to 11 seconds with 1 second of warm-up.

Protocols: We evaluated the proposed similarity-based technique using both User Datagram Protocol (UDP) and TCP because of their popularity.

Comparison Metrics: We use speed-up, MAPE, number of calls to the

analytical model, and absolute runtime to demonstrate how similarity decreases performance estimation time using the analytical model. We first show speed-up of the technique with respect to simulation for each configuration. Then we show the significant reduction in calls to the analytical model using three different similarity thresholds ($S_T = 0\%, 5\%, 15\%$) versus no similarity. Similarity threshold of 0% means that each flow-split is compared and clustered only if they match exactly whereas no similarity indicates no comparisons or clustering of flow-splits. Finally, we show the impact on MAPE of applying similarity ($S_T = 0\%, 5\%, 15\%$) vs no similarity.

Total Number of Simulations: Table 5.2 summarizes the parameters used for evaluation. We ran 1992 different simulations for 3-tier fat-tree, 1733 for 2-tier, for a total of 3725 evaluated simulations.

Evaluations on a 2-Tier and 3-Tier Fat-tree

This section extensively evaluates the proposed similarity-based technique using synthetic traffic on 3-tier and 2-tier fat-tree topologies.

Number of Calls to Analytical Model:

Figures 5.5 and 5.8 (3-tier and 2-tier respectively) show the number of calls to the analytical model with no similarity and with similarity thresholds ($S_T = 0\%$, $S_T = 5\%, 15\%$). For no similarity, we expect two trends in the number of calls to the analytical model. Firstly, for a given communication

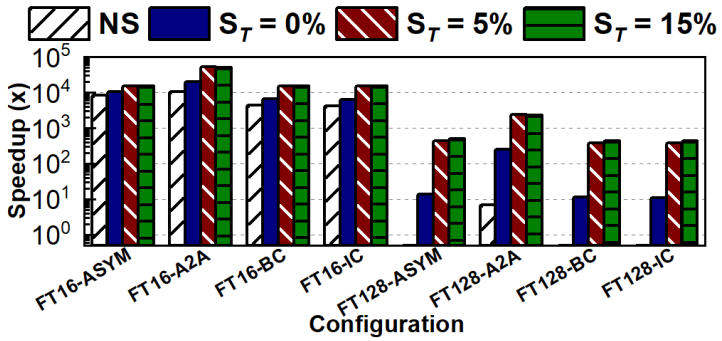

Figure 5.7: Performance analysis speed-up of 2-tier fat-tree with three different similarity thresholds (S_T) as compared to ns-3.

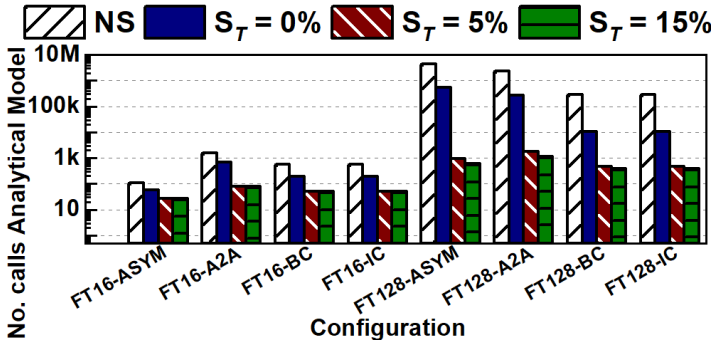

Figure 5.8: Number of times the analytical model is called for different similarity thresholds (S_T) in a 2-tier fat-tree.

pattern, we expect the number of calls to grow with network size. As seen in Figure 5.5 the trend holds for all communication patterns up to fat-tree 128 (FT128). However, as described in 5.4, to obtain practical ns-3 simulation runtimes, we apply a density factor to reduce the number of flows. For the fat-tree-432 all-to-all (FT432-A2A) workload, a 6% density results in approximately 11k flows, which is slightly lower than the number of flows in the smaller FT128-A2A with 100% density.

