UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BARCELONATECH

THESIS FOR THE DOCTORAL DEGREE IN COMPUTER ARCHITECTURE

Network Modeling using Graph Neural Networks

---

by:

Miquel Ferriol Galmés

Advisor: Dr. Albert Cabellos-Aparicio
Co-Advisor: Dr. Pere Barlet-Ros

Thesis for the Doctoral Degree in Computer Architecture

Barcelona Neural Networking Center (BNN)
Departament d'Arquitectura de Computadors (DAC)

February 2024

"The common facts of today are the products of yesterday's research."

Duncan MacDonald

Acknowledgements

I would like to extend my sincere gratitude to my advisors, Prof. Albert Cabellos and Prof. Pere Barlet, for their unwavering guidance and support throughout my PhD journey. Their expertise, encouragement, and dedicated support have been indispensable in helping me navigate and successfully complete my thesis. I appreciate the opportunities they provided for presenting my work and their support in overcoming challenges. I am truly thankful for the knowledge and experience I have gained under their supervision, and I will always be grateful for their mentorship.

I would like to extend my gratitude to Prof. Krzysztof Rusek, whose invaluable work, expertise, and assistance have served as a crucial pillar for this thesis. His contributions and guidance have greatly enriched the research, and I am thankful for the support and insights he provided throughout the process.

Thanks to all the members and former members of the Barcelona Neural Networking Center and the Computer Architecture Department: Sergi Abadal, José Suarez, Jordi Paillisé, Paul Almasan, Albert López, Guillermo Bernárdez, David Pujol, Hamid Latif, Berta Serracanta, Axel Wassington, and Carlos Güemes that helped and guided me during the research done in this work. Your kindness, generosity, and camaraderie have made my PhD journey a truly memorable experience.

I would also like to express my deepest appreciation to all the members of Hospital Clínic-IDIBAPS where I spent 6 months developing my skills and collaborating with great researchers. Thanks for giving me this unique opportunity to learn about the health domain. I am especially grateful to Dr. Laura Mezquita, for their valuable guidance and support during my stay.

This thesis has been partially funded by the Spanish I+D+i project TRAINER-A (ref.PID2020-118011GB-C21), funded by MCIN/ AEI/10.13039/ 501100011033. This work is also partially funded by the Catalan Institution for Research and Advanced Studies (ICREA) and the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia and the European Social Fund.

I would like to express my sincere gratitude to all my family and friends for your unwavering love and encouragement throughout my PhD

journey. Your support has been a source of comfort and strength during the challenging times, and I am truly grateful to have you all in my life. I could not have accomplished this milestone without your support, and I am deeply thankful for all that you have done for me.

Finalment, m'agradaria agrair l'immens suport i motivació rebut per la meva familiar més directe. Per una part, als meus pares Guillem i Catalina Inés per haver-me ajudat a superar els moments difícils que han aparegut durant el desenvolupament d'aquesta tesis. Als meus germans Toni i Maria Margalida i al meu cunyat Pedro Carreres pels bons moments compartits que han estat una part indispensable per superar aquest camí. Finalment, als nouvinguts Josep i Maria Antònia Carreras, dessitjar-li el millor d'aquest món, esper que la vida us somrigui tal i com m'ha fet a mi. *Gràcies!*

Abstract

Network modeling is central to the field of computer networks. Models are useful in researching new protocols and mechanisms, allowing administrators to estimate their performance before their actual deployment in production networks. Network models also help to find optimal network configurations, without the need to test them in production networks.

Arguably, the most prevalent way to build these network models is through the use of discrete event simulation (DES) methodologies which provide excellent accuracy. State-of-the-art network simulators include a wide range of network, transport, and routing protocols, and are able to simulate realistic scenarios. However, this comes at a very high computational cost that depends linearly on the number of packets being simulated. As a result, they are impractical in scenarios with realistic traffic volumes or large topologies. In addition, and because they are computationally expensive, they do not work well in real-time scenarios.

Another network modeling alternative is Queuing Theory (QT) where networks are represented as inter-connected queues that are evaluated analytically. While QT solves the main limitation of DES, it imposes strong assumptions on the packet arrival process, which typically do not hold in real networks.

In this context, Machine Learning (ML) has recently emerged as a practical solution to achieve data-driven models that can learn complex traffic models while being extremely accurate and fast. More specifically, Graph Neural Networks (GNNs) have emerged as an excellent tool for modeling graph-structured data showing outstanding accuracy when applied to computer networks. However, some challenges still persist:

1. Queues and Scheduling Policies: Modeling queues, scheduling policies, and Quality-of-Service (QoS) mappings within GNN architectures poses another challenge, as these elements are fundamental to network behavior.

2. Traffic Models: Accurately modeling realistic traffic patterns, which exhibit strong autocorrelation and heavy tails, remains a challenge for GNN-based solutions.

3. Training and Generalization: ML models, including GNNs, require representative training data that covers diverse network operational scenarios. Creating such datasets from real production networks is unfeasible, necessitating controlled testbeds. The challenge lies in designing GNNs capable of accurate estimation in unseen networks, encompassing different topologies, traffic, and configurations.

4. Generalization to Larger Networks: Real-world networks are often significantly larger than testbeds. Scaling GNNs to handle networks with hundreds or thousands of nodes is a pressing challenge, one that requires leveraging domain-specific network knowledge and novel architectural approaches.

This dissertation represents a step forward in harnessing Graph Neural Networks (GNN models) for network modeling, by proposing a new GNN-based architecture with a focus on addressing these critical challenges while being fast and accurate. Additionally, this thesis explores how the proposed GNN architecture can serve as a Network Digital Twin (NDT), which, when paired with an optimizer, illustrates the potential of these types of models for SLA-driven optimization.

Resumen

El modelado de redes es fundamental para el campo de las redes informáticas. Los modelos son útiles para investigar nuevos protocolos y mecanismos, lo que permite a los administradores estimar su rendimiento antes de su implementación real en las redes de producción. Los modelos de red también ayudan a encontrar configuraciones de red óptimas, sin necesidad de probarlas en redes de producción.

Podría decirse que la forma más frecuente de construir estos modelos de red es mediante el uso de metodologías de simulación de eventos discretos (DES) que proporcionan una precisión excelente. Los simuladores de redes de última generación incluyen una amplia gama de protocolos de red, transporte y enrutamiento y son capaces de simular escenarios realistas. Sin embargo, esto tiene un costo computacional muy alto que depende linealmente de la cantidad de paquetes que se simulan. Como resultado, no resultan prácticos en escenarios con volúmenes de tráfico realistas o topologías grandes. Además, y debido a que son computacionalmente costosos, no funcionan bien en escenarios de tiempo real.

Otra alternativa de modelado de redes es la teoría de colas (QT), donde las redes se representan como colas interconectadas que se evalúan analíticamente. Si bien QT resuelve la principal limitación de DES, impone fuertes suposiciones sobre el proceso de llegada de paquetes, que normalmente no se cumplen en las redes reales.

En este contexto, el aprendizaje automático (ML) ha surgido recientemente como una solución práctica para lograr modelos basados en datos que pueden aprender modelos de tráfico complejos y al mismo tiempo ser extremadamente precisos y rápidos. Más específicamente, las redes neuronales gráficas (GNN) se han convertido en una excelente herramienta para modelar datos estructurados en gráficos que muestran una precisión excepcional cuando se aplican a redes informáticas.

Esta disertación pretende ser un paso adelante en la aplicación de Graph Neural Networks para el modelado de redes. Sin embargo, aún persisten algunos desafíos:

1. Colas y políticas de programación: modelar colas, políticas de programación y asignaciones de calidad de servicio (QoS) dentro de las arquitecturas GNN plantea otro desafío, ya que estos elementos son fundamentales para el comportamiento de la red.

2. Modelos de tráfico: modelar con precisión patrones de tráfico realistas, que exhiben una fuerte autocorrelación y colas pesadas, sigue siendo un desafío para las soluciones basadas en GNN.

3. Capacitación y generalización: los modelos de ML, incluidos los GNN, requieren datos de capacitación representativos que cubran diversos escenarios operativos de red. Crear tales conjuntos de datos a partir de redes de producción reales es inviable y requiere bancos de pruebas controlados. El desafío radica en diseñar GNN capaces de realizar estimaciones precisas en redes invisibles, que abarquen diferentes topologías, tráfico y configuraciones.

4. Generalización a redes más grandes: las redes del mundo real suelen ser significativamente más grandes que los bancos de pruebas. Escalar las GNN para manejar redes con cientos o miles de nodos es un desafío apremiante, que requiere aprovechar el conocimiento de la red específico del dominio y enfoques arquitectónicos novedosos.

Esta disertación representa un paso adelante en el aprovechamiento de Graph Neural Networks para el modelado de redes, con un enfoque en abordar estos desafíos críticos.

Resum

El modelatge de xarxes és fonamental en el camp de les xarxes informàtiques. Els models són útils per investigar nous protocols i mecanismes, permetent als administradors estimar el seu rendiment abans del seu desplegament real a les xarxes de producció. Els models de xarxa també ajuden a trobar configuracions de xarxa òptimes, sense necessitat de provar-les a les xarxes de producció.

Sens dubte, la manera més freqüent de construir aquests models de xarxa és mitjançant l'ús de metodologies de simulació d'esdeveniments discrets (DES) que proporcionen una precisió excel·lent. Els simuladors de xarxa d'última generació inclouen una àmplia gamma de protocols de xarxa, transport i encaminament, i són capaços de simular escenaris realistes. Tanmateix, això comporta un cost computacional molt elevat que depèn linealment del nombre de paquets que s'estimulen. Com a resultat, són poc pràctics en escenaris amb volums de trànsit realistes o topologies grans. A més, i com que són computacionalment costosos, no funcionen bé en escenaris en temps real.

Una altra alternativa de modelització de xarxes és la teoria de la cua (QT) on les xarxes es representen com a cues interconnectades que s'avaluen analíticament. Tot i que QT soluciona la principal limitació del DES, imposa supòsits forts sobre el procés d'arribada de paquets, que normalment no es compleixen a les xarxes reals.

En aquest context, l'aprenentatge automàtic (ML) ha sorgit recentment com una solució pràctica per aconseguir models basats en dades que poden aprendre models de trànsit complexos alhora que són extremadament precisos i ràpids. Més concretament, les xarxes neuronals de gràfics (GNN) han sorgit com una excel·lent eina per modelar dades estructurades en gràfics que mostren una precisió excepcional quan s'apliquen a xarxes d'ordinadors.

Aquesta tesi pretén ser un pas endavant en l'aplicació de les xarxes neuronals de gràfics per al modelatge de xarxes. Tanmateix, encara persisteixen alguns reptes:

1. Cues i polítiques de programació: modelar cues, polítiques de programació i mapes de qualitat de servei (QoS) dins de les arquitectures GNN suposa un altre repte, ja que aquests elements són fonamentals per al comportament de la xarxa.

2. Models de trànsit: modelar amb precisió patrons de trànsit realistes, que presenten una forta autocorrelació i cues pesades, segueix sent un repte per a les solucions basades en GNN.

3. Entrenament i generalització: els models ML, incloses les GNN, requereixen dades d'entrenament representatives que cobreixen diversos escenaris operatius de la xarxa. La creació d'aquests conjunts de dades a partir de xarxes de producció reals és inviable, ja que requereix bancs de proves controlats. El repte rau a dissenyar GNN capaços d'estimar amb precisió en xarxes no vistes, que abastin diferents topologies, trànsit i configuracions.

4. Generalització a xarxes més grans: les xarxes del món real sovint són significativament més grans que els bancs de proves. Escalar les GNN per gestionar xarxes amb centenars o milers de nodes és un repte urgent, que requereix aprofitar el coneixement de la xarxa específic del domini i enfocaments arquitectònics nous.

Aquesta tesi representa un pas endavant en l'aprofitament de les xarxes neuronals de grafs per al modelatge de xarxes, amb un enfocament a abordar aquests reptes crítics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network modeling is arguably one of the key tools when designing, building, and evaluating computer networks, even since the early days of networking [1]. Network models are used in protocol design, performance evaluation, or network planning to cite a few examples. The two most widespread network modeling techniques are analytical models based on Queuing Theory (QT), and packet-level simulators [2, 3].

However, the evolution of computer networks, especially concerning complexity and traffic characteristics, highlights some of the limitations of classical modeling techniques. Despite their tremendous success and general usage, some scenarios require more advanced techniques capable of accurately modeling complex traffic characteristics, while scaling to large real-world networks.

Especially, two relevant applications can benefit from advanced network modeling techniques: Network Digital Twins (NDT) [4], and network optimization tools. Commonly, an NDT is referred to as a virtual replica of a physical network that can accurately mimic its behavior and can make performance predictions for any given input condition (e.g., traffic, topology change, or new routing configuration). In other words, an NDT is an accurate network model that can support a wide range of network configurations and that can accurately model the complex non-linear behaviors behind real-world networks. As a result, NDTs can be used to produce accurate performance predictions, carry out what-if analysis, or perform network optimization by pairing it with an optimization algorithm [5, 4].

In the context of network optimization, *we can only optimize what we can*

model. Optimization algorithms operate by searching the network configuration space (e.g., to find an alternative routing scheme). For each configuration, a network model is used to estimate the resulting performance to see if it fulfills the optimization goal (e.g., minimize delay [6]). To achieve efficient online optimization, it is essential an accurate and fast network model.

Arguably, the most prevalent way to build these network models is through the use of discrete event simulation (DES) methodologies. Notable examples include ns-3 [7] and OMNeT++ [8]. While DES-based models can offer a degree of accuracy in certain cases, they also come with significant limitations. The main one is their computational performance which comes at a high computational cost. The cost of a simulator depends linearly on the number of packets forwarded, which can be in the range of millions per second on a single 1Gbps link. In consequence, they are slow and impractical when considering large networks with realistic traffic volumes. This also severely limits its applicability to online network optimization, given the hard time constraints of such types of applications.

Among the various network modeling techniques available, Queueing Theory (QT) [9] stands out as one of the most widely recognized and utilized methods. QT is a widely recognized and extensively applied network modeling technique, as computer networks inherently resemble interconnected queues where data packets await service. However, it imposes strong assumptions on the packet arrival process (Poisson traffic generation), which often is not sufficient to model real-world networks [10]. Internet traffic has been extensively analyzed in the past two decades [11, 12, 13, 14, 15] and, despite the community has not agreed on a universal model, there is consensus that in general aggregated traffic shows strong autocorrelation and a heavy-tail [16].

Another commonly used technique is fluid models that have become a popular alternative for network optimization [17, 18, 19, 20, 21], as they are simple but practical in some cases. However, fluid models assume constant per-link delays and do not consider the effects of queuing delays or network losses.

Machine Learning (ML) [22] provides a new breed of mechanisms to model complex systems. In particular, Deep Learning (DL) [23] has shown

the ability to extract deep knowledge from human-understandable descriptions of a system. This approach has proven to achieve unprecedented accuracy in modeling properties of complex systems, like proteins [24].

The main advantage of DL models is that they are *data-driven*. DL models can be trained with real-world data, without making assumptions about the system. This enables the building of models with unprecedented accuracy by effectively modeling the entire range of non-linear and multi-dimensional system characteristics. Computationally, DL is based on linear algebra and can take advantage of massive parallelism by leveraging dedicated hardware and compilers.

Network models using DL techniques have been attempted in the past [25, 26]. However, these proposals are based on traditional Neural Network (NN) architectures, such as Feed-Forward or Convolutional. Computer networks are inherently a network of queues, and as such are represented by *graphs*. Both Feed-forward and convolutional NN architectures are not designed to learn relational information and thus, the resulting network models are strongly limited. These models only function effectively within the networks they were trained on, rendering them impractical for accommodating even minor changes in network topology or routing configurations.

Within the field of DL, Graph Neural Networks (GNN) [27] have recently emerged as an effective technique to model graph-structured data. GNNs are tailored to understand the complex relationships between the elements of a graph. The main novelty of GNNs is that their internal architecture is dynamically assembled based on the elements and connections of input graphs, and this enables them to learn universal modeling functions that are invariant to graph isomorphism. GNNs are thus able to *generalize* over unseen data, which means that they can produce accurate estimates in different graphs not seen during training.

The novel GNN paradigm finally allows the application of ML in domains where data is essentially represented as graphs. As a consequence, at the time of this writing, substantial research efforts are being devoted to applying GNNs to different fields where data is fundamentally represented as graphs, such as chemistry [28], physics [29] and others [30] [31].

This dissertation aims to explore GNNs as a new network modeling

language with attractive advantages and characteristics. GNNs are purpose-built to comprehend graphs, and given that computer networks are essentially graphs of interconnected queues, they provide a well-suited modeling framework. Furthermore, this dissertation delves into the study of GNNs as a modeling tool, emphasizing the need for in-depth research and design that aligns with the fundamental behaviors of computer networks, as opposed to treating GNNs as black-box models that merely map data inputs to outputs. In contrast to traditional Deep Learning models, where the architecture is defined by the number of layers and neurons, GNNs are constructed on an ad-hoc basis, based on the elements and connections of the input graphs.

1.1 Motivation and Objectives

Even though in recent years notable breakthroughs have been made in the network modeling scene, these advancements have brought to light a series of challenges that GNN-based network modeling tools need to accomplish:

Queues and Scheduling policies: A fundamental aspect when modeling networks is considering the behavior of queues (e.g., number, size), scheduling policies (e.g., WFQ, DRR), and the mapping of traffic flows to different Quality-of-Service classes if any. QT is a well-established technique, and models have been developed to support a wide range of scheduling policies [32, 33]. The challenge is, how to represent queues and scheduling policies inside the GNN architecture.

Traffic models: A model is an abstraction of a system able to define the essential aspects of the system. In the case of computer networks, which handle a multitude of packets, an effective model necessitates a pertinent abstraction for these packets. This underlines the significance of supporting arbitrary stochastic traffic models. Experimental observations show that traffic on the Internet has strong autocorrelation and heavy-tails [16]. In this context, it is well-known that a main limitation of QT is that it fails to provide accurate estimates on realistic Markovian models with continuous state space, or non-Markovian traffic models. Analytical models for queues with general arrival processes are limited to infinite buffers [34], or they make some sort of approximation (e.g., asymptotic), which greatly differs from the

actual behavior of computer networks. The challenge for GNN-based modeling is, how can we design an architecture that can accurately model realistic traffic models?

Training and Generalization: One of the main differences between analytical modeling (e.g., QT) and ML-based modeling is that the latter requires training. In ML, training involves obtaining a *representative* dataset with network measurements. The dataset needs to include a broad spectrum of network operational regimes. In practice, this means testing how different congestion levels affect performance metrics (delay, jitter, and losses), evaluating how different queuing policies affect performance, or testing different routing policies, among others. Without this, the ML model is unable to learn and provide accurate estimates. Generating this training dataset from networks in production is typically unfeasible, as it would require artificially generated configurations (e.g., queue scheduling, routing) that lead to service disruption. A reasonable alternative is to create these datasets in controlled testbeds, where it is possible to use different traffic models and implement a broad set of configurations. Thus, the GNN model can be trained on samples from this testbed, and then be applied to real networks. Hence, the research challenge is: how to design a GNN able to provide accurate estimates in networks not seen during training? This includes topologies, traffic, and configurations (e.g., queue scheduling, routing) different from those seen in the training network testbed.