Secondly, for a given network size, we expect the all-to-all commu-

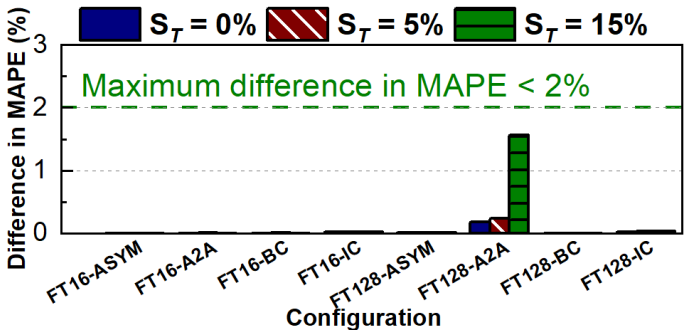

Figure 5.9: Comparison of baseline model MAPE with the MAPE obtained through different similarity thresholds (S_T) in a 2-tier fat-tree.

nication pattern to have the largest number of calls, as it has the most flow-splits (proportional to N nodes communicating to all other $N-1$ nodes). This trend holds for the 3-tier fat-tree. However, for 2-tier, the all-to-all workload at 128 nodes (FT128-A2A) has fewer calls than the asymmetric communication pattern. This time we apply a density factor (6%) to achieve tenable runtimes for the *analytical* model. As mentioned in Section 5.4, the 2-tier FT128 has a significantly higher degree of parallelism, $p = 16$, which increases the algorithmic complexity of the analytical model, and similarly the number of calls, by a factor of p^2, resulting in the untenable runtimes.

As the number of calls directly impacts speed-up, we leave further observations regarding the number of calls to the next section.

Speed-up of the Analytical Model versus ns-3:

The proposed similarity technique is implemented in C++ to fairly compare the execution time with ns-3 simulator. Figure 5.4 shows the speed-up

of each configuration in 3-tier fat-tree with different similarity thresholds, and Figure 5.7 shows the results for the customized 16-node and 128-node 2-tier fat-tree.

As mentioned in the previous section, for no similarity, as the network size grows, the number of flow-splits (i.e., calls to analytical model) increases exponentially. Consequently, the speed-up compared to packet-level simulation degrades proportionally to network size. This impacts the 2-tier to a greater degree than 3-tier, with a *slowdown* for some configurations starting at a small network size of 128 nodes. Hence, the similarity method is critical to enabling performance analysis at scale.

The speed-up improvements for fat-tree-16 with similarity ($S_T = 0\%$, 5%, 15%) compared to no similarity are small because the number of flow-splits in the network is small in both cases. For larger network sizes, however, Figures 5.5 and 5.8 show that both similarity thresholds ($S_T = 5\%, 15\%$) drastically decrease the number of calls to the model and show a corresponding increase in speed-up (Figures 5.4 and 5.7). Specifically, at the highest similarity threshold ($S_T = 15\%$), the proposed technique applied to the 3-tier and 2-tier achieve an average speed-up in execution time compared to ns-3 of 13000$\times$ and 12000$\times$ respectively. While the analytical model with no similarity achieves an average speed-up of only 3000$\times$ for both 3-tier and 2-tier, but 2-tier is limited to a maximum node count of 128.

As such, Figures 5.4 and 5.5 corroborate our hypothesis of flow-splits

having similar performance characteristics due to the high amount of symmetry and load balancing in DCNs.

MAPE of the Analytical Model:

The similarity analysis methodology is more approximate than the baseline queueing theoretic approach, because of using a cluster to approximate other similar flow-splits. This approximation impacts MAPE. The similarity threshold is configurable, allowing the user to trade off speedup for MAPE impact. We evaluate the approach at 3 increasingly stringent thresholds and demonstrate that the MAPE difference for 3-tier and 2-tier fat-tree, even at a threshold of 15% yields $<$ 2% MAPE difference vs the baseline approach (as shown in Figure 5.6 and 5.9). Furthermore, in both the fat-tree topologies, the increase in the error is due to the exorbitantly high number of flow-splits coupled with a higher similarity threshold (S_T = 15%), which results in a higher approximation error compared to the similarity threshold (S_T = 5%).

Absolute Run Time Analysis

This section presents the absolute runtimes of the analytical model with no similarity and with similarity. Table 5.3 provides the range (Maximum - Minimum) of the runtimes among all 1992 simulations for 3-tier fat-tree and 1733 runs for 2-tier fat-tree. The maximum run time taken by ns-3 simulation is 49 hours and 5 hours for 3-tier and 2-tier fat-tree respectively

whereas the analytical model with no similarity takes a maximum of 3 hours and 12 hours. Our proposed technique brings down the runtime of 3-tier and 2-tier fat-tree from a maximum of 49 hours and 5 hours to 72 seconds and 12 seconds respectively. The speed-up seen in a 3-tier and 2 tier fat-tree is around 2500× and 3500× respectively. These speed-ups are expected as the number of calls to the analytical model in a 3-tier and 2 tier fat-tree is lesser by around 2000× and 3000× compared to ns-3, respectively. Additionally, due to the p^2 parallelism seen in a 2-tier fat-tree the total number of calls with no similarity is significantly higher than 3-tier fat-tree.

5.5 Conclusion

Data centers are essential in providing computing, storage, and memory resources to large organizations serving many users. Fast and accurate

Table 5.3: Range of absolute runtime between simulation and analytical model (with and without similarity).

Topology (Max -Min Nodes)	Absolute Run Time (Max-Min)				
	Simulation (ns-3)	Analytical (NS)	Analytical ($S_T = 0\%$)	Analytical ($S_T = 5\%$)	Analytical ($S_T = 15\%$)
3-tier (2000 -16)	49hrs - 56sec	3hrs - 4ms	9min - 2ms	72sec - 2ms	72sec - 2ms
2-tier (128 -16)	5hrs - 5sec	12hrs - 1ms	3hrs - 1ms	23sec - 1ms	12sec - 1ms

performance analysis techniques are vital for rapidly exploring design options and optimizing architectural configurations, especially in datacenter networks with thousands of nodes. To tackle this challenge, we propose a novel similarity-based method that leverages the high degree of similarity available in the network. We implement a technique that clusters flows with similar characteristics based on a user-defined threshold. This technique significantly accelerates performance analysis, achieving speed-ups of around 3000x compared to the analytical model without similarity. In our evaluations, this technique achieves around 3000$\times$ speed-up while maintaining less than 1% degradation in MAPE accuracy. We evaluated the technique for 2-tier and 3-tier fat-tree topologies with sizes ranging from 16 to 2000 nodes.

6 CONCLUSION OF THE DISSERTATION

In conclusion, this dissertation makes significant contributions to the field of modern processor analysis, with a primary focus on Network-on-Chip (NoC) architectures and data center networks (DCNs). The key findings and advancements can be summarized as follows:

NoC Performance Analysis: Introduces an analytical model addressing the limitations of current NoC performance analysis models. Proposes a technique for multi-layer priority-aware NoCs with deflection routing, showcasing superior accuracy in assessing various traffic scenarios.

Proactive Congestion Control in NoCs: Presents a proactive congestion control technique for NoCs using a supervised learning framework and time reversal technique. Achieves substantial improvement (up to 114%) in memory read bandwidth for realistic workloads with minimal overhead (less than 0.01%).

Scalable Performance Analysis for DCNs: Introduces MQL, a scalable methodology for performance analysis in DCNs, combining queuing theory and machine learning. Achieves an impressive speedup (up to four orders of magnitude) compared to traditional simulations, estimating latencies with less than 3% average error.

Similarity-Based Method for DCN Performance Analysis: Proposes a novel similarity-based method to accelerate performance analysis in DCNs by clustering flows with similar characteristics. Demonstrates a remarkable

3000x speed-up compared to analytical models without similarity, with less than 1% degradation in MAPE accuracy.

Future Work: The dissertation opens avenues for future research to further enhance and extend its contributions: Explore the application of the proposed techniques on diverse network topologies beyond fat-tree in data center networks. Validate and optimize the proposed methodologies in scenarios with highly constrained queue buffer resources to ensure scalability. Develop analytical performance analysis techniques for modeling TCP with congestion control in data center networks, considering varied DCN requirements.