Generalization to larger networks: From a practical standpoint, the GNN model must also generalize to *larger* networks. Real-world networks include hundreds or thousands of nodes, and building a network testbed at this scale is typically unfeasible. As a result, the GNN model should be able to generalize from small network testbeds to considerably larger networks. Generalizing to larger networks – or graphs, in general – is currently an open research challenge in the field of GNNs.

The objective of this dissertation is to develop a new efficient network model leveraging Graph Neural Networks. Specifically, it focuses on the design of a novel architecture capable of supporting different scheduling policies and traffic models, while being able to generalize not only to topologies of similar sizes but to larger ones.

1.2 Contributions

This thesis aims to contribute meaningfully to the field of network modeling by leveraging the capabilities of Graph Neural Networks (GNNs). Within this pursuit, it presents a series of contributions that seek to enhance the state of network modeling in various important points:

Advanced GNN-based Modeling: The thesis leverages the use of Graph Neural Networks (GNNs) as a powerful tool for network modeling. GNNs offer an innovative approach, designed to understand and model the intricate relationships within network graphs, taking into account several elements found in a real network (e.g., queues, links, traffic flows).

Enhanced Traffic Modeling: The thesis addresses the challenge of accurately modeling network traffic. Unlike traditional approaches limited to Markovian models, the GNN-based models developed in this thesis support a wide range of traffic models, including those with strong autocorrelation and high variance, which more faithfully capture real-world traffic behaviors.

Effective Generalization and Scalability: One of the main limitations of existing models, particularly in Deep Learning (DL), is their inability to generalize beyond the training data. This constraint often arises from the fixed inputs commonly used in DL models, which typically entail static topologies or predetermined vector/matrix structures, such as those found in fixed-size images. This thesis overcomes this limitation by designing GNN architectures that provide accurate estimates in network scenarios not seen during training. Notably, these models are trained in controlled small-scale testbeds and demonstrate the remarkable ability to generalize their findings to network scenarios orders of magnitude larger. This includes various topologies, traffic patterns, and configurations, making them highly scalable and applicable to real-world large-scale networks.

Optimization Techniques: The thesis expands the scope of network modeling by introducing optimization techniques. These techniques are combined with GNN-based models to navigate complex scenarios, ensuring the fulfillment of Service Level Agreements (SLAs) even under highly congested network conditions.

Flow-aware Modeling: The thesis introduces flow-aware modeling, enabling the understanding and modeling of network flows in the time domain. This approach supports dynamic flow creation, modification, and termination, as well as changes in flow characteristics. It provides per-flow metrics, such as delay and jitter, influenced by flow dynamics and network configurations.

In short, this thesis proposes some advances in the state of network modeling, leveraging the capabilities of GNNs to tackle complex challenges in modeling traffic, achieving effective generalization and scalability to larger networks, optimizing network performance, and accurately modeling network flows.

1.3 Outline of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2 - Background

In this chapter, we explore the application of Graph Neural Networks (GNNs) in various specific domains. We delve into how GNNs offer unique advantages for modeling networked scenarios, with their ability to generalize across diverse network configurations, topologies, and beyond their training domain. Additionally, we examine the utilization of Machine Learning in computer networks, emphasizing the pivotal role of GNNs in this context. The chapter provides an in-depth understanding of GNNs, their message-passing interface, and their relevance to computer networks, setting the stage for the research conducted in this thesis.

Chapter 3 - Limitations and Challenges of Current Network Modeling Techniques

In this chapter, we embark on a comprehensive exploration of current network modeling techniques, ranging from packet-level simulators to analytical models and DL ones. The primary objective of this chapter is to assess the practicality of these established methods, showcasing their strengths and limitations. By dissecting these techniques, we aim to clear up the unresolved challenges and constraints that are intrinsic to network modeling.

This in-depth analysis serves to provide a foundation for understanding the landscape in which our research operates, focusing on the specific concerns addressed in this thesis.

Chapter 4 - RouteNet-Darwin: Advancing Towards Quality of Service-Aware Network Modeling

In the realm of network modeling, this chapter introduces RouteNet-Darwin, a network model designed to provide precise predictions of network performance. RouteNet-Darwin leverages Graph Neural Networks (GNNs) to understand the intricate relationships between network topology, routing configurations, queue scheduling, and traffic matrix. This chapter provides a comprehensive insight into RouteNet-Darwin's architecture, notation, and inner workings, demonstrating its capacity to overcome the complexities of real-world networks.

Chapter 5 - RouteNet-Erlang: Enhancing Network Modeling through Scheduling, Traffic Models, and Generalization

This chapter presents RouteNet-Erlang (RouteNet-E), a groundbreaking GNN-based architecture designed for highly accurate performance evaluation in computer networks. Unlike conventional Queueing Theory models, RouteNet-E can handle a wider array of traffic models, including complex non-Markovian ones, closely mirroring real-world network traffic.

A notable strength of RouteNet-E is its ability to generalize effectively, providing precise performance estimations even for larger, structurally distinct networks not encountered during training. The chapter extensively benchmarks RouteNet-E against a state-of-the-art QT model across diverse network scenarios. RouteNet-E consistently outperforms the QT model, delivering remarkable accuracy in predicting delay, jitter, and packet losses.

Chapter 6 - RouteNet-Fermi: Unifying Scheduling, Traffic Models, and Generalization in Network Modeling

In this chapter, we introduce the core contribution of this thesis: RouteNet-F. This novel model represents the culmination of insights gained from earlier iterations of RouteNet. RouteNet-F is a powerful GNN that encompasses the ability to comprehend diverse traffic models and intricate queue scheduling policies, all while demonstrating exceptional generalization capabilities. Notably, it showcases the capacity to extrapolate its knowledge to

scenarios vastly larger than those encountered during its training, setting it apart as a versatile and adaptable tool for network modeling.

Chapter 7 - ST-RouteNet: Adding the Temporal Dimension

In this chapter, we discuss the limitations of existing models in representing network traffic as "Traffic Matrices." These models excel in modeling network performance metrics but are constrained by their oversimplification. We delve into the significance of representing network traffic as "flows" and the role they play in networking, with an emphasis on their use in applications and network optimization. Notably, this chapter addresses the unique challenges posed by flows' dynamic nature and introduces the need for understanding the time dimension in modeling network traffic.

Here, we present a novel GNN-based model that operates at the flow level. This model is designed to understand and model flows, working in the time domain. It accommodates dynamic flows, considers their creation and destruction, and handles flows with changing characteristics. The chapter discusses how this model enables the computation of per-flow metrics, such as delay and jitter, based on the input flow dynamics and network configurations, including routing and network topology.

Chapter 8 - Network Performance Digital Twins

In this chapter, we delve into the concept of Network Performance Digital Twins (NPDTs) within the context of the evolving digital landscape. Digital Twins, a transformative concept spanning various industries, have made significant inroads. In particular, we focus on the Network Digital Twin (NDT) and its most common iteration, the Network Performance Digital Twin (NPDT), which predicts network performance metrics. The NPDT's pivotal role in understanding, optimizing, and proactively managing real-world networks is emphasized. It enables decision-makers to explore scenarios, perform what-if analyses, and optimize network configurations without impacting real networks. We discuss how Machine Learning (ML) models offer a promising solution for constructing reliable and efficient NPDTs, striking a balance between accuracy and speed. Additionally, we explore the potential of RouteNet family models as architectural candidates for NPDTs and outline the requirements for implementing this form of Digital Twin.

Chapter 9 - Conclusions and Future Work

In this final chapter, we draw conclusions from the dissertation's findings and outline potential avenues for future research in the field of network performance modeling.

Chapter 2

Background

In the context of network modeling, this thesis adopts an innovative approach by harnessing the power of a special Machine Learning (ML) mechanism. At its core, this thesis employs Graph Neural Networks (GNNs) as a fundamental component for modeling networked scenarios. GNNs are a class of neural network architectures explicitly crafted to work with graph-structured data [35]. Their unique ability to generalize effectively across diverse network scenarios, encompassing topologies and configurations that lie beyond their training domain, makes them a formidable tool suited for network modeling. Notably, GNNs also shine in terms of operational efficiency, offering near real-time processing capabilities, often operating at millisecond timescales. In this thesis, GNNs will be employed as the fundamental framework for the research, serving as a strong basis for addressing the challenges aforementioned.

2.1 Graph Neural Networks

Graphs are a data structure that captures collections of objects, referred to as vertices, and the relationships between them, denoted by edges. This fundamental representation offers a simple yet powerful representation for modeling numerous phenomena and complex systems found in nature. Some examples of these systems are molecules and compounds in chemistry, physical systems exhibiting interactions among objects (e.g., gravitational forces in celestial bodies), social network structures capturing user relationships, and the components within computer network topologies (e.g., forwarding devices, links, routing paths). In general, a graph can provide a structured,

dense, and independent representation of a system composed of heterogeneous elements with arbitrary relationships between them.

In the context of Deep Learning (DL), a lot of effort has been devoted to designing neural network architectures, which are very suitable for understanding and extracting in-depth knowledge from various widely used data structures. For example, Recurrent Neural Networks (RNNs) [36] have been proven to be an effective way to model sequential data, that can understand and maintain the temporal relationship.

Convolutional Neural Network (CNN) [37] is another well-known neural network architecture with many applications in the field of image processing. These neural networks are based on convolution operations, which are applied to spatially ordered grids of elements. CNNs are invariant to spatial translation, so they can apply their previous knowledge to images that have modified the position and orientation of certain objects.

These two neural network architectures are designed with a special focus on understanding and learning the inherent relationship information present within the data structure itself. Specifically, RNNs excel at capturing temporal relationships within sequences, while CNNs are tailored to discern spatial relationships in images.

In this context, GNNs [27] [35] belong to a relatively recent family of neural networks. GNNs are purposefully crafted to effectively process and analyze data organized in graph structures. Notably, GNNs feature a modular neural network architecture that explicitly represents the individual elements (nodes) within graphs and the connections (edges) that link them together.

GNNs provide the unique capability to handle input graphs of varying sizes and accommodate arbitrary relationships between their elements. Perhaps even more notably, GNNs exhibit invariance to both node and edge permutations, a quality that enables them to abstract intricate knowledge regarding the connections among graph elements. This acquired knowledge can then be leveraged to make inferences about entirely new graphs that have not been encountered previously. GNNs have already demonstrated their effectiveness in solving diverse problems across various domains. For instance, they have been successfully applied in chemistry, specifically for

tasks like predicting molecular properties [28] or in physics, such as predicting trajectories in n-body systems [29]).

2.1.1 Message-passing Interface

One feature that distinguishes GNNs from other well-known neural network families is that their architecture is not fixed, but depends on the structure of the input graph. At the core of a GNN lies the fundamental concept of an iterative message-passing procedure.

In this procedure, each node within the input graph possesses a hidden state, denoted as h_i, which is typically represented as an n-element vector. This hidden state is initially set using the node's features during the Initialization Phase. Subsequently, during each iteration of the message-passing, nodes in the graph generate messages in the Message Phase, using their respective hidden states. These messages are then employed to update the current states of their neighboring nodes during the Update Phase. This iterative process is repeated multiple times until the node states reach a stable, fixed value. This mechanism enables GNNs to capture and propagate information across the graph structure, facilitating the extraction of meaningful insights from complex relational data. Ultimately, the updated and converged hidden states, obtained after the iterative message-passing process, are harnessed to make predictions. These predictions capture the essential information and insights derived from the input graph, allowing GNNs to perform various tasks, such as classification, regression, or graph generation, depending on the specific application and problem domain.

Initialization Phase

Initially, the GNN's input representation is initialized based on the information contained within the graph. This involves setting up the hidden states (h_i) of individual elements (nodes) in the graph. These hidden states are typically initialized using features (x_i) specific to each node as seen in Figure 2.1. Additionally, the edges (e_{ij}) in the graph can contain valuable information pertaining to the type of relationship between nodes.

Given that the hidden state (h_i) is typically represented as an n-element vector, which is often larger than the initial feature vector (x_i), it's common practice to initialize these vectors with zeros. The size of the h_i vector corresponds to the capacity to encode and store information about each element within the graph, allowing for more intricate representations as needed.

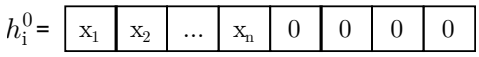


FIGURE 2.1: Hidden state (h_i^0) Initialization for a single node i.

Message-passing Phase

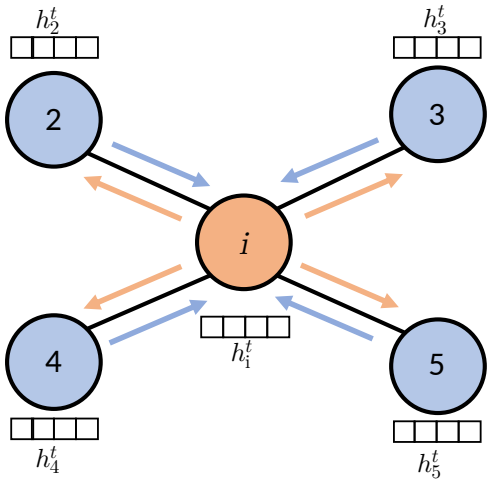


FIGURE 2.2: Message-passing scheme for a single node i and its neighbors.

The message-passing phase is a critical component of the GNN framework and involves an iterative process executed during T iterations. In Figure 2.3, a simplified illustration of a single iteration of this process for one node is provided. During this phase, three key functions are executed across the elements within the graph:

1. Message Function (M): This function is responsible for creating a message (m_{ij}^t) that encapsulates information regarding the various relationships existing between graph elements. To generate this message, it takes as input the hidden states of two neighboring nodes (h_i^t and h_j^t) and the properties that characterize the relationship between them (e_{ij}). This can be expressed as follows: $m_{ij}^t = M_{ij}(h_i^{t-1}, h_j^{t-1}, e_{ij})$.

2. Aggregation Function ($Aggr$): Once all messages have been created, they are combined using an aggregation function ($Aggr$), resulting in a fixed-size vector ($Aggr_i^t$) as output. Commonly, element-wise summation is employed as the aggregation function.

3. Update Function (U): The update function is applied to each node's hidden state individually (h_i^t) and takes into account the aggregation computed for that node during the current message-passing iteration ($Aggr_i^t$). Subsequently, it produces an updated hidden state for the node (h_i^t), which will be utilized in the subsequent message-passing iteration.

Importantly, it's worth noting that during each message-passing iteration, a node exclusively receives information from its neighboring nodes within the graph. This localized information exchange process enables the GNN to gradually refine its understanding of the relationships and patterns within the graph.

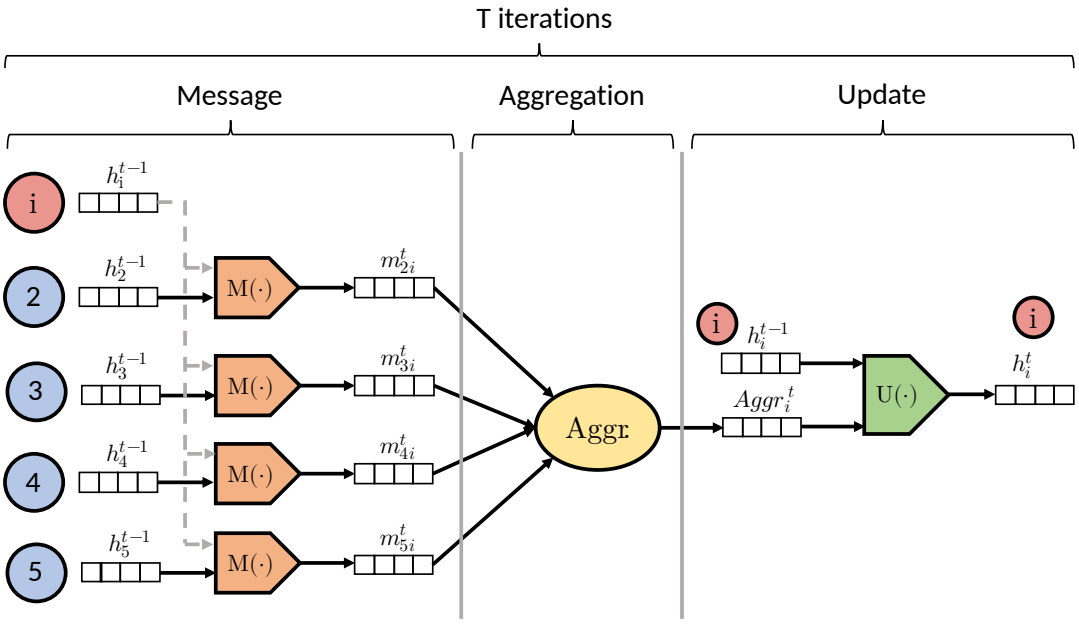


FIGURE 2.3: Message-passing Phase for a single graph node.

Readout Phase

Once the message-passing phase is completed, the resulting hidden states are passed through the readout phase. This last phase is responsible for converting the encoded information in the hidden states into the output values or

labels that the GNN eventually produces. It is important to note that GNNs may produce two different types of outputs: node-level outputs and global outputs. In the first case, node-level outputs are typically obtained by applying a readout function (R_i) individually to the hidden state of every node in the graph (Fig. 2.4, top), while global outputs (Fig. 2.4, bottom) are typically computed by first aggregating the hidden states of all the nodes and then applying a global readout function (R_G). In this latter case, the application of an aggregation function allows the GNN to operate over an arbitrary number of nodes in the readout phase.

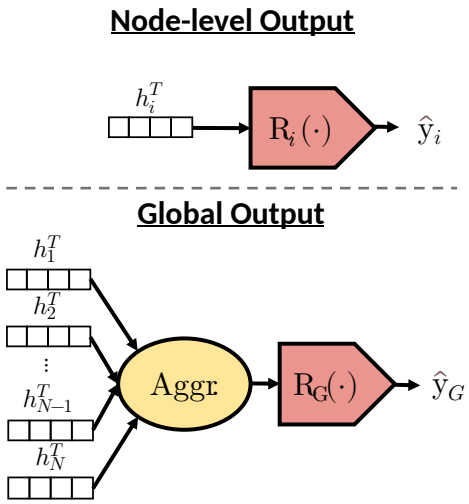


FIGURE 2.4: Readout Phase.

A GNN architecture is essentially characterized by the combination of the four functions outlined earlier (M, $Aggr$, U, and R). These functions are unique and are replicated throughout the GNN architecture, based on the input graph structure. Typically, M, U, and R are functions that can be approximated using independent neural networks, such as fully-connected layers. On the other hand, the $Aggr$ function often involves mathematical operations like element-wise summation.

It's important to note that while GNNs encompass a variety of architectural variants, as seen in previous works (e.g., [27], [29], [38], [38], [39], [40]), they all share the fundamental principle of message passing between graph elements, as described earlier (i.e., message-aggregation-update).

2.2 Machine Learning applied to Computer Networks

In recent years, two paradigms have reshaped the traditional landscape of network management. First, Software-Defined Networking (SDN) [41] addresses the inherent challenges of static and decentralized network architectures. By decoupling the data plane from the control plane and centralizing network intelligence, SDN introduces dynamic and programmatically efficient network configurations. This approach significantly enhances network performance and monitoring capabilities.

Simultaneously, Network Analytics (NA) has emerged as a pivotal technology for gaining insights into networked environments. NA empowers organizations to collect comprehensive data about network devices and their communication patterns. This wealth of data is then subjected to in-depth analysis, informing critical decisions related to network operations.

The convergence of these paradigms, coupled with the growing prominence of Machine Learning, has encouraged the integration of machine learning techniques into network management practices. Furthermore, a novel paradigm known as Knowledge-Defined Networking (KDN) [42] has emerged. KDN harnesses the combined strengths of SDN, NA, and Machine Learning to realize the vision of automated network control. This holistic approach aims to revolutionize network management by leveraging data-driven insights, dynamic configurations, and intelligent decision-making processes.

2.2.1 An Overview

Within the existing literature, one can readily discover a multitude of surveys that offer comprehensive insights into the application of machine learning techniques in the field of networking.

R. Boutaba et al. [43] differentiate 8 fields where the different ML techniques can be used:

- Traffic Prediction: Machine Learning can be used to forecast future traffic volumes based on historical traffic data. This prediction capability

aids in proactive network management and resource allocation.

- Traffic Classification: ML techniques shine at categorizing network traffic into specific classes or applications, such as distinguishing between social media traffic (e.g., Facebook) and communication protocols (e.g., WWW or FTP). Traffic classification is pivotal for security, quality of service (QoS), and resource allocation.

- Traffic Routing: ML algorithms are deployed to select optimal data transmission paths based on network performance metrics, including cost minimization, link utilization, latency, jitter, and QoS. Efficient traffic routing enhances overall network performance.

- Congestion Control: Machine learning can enhance traditional congestion control mechanisms like TCP or queue management. By using ML-based congestion control, network operators can mitigate congestion and improve network functionality.

- Resource Management: Predicting demand and dynamically configuring network resources, such as CPU, memory, switches, and routers, facilitates network adaptation to changing service requirements, optimizing resource utilization.

- Fault management: ML-driven fault management focuses on the detection, isolation, and rectification of network anomalies or irregularities, such as disabled links or malfunctioning switches, ensuring network reliability.

- QoS and QoE Management: Quality of Service (QoS) encompasses objective network performance metrics like bandwidth and latency, while Quality of Experience (QoE) represents the user's subjective perception of network quality. Machine learning bridges the gap between QoS and QoE, enabling the estimation of QoE based on QoS data.

- Network security: Machine learning serves as a powerful tool for identifying security threats within the network. It can detect known network attacks like Denial of Service (DoS) and identify anomalies that may be indicative of novel attack vectors, bolstering network security.

The primary objective of this dissertation is to develop a model capable of accurately estimating network performance metrics based on the

network's topology and current state. This model holds the potential to find relevance and application across various domains where such predictive capabilities are valuable.

For instance, one significant application area lies within network optimization. By providing an efficient and precise estimation of network performance under specific routing configurations, the model can serve as a valuable tool for optimizers. They can utilize this model to make informed decisions without the need for risky real-world network experiments, time-consuming packet simulation, or reliance on simpler heuristics like link utilization. This practicality positions the project within the realms of Quality of Service (QoS) and Quality of Experience (QoE) management, as well as traffic routing optimization.

2.2.2 GNNs applied to Computer Networks

In the computer networks field, recent attempts show the feasibility of constructing efficient GNN-based network models able to predict performance metrics in networks[44] [45] and combine them with optimization algorithms to address network-related problems.

The adoption of Graph Neural Networks (GNNs) in the domain of computer networks has been relatively limited until recently, primarily due to their relatively recent emergence. However, there have been noteworthy initiatives in the field of computer network security, such as [46], which leveraged GNNs' capacity to comprehend structured graph data to rank attack graphs. These pioneering efforts highlight the potential of GNNs in addressing network security challenges.

Beyond these early work and to the best of our knowledge, the research on GNNs applied to computer networking has been conducted by Fabien Geyer, from the Technical University of Munich, who used these models to predict the performance evaluation of network topologies [47], to learn and generate distributed routing protocols [45], and to improve Network Calculus models [48] [49].

In the realm of computer modeling, pioneering efforts have been made in different works. They encompass diverse areas, including delay prediction in queuing networks and the development of the RouteNet model [44][50][51].

Notably, this dissertation's research can be considered a successor to RouteNet, as it addresses various challenges in network modeling, thereby expanding the utility and capabilities of Graph Neural Networks (GNNs) in computer networks.

Chapter 3

Limitations and Challenges of Current Network Modeling Techniques

This chapter delves into the practicality of current network modeling techniques, beginning with packet-level simulators, and evaluates the effectiveness of various Deep Learning (DL) models. We'll examine the primary constraints of these established network modeling methods and delve into the unresolved challenges. The aim is to provide insight into the strengths and weaknesses of existing network modeling techniques while highlighting the ongoing issues that require attention, with a particular focus on the concerns addressed in this thesis.

3.1 Simulation as a Network Modeling Technique

Network simulators reproduce the network behavior at the granularity of packet events [3, 52]. This way, they can offer excellent accuracy and can be easily extended to include virtually any feature, such as packet scheduling, wireless networks, etc. Some simulators, such as OMNET++ [8] or ns3 [7], are widely used and maintained.

However, their main limitation is the simulation time, especially for networks with high-speed links (10Gbps and above). Hence, depending on the amount of traffic found in the target network, it may become unfeasible to simulate the network [53].

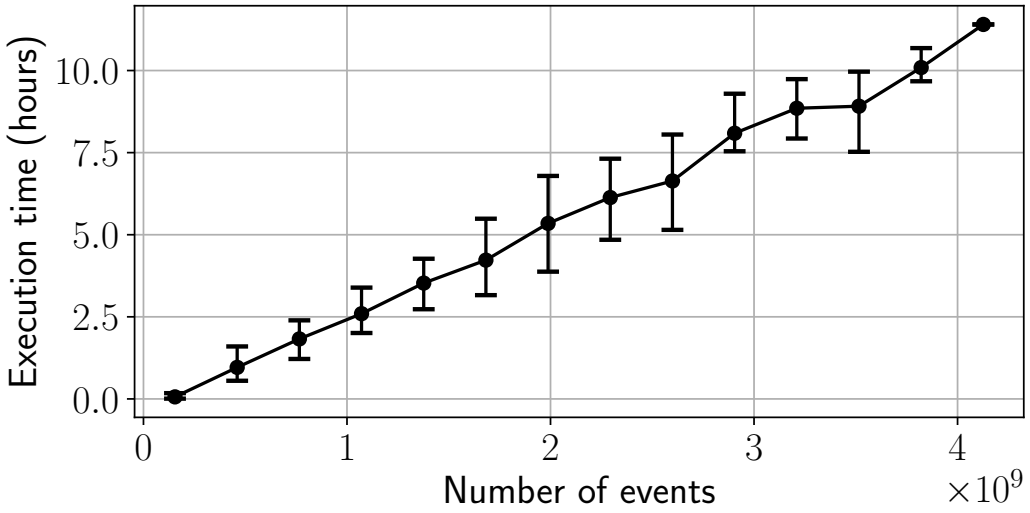


FIGURE 3.1: Simulation time depending on the number of processed events.

To illustrate this limitation, we simulate different topologies using the OMNET++ simulator to calculate the delay of a set of source-destination flows [CPU Intel Xeon Silver 4210R @ 2.40 GHz]. Network topologies are artificially generated using the Power-Law Out-Degree Algorithm from [54] and a traffic distribution that follows a Poisson process.

Figure 3.1 shows the simulation time of such networks depending on the number of events. Here, an event refers to a transition in the status of the network (e.g., adding a new packet to a queue). We can see that the simulation time increases linearly and that simulating 4 billion events takes more than 11 hours. Although 4 billion events may appear a large figure, consider that a 10 Gbps link transmitting regular Ethernet frames translates to ≈820k events per second or 247 million events in 5 minutes of network activity for a single link. For example, in our experiments, the simulator takes around 8 hours to compute the performance metrics of a 300-node network.

So, the main limitation of packet-level simulators is the simulation time. On the contrary, packet simulators offer unrivaled accuracy and can simulate virtually any scenario, from different routing configurations to replaying packet traces to simulate unknown traffic models. Because of this, hereafter, we consider the results from the simulator as the ground truth for the evaluations in the following sections.

3.1.1 Simulation Setup

To train, validate, and test the existing network modeling techniques we use as ground truth a packet-level network simulator (OMNeT++ v5.5.1 [8]), where network samples are labeled with performance metrics, including the flows' mean delay, jitter and losses, and queue-level statistics (e.g., occupation, packet loss). To generate these datasets, for each sample, we randomly select a combination of input features (traffic model and topology) according to the descriptions below:

Traffic models

Traffic is generated using five different models with increasing levels of complexity, which range from a basic Poisson generation process to more realistic traffic models with strong autocorrelation and heavy-tails [16]. We define below the implementation details of these models (except for the well-known Poisson and Constant Bitrate, whose only configurable parameter is the traffic intensity level):

On-Off This model defines two possible states (On or Off). The lengths of On and Off periods are randomly selected between 5 and 15 seconds. Likewise, during the On period, packets are generated using an exponential distribution.

Autocorrelated exponentials This model generates autocorrelated exponentially distributed traffic starting from the following auto-regressive (AR) process: z_{t+1}=az_t+ε, $\varepsilon\sim$N$(0,\sigma^2)$ where $a\in(-1,1)$ controls the level of autocorrelation. The marginal distribution of z is N$(0, s^2 = \sigma^2/(1-a^2))$, so z can be negative. In order to construct positive inter-arrival times, z is mapped by a nonlinear transformation: $\delta_t = F_E^{-1}\left(\lambda, F_N\left(0, s^2, z_t\right)\right)$, where $F_N(0, s^2, \cdot)$ and $F_E(\lambda, \cdot)$ are respectively a CDF of the normal distribution with $\mu = 0$ and variance s^2=$[3, 12]$, and an exponential distribution with parameter λ=$[40, 2000]$. The first transformation changes the distribution from normal to uniform on $(0, 1)$, while the second maps it into an exponential distribution. As a result, δ_t follows an exponential distribution with autocorrelation. Such a model can be interpreted as a copula [55].

Modulated exponentials This model represents an alternative autocorrelated model with higher complexity than the one above and is inspired by observation from [12]. Particularly, the inter-arrival times are set by a hierarchical model. Inter-arrivals follow an exponential distribution (Exp) whose rate is controlled by the value of a hidden AR model. Formally, we can describe the model as $\delta_t|z_t \sim \text{Exp}(\lambda_t = Ae^{z_t})$, where A is scaling constant and z is an AR model as in the previous traffic model.

The traffic matrix represents a source-destination pair, defining the flow of data between two network points. Performance metrics like delay, jitter, and losses are computed once the network reaches a stationary state, discarding the transitory phase. Also, in all the previous models, average traffic rates on src-dst flows are carefully set to cover low to quite high congestion levels across different samples, where the most congested samples have ≈3% of packet losses.

Topologies

To train, test and validate the models we used three different real-world topologies: NSFNET (14 nodes) [56], GEANT (24 nodes) [57], and GBN (17 nodes) [58]. These topologies have variable link capacities that are distributed according to the link centrality in the topology.

3.2 Analytical Models: Queueing Theory

Among all existing network modeling techniques, Queueing Theory (QT) [9] is probably one of the most known and used since computer networks can be understood as a set of queues connected, where the packets wait until they get served. For this, QT has been successfully applied in a wide variety of use cases [59].

This section explains the design and implementation of one state-of-the-art model that will be used as a future benchmark. The model is also evaluated in a wide variety of scenarios showing how, QT models in general, fail at predicting results when the Poisson arrival process is not met.

3.2.1 Design

In the holistic approach, the network is modeled as a single system, like in Jackson Networks [60] or more general BCMP queuing networks [61]. For those systems, the product form of the stationary distribution greatly simplifies the solution, however, the assumption of infinite buffers makes those models unrealistic and unable to estimate the packet loss ratio.

In our approach, all the queues along the path are modeled independently. Further, we assume that arrival to each queue is approximated by the Poisson process. Service times are assumed to be independent and exponentially distributed. Under those assumptions, we can derive analytical results for queue throughput, delay distribution, and blocking probability.

The aforementioned model also suffers from circular dependencies. Packet loss on a particular queue depends on its load, so it also depends on the throughput of other queues feeding this particular one. The throughput, however, depends on packet loss so we end up with circular dependence. We fixed this problem by a *map-reduce* inspired algorithm.

The algorithm consists of there functions: *map_queues*, *map_paths* and *reduce*. The *map_queues* function updates packet loss for each queue, given the total traffic (external demands plus within network transfer). The function also computes the remaining QoS parameters (jitter and delay). The *map_paths* function updates the traffic knowing the packet loss on every queue. Finally, the *reduce* function aggregates per path delay, jitter, and packet loss. In the first iteration, we assume no packet loss. Given the first approximation, we can compute the loss probability (*map_queues*) and update the intensities to account for the losses (*map_paths*). After a few iterations, the algorithm converges to a fixed point and the final values are reduced (*reduce*).

For an $M/M/1/b$ system, we used known formulas for blocking probabilities and delay distribution to get average delay and jitter. For a network with scheduling, we designed *map_queues* functions based on the Markov chain model described below. Because scheduling couples the queues, the corresponding *map_queues* operates on groups of queues assigned to the same link.

Let us begin with a strict priority scheduler. Consider p priority class customers arriving at rate λ_i and requiring service time with mean $1/\mu_i$. Each

class waits in the independent virtual queue limited by b_i and is served in non-preemptive First In, First-Out (FIFO) order. Such a system can be modeled as a continuous-time Markov chain on the state space $\mathcal{S}_{SP} = \{(s, q = (q_1, q_2, \ldots, q_p)\}$, where s denotes the priority class currently being served or 0 if the system is empty. The remaining part of the state: p-tuple q encodes the number of customers for each priority. For convenience let us define $q_{i+} := (q_1, \ldots, q_i + 1, \ldots, q_p)$, $q_{i-} := (q_1, \ldots, q_i - 1, \ldots, q_p)$ and $q^0 = (0, \ldots, 0)$. The model is based on [33] and modified to allow for per-priority class buffer size. The time evolution of the continuous-time Markov chain (CTMC) is characterized by the generator matrix $\boldsymbol{Q}$ whose elements follow the rules:

$$\boldsymbol{Q}[(0, q), (i, q_{i+})] = \lambda_i \quad 0 < i \leq p \tag{3.1}$$

$$\boldsymbol{Q}[(s, q), (s, q_{i+})] = \lambda_i I_{q_i < b_i} \quad 0 < i \leq p \tag{3.2}$$

$$\boldsymbol{Q}[(s, q^0_{s+}), (0, q^0)] = \mu_s \quad 0 < s \leq p \tag{3.3}$$

$$\boldsymbol{Q}[(s, q), (\min\{i : q_i > 0\}, q_{s-})] = \mu_s \tag{3.4}$$

where I_A is an indicator function and $\boldsymbol{Q}[.,.]$ denotes entry in generator matrix. If neither rule matches states pair a general rules $\boldsymbol{Q}[s, s'] = 0, \quad s \neq s'$ and $\boldsymbol{Q}[s, s] = -\sum_{s' \neq s} \boldsymbol{Q}[s, s']$ apply. A similar model can be constructed for Weighted Fair Queueing (WFQ) and Deficit Round Robin (DRR). Since both scheduling policies approximate an ideal Generalized Processor Sharing the same model is used for WFQ and DRR. The CTMC is similar to Equations 3.1-3.4 with the exception that the queue i is served at rate μ_i if other queues are empty, otherwise the rate scales proportionally to the positive weight w_i. State space $\mathcal{S}_{GPS}$ is also simplified and it is formed solely of p tuples q defined as for Strict Priority (SP). The resulting CTMC is based on [32] and evolves according to the following generator:

$$\boldsymbol{Q}[q, q_{i+}] = \lambda_i I_{q_i < b_i} \quad 0 < i \leq p \tag{3.5}$$

$$\boldsymbol{Q}[q, q_{i-}] = \frac{I_{q_i > 0} w_i}{\sum_{q_i > 0} w_i} \mu_i \quad 0 < i \leq p \tag{3.6}$$

Given the generator matrix $\boldsymbol{Q}$, we can develop either an analytical solution for queue characteristics as in [32, 33] or use a direct approach and obtain them numerically. We chose the latter and computed packet loss,

delay, and jitter assuming the CTMC has reached stationary distribution $\boldsymbol{\pi}$ computed from global balance equations (GBE) [60] that form a sparse linear system.

We obtained $\boldsymbol{\pi}$ from sparse eigenvalue decomposition via Arnoldi method [62] with a general sparse linear solver as a fallback in case of numerical instabilities. Given the $\boldsymbol{\pi}$, the packet loss ratio for class i ($\boldsymbol{p}_b[i]$) is the sum of all state probabilities where queue i is full. The delay is computed from the average queue size (with respect to $\boldsymbol{\pi}$) using Little's law. The computation of jitter requires a more sophisticated approach. We pose this as the first passage time problem in CTMC [63]. The delay of a class i customer is the first passage time to any state where the queue i is empty provided that no new customers can arrive so $\lambda_i = 0$ for GPS or $\lambda_j = 0, j \leq i$ for SP. Its conditional distribution can be calculated from $\boldsymbol{Q}$ using Laplace transform [63]. The final delay distribution and jitter are obtained from the total probability theorem. It is assumed that a packet of class i observing state s at his arrival experiences a delay equal to the first passage time from the state just after his arrival s_{i+}. From the PASTA property, the probability of such an event is $\boldsymbol{\pi}[s]/(1 - \boldsymbol{p}_b[i])$, here we condition the event that packet is not dropped.

3.2.2 Evaluation

Table 3.1 presents the error when predicting the network delay with respect to the results produced by the network simulator, including several traffic models. Particularly, we show four different metrics: (i) Mean Absolute Percentage Error (MAPE), (ii) Mean Squared Error (MSE), (iii) Mean Absolute Error (MAE), and (iv) Coefficient of Determination (R^2). We can see that the QT model offers good accuracy for Poisson traffic as expected. However, as the complexity of the traffic model increases the error also increases significantly achieving a maximum error of 68.1% for the more complex one.