These future directions aim to refine and broaden the applicability of the proposed models and methodologies, paving the way for continued advancements in the design, analysis, and optimization of modern processors and data center networks.

A.1 Appendix

Implementation of a steady-state maximum entropy solution in a stochastic system requires estimation of Lagrangian coefficients g, x, and y related to output parameters such as utilization and mean queue length. We make this estimation by using a related closed network at equilibrium with infinite queue [29] Therefore, we find the Lagrangian coefficient x by using the mean infinite queue length $< n >$ from Equation A.1, in Equation A.2 [30].

$$< n >= \lambda \widehat{W} + \rho \tag{A.1}$$

$$x = \frac{< n > -\rho}{< n >} \tag{A.2}$$

The Lagrangian coefficient x is then used to find the the mean finite queue length $< n >_N$ given by the Equation A.3.

$$< n >_N = \frac{\rho}{1 - \rho^2 x^{N-1}} \left\{ \frac{1 - x^N}{1 - x} - N\rho x^{N-1} \right\} \tag{A.3}$$

Using the mean finite queue length $< n >_N$ from Equation A.3, we find the Lagrangian coefficient x_N, y, and g as shown in Equation A.4, A.5

and A.6 respectively.

$$x_N = \frac{< n >_N - \rho}{< n >_N} \tag{A.4}$$

$$y = \frac{1-\rho}{1-x_N} \tag{A.5}$$

$$g = \frac{\rho(1-x_N)}{x_N(1-\rho)} \tag{A.6}$$

These coefficients then help us in finding the queue occupancy distribution given by the Equation A.7.

$$p_N(n) = \begin{cases} C_N \widehat{p}(n), & \text{for } n = 0, 1, \ldots, N-1 \\ 1 - \sum_{n=0}^{N-1} p_N(n), & \text{for } n = N \end{cases} \tag{A.7}$$

where N is the queue length and the intermediate variables C_N and $\widehat{p}(n)$ are found as follows:

$$C_N = \frac{p_N(0)}{1-\rho} \tag{A.8}$$

$$\widehat{p}(n) = (1-\rho) g^{h(n)} x_{N\,ds}^{n} \tag{A.9}$$

Here, $h(n)$ is the unit step function as shown in Equation A.10.

$$h(n) = \begin{cases} 0, & \text{for } n = 0 \\ 1, & \text{for } n > 0 \end{cases} \tag{A.10}$$

From the resulting probability distribution given by Equation A.7,

we acquire the probability of the queue being full as $p_N(N)$, where N represents the queue size.

BIBLIOGRAPHY

[1] R. Akbar and F. Safaei. A Novel Heterogeneous Congestion Criterion for Mesh-based Networks-on-Chip. *Microprocessors and Microsystems*, 84, 2021.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[3] M. Arafa and thers. Cascade Lake: Next Generation Intel Xeon Scalable Processor. *IEEE Micro*, 39(2):29–36, 2019.

[4] B. A. P. C. (BAPCo). Benchmark, sysmark2014. `http://bapco.com/products/sysmark-2014`, accessed 27 May 2020.

[5] Y. Ben-Itzhak, I. Cidon, and A. Kolodny. Delay Analysis of Wormhole Based Heterogeneous NoC. In *Proc. ACM/IEEE Int. Symp. on Networks-on-Chip*, pages 161–168, 2011.

[6] D. Bertsekas and R. Gallager. *Data networks*. Athena Scientific, 2021.

[7] D. P. Bertsekas, R. G. Gallager, and P. Humblet. *Data Networks*, volume 2. Prentice-Hall International New Jersey, 1992.

[8] M. Besta and T. Hoefler. Slim Fly: A Cost Effective Low-Diameter Network Topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359, 2014.

[9] N. Binkert et al. The Gem5 Simulator. *SIGARCH Comp. Arch. News*, May. 2011.

[10] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 2006.

[11] J. Bucek, K.-D. Lange, and J. v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018.