3.3 Neural Networks as Network Modeling Techniques

From the section provided, it can be concluded that analytical models like Queueing Theory (QT) may not perform effectively when network traffic distributions deviate from a Poisson process. This limitation arises from QT's reliance on specific traffic distribution assumptions. The subsequent sections review the performance of three neural network (NN) architectures, each increasing in complexity. Unlike analytical models, NNs do not make stringent assumptions about traffic distributions; instead, they learn from data to model network performance more flexibly.

First, we evaluate the Multilayer Perceptron, one of the simplest NNs. Next, Recurrent Neural Networks which are designed to work with sequences. Finally, we directly input the network into a Graph Neural Network specifically designed to work with graphs. The objective is to create a network model with the NN that can predict performance parameters for input networks with a wide range of characteristics. We are especially interested in the following parameters:

- **Accuracy:** How close is the prediction to simulation values?
- **Different Routing:** Does the accuracy degrade if we change the routing configuration?
- **Link failures:** Quantify if link failures affect the quality of predictions.

	QT			
	MAPE	MSE	MAE	R^2
Poisson	17.9%	0.015	0.080	0.988
Deterministic	22.42%	0.715	0.321	0.611
On-Off	23.09%	0.784	0.363	0.613
A. Exponentials	21.11%	0.686	0.316	0.618
M. Exponentials	68.1%	1.10	0.798	0.145
Mixed	35.15%	0.721	0.430	0.560

TABLE 3.1: Delay prediction using the QT model. The error is computed w.r.t. simulation results.

We train and test the three neural networks with the same dataset, obtained from simulations with OMNET++. The input values are the network characteristics (topology, routing configuration, traffic model and intensity, etc.), and the output values are the delay for each path. Hence, all the errors are computed with respect to the values of the simulator. We use four different datasets:

- **Traffic Models:** In it, we consider traffic models that are non-Poisson, auto-correlated, and with heavy tails (See Subsection 3.1.1 Traffic Models).
- **Same Routing:** Where the testing and training datasets contain networks with the *same* routing configurations.
- **Different Routing:** Where the training and testing datasets contain networks with *different* routing configurations.
- **Link failures:** Here, we iteratively remove one link of the topology to replicate a link failure, until we transform the network graph into a connected acyclic graph. This scenario is the most complex since a link failure triggers a change both in the routing and the topology.

To compare the different techniques, we compute the prediction error with respect to the accurate performance values produced by the simulator. Similar to before, we use the following error metrics: (i) Mean Absolute Percentage Error (MAPE), (ii) Mean Squared Error (MSE), (iii) Mean Absolute Error (MAE), and (iv) Coefficient of Determination (R^2).

3.3.1 Multilayer Perceptron

A Multilayer Perceptron (MLP) is a basic kind of NN from the family of feedforward NNs [64]. In short, input data is propagated unidirectionally from the input neuron layer to the output layer. There may be an arbitrary number of hidden layers between these two layers, and this determines how deep is the NN.

	MLP				RNN			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
Poisson	12.3%	0.103	0.122	0.801	10.0%	0.071	0.084	0.862
Deterministic	23.9%	0.309	0.160	0.044	13.1%	0.083	0.070	0.743
On-Off	30.4%	0.438	0.240	0.002	15.2%	0.065	0.082	0.851
A. Exponentials	84.5%	1.013	0.308	-1.935	14.0%	0.070	0.072	0.7961
M. Exponentials	57.1%	1.058	0.363	-1.679	57.8%	0.528	0.457	-0.338
Mixed	41.2%	0.351	0.269	0.052	17.5%	0.036	0.080	0.900

TABLE 3.2: Delay prediction using an MLP and an RNN for different traffic models. The error is computed w.r.t. simulation results.

Design

Several works have leveraged an MLP to predict network performance metrics [65, 25, 26]. Based on this work, we have built an MLP to predict the mean delay for each source-destination pair of nodes of a given network. The MLP has 8,280 inputs and two hidden layers with 4096 neurons and uses Rectified Linear Units (ReLU) as activation functions.

Evaluation

Table 3.2 presents the error when predicting the network delay with respect to the results produced by the network simulator, including several traffic models. We can see that the MLP offers good accuracy for Poisson traffic, but the error increases significantly for the rest of the traffic models showing a MAPE between 23% and 84%.

Likewise, Table 3.3 shows the error of predicting the delay for the datasets with the same/different routing and link failures. We can see that the MLP cannot offer an accurate estimate when predicting the delay of a previously unseen routing configuration (1150% of error). This is due to the internal architecture of the MLP. During training, the MLP performs overfitting, meaning that the model *only* learns about the initial network topology used for training and not for any others. When we input a new topology, it does not have sufficient information to make an accurate prediction.

	MLP				RNN			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
Same Routing	12.3%	0.103	0.122	0.801	10.0%	0.071	0.084	0.862
Diff. Routing	1150%	28.3	2.96	-40.0	30.5%	0.553	0.282	0.197
Link Failures	125%	3.69	1.03	-0.191	63.8%	2.971	0.870	0.0417

TABLE 3.3: Delay prediction using an MLP and an RNN for the same and different routing configurations w.r.t. those seen during training, and considering various link failures. The error is relative to simulation results.

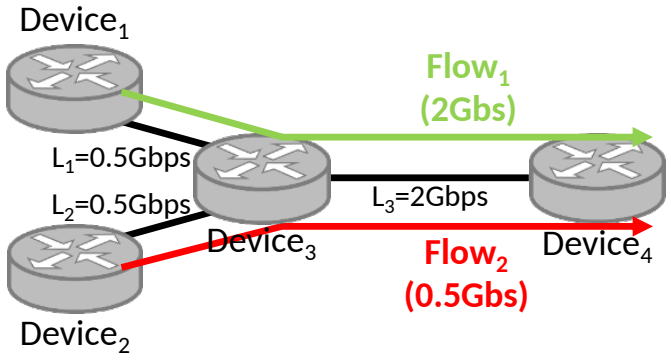


FIGURE 3.2: Sample Topology with 4 nodes, three links, and two flows.

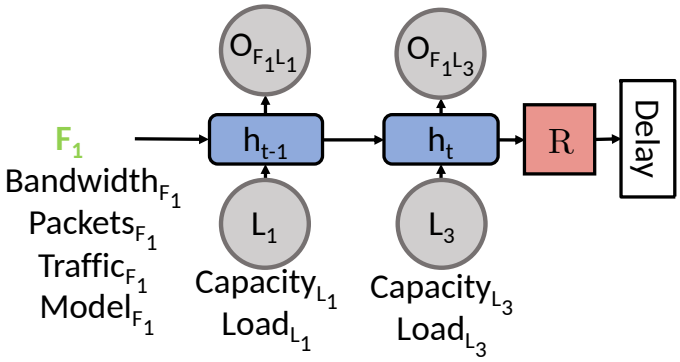


FIGURE 3.3: Recurrent Neural Network model for the Sample Topology (Figure 3.2)

3.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a more advanced type of NN. They have shown excellent performance when processing sequential data [66]. This is mainly because they connect some layers to the previous ones, which gives them the ability to keep the state along sequences.

Design

Several works [67, 68, 69] propose RNNs as a way to predict network performance. In this experiment, we build a sequential model with an RNN (Figure 3.3). Particularly, we choose a Gated Recurrent Unit (GRU).

We initialize the state of each path with the sequence of nodes in the path and the features of the traffic model (e.g., packets, bandwidth, λ, ϵ, α, or on-off time), and we update the state of each link across the path. As an example, Figure 3.3 shows the structure of an RNN to model the sample network from Figure 3.2. We can see that the path of $flow_1$ is composed of L_1 and L_3. Once the path state has been computed, an MLP with 2 hidden layers computes the final output.

Evaluation

We train the RNN with the same datasets as the previous subsection. Although the RNN supports better different traffic models than the MLP (Table 3.2), it still struggles to produce accurate predictions when there are routing or topology changes (Table 3.3), especially for different routing configurations (30% error), or when removing links (63%).

The reason behind the lack of capability of RNNs to understand routing changes and link failures is due to its internal architecture. RNNs can accommodate different end-to-end paths in the network (i.e., series of routers and links), thereby, making it easier to perform predictions for paths never seen in the training phase. However, this structure cannot store and update the status of individual links in the topology due to the inter-dependency between links and traffic flows (i.e., routing). In other words, if the status of a link changes, it affects several flows, and vice-versa. This generates circular dependencies that RNNs are not able to model.

3.3.3 Graph Neural Networks

Networks are fundamentally represented as graphs where networking devices are the graph nodes and the links connecting devices are the graph edges. This interconnection translates to the fact that the different elements

in the network are dependent on each other. Since most standard DL models (e.g., MLP, RNN) assume independent flow-level data points, this renders them inaccurate for our use case. Hence, a model that is capable of processing directly the network graph is arguably more desirable for network modeling, because it will be able, not only to obtain information from the individual nodes and edges but also from the underlying data structure (i.e., the relationships between the different elements) [70].

GNNs [27] are a type of neural network designed to work with graph-structured data. They have two key characteristics that make them a good candidate for a network model. First, GNNs have the ability to store node-level hidden states and update them in each iteration. Second, they process the input data directly as a graph, both during training and inference. This means that the internal structure of the neural network depends on the input graph. Hence, they can dynamically adapt to the underlying dependencies between the different elements of a network [35, 30]. The latter is of special importance, since changes in the network graph (e.g., routing modifications, link failures) are the main limitation of other DL-based models, such as MLPs and RNNs, as we have seen in the previous subsections.

In this section, we build a standard GNN model to predict the mean per-flow delay in networks.

Design

We implement a Message-Passing Neural Network (MPNN), a powerful state-of-the-art GNN architecture that can efficiently capture dependencies between the elements of input graphs [28]. We define a graph G described as a set of vertices (or nodes) V and a set of edges (or links) E. Each node has a set of features x_v, and edges also have some features e_{vw}. The execution of an MPNN can be divided into three phases, an initialization phase, a message-passing phase, and a readout phase. The first one defines a hidden state (h_v^0) using the node features (x_v). The second one is an iterative process that runs for T time steps and that is defined by two functions: the message function M_t and the update function U_t. During this phase, node hidden states (h_v^t), represented as fixed-size vectors, attempt to encode some *meaningful* information, and are iteratively updated by exchanging messages m_v^{t+1} with their neighbors:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{3.7}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{3.8}$$

where $N(v)$ represents the neighbors of v in graph G. Finally, the readout phase computes the output vector using a readout function R that takes as input the final hidden states h_v^T:

$$\hat{y} = R(\{h_v^T | v \in G\}) \tag{3.9}$$

In our case, the input of the MPNN is the network topology graph. It performs T=4 message-passing iterations and the hidden state dimension is 32. The Message function is implemented as a two-layer fully connected NN with ReLUs as activation functions. For the Update function, a GRU layer is used. Finally, the Readout function is implemented using a two-layer fully connected NN with ReLUs as activation functions for the hidden layers and a linear activation for the output layer.

Evaluation

We evaluate the accuracy of the MPNN model when predicting the mean flow-level delay, as in previous subsections. Table 3.4 presents the delay prediction errors in the same scenarios of the previous experiments: routing configurations, both seen and not seen during training, and link failures. Unfortunately, the results are similar to those of the RNN: the routing configurations from the training dataset are easy to predict, with an error as low as 3%, while new routing configurations and link failures increase the error significantly (respectively, 50% and 125% of MAPE), thus showing even larger errors than the RNN model.

The main reason behind the poor accuracy of the MPNN model is that the architecture of this model is directly built based on the network topology without taking into account the paths used by different traffic flows (i.e., the routing configuration), which is a fundamental property to understand inter-dependencies between flows and links. More specifically, when we test

the model with the same routing configuration, it learns the relationships between flows and links. However, when we change this configuration, those previously learned relationships are no longer valid, and the model is not able to capture the relationships between elements in the new scenario.

This intuition is better understood with an extreme packet loss example. Let's suppose we have the sample network in Figure 3.2. The first flow ($Flow_1$) is transmitting at a rate of 2Gbps, and the second one ($Flow_2$) is transmitting at a rate of 0.5Gbps. As we can see, L_1 has a maximum capacity of 0.5Gbps. Because of this, $Flow_1$ will experience a large packet loss, which causes the traffic of $Flow_1$ at L_3 to be at most 0.5Gbps. Hence, instead of having 2Gbps+0.5Gbps of aggregated traffic at L_3, it will only have 1Gbps. Now, $Flow_2$, which initially could have experienced a lot of network congestion when going through the 2Gbps link will experience none as the state of the link changed.

Knowing this, we can see how there is a circular dependency between the flows and links found in the network. At the same time, the state of a flow depends on the state of the links they traverse, and the state of the links depends on the state of the flows passing through them.

If we apply a standard GNN over this example, the state of each flow is not updated at each hop. Therefore, the GNN does not have a structure that represents how the delay depends on both the links (topology) and the flows that go through each specific router.

Hence, we conclude that feeding the MPNN directly with the network topology graph is not sufficient to accurately perform network modeling. However, more complex and customized GNN-based architectures can still

	MAPE	MSE	MAE	R^2
Same Routing	3.0%	0.002	0.016	0.994
Diff. Routing	50.0%	0.609	0.307	0.115
Link Failures	82.2%	3.41	0.949	-0.099

TABLE 3.4: Delay prediction using an MPNN for the same and different routing configurations w.r.t. those seen during training, and considering various link failures. The error is relative to simulation results.

be powerful for modeling the inter-dependencies between the different network elements and generalizing over new network scenarios by exploiting the underlying graph structure.

3.4 Challenges of data-driven Network Modeling

In previous sections, we have seen how several data-driven DL-based architectures fail to address changes in the network topology. The last section shows how, even if GNNs are a potential tool to solve these problems, they need to be specifically built to the problem and how there is no specific architecture that solves all the problems. This section describes the main challenges that data-driven solutions need to address for network modeling. These challenges drove the core design of the models that constitute one of the major contributions of this dissertation.

Quality of Service and Scheduling policies: A key requirement of modern networks is supporting Quality of Service (QoS), a parameter typically implemented via scheduling policies and mappings of traffic flows to QoS classes. Hence, a DL model should be able to predict the performance of the input traffic flows with their associated QoS class, similarly to how QT models support a wide range of scheduling policies [71, 33].

Traffic models: Networks carry different types of traffic, so, supporting arbitrary stochastic traffic models is crucial. Experimental observations show that traffic on the Internet has strong autocorrelation and heavy-tails [16]. In this context, it is well-known that the main limitation of Queuing Theory is that it fails to provide accurate estimates on realistic Markovian models with continuous state space, or non-Markovian traffic models. The challenge for DL-based modeling is: How can we design a neural network architecture that can accurately model realistic traffic models?

Training and Generalization: One of the main differences between analytical modeling (e.g., QT) and data-driven modeling is that the latter requires training. In DL, training involves obtaining a *representative* dataset of network measurements. The dataset needs to include a broad spectrum of network operational regimes, ranging from different congestion levels to various routing configurations, among others. In other words, the DL model

can predict only scenarios it has previously seen. Note that this is a common property of all neural network architectures.

Ideally, we would obtain this training dataset from a production network, since they commonly have systems in place to measure performance. However, it would be difficult to obtain a *representative* dataset. As we mentioned previously, we would need to measure the production network when it is experiencing extreme performance degradation as the result of link failures, incorrect configurations, severe congestion, etc. However, these situations are not common in production networks, which limits the ability to generate the training dataset. A reasonable alternative is creating these datasets in controlled testbeds where it is possible to use different traffic models, implement a broad set of configurations, and replicate a wide range of network failures. Thus, the DL model can be trained on samples from this testbed and then, applied to production networks. Hence, the research challenge is: how to design a DL model that can provide accurate estimates in networks not seen during training? This includes topologies, traffic, and configurations (e.g., queue scheduling, routing) different from those seen in the training network testbed.

Leveraging a testbed that is smaller than a production network creates another challenge: the generalization to *larger* networks. Real-world networks include hundreds or thousands of nodes, and building a network testbed at this scale is typically unfeasible. As a result, the DL model should be able to learn from datasets with samples of small network testbeds and predict metrics for considerably larger networks, e.g., by a factor of 10-100x. Generalizing to larger networks, or graphs in general, is currently an open research challenge in the field of GNNs [72, 73].

Chapter 4

RouteNet-Darwin: Advancing Towards Quality of Service-Aware Network Modeling

As seen in Chapter 3, computational models (e.g., network simulators), provide excellent accuracy. State-of-the-art network simulators include a wide range of network, transport, and routing protocols, and are able to simulate realistic scenarios. However, this comes at a very high computational cost that depends linearly on the number of packets being simulated. As a result, they are impractical in scenarios with realistic traffic volumes or large topologies. In addition, and because they are computationally expensive, they do not work well in real-time scenarios.

In addition, Queuing Theory (QT) [9] is one of the most popular modeling techniques, where networks are represented as inter-connected queues that are evaluated analytically. This represents a well-established framework that can model complex and large networks. Its main limitation is that it imposes strong assumptions on the packet arrival process, which typically do not hold in real networks [10]. Internet traffic has been extensively analyzed in the past two decades [11, 12, 13, 14, 15] and, despite the community has not agreed on a universal model, there is consensus that in general aggregated traffic shows strong autocorrelation and a heavy-tail [16].

Machine Learning (ML) [22] has provided a new breed of mechanisms to model complex systems. In particular, Deep Learning (DL) [23] has been demonstrated to extract deep knowledge from human-understandable descriptions of a system. This approach has proven to achieve unprecedented

accuracy in modeling properties of complex systems, like proteins [24].

The main advantage of DL models is that they are *data-driven*. DL models can be trained with real-world data, without making assumptions about the system. This enables the building of models with unprecedented accuracy by effectively modeling the entire range of non-linear and multi-dimensional system characteristics. Computationally, DL is based on linear algebra and can take advantage of massive parallelism by leveraging dedicated hardware and compilers.

Within the field of DL, Graph Neural Networks (GNN) [27] have recently emerged as an effective technique to model graph-structured data. GNNs are tailored to understand the complex relationships between the elements of a graph. The main novelty of GNNs is that their internal architecture is dynamically assembled based on the elements and connections of input graphs, and this enables them to learn universal modeling functions that are invariant to graph isomorphism. GNNs are thus able to *generalize* over graphs, which means that they can produce accurate estimates in different graphs not seen during training. As we will show in this chapter, this is a critical advantage of GNNs in the context of network modeling.

The novel GNN paradigm finally allows the application of ML in domains where data is essentially represented as graphs. As seen in Chapter 3, this dissertation argues that GNNs represent a new network modeling language with attractive advantages and characteristics. GNNs are designed to learn graphs, and computer networks are fundamentally graphs of connected queues. However, GNNs are not a black box that maps data inputs to outputs, it is actually a *modeling tool* that needs to be researched and designed to account for the core behavior of computer networks. In contrast to more classical DL models, where the architecture is basically defined by the number of layers and neurons, GNNs are assembled ad-hoc, based on the elements and connections of the input graphs. These components represent the GNN modeling language, and they need to be carefully designed to reflect the relevant properties of the system being modeled.

In response, [50] presents a tailored solution explicitly designed to predict end-to-end performance metrics based on network topology, traffic matrix, and routing configuration. This method introduces a Message Passing Neural Network (MPNN) model comprising two fundamental elements:

links and paths. Links represent the physical connections between two forwarding devices in a computer network, primarily responsible for establishing communication paths between source and destination nodes. Paths, on the other hand, denote the routes followed by data flows as they traverse from one network device to another. The network's routing configuration dictates these paths, often involving the traversal of specific links and nodes. While this approach offers accuracy and generalization, it does not fully encapsulate the intricacies of real-world networks. The presence of various forwarding devices (e.g., routers, switches, firewalls) and their configurations (e.g., queue sizes, scheduling policies) remains unaccounted for, resulting in a simplification that limits its practicality for application to real network environments.

Inspired by the work described at [50], this Chapter presents RouteNet-Darwin (from now on referenced as RouteNet-D), a model specifically designed to overcome this limitation. RouteNet-D models the complex relationship between topology, routing, queue scheduling, and input traffic, in order to produce accurate estimates of per-flow QoS metrics (e.g., delay, jitter, loss). RouteNet-D is able to accurately estimate the delay in paths traversing arbitrary concatenations of queuing policies, with different routing configurations, traffic matrices, and network topologies. A critical feature of RouteNet-D is its ability to generalize to unseen networks. This means that it can provide accurate estimates in networks with different characteristics to those seen in training.

4.1 RouteNet-Darwin

In this section, we describe RouteNet-D, a GNN-based model that accurately infers the impact of different routing and queueing configurations (scheduling algorithm and queue parameters) on network performance. Likewise, it is tailored to generalize to different network topologies and traffic intensities not seen before.

4.1.1 Overview

RouteNet-D implements a novel and custom GNN architecture inspired by the inherent behavior of computer networks, where there are different components (e.g., forwarding devices, configuration, traffic) that interact with each other and have a complex non-linear impact on network performance.

The main intuition behind this architecture is as follows. The model considers an input graph with three main network components: (i) the links that shape the network topology, i.e. connections between network devices, (ii) the queues on each output port of network devices, and (iii) the src-dst paths resulting from the input routing configuration. Each of these elements is explicitly represented in the GNN with n-element vectors that encode their hidden states (h_l, h_q, and h_p respectively). They are combined through a message-passing algorithm that aims to capture the relation between the topology, traffic, routing, and queueing policy of the input network scenario.

Figure 4.1 represents how RouteNet-D models these three components. First, the state of paths is affected by the concatenation of the queues and the links they traverse. For instance, in Fig. 4.1, $path_1$ follows the sequence: $[queue_3, link_1, queue_5, link_2...]$. At the same time, the state of queues and links depends on all the paths passing through them. Hence, there is a circular dependency between the states of paths, links, and queues that the GNN model must resolve to eventually produce accurate per-path QoS estimates. Note that we assume that the forwarding engine of network devices is constant and ideal, hence it does not introduce any other potential hardware-level effects on the delay of queues.

4.1.2 Notation

We define the network topology as a set of links $L = \{l_i : i \in (1, ..., n_l)\}$ and the queues on output ports of network devices $Q = \{q_i : i \in (1, ..., n_q)\}$ and a set of source-destination paths $P = \{p_i : i \in (1, ..., n_p)\}$. Let us also consider a path as a sequence of tuples with the queues and links they traverse defined by the routing scheme. Hence, we define the paths as: $p_i = \{(q_{PQ(p_i,0)}, l_{PL(p_i,0)}), ..., (q_{PQ(p_i,|p_i|)}, l_{PL(p_i,|p_i|)})\}$, where $PQ(p_i, j)$ and $PL(p_i, j)$ respectively return the index of the j-th queue or link along the path p_i. Let

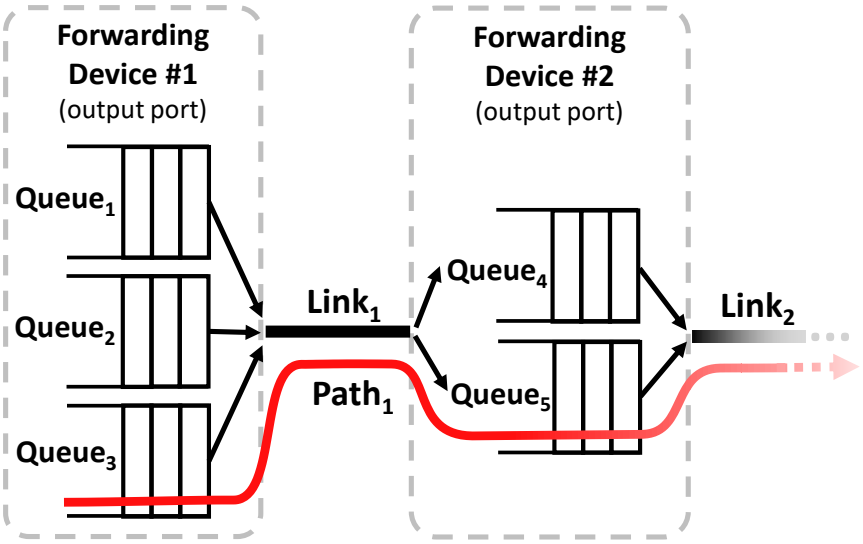


FIGURE 4.1: Schematic representation of the network model implemented by RouteNet-D.

us also define $Q_p(q_i)$ as a function that returns all the paths passing through queue q_i, and $L_q(l_i)$ as a function that returns the queues injecting traffic into link l_i – i.e., the queues at the output port to which the link is connected.

Each queue, link, and path is initialized with some features x_{l_i}, x_{q_j} and x_{p_k}, respectively. In our particular case, we set the initial features of links (x_l) as (i) the link capacity, and (ii) the scheduling policy (FIFO, Strict Priority, WFQ, or DRR) configured in the egress port that injects traffic into the link, using one-hot encoding. The initial features of queues (x_q) are: (i) buffer size, (ii) priority order (one-hot encoding), and (iii) weight (only for WFQ and DRR). Lastly, we set the initial path features (x_p) as the traffic volume (bits and packets) sent from the source to the destination node of the path.

4.1.3 Network Model

We initialize the state of links h_l, queues h_q, and paths h_p respectively with their initial feature vectors (x_l, x_q and x_p), and apply zero-padding to fit the size of the target vectors, which is a configurable parameter of the GNN. After the message-passing phase, these hidden states are expected to encode some meaningful information about links (e.g., utilization), queues (e.g., load, packet loss rate), and paths (e.g., end-to-end delay, packet loss) based on the information exchanged along the graph. Thus, RouteNet-D is based on these basic principles:

1. The state of a path depends on the states of all the queues and links that it traverses.
2. The state of a link depends on the states of all the queues that inject traffic into the link.
3. The state of a queue depends on the states of all the paths that inject traffic into the queue.

These principles can be mathematically formulated as follows:

$$h_{q_i} = f_q(h_{p_1}, ..., h_{p_m}), \quad q_i \in p_k, k = 1, ..., j \tag{4.1}$$

$$h_{l_j} = f_l(h_{q_1}, ..., h_{q_m}), \quad q_m \in L_q(l_j) \tag{4.2}$$

$$h_{p_k} = f_p(h_{q_{k(0)}}, h_{l_{k(0)}}, ..., h_{q_{k(|f_k|)}}, h_{l_{k(|f_k|)}}) \tag{4.3}$$

where f_q, f_l, and f_p are some unknown functions.

A direct approximation of functions f_q, f_l and f_p is complex given that: (i) equations 4.1, 4.2 and 4.3 define a complex non-linear system of equations with the states being hidden variables, (ii) they encode complex mutual dependencies between different network components (topology, routing, queueing policies, traffic), and (iii) the dimensionality of possible states is extremely large.

GNNs have shown an outstanding capability to work as universal approximators over graph-structured data [35, 30]. As a result, thanks to its internal GNN-based architecture, RouteNet-D is able to approximate flexible f_q, f_l, and, f_p functions, which can later be applied to unseen topologies, routing schemes, queueing configurations, and traffic distributions.

4.1.4 Proposed GNN Architecture

Algorithm 1 describes the internal architecture of RouteNet-D. This custom GNN architecture is specially designed to solve the circular dependencies described in Equations 4.1, 4.2 and 4.3 by executing an iterative message-passing process. First, the hidden states h_l, h_q, and h_p are initialized (lines 1-3) using x_l, x_q, and x_p respectively and padded with zeros to the specific hidden state dimension. After the hidden state initialization, the message-passing

Algorithm 1 Internal architecture of RouteNet-D

Input: L, Q, P, x_q, x_l, x_p
Output: h_q^T, h_l^T, h_p^T, $\hat{y_p}$

1: **for each** $l \in L$ **do** $h_l^0 \leftarrow [x_l, 0...0]$
2: **for each** $q \in Q$ **do** $h_q^0 \leftarrow [x_q, 0...0]$
3: **for each** $p \in P$ **do** $h_p^0 \leftarrow [x_p, 0...0]$
4: **for** t = 0 to T-1 **do**
5: **for each** $p \in P$ **do**
6: **for each** $q, l \in p$ **do**
7: $h_p^t \leftarrow RNN_p(h_p^t, [h_q^t, h_l^t])$
8: $\widetilde{m}_p^{t+1} \leftarrow h_p^t$
9: $h_p^{t+1} \leftarrow \widetilde{h}_p^t$
10: **for each** $q \in Q$ **do**
11: $M_q^{t+1} \leftarrow \sum_{p \in Q_p(q)} \widetilde{m}_{p,q}^{t+1}$
12: $h_q^{t+1} \leftarrow U_q(h_q^t, \widetilde{M}_q^{t+1})$
13: $\widetilde{m}_q^{t+1} \leftarrow h_q^{t+1}$
14: **for each** $l \in L$ **do**
15: **for each** $q \in L_q(l)$ **do**
16: $h_l^t \leftarrow RNN_l(h_l^t, \widetilde{m}_q^{t+1})$
17: $h_l^{t+1} \leftarrow h_l^t$
18: $\hat{y_p} \leftarrow F_p(h_p)$

phase begins. During this step, each state is combined with its connected elements according to the relations described in the input graph. This process is repeated T iterations (loop from line 4). Thus, by the end of the message-passing execution, hidden states h_l, h_q, and h_p should eventually converge to some fixed values after exchanging information with their neighbors in the graph [27].

Unlike standard GNN models, RouteNet-D implements a more complex message-passing that can be divided into three different stages involving message exchanges between heterogeneous graph elements. The loops from line 5, line 10, and line 14 in Algorithm 1 represent these different message-passing stages, where for each path (line 5), for each queue (line 10), and for each link (line 5), the hidden states are exchanged with their connected elements and updated based on the information received. More specifically, each path collects messages from all the queues and links that it crosses (loop from line 6), then each queue receives messages from all the

paths that pass through it (summation from line 11) and lastly, each link collects information from all the queues that inject traffic into it (loop from line 15). To aggregate the paths' hidden states on queues (line 11) we use an element-wise summation. In the case of links and queues, it is important to consider that there is a sequential dependence on the elements connected. For instance, the order of queues and links that a path traverses is important in case there is packet loss, as the packets dropped by one queue will not be injected into the subsequent links and queues. For this reason, we use a Recurrent Neural Network (RNN) to aggregate the sequences of queues and links on the paths' hidden states (line 7). Similarly, the model aggregates the queue states on their related links using another RNN (line 16), as it is important to maintain the order of queues to model the behavior of the queuing policy (e.g., the priority order). For simplicity of the GNN architecture, we implement some message and update functions as direct variable assignments, except for the case of the update function for queues U_q (line 12), which is implemented using a Gated Recurrent Unit to facilitate the convergence of the algorithm [74].

Finally, the function F_p (line 18) represents a readout function that is applied individually on each path hidden state h_p and, in this case, is used to finally produce the estimated mean per-path delays ($\hat{y}_p$). Particularly, we modeled the readout function F_p with a fully connected NN using a SELU activation function [75].

This architecture provides flexibility to represent any routing configuration and queuing policy (including QoS-aware scheduling algorithms with multiple queues). This is achieved by the direct mapping of the paths resulting from the routing configuration P to specific message-passing operations with queues, links, and other paths.

4.2 Prototype Implementation

We implemented a prototype of the full RouteNet-D message-passing structure using TensorFlow. The source code of RouteNet-D and the training and evaluation datasets used in this chapter are publicly available at [76].

4.2.1 Simulation Setup

In order to train our GNN-based model, we built the ground truth leveraging a packet-level network simulator (OMNeT++ v5.5.1 [8]). Each sample in the training set corresponds to the simulation of a specific network scenario, defined by a topology, a src-dst traffic matrix, and a routing and queuing policy. Then, the simulator labels this sample with the mean per-packet delay measured on each source-destination path. Regarding the training dataset, each sample represents a random selection of input features (topology, traffic, routing, and queuing configuration) according to the following descriptions:

Traffic

We generate the input Traffic Matrices (TM) following the same approach described in [77]. In our particular case, traffic matrices are generated to cover a wide range of operational scenarios from low traffic loads to highly loaded networks. Depending on the capacities of the links, as well as the routing configuration of the network, this TM will result in a certain packet loss. Particularly, we generated these TMs to obtain a maximum packet loss of 3% according to [78]. Since some TMs lead to traffic aggregates that exceed the capacity on some links, they will cause congestion and packet loss due to the accumulation of packets in the queues. Note that we also use traffic extracted from real packet traces, which is later described in Section 4.3.5. Finally, we randomly assign a Type-of-Service (ToS) label to each source-destination traffic flow (ToS$\in$[0-9]), which is then used to map traffic flows to specific queues at egress ports of network devices – as shown below.

Queueing Configuration

Each node is configured randomly with (i) a queue scheduling policy, that can be First In First Out (FIFO), Strict Priority (SP), Weighted Fair Queueing (WFQ), or Deficit Round Robin (DRR), (ii) a random number of queues, and (iii) a random queue size. For WFQ and DRR, we define a set of random queue weights. Finally, we map ToS labels to queues in decreasing order of priority, including a random component and depending on the number of queues. Hence, lower ToS labels are assigned to higher-priority queues. Note that the dataset contains samples of a wide range of queuing configurations

from the simulator, this helps the GNN model understand their effect on network performance.

Topologies

In order to train and evaluate the model, we use three different real-world topologies (NSFNET [56], GEANT [57] and GBN [58]). Later, in Section 4.3.4, we also evaluate the generalization properties of our solution with 106 real-world topologies from the Internet Topology Zoo [79].

4.2.2 Machine Learning Framework

We train the model using 100k samples from each training network (NSFNET and GEANT). Note that for a given network topology, a data sample is defined as a random combination of routing, queuing policy, and traffic matrix. The randomly generated configurations are within the operational ranges defined in Section 4.2.1.

Our model has two relevant hyper-parameters that can be fine-tuned: (i) The size of the hidden states h_l, h_q and h_p, and (ii) the number of message-passing iterations (T). Based on preliminary experiments we use 32-element vectors for all the hidden states, and T=8 iterations. In this context, T should correlate to the network diameter, which in real networks approximately scales with $log(N)$, where N is the number of nodes of the input graph [80].

We choose the Mean Squared Error (MSE) as a loss function, which is minimized using an Adam optimizer with an initial learning rate of 0.001 and a decay rate of 0.6 executed every 80,000 steps. In addition to this, we added a $L2$ regularization loss of λ=0.1.

Note that, these parameters have been selected following an exhaustive grid search, aiming to obtain a balance between computational efficiency and achieving low error rates for the model.

Figure 4.2 shows the CDF of the relative error when predicting the mean per-path delay for the three topologies. In the two topologies seen in training (NSFNET and GEANT), RouteNet-D obtains a MAPE of 2.59% and 3.01% respectively. Int the GBN topology, only used in the evaluation,

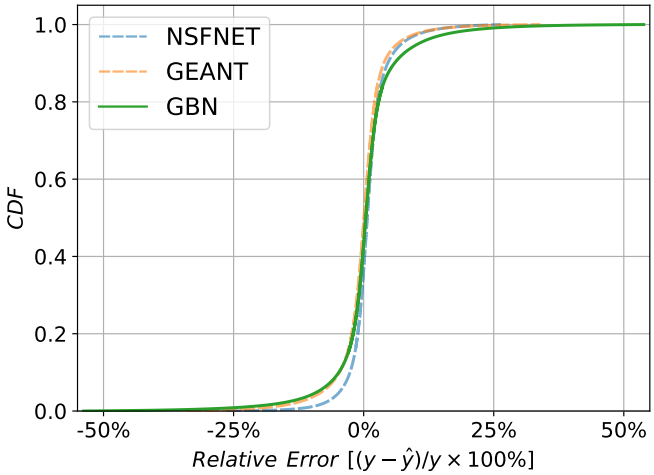


FIGURE 4.2: CDF of relative error for RouteNet-D. y represents the true delay, while $\hat{y}$ denotes the predicted one.

obtains a $MAPE$=3.88%. Also, the error distribution is centered around 0, which means that the model predictions are not biased towards under or overshooting.

4.3 Evaluation

Our evaluation focuses on several relevant properties of RouteNet-D including (i) the accuracy of the delay prediction of source-destination pairs, in a wide variety of topologies, routing, queueing configurations, and traffic matrices with various load levels, (ii) the generalization capabilities of the model in networks never seen during training, (iii) the accuracy when working with real-world data, and (iv) the speed and the scalability of its estimations.

4.3.1 Baseline

To benchmark the performance of RouteNet-D we compare it against [77] which from now on will be referred to as GNN-Baseline. Similar to RouteNet-D, GNN-Baseline models the network by performing a two-stage message passing between the links and paths in the network.

4.3.2 Performance Metrics

Performance metrics are a critical component of the evaluation frameworks in Machine Learning. They are mainly used to monitor and measure the performance of a model. Since we are in a regression problem and following the approach described in [81], we provide 4 different metrics. Two absolute metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE), as well as two relative metrics: Mean Absolute Percentage Error (MAPE) and the Coefficient of Determination (R^2).

We believe that combining the four of them provides a good picture of the performance of the different models evaluated in the following sections. Mainly, we focus on MAPE as, in contrast to MAE and MSE, it is a relative metric that does not depend on the units of the predicted variable (delay).

4.3.3 Accuracy

We evaluate the accuracy of RouteNet-D and GNN-Baseline using 100k samples of each of the aforementioned topologies (NSFNET, GEANT, and GBN). Note that the GBN topology has never been seen during the training phase for any one of the models.