[12] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching Output Queueing with a Combined Input/Output-Queued Switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999.

[13] B. K. de Aquino Afonso and L. Berton. QT-RouteNet: Improved GNN Generalization to Larger 5G Networks by Fine-Tuning Predictions from Queuing Theory. *arXiv e-prints*, pages arXiv–2207, 2022.

[14] A. Dietmüller, S. Ray, R. Jacob, and L. Vanbever. A New Hope for Network Model Generalization. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 152–159, 2022.

[15] J. Doweck et al. Inside 6th-generation Intel Core: New Microarchitecture Code-named Skylake. *IEEE Micro*, (2):52–62, 2017.

[16] M. Ferriol-Galmés, K. Rusek, J. Suárez-Varela, S. Xiao, X. Shi, X. Cheng, B. Wu, P. Barlet-Ros, and A. Cabellos-Aparicio. RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation. In *IEEE Conference on Computer Communications*, pages 2018–2027, 2022.

[17] D. H. Hagos, P. E. Engelstad, A. Yazid, and C. Griwodz. A Deep Learning Approach to Dynamic Passive RTT Prediction Model for TCP. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pages 1–10. IEEE, 2019.

[18] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[19] C. Hopps and D. Thaler. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991, Nov. 2000.

[20] B. Jaeger, M. Helm, L. Schwegmann, and G. Carle. Modeling TCP Performance using Graph Neural Networks. In *Proceedings of the 1st International Workshop on Graph Neural Networking*, pages 18–23, 2022.

[21] R. Jain, K. Ramakrishnan, and D.-M. Chiu. Congestion Avoidance in Computer Networks with a Connectionless Network Layer. *arXiv preprint*, 1998.

[22] J. Jeffers, J. Reinders, and A. Sodani. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann, 2016.

[23] N. Jiang et al. A Detailed and Flexible Cycle-accurate Network-on-chip Simulator. In *2013 IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96.

[24] Z. Jiong, Z. Shu-jing, and Z. Qi-gang. An Adapted Full Model for TCP Latency. In *2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM'02. Proceedings.*, volume 2, pages 801–804. IEEE, 2002.

[25] S. A. Kassing. Static yet flexible: Expander data center network fabrics. Master's thesis, ETH-Zürich, 2017.

[26] A. E. Kiasari, Z. Lu, and A. Jantsch. An Analytical Latency Model for Networks-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, 2012.

[27] J. Kim, W. Dally, S. Scott, and D. Abts. Cost-Efficient Dragonfly Topology for Large-Scale Systems. *IEEE Micro*, 29(1):33–40, 2009.

[28] J. Kim, W. J. Dally, and D. Abts. Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks. In *Proceedings of the Annual International Symposium on Computer Architecture*, pages 126–137, 2007.

[29] D. D. Kouvatsos. Maximum Entropy and the G/G/1/N Queue. volume 23, pages 545–565. Springer, 1986.

[30] D. D. Kouvatsos. Entropy Maximisation and Queuing Network Models. *Annals of Operations Research*, 48(1):63–126, 1994.

[31] D. D. Kouvatsos, S. Assi, and M. Ould-Khaoua. Performance Modelling of Wormhole-routed Hypercubes with Bursty Traffic and Finite Buffers. 2005.

[32] D. D. Kouvatsos, P. H. E. Georgatsos, and N. M. Tabet-Aouel. *A Universal Maximum Entropy Algorithm for General Multiple Class Open Networks with Mixed Service Disciplines*, pages 397–419. Springer US, Boston, MA, 1989.

[33] K. Lakhotia, M. Besta, L. Monroe, K. Isham, P. Iff, T. Hoefler, and F. Petrini. PolarFly: A Cost-Effective and Flexible Low-Diameter Topology. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 146–160. IEEE Computer Society, 2022.

[34] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron. Understanding Rack-Scale Disaggregated Storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, 2017.

[35] C.-J. M. Liang, Z. Fang, Y. Xie, F. Yang, Z. L. Li, L. L. Zhang, M. Yang, and L. Zhou. On Modular Learning of Distributed Systems for Predicting {End-to-End} Latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1081–1095, 2023.