	NSFNET				GEANT			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
GNN-Baseline	0.667	4.366	0.847	-0.193	0.581	1.172	0.336	-0.093
RouteNet-D	**0.033**	**0.077**	**0.06**	**0.978**	**0.039**	**0.028**	**0.034**	**0.973**

(A) Performance comparison for NSFNET and GEANT networks, seen during training.

	GBN			
	MAPE	MSE	MAE	R^2
GNN-Baseline	0.533	3.124	0.523	-0.089
RouteNet-D	**0.034**	**0.067**	**0.046**	**0.976**

(B) Performance comparison for GBN network, never seen during training.

TABLE 4.1: Comparison of performance metrics.

Table 4.1 shows the results for the three topologies. In this particular case, we show the Mean Absolute Percentage Error (MAPE), the Mean Squared Error (MSE), the Mean Absolute Error (MAE), and the Coefficient

of determination (R^2). As we can see, RouteNet-D clearly outperforms the baseline, achieving a MAPE of 3.8% and an R^2 of 0.976 for the GBN topology.

4.3.4 Generalization Capabilities

GNN models have shown a great potential to generalize over data structured as graphs [35, 30]. This section presents an analysis of the generalization capabilities of RouteNet-D. Particularly, we refer to generalization as the capability of the model to make accurate predictions in new network scenarios unseen during the training phase. In our case, it involves different topologies, routing and queuing configurations, and traffic distributions never seen during training.

To this end, we evaluate the accuracy of the proposed GNN-based model as well as the GNN-Baseline, with 106 real-world topologies from the Internet Topology Zoo [79] that were not present in the training set. For each topology, we use the network simulator previously described (Sec. 4.2.1) to generate the delay labels for the ground truth, considering a variety of traffic matrices and configurations. Then, we analyze the accuracy of the model trained only with the NSFNET and GEANT topologies (Sec. 4.2.2).

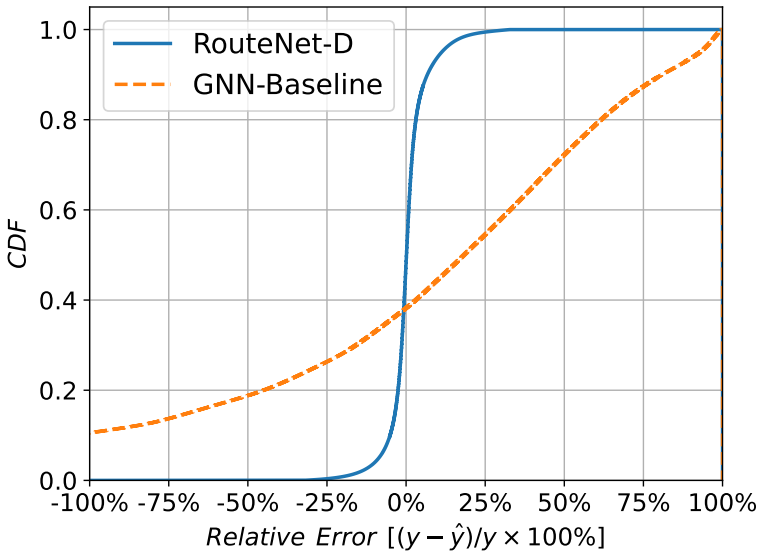


FIGURE 4.3: CDF of relative error over 106 unseen real-world topologies.

Figure 4.3 shows the CDF of the relative error for this experiment. As the figure shows, our model provides good generalization capabilities,

achieving a $MAPE$=3.8% over the 106 real-world topologies never seen during training. As expected, the GNN-Baseline performs poorly in generalization scenarios with a MAPE greater than 60%.

	TOPOLOGY ZOO			
	MAPE	MSE	MAE	R^2
GNN-Baseline	0.643	14.44	0.322	-0.006
RouteNet-D	**0.038**	**0.476**	**0.0258**	**0.966**

TABLE 4.2: Performance metrics comparison over 106 real-world topologies never seen during training.

Table 4.2 shows a summary of the different metrics evaluated on the 106 real-world topologies obtained from the Internet Topology Zoo. Again, RouteNet-D outperforms the baseline, which performs poorly when the topologies evaluated are different from the ones seen during training.

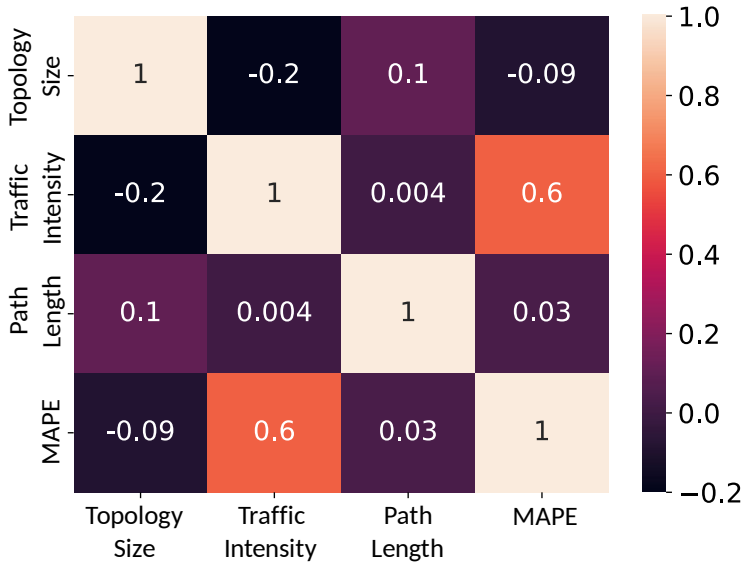


FIGURE 4.4: Correlation matrix of the topology size, traffic intensity, path length, and MAPE.

We have experimentally analyzed what features have more impact on the model's accuracy, and have found that there is little variability in the error across the different topologies (≈0.8% MAPE between the topologies with less and more error). Figure 4.4, shows the correlation matrix between several properties of the experiment. It can be seen how the highest correlation is found between traffic intensity and the MAPE. This is in line with the accuracy results we obtained in previous experiments where the model showed slightly less accuracy in samples from highly congested networks.

As an example, networks with very low traffic obtain an average error of 1.5%, while networks with a high level of congestion produce an error of 5.3% on average.

4.3.5 Experiments with Real Traffic

In the previous experiments, we evaluated RouteNet-D with synthetic traffic. In this section, we validate the accuracy of this model when applied to real traffic, without retraining the model.

For this purpose, we use real-world traffic matrices from the SNDlib library [82]. Since the traffic matrices only contain traffic aggregates of each source-destination pair, we use a recent snapshot from the MAWI repository (SamplePoint-F, Oct. 2020) [83] to extract realistic packet inter-arrival times. Then, we scale these inter-arrivals according to the values in the traffic matrices. Regarding the mapping of source-destination flows to ToS classes, we follow the same distribution from a real ISP [84].

We create a new dataset only for evaluation, not training. This dataset contains three different topologies: GBN (NOBEL), GEANT, and ABILENE, extracted from SNDlib [82], and the aforementioned traffic matrices. Note that the dataset contains: (i) two new topologies (GBN and ABILENE), not present in the training dataset, and (ii) traffic matrices completely different from the ones used in training. We evaluate the accuracy of the previous model from Sec. 4.3.3 directly in this dataset, without retraining it.

Figure 4.5 shows the CDF of the relative error in this new scenario. As we can observe, RouteNet-D produces accurate delay estimates even in scenarios emulating real traffic. Particularly, the model achieves a MAPE of 5.6% for the ABILENE topology, 7.1% for GEANT, and 8.9% for GBN. As mentioned previously, the model has been trained with synthetic traffic, and here we test it using real traffic. Despite the traffic seen by the model following a slightly different traffic distribution than that seen during training, the model still achieves good accuracy. Compared with the results from Figure 4.2, the MAPE increases from 3.9% (results with synthetic traffic) to 5.6% - 8.9% MAPE (best and worst case results with real traffic).

Finally, we evaluate RouteNet-D using a dataset that combines the 106

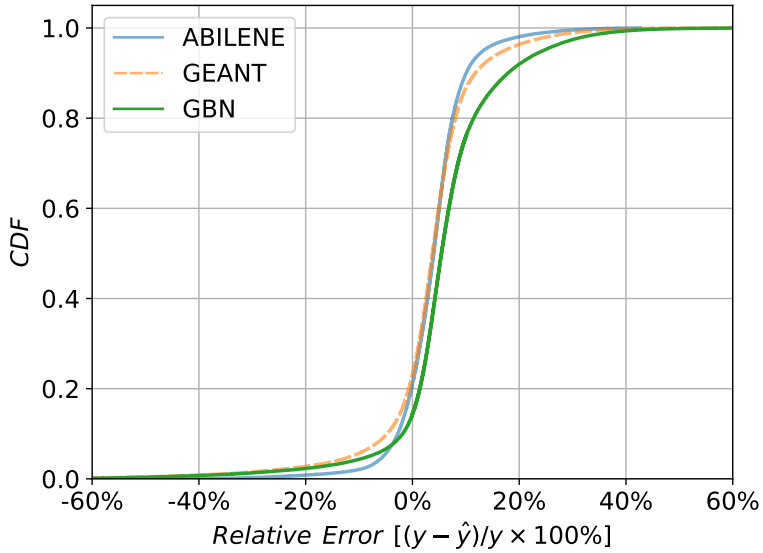


FIGURE 4.5: CDF of the relative error using real traffic.

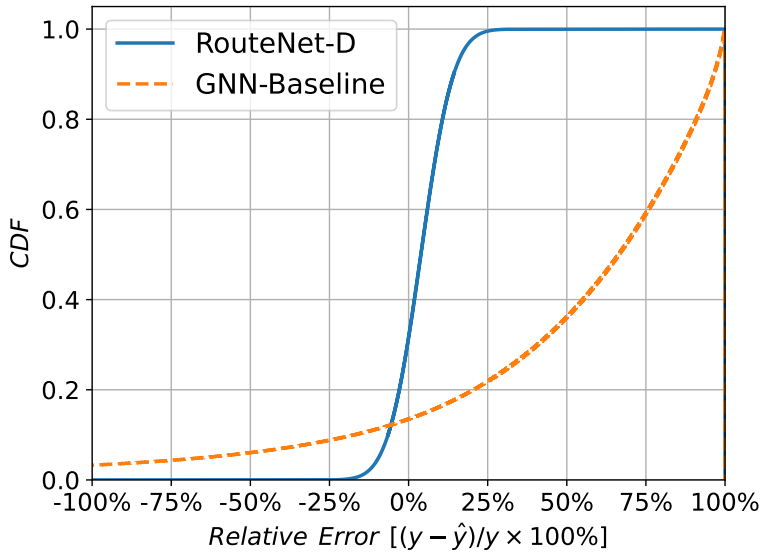


FIGURE 4.6: CDF of relative error over 106 unseen real-world topologies with realistic traffic.

	TOPOLOGY ZOO WITH REALISTIC TRAFFIC			
	MAPE	MSE	MAE	R^2
GNN-Baseline	0.686	14.48	0.351	-0.008
RouteNet-D	**0.071**	**0.104**	**0.028**	**0.992**

TABLE 4.3: Performance metrics comparison over 106 real-world topologies with realistic traffic never seen during training.

real network topologies previously used (Sec. 4.3.4), and with traffic matrices that follow the inter-arrivals times and ToS classes mentioned used in the

previous experiment [83, 84]. Figure 4.6 and Table 4.3 show the performance of RouteNet-D compared to the baseline. Following the trend of the previous experiments, RouteNet-D outperforms the GNN-baseline, achieving a MAPE of 7.1%.

4.4 Use Case: Optimization

Network optimization is typically achieved by combining two main elements: (i) a network model and (ii) an optimization algorithm (e.g., [85]). The model predicts the performance (e.g., per-path delay) for a specific configuration (e.g., routing), while the optimization algorithm generates configurations that can potentially meet the expected performance, for example, according to a Service Level Agreement (SLA).

Emerging use cases have renewed interest in SLA optimization [86]. SD-WAN [87], 5G network slicing [88], networked control of industrial systems, such as Industry 4.0 [89] and Tactile Internet [90, 91], require new stringent SLA requirements. In addition, novel forms of communication, such as AR/VR or holographic telepresence, demand ultra-low deterministic latency [92, 93]. In order to efficiently offer such SLAs, network optimizers must consider both routing *and* queue scheduling mechanisms (e.g., Strict Priority, Weighted Fair Queueing, Deficit Round Robin).

A fundamental aspect of network optimization is that *we can only optimize what we can model*. For example, to optimize the delay of a path traversing some nodes with different queuing policies, the model must be able to understand how delay relates to queuing policies and traffic.

In this section, we demonstrate the practical applications of RouteNet-D when coupled with an optimizer in various real-world use cases. This showcases the potential of GNN-based models in efficiently identifying optimal configurations. It's important to note that the effectiveness of network optimization is inherently linked to the quality of the underlying network model. RouteNet-D excels in this regard, as it comprehensively captures the intricate relationships between network parameters (as shown in previous Sections), making it a valuable tool for exploring and finding optimal network configurations.

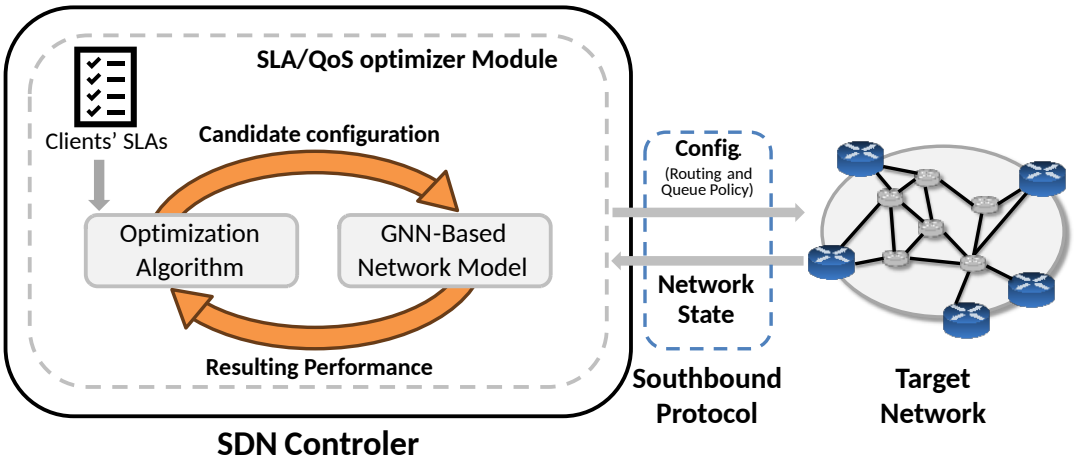


FIGURE 4.7: Network scenario for SLA-based optimization.

4.4.1 Network Scenario

Novel network applications (e.g., Industry 4.0 [89], Tactile Internet [90]), are pushing further the requirements of network-offered SLAs. At the time of this writing, there are substantial research efforts to accommodate such challenging SLAs with new protocols and architectures [92, 93, 86]. Many of such architectures take advantage of the Software-Defined Networking paradigm, which enables a new breed of centralized optimization algorithms [21]. Centralization enables full visibility of the network configuration and state, as well as fine-grained flow control.

In this section, we consider a Wide Area Network (WAN) that implements a classical SDN architecture: a centralized controller, and a southbound protocol that allows configuring the devices and collecting network performance metrics, such as OpenFlow or NETCONF (Fig. 4.7). The SDN controller has an SLA/QoS optimizer application that guarantees fine-grained SLAs by adjusting the routing and queuing policies according to the network state. First, the network administrator defines the desired set of SLAs, e.g. the *maximum* mean delay of flows. The granularity of the flows depends on the requirements of the administrator, hence, we consider different flow granularities, ranging from source-destination to 5-tuple flows. Then, SLAs are mapped to the data plane by tagging the packets of each flow using common fields from the IP header: Type-of-Service (ToS) for IPv4, and Differentiated Services for IPv6.

Second, a network model – RouteNet-D in this case – is paired with

an optimizer running on the SDN controller. The controller has visibility of the local configuration of each data-plane element as well as up-to-date measurements of the network state: bandwidth, mean delay of each source-destination pair, and link utilization. This can be achieved using readily available telemetry methods [94, 95].

Leveraging this information, the optimizer explores alternative configurations that can meet the SLA for the current traffic load. In RouteNet-D, configurations are combinations of source-destination routing and per-interface queuing configurations, i.e., scheduling algorithm and queue parameters. Each configuration produced by the optimizer is tested by RouteNet-D, which produces fast and accurate estimates of flow delays. Once the optimizer finds a configuration that meets the SLAs, it is applied to the data-plane elements.

For this reason, the abilities of RouteNet-D of (i) Generating accurate predictions of SLA metrics, (ii) Achieving fast operation, to quickly adapt the configuration to traffic changes, and (iii) *Generalizing* to other network scenarios not seen during training, are of critical importance.

The latter feature is of critical importance. In the context of computer networks, training a Network Model requires generating a large diversity of network scenarios (e.g., random routing and queuing configurations, simulating link failures, etc), which could render the network unusable. This would be infeasible in production networks. Thus, we argue that a practical way to build a Network Model is to train them in controlled environments (e.g., in a networking lab). Then, they can take advantage of their generalization capabilities to operate in real network topologies unseen in advance.

4.4.2 Limitations of State-of-the-Art Optimizers

State-of-the-art network optimization solutions mainly rely on fluid models [17, 18, 19, 20, 21]. In order to discuss the limitations of such models when dealing with complex SLA scenarios, we take DEFO [17] as a representative of the state-of-the-art in this area [96].

DEFO is a constraint programming framework for network optimization. This optimizer allows network administrators to set constraints to the optimization problem, including maximum end-to-end delays. To estimate

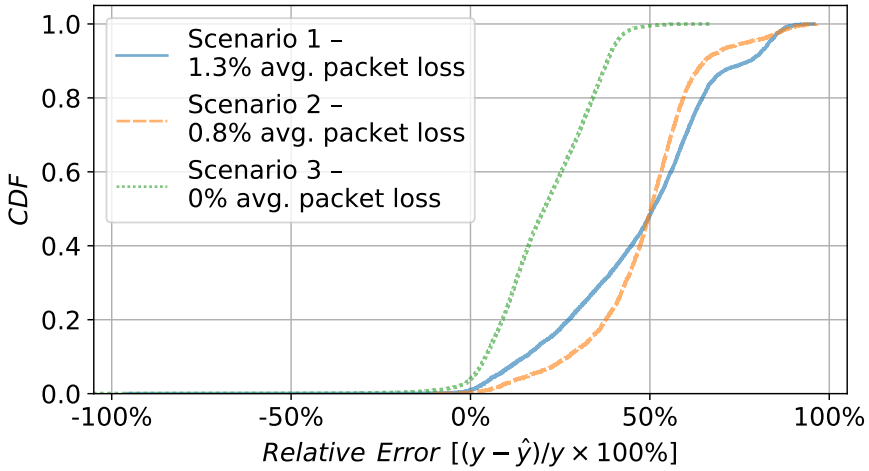


FIGURE 4.8: CDF of the relative error of the fluid model under various traffic loads.

the performance of candidate configurations, DEFO uses a fluid model of the network. In DEFO, the per-link delay is a fixed input variable of the network optimizer. Particularly, this value is either provided by the network operator in real topologies or computed manually according to the link distance in synthetic topologies [17]. Then, the delay of a path is assumed to be equal to the sum of transmission delays of the links that it traverses, without considering any queuing delay.

To test the accuracy of the fluid model used by DEFO, the actual delay is measured with an accurate packet-level simulator based on OMNET++ [8]. In this particular case, the simulation of the network scenario is defined by a topology, a src-dst traffic matrix, and a routing configuration. Similar to before, we use three different real-world topologies (NSFNET [56], GEANT [57] and GBN [58]). The traffic matrix is defined following the same approach as described in [77], generating three network scenarios containing 0%, 0.8%, and 1.3% off packet losses to see the effect of the network congestion on the accuracy of the fluid model. Finally, the routing configurations are the ones returned by DEFO after running its optimization process.

Figure 4.8 plots the Cumulative Distribution Function (CDF) of the relative error produced by the fluid model when estimating the per-path delay in a network scenario from [17], optimized with DEFO. The figure plots three different distributions according to the network load (average packet loss from 0% to 1.3%). These results show that actual delays are different from those estimated by the fluid model. Even for scenarios without packet loss, the fluid model has a Mean Absolute Percentage Error (MAPE) of 21%,

while estimations degrade significantly with increasing network load (≈50% MAPE). This is due to the fact that fluid models do not consider the queuing delay, which becomes an important component of the end-to-end delay in the presence of congestion.

Finally, it is worth noting that DEFO is just used in this section to illustrate the limitations of fluid models for SLA-aware optimization, but the contributions described in [17] go well beyond the network model used for optimization. Indeed, the DEFO optimizer could easily support more complex network models, like the one proposed in Section 4.1.

4.4.3 SLA-driven Optimization Use Cases

Here, we present several relevant use cases that illustrate the potential of RouteNet-D for SLA-driven optimization. In all of them, RouteNet-D is paired with an optimization algorithm in order to produce routing and queuing configurations that meet a set of SLAs.

Methodology

We combine RouteNet-D with an optimization algorithm based on Direct Search. This algorithm uses a custom search heuristic based on common network metrics (link utilization, traffic, path length, etc), which guides the exploration within the high-dimensional space of solutions. Note that more sophisticated optimization algorithms can be paired with RouteNet-D. However, we leave this out of the scope of this dissertation, as the goal of this section is to showcase how RouteNet-D can be effectively used for SLA-aware optimization.

In all the experimental setups, we generate traffic flows with two different SLA levels (ToS0=Top priority, ToS1=high priority), and some background traffic labeled as *Best effort*. The goal for the optimizer is to find a configuration that fulfills the predefined SLAs for ToS0 and ToS1 while minimizing the mean delay for the best-effort traffic. In all the experiments, we use the RouteNet-D model previously trained in NSFNET and GEANT (Sec. 5.1.3) and perform optimization over scenarios not seen during training (in the GBN topology).

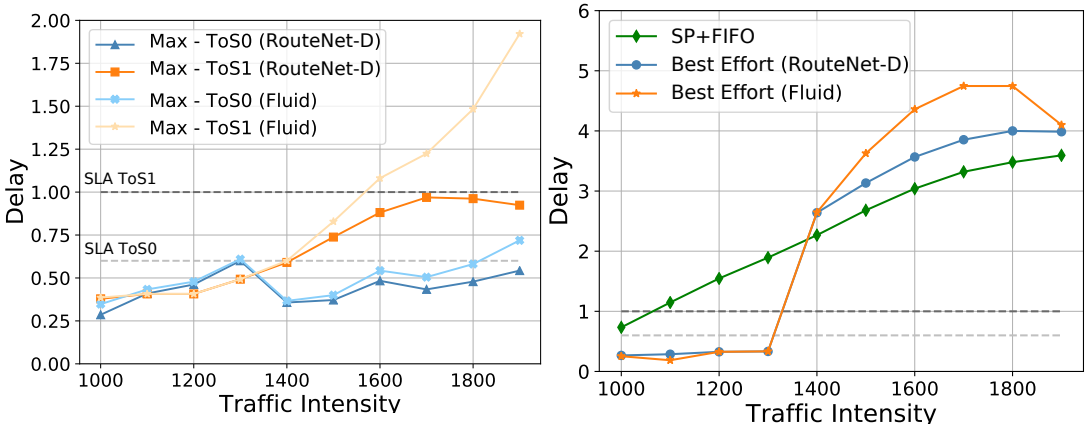
As a benchmark, we also show the performance of a shortest-path policy using FIFO scheduling on all network devices, as well as the configuration that results from running an optimizer that integrates a fluid model instead of RouteNet-D.

Routing

In this use case, the goal is to optimize the mean delay of the best-effort traffic while satisfying the set of SLAs for each ToS by only modifying the (src-dst) routing scheme. We evaluate the performance across various traffic intensities ranging from 1000 (middle traffic load) to 1900 (high network congestion).

We compare the results obtained using RouteNet-D with those achieved using a traditional fluid model. This latter case represents a baseline of state-of-the-art optimization tools that rely on fluid models, such as DEFO [17]. However, note that this reference benchmark is not exactly the same as DEFO, as we do not consider middle-point routing or ECMP. Our focus is to make a direct comparison of both network models, under the same conditions. Hence, we use the same optimization algorithm. In both cases, the exploration is guided by the delay estimates of the network model: RouteNet-D or fluid models. Additionally, we compare the results with the performance of a traditional Shortest Path (SP) policy.

Figures 4.9a and 4.9b show the results of the optimization. Figure 4.9a



(A) SLA flows - Routing (max. flow mean delay)

(B) Best Effort - Routing (avg. flow mean delay)

FIGURE 4.9: Routing-based SLA Optimization.

shows how the RouteNet-D-based optimizer finds a solution that fulfills the SLA requirements for both ToS (top and high priorities). Figure 4.9b shows that the optimizer paired with RouteNet-D achieves remarkable performance when minimizing the mean delay of the best-effort traffic, outperforming both the fluid and the SP for low traffic intensities. For high traffic load, RouteNet-D manages to fulfill the SLAs for both ToS (Fig. 4.9a), while this has an impact on the mean delay of best-effort traffic (Fig. 4.9b).

Regarding the fluid model, we can observe that the optimizer could not find solutions that satisfy the SLAs for medium to high traffic intensities. The reason for this is that despite the fluid model estimates that the delay experienced by flows is within the SLA terms, the packet-level simulator shows that such estimates are not accurate –as shown earlier in Sec. 4.4.2– and exceed the SLA thresholds. This is because the fluid model ignores queuing delay. Thus, in highly congested scenarios, where the queuing delay becomes more significant, the fluid model-based optimizer leads to SLA violations.

Scheduling

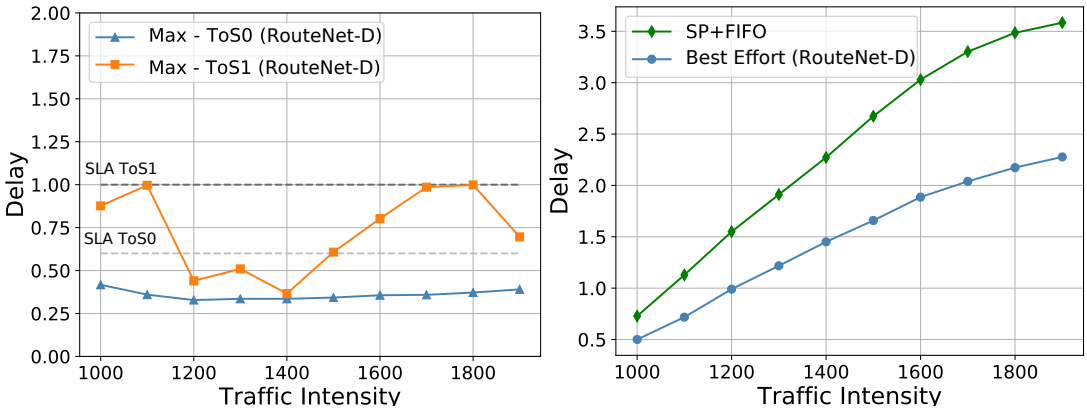
This use case also attempts to minimize the mean delay on best-effort traffic, while satisfying the SLAs assigned to each ToS. However, in this case, we aim to evaluate the potential of optimizing the queuing configuration using RouteNet-D. The main difference with the previous experiments is that now the optimizer only explores different queuing configurations, while routing is fixed to a standard shortest path scheme. Then, we leverage the flexibility of RouteNet-D to make delay estimates under different queue scheduling policies and parameters. Note that in this case, we cannot compare the results with a fluid model given that it does not have support for queuing policies beyond FIFO. Because of this, optimizers based on fluid models are typically limited to exploring different routing configurations.

Figures 4.10a and 4.10b show that the optimizer is able to fulfill all SLAs, while also minimizing the delay of best-effort traffic. More importantly, if we compare these results with the previous ones, we can observe that by modifying the queuing configuration we achieve better results than modifying the routing. These results illustrate the remarkable impact of the queue scheduling configuration on the overall network performance, particularly in the presence of different ToS.

Routing and Scheduling

Based on the previous results, in this scenario, we aim to evaluate the optimization potential when optimizing both the routing and queuing configuration. The objective is the same as before, minimizing the mean delay on best-effort traffic while satisfying the SLAs.

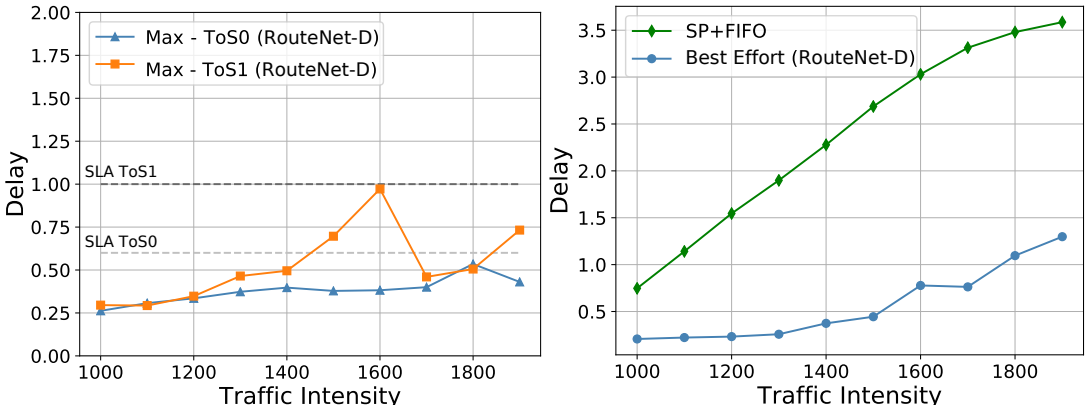
Figures 4.11a and 4.11b show the evaluation results. As we can observe the improvement is remarkable compared to the previous results. The RouteNet-D-based optimizer satisfies all SLAs while pushing the mean delay of best-effort traffic even lower. For instance, in the scenario with the highest traffic load, it achieves a reduction on the mean delay of ≈60% with respect to the SP+FIFO policy.



(A) SLA flows - Scheduling (max. flow mean delay)

(B) Best Effort - Scheduling (avg. flow mean delay)

FIGURE 4.10: Scheduling-based SLA Optimization.



(A) SLA flows - Routing & Scheduling (max. flow delay)

(B) Best Effort - Routing & Scheduling (avg. flow delay)

FIGURE 4.11: Routing and Scheduling-based SLA Optimization.

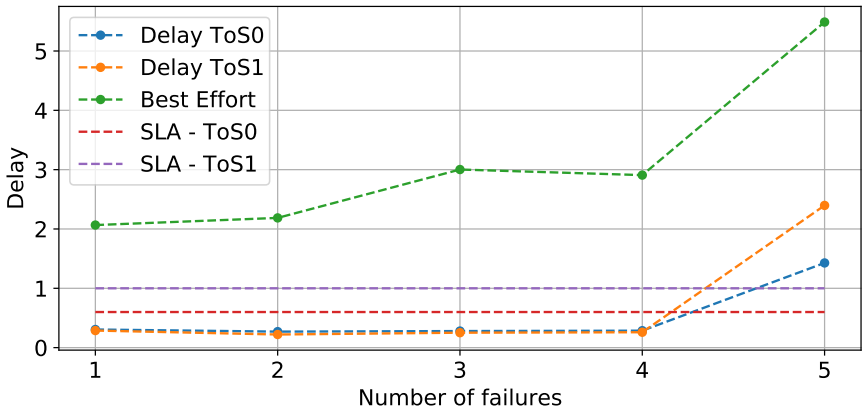


FIGURE 4.12: SLA-driven optimization with link failures.

Robustness Against Link Failures

In this use case, we evaluate if our model is able to generalize in the presence of link failures. When a certain link fails, we need to find a new routing and queuing configuration to avoid such a link. As the number of link failures increases, fewer paths are available and the network becomes more saturated.

We run our RouteNet-D-based optimizer in scenarios with a number of random link failures. The initial network scenario is the same as in the previous experiments (i.e., routing & scheduling optimization), considering the highest traffic intensity ($TI = 1900$).

Figure 4.12 shows the optimized mean delay with respect to the number of link failures (n). Each point in the plot corresponds to the optimal delay obtained over 10 experiments with n random link failures. We observe that the mean delay increases gradually on best-effort traffic as there are more link failures and the network becomes increasingly congested. Nevertheless, the optimizer is able to meet all the SLAs even with up to 4 link failures.

What-if: Budget-constrained Network Upgrade

In this last use case, we show how RouteNet-D can be used to reason about the network and provide recommendations. Particularly, we use it to answer the following question: *What is the optimal link upgrade in the network?* The question is put in the context of a network administrator that has a static and well-known traffic matrix and that is willing to upgrade the capacity of the network by adding one link that minimizes the mean traffic delay.

Traffic matrix	Optimal new link placement	Previous delay	Delay with new link	Delay reduction
TM_1	(6,16)	0.446	0.259	41.9%
TM_2	(8,13)	0.508	0.303	40.3%
TM_3	(7,15)	0.409	0.239	41.5%
TM_4	(6,14)	0.499	0.253	49.3%
TM_5	(7,15)	0.551	0.322	41.5%
TM_6	(6,16)	0.458	0.209	54.3%
TM_7	(5,12)	0.419	0.251	40.1%
TM_8	(6,14)	0.590	0.312	47.1%

TABLE 4.4: Optimal link placement with various Traffic Matrices (TM_i).

To answer this question we use our RouteNet-D-based optimizer, constrained to adding only one link. Table 4.4 shows the results for scenarios with 8 different random traffic matrices of traffic intensity TI=1500. We can see that the optimizer achieves a considerable delay reduction (40.1% - 54.3%) by properly selecting the best link placement.

4.5 Discussion and Concluding remarks

Network models are a central component for network control and optimization, as they enable the evaluation of network performance under alternative configurations and what-if scenarios. Reproducing the behavior of real networks involves a series of complex non-linear relationships among multiple components that define the network state (e.g., topology, routing, queuing configuration, traffic).

In this Chapter, we presented RouteNet-D, a GNN-based model that implements a custom message-passing architecture particularly designed to model these complex relationships. For this purpose, RouteNet-D explicitly defines network elements and their relations in its internal neural network architecture and learns how to reason on this graph-structured information. We applied RouteNet-D to estimate per-path delays in networks, covering a wide variety of topologies, configurations (routing and queuing), and traffic load levels. Our evaluation shows that RouteNet-D is able to generalize accurately to samples of 106 real-world networks (from Internet Topology Zoo)

unseen in advance, after being only trained on samples of two different networks (NSFNET and GEANT). Also, our evaluation shows how RouteNet-D is capable of understanding the complex relationships in a real traffic scenario. This reflects the capability of this model to abstract deep insights from network scenarios seen during training and then apply this knowledge effectively to new network topologies, including different traffic matrices, routing, and queuing configurations.

Finally, we presented several relevant use cases that illustrate the potential of the proposed model for SLA-driven optimization. To this end, we paired RouteNet-D with an optimization algorithm to produce routing and queuing configurations that meet a set of SLAs in a specific network scenario. We also used it to test the robustness of the network against link failures and to find the optimal link placement in a network planning use case.

Chapter 5

RouteNet-Erlang: Enhancing Network Modeling through Scheduling, Traffic Models, and Generalization

In the previous Chapter, we delved into the challenge of Quality of Service and Scheduling policies by leveraging a custom GNN-based architecture that solves the complex relationships found in the different elements encountered on the network. Nevertheless, with this solution, the challenges of the traffic models and the generalization to larger topologies keep being there.

In this Chapter, we present RouteNet-Erlang (RouteNet-E), a novel GNN-based architecture designed for the performance evaluation of computer networks that solves the aforementioned challenges. RouteNet-E shares the same goals as QT models: it is also able to model a network of queues, with different sizes and scheduling policies while providing accurate estimates of delay, jitter, and losses. Interestingly, RouteNet-E is not limited to Markovian traffic models as QT, but rather it supports arbitrary traffic models including more complex ones with strong autocorrelation and high variance, which better represent the properties of real-world traffic [16]. It also shows that RouteNet-E overcomes one of the main limitations of existing ML-based models: *generalization to larger topologies*. RouteNet-E is able to make accurate estimates in samples of unseen topologies one order of magnitude larger than those seen during training.

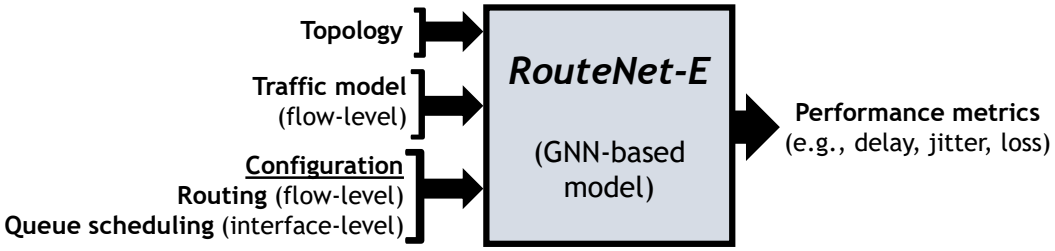


FIGURE 5.1: Black-box representation of RouteNet-E.