[36] J. D. Little. A proof for the queuing formula: L= λ w. *Operations research*, 9(3):383–387, 1961.

[37] J. D. Little and S. C. Graves. Little's Law. In *Building Intuition*, pages 81–100. Springer, 2008.

[38] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu. Memory Disaggregation: Research Problems and Opportunities. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 1664–1673, 2019.

[39] W. LLC. Skylake (server) - microarchitectures - intel. `https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)`, accessed 22 Sept 2022.

[40] S. K. Mandal, R. Ayoub, M. Kishinevsky, M. M. Islam, and U. Y. Ogras. Analytical Performance Modeling of NoCs under Priority Arbitration and Bursty Traffic. *IEEE Embedded Systems Letters*, 2020.

[41] S. K. Mandal, R. Ayoub, M. Kishinevsky, and U. Y. Ogras. Analytical performance models for nocs with multiple priority traffic classes. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–21, 2019.

[42] S. K. Mandal, A. Krishnakumar, R. Ayoub, M. Kishinevsky, and U. Y. Ogras. Performance analysis of priority-aware nocs with deflection routing under traffic congestion. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[43] P. Maniotis, L. Schares, B. G. Lee, M. A. Taubenblatt, and D. M. Kuchta. Toward Lower-Diameter Large-Scale HPC and Data Center Networks with Co-Packaged Optics. *Journal of Optical Communications and Networking*, 13(1):A67–A77, 2020.

[44] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives. *IEEE Trans. on Computer-Aided Design of Integrated Circ. and Syst.*, 28(1):3–21, 2008.

[45] C. Minkenberg, N. Farrington, A. Zilkie, D. Nelson, C. P. Lai, D. Brunina, J. Byrd, B. Chowdhuri, N. Kucharewski, K. Muth, A. Nagra, G. Rodriguez, D. Rubi, T. Schrans, P. Srinivasan, Y. Wang, C. Yeh, and A. Rickman. Reimagining Datacenter Topologies with Integrated Silicon Photonics. *Journal of Optical Communications and Networking*, 10(7):126–139, 2018.

[46] M. Mitzenmacher and R. Rajaraman. Towards More Complete Models of TCP Latency and Throughput. *The Journal of Supercomputing*, 20:137–160, 2001.

[47] S. Y. Narayana, J. Tong, A. Krishnakumar, N. Yildirim, E. Shriver, M. Ketkar, and U. Y. Ogras. MQL: ML-Assisted Queuing Latency Analysis for Data Center Networks. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 50–60. IEEE, 2023.

[48] J. Navaridas, J. A. Pascual, A. Erickson, I. A. Stewart, and M. Luján. INRFlow: An Interconnection Networks Research Flow-level simulation framework. *Journal of parallel and distributed computing*, 130:140–152, 2019.

[49] N. Nikitin and J. Cortadella. A Performance Analytical Model for Network-on-Chip with Constant service time Routers. In *Proc. Int. Conf. on Computer-Aided Design*, pages 571–578, 2009.

[50] nsnam. ns-3, 2017. `https://www.nsnam.org`. [Online; last accessed 01-Dec-2022.].

[51] G. Nychis, C. Fallin, T. Moscibroda, and O. Mutlu. Next Generation On-chip Networks: What Kind of Congestion Control do we Need? In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[52] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip Networks from a Networking perspective: Congestion and Scalability in Many-core Interconnects. *ACM SIGCOMM computer communication review*, 42(4):407–418, 2012.

[53] U. Y. Ogras, P. Bogdan, and R. Marculescu. An Analytical Approach for Network-on-Chip Performance Analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):2001–2013, 2010.

[54] U. Y. Ogras and R. Marculescu. Prediction-based Flow Control for Network-on-Chip Traffic. In *Proceedings of the 43rd annual design automation conference*, pages 839–844, 2006.

[55] OpenSim Ltd. OMNeT++, 2018. `https://omnetpp.org`. [Online; last accessed 01-Dec-2022.].

[56] M. S. Papamarcos and J. H. Patel. A Low-overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proceedings of the 11th annual international symposium on Computer architecture*, pages 348–354, 1984.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[58] A. Petlund, P. Halvorsen, P. F. Hansen, T. Lindgren, R. Casais, and C. Griwodz. Network Traffic from Anarchy Online: Analysis, Statistics and Applications: A Server-Side Traffic Trace. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 95–100, 2012.

[59] J. Postel. The TCP Maximum Segment Size and Related Topics. RFC 879, Nov. 1983.

[60] G. Pujolle and W. Ai. A Solution for Multiserver and Multiclass Open Queueing Networks. *INFOR: Information Systems and Operational Research*, 24(3):221–230, 1986.

[61] Z. Qian, P. Bogdan, C.-Y. Tsui, and R. Marculescu. Performance Evaluation of NoC-based Multicore Systems: From Traffic Analysis to NoC Latency Modeling. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 21(3):1–38, 2016.

[62] E. Rotem and S. P. Engineer. Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency. In *Intel Developer Forum*, 2015.

[63] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

[64] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[65] S. Sarangi and B. Baas. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-submicron Era. In *2021 IEEE ISCAS*, pages 1–5, 2021.

[66] A.-H. Smai and L.-E. Thorelli. Global Reactive Congestion Control in Multicomputer Networks. In *Proc. Fifth International Conf. on High Perform. Comput. (Cat. No. 98EX238)*, pages 179–186. IEEE, 1998.

[67] A. Sriraman and A. Dhanotia. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 733–750, 2020.

[68] B. K. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Madnani. Genesis: a system for large-scale parallel network simulation. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, pages 89–96, 2002.

[69] S. M. Tam et al. SkyLake-SP: A 14nm 28-Core Xeon® Processor. In *2018 IEEE ISSCC*, pages 34–36, 2018.

[70] M. Taubenblatt, P. Maniotis, and A. Tantawi. Optics enabled networks and architectures for data center cost and power efficiency. *Journal of Optical Communications and Networking*, 14(1):A41–A49, 2022.

[71] L. N. Vaserstein. Markov Processes over Denumerable Products of Spaces, Describing Large Systems of Automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[72] B. Wang, Z. Lu, and S. Chen. ANN Based Admission Control for on-chip Networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[73] Y. Wang, D. Dong, and F. Lei. Understanding Node Connection Modes in Multi-Rail Fat-tree. *Journal of Parallel and Distributed Computing*, 2022.

[74] K. Wen, P. Samadi, S. Rumley, C. P. Chen, Y. Shen, M. Bahadori, K. Bergman, and J. Wilke. Flexfly: Enabling a Reconfigurable Dragonfly through Silicon Photonics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 166–177, 2016.

[75] Q. Yang, X. Peng, L. Chen, L. Liu, J. Zhang, H. Xu, B. Li, and G. Zhang. DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility. In *Proceedings of the ACM SIGCOMM Conference*, pages 441–457, 2022.

[76] Q. Zhang, K. K. Ng, C. Kazer, S. Yan, J. Sedoc, and V. Liu. MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning. In *Proceedings of the ACM SIGCOMM Conference*, pages 287–304, 2021.

[77] H. Zhao, N. Bagherzadeh, Q. Wang, and Y. Wang. A Fine-Grained Source-Throttling Method for Mesh Architectures. *IEEE Access*, 8:33101–33112, 2020.

[78] K. Zhao, P. Goyal, M. Alizadeh, and T. E. Anderson. Scalable Tail Latency Estimation for Data Center Networks. arXiv, 2022. Available `https://arxiv.org/abs/2205.01234`.

[79] K. Zhao, P. Goyal, M. Alizadeh, and T. E. Anderson. Scalable tail latency estimation for data center networks. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 685–702, 2023.

[80] Z. Zhu, A. X. Liu, F. Zhang, and F. Chen. Fpga resource pooling in cloud computing. *IEEE Transactions on Cloud Computing*, 9(02):610–626, 2021.