We benchmark RouteNet-E against a state-of-the-art QT model (Chapter 3), over a wide variety of network samples covering several different traffic models, from basic Poisson to more realistic and complex models with strong autocorrelation and approximated heavy-tails. Our evaluation results show that the proposed model outperforms the QT benchmark in *all* the network scenarios evaluated, always producing accurate delay predictions with a worst-case error of 6% (for QT is 68%). We also show RouteNet-E's remarkable performance in hundreds of random network topologies not seen during training. Lastly, we measure its inference speed, which is in the order of milliseconds, in line with the QT benchmark.

5.1 RouteNet-Erlang

This section describes RouteNet-E, a novel GNN-based solution tailored to accurately model the behavior of real network infrastructures. RouteNet-E implements a novel three-stage message-passing algorithm that explicitly defines some key elements for network modeling (e.g., traffic models, queues, paths), and offers support for a wide variety of features introduced in modern networking trends (e.g., complex QoS-aware queuing policies, overlay routing).

Figure 5.1 shows a black-box representation of the proposed GNN-based network model. The input of RouteNet-E is a network state sample, defined by: a network topology, a set of traffic models (flow level), a routing scheme (flow level), and a queuing configuration (interface level). As output, this model produces estimates of relevant performance metrics at a flow-level granularity (e.g., delay, jitter, losses).

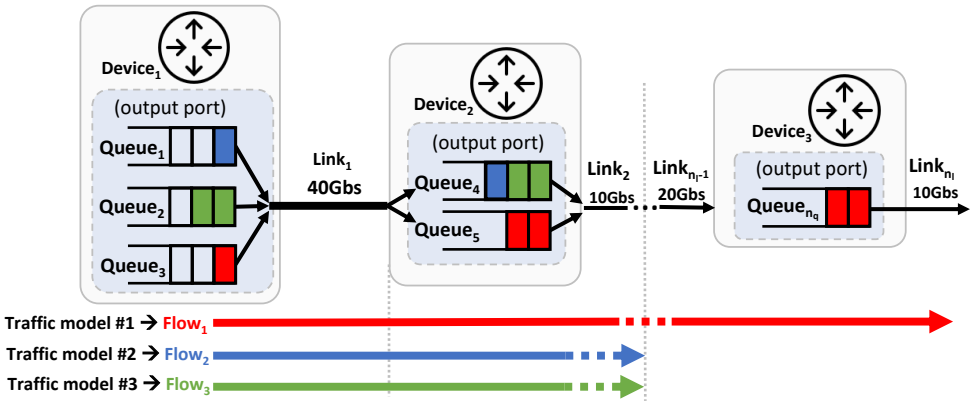


FIGURE 5.2: Schematic representation of the network model implemented by RouteNet-E.

5.1.1 Model Description

RouteNet-E has two main building blocks: (1) Finding a good representation for the network components supported by the model – e.g., traffic models, routing, queue scheduling –, and (2) Exploit scale-independent features of networks, in order to achieve good generalization power to larger networks than those seen during training, which is an important open challenge previously discussed in Section 3.4.

1) Representing network components and their relationships:

First, let us define a network as a set of links $L = \{l_i : i \in (1, ..., n_l)\}$, a set of queues on $Q = \{q_i : i \in (1, ..., n_q)\}$, and a set of source-destination flows $F = \{f_i : i \in (1, ..., n_f)\}$. According to the routing configuration, flows follow a source-destination path. Hence, we define flows as sequences with tuples of the queues and links they traverse $f_i = \{(q_{F_q(f_i,0)}, l_{F_l(f_i,0)}), ..., (q_{F_q(f_i,|f_i|)}, l_{F_l(f_i,|f_i|)})\}$, where $F_q(f_i, j)$ and $F_l(f_i, j)$ respectively return the index of the j-th queue or link along the path of flow f_i. Let us also define $Q_f(q_i)$ as a function that returns all the flows passing through queue q_i, and $L_q(l_i)$ as a function that returns the queues injecting traffic into link l_i – i.e., the queues at the output port to which the link is connected.

Following the previous notation, RouteNet-E considers an input graph with three main components: (i) the physical links L that shape the network topology, (ii) the queues Q at each output port of network devices, and (iii) the active flows F in the network, which follow some specific src-dst paths (i.e., sequences of queues and links), and whose traffic is generated from a given traffic model. Figure 5.2 shows a schematic representation of

the network model internally considered by RouteNet-E, which is derived from the several mechanisms that affect performance in real networks. From this model, we can extract three basic principles:

(i) The state of flows (e.g., throughput, losses) is affected by the state of the queues and links they traverse (e.g., queue/link utilization).

(ii) The state of queues (e.g., occupation) depends on the state of the flows passing through them (e.g., traffic model).

(iii) The state of links (e.g., utilization) depends on the states of the queues at the output port of the link, and the queue scheduling policy applied over these queues.

Formally, these principles can be formulated as follows:

$$h_{f_k} = g_f(h_{q_{k(0)}}, h_{l_{k(0)}}, ..., h_{q_{k(|f_k|)}}, h_{l_{k(|f_k|)}}) \tag{5.1}$$

$$h_{q_i} = g_q(h_{p_1}, ..., h_{p_m}), \quad q_i \in p_k, k = 1, ..., j \tag{5.2}$$

$$h_{l_j} = g_l(h_{q_1}, ..., h_{q_m}), \quad q_m \in L_q(l_j) \tag{5.3}$$

Where g_f, g_q, and g_l are some unknown functions, and h_f, h_q and h_l are latent variables that encode information about the state of flows F, queues Q, and links L respectively. Note that these principles define a circular dependency between the three network components (F, Q, and L) that must be solved to find latent representations satisfying the equations above.

Based on the previous network modeling principles, we define the architecture of RouteNet-E (see Algorithm 2). Our GNN-based model implements a custom three-stage message-passing algorithm that combines the states of flows, queues, and links according to Equations (5.1)-(5.3), thus aiming to resolve the circular dependencies defined in such functions. First, the hidden states h_l, h_q, and h_f – represented as n-element vectors – are initialized with some features (lines 1-3), denoted respectively by x_{l_i}, x_{q_j} and x_{f_k}. In our case, we set the initial features of links (x_l) as (i) the link capacity (C_i), and (ii) the scheduling policy at the output port of the link (FIFO, SP, WFQ, or DRR), using one-hot encoding. For the initial features of queues (x_q) we include: (i) the buffer size, (ii) the priority level (one-hot encoding), and (iii) the weight (only for WFQ and DRR). Lastly, the initial flow features (x_f) are

Algorithm 2 Internal architecture of RouteNet-E

Input: F, Q, L, x_f, x_q, x_l
Output: h_q^T, h_l^T, h_f^T, $\hat{y}_f$, $\hat{y}_q$, $\hat{y}_l$

1: **for each** $l \in L$ **do** $h_l^0 \leftarrow [x_l, 0...0]$
2: **for each** $q \in Q$ **do** $h_q^0 \leftarrow [x_q, 0...0]$
3: **for each** $f \in F$ **do** $h_f^0 \leftarrow [x_f, 0...0]$
4: **for** t = 0 to T-1 **do** ▷ Message Passing Phase
5: **for each** $f \in F$ **do** ▷ Message Passing on Flows
6: **for each** $(q, l) \in f$ **do**
7: $h_f^t \leftarrow FRNN(h_f^t, [h_q^t, h_l^t])$ ▷ Flow: Aggr. and Update
8: $\widetilde{m}_{f,q}^{t+1} \leftarrow h_f^t$ ▷ Flow: Message Generation
9: $h_f^{t+1} \leftarrow h_f^t$
10: **for each** $q \in Q$ **do** ▷ Message Passing on Queues
11: $M_q^{t+1} \leftarrow \sum_{f \in Q_f(q)} \widetilde{m}_{f,q}^{t+1}$ ▷ Queue: Aggregation
12: $h_q^{t+1} \leftarrow U_q(h_q^t, M_q^{t+1})$ ▷ Queue: Update
13: $\widetilde{m}_q^{t+1} \leftarrow h_q^{t+1}$ ▷ Queue: Message Generation
14: **for each** $l \in L$ **do** ▷ Message Passing on Links
15: **for each** $q \in L_q(l)$ **do**
16: $h_l^t \leftarrow LRNN(h_l^t, \widetilde{m}_q^{t+1})$ ▷ Link: Aggr. and Update
17: $h_l^{t+1} \leftarrow h_l^t$
18: $\hat{y}_f \leftarrow R_f(h_f^T)$ ▷ Readout phase
19: $\hat{y}_q \leftarrow R_q(h_q^T)$

a descriptor of the traffic model used in the flow (T_i). Once the states are initialized, the message-passing phase is iteratively executed T times (loop from line 4), where T is a configurable parameter. Each message-passing iteration is in turn divided into three stages, that respectively represent the message passing and update of the hidden states of flows h_f (lines 5-9), queues h_q (lines 10-13), and links h_l (lines 14-17).

Finally, functions R_f (line 18) and R_q (line 19) represent independent readout functions that can be respectively applied to the hidden states of flows h_f or queues h_q. In our experiments in Section 5.2, we use R_f and R_q to predict the flow-level delay, jitter and losses – as described later in this section.

The main motivation to use data-driven methods, such as RouteNet-E, instead of traditional QT is to achieve accurate modeling of complex traffic models that better reflect real-world traffic – as previously introduced in

Section 3.4. Hence, in RouteNet-E the representation of the traffic model descriptors (T_i) is central to achieve accurate modeling of different traffic patterns, and capturing their intrinsic properties. Particularly, we define T_i as an n-element vector that includes the specific parameters that shape each traffic model. Find more details about the parameters of each model in the previous section 3.1.1.

2) Scaling to larger networks: scale-independent features

As previously discussed in Section 3.4, generating datasets directly from networks in production would imply testing configurations that may break the correct operation. As a result, GNN-based network models should be typically trained with data from network testbeds, which are usually much smaller than real networks. In this context, it is essential for our GNN to effectively scale to larger networks than those of the training dataset – by at least a 10x factor.

GNNs have shown an unprecedented capability to generalize over graph-structured data [35, 30]. In the context of generalizing to larger graphs, it is well known that these models keep good generalization capabilities as long as the spectral properties of graphs are similar to those seen during training [97]. In the case of RouteNet-E, its message-passing algorithm can analogously generalize to graphs with similar structures to those seen during the training phase – e.g., similar number of queues at output ports, or similar number of flows aggregated in queues. In this vein, generating a representative dataset for RouteNet-E in small networks, covering a wide range of graph structures, does not imply any practical limitation to then achieve good generalization properties to larger networks. It can be done by simply adding a broad combination of realistic network samples with a wide variety of traffic models, routing schemes, and queuing policies as in the process described later in section 5.1.2.

However, from a practical standpoint, scaling to larger networks often entails a broader definition beyond the topology size and structure. In particular, there are two main properties we can observe as networks become larger: (i) *higher link capacities* (as there is more aggregated traffic in the core links of the network), and (ii) *longer paths* (as the network diameter becomes larger). This requires devising mechanisms to effectively scale on these two features.

Scaling to larger link capacities: If we observe the internal architecture of RouteNet-E (Algorithm 2), we can find that the link capacity C is only represented as an initial feature of links' hidden states x_{l_i}. The fact that C is encoded as a numerical feature in the model introduces inherent limitations to scale to larger capacity values. Indeed, scaling to out-of-distribution numerical values is widely recognized as a generalized limiting factor among all neural networks [98, 99]. Thus, our approach is to exploit particularities from the network domain to find scale-independent representations that can define link capacities and how they relate to other link-level features that impact performance (e.g., the aggregated traffic in the link), as the final goal of RouteNet-E is to accurately estimate performance metrics (e.g., delay, jitter, losses). Inspired by traditional QT methods, we aim to encode in RouteNet-E the relative ratio between the arrival rates on links (based on the traffic aggregated in the link), and the service times (based on the link capacity), thus enabling the possibility of inferring the output performance metrics of our model from scale-independent values. As a result, we define link capacities (Cap_{link}) as the product of a *virtual reference link capacity* (C_{ref}) and a *scale factor* (S_f) – i.e., $Cap_{link} = C_{ref} * S_f$.

This representation enables to define arbitrary combinations of scale factors and reference link capacities to define the actual capacity of links in networks. Hence, in RouteNet-E we introduce the capacity feature (C_i) as a 2-element vector defined as C_i=$[C_{ref}, S_f]$, which is included in the initial feature vector of links (x_l). Note that this feature will eventually be encoded in the hidden states of links (h_l). In the internal architecture of RouteNet-E (Algorithm 2), this factor will mainly affect the update functions of flows and links (lines 7 and 16), as they are the only ones that process directly the hidden states of links (h_l). As a result, the RNNs approximating these update functions can potentially learn to make accurate estimates on any combination of C_{ref} and S_f as long as these two features are within the range of values observed *independently* for each of them during the training phase (i.e., $S_f \in [s_{f_{min}}, s_{f_{max}}]$ and $C_{ref} \in [C_{ref_{min}}, C_{ref_{max}}]$). Thus, we exploit this property to devise a custom data augmentation method, where we take samples from small networks with limited link capacities and generate different combinations of C_{ref} and S_{factor} that enable us to scale accurately to considerably larger capacities. Note that in this process, the numerical values seen by RouteNet-E (C_{ref} and S_{factor}) are kept in the same ranges both in the training on small networks and the posterior inference on larger networks, thus

overcoming the practical limitation of out-of-distribution predictions [98, 99]. More details about the proposed data augmentation process are given in Sec. 5.1.3.

The previous mechanism enables us to keep scale-independent features along with the message-passing phase of our model (loop lines 4-17 in Algorithm 2), while it is still needed to extend the scale independence to the output layer of the model. Particularly, we use RouteNet-E to predict the flows' delay, jitter, and losses. Note that the distribution of these parameters can also vary for flows traversing links with higher capacities, thus leading again to out-of-distribution values. Based on the fundamentals of QT, we overcome this potential limitation by inferring delays/jitter indirectly from the occupation of queues in the network $O_{q_i} \in [0, 1]$, using the $\hat{y_q} = R_q(h_q)$ function of RouteNet-E (Algorithm 2). Then, we infer the flow delay/jitter as a linear combination of the waiting times in queues (inferred from O_{q_i}) and the transmission times of the links the flow traverses. Note that a potential advantage with respect to traditional QT models is that the queue occupation estimates produced by RouteNet-E can be more accurate, especially for complex traffic models resembling real-world traffic – as shown later in our experimental results of Section 5.2. Likewise, for packet loss, RouteNet-E predicts directly the percentage of packets dropped with respect to the packets that were sent by the source of the flow, thus producing a bounded value $D_{f_i} \in [0, 1]$, that is estimated with the $\hat{y}_f = R_f(h_f)$ function of Algorithm 2.

Scaling to longer paths: In the internal architecture of RouteNet-E, the path length only affects the RNN function of line 7 (Algorithm 2), which collects the state of queues (h_q) and links (h_l) to update flows' states (h_f). The main limitation here is that this RNN can typically see during training shorter link-queue sequences than those it can find then in larger networks, that can potentially have longer paths. As a result, we define L_{max} as a configurable parameter of our model that defines the maximum sequence length supported by this RNN. Then, we split flows exceeding L_{max} into different queue-link sequences that are independently digested by the RNN. To keep the state along with the whole flow, in case it is divided into more than one sequence, we initialize the initial state of the RNN with the output resulting from the previous sequence.

5.1.2 Simulation Setup

To train, validate, and test RouteNet-E we use as ground truth a packet-level network simulator (OMNeT++ v5.5.1 [8]), where network samples are labeled with performance metrics, including the flows' mean delay, jitter and losses, and queue-level statistics (e.g., occupation, packet loss).

To generate these datasets, for each sample, we randomly select a combination of input features (traffic model, topology, and queuing configuration). Table 5.1 summarizes the possible input features.

Similar to Chapter 4.4.2, traffic is generated using five different models with increasing levels of complexity, which range from a basic Poisson generation process to more realistic traffic models with strong autocorrelation and heavy-tails [16].

Each forwarding device is configured with a different scheduling policy that depends on the particular scenario of our evaluation (more details in Sec. 5.2). Overall, we use four different queue scheduling policies: First In First Out (FIFO), Strict Priority (SP), Weighted Fair Queueing (WFQ), and Deficit Round Robin (DRR). We consider three queues per output port (except for FIFO, with only one queue), and queues have a size of 8, 16, 32, or 64 packets. For WFQ and DRR, we define random queue weights.

Finally, to train the GNN model we used two different real-world topologies: NSFNET (14 nodes) [56], and GEANT (24 nodes) [57]. Then, we validate the accuracy of RouteNet-E in GBN (17 nodes) [58]. Later, in Section 5.2.4, to test the generalization capabilities to larger networks of our model, we use a wide set of topologies of variable size (from 25 to 300 nodes). All these topologies have been artificially generated using the Power-Law Out-Degree Algorithm described in [54], where the ranges of the α and β parameters have been extrapolated from real-world topologies of the Internet Topology Zoo repository [79].

5.1.3 Training

We implement RouteNet-E in TensorFlow. All the datasets, the code, and the trained models are publicly available [100]. To train the model, we use

a custom data augmentation approach that, given a link capacity (Cap_{link}), covers a broad combination of S_f and C_{ref} values, in order to eventually make the model generalize over samples with larger link capacities. Particularly, given a link capacity, in some samples, we use low values of S_f with higher values of C_{ref}, while in other samples we make it in the opposite way. As an illustrative example, if the model is trained over samples with 1Gbps links, we can represent these capacities in different samples as Cap_{link}=10*100$Mbps$=1$Gbps$, or Cap_{link}=1*1$Gbps$=1$Gbps$. Thus, after training the model should be able to make accurate inferences on samples that combine the maximum S_f and C_{ref} values seen during training – i.e., Cap_{link}=10*1$Gbps$=10$Gbps$. In practice, this means that the model can be trained with samples with a maximum link capacity of 1 Gbps, and then scale effectively to samples with link capacities up to 10 Gbps. Note that these numbers are just illustrative, while this data augmentation method is sufficiently general to produce in the training dataset wider ranges of S_f and C_{ref} given a maximum link capacity. Thus, it can be potentially exploited to represent combinations leading to arbitrarily larger capacities.

After making some grid search experiments, we set a size of 32 elements for all the hidden state vectors (h_f, h_q, h_l), and T=8 message-passing iterations. We implement $FRNN$, $LRNN$, and U_q as Gated Recurrent Units

Topology	NSFNET [56], GEANT [57], GBN [58], and scale-free synthetic topologies following the Power-Law Out-Degree algorithm [54].
Traffic Model	6 options: Poisson, On-Off, Constant Bitrate, Autocorrelated Exponentials, Modulated Exponentials (according to 3.1.1), and all models mixed.
Traffic Intensity	Random traffic intensities to generate packet loss between 0% and 3%.
Queuing Configuration	1, 2, or 3 queues per port. Queue size: 8, 16, 32, or 64 kbits. Policy: First In First Out, Strict Priority, Weighted Fair Queuing, and Deficit Round Robin.

TABLE 5.1: Simulation variables.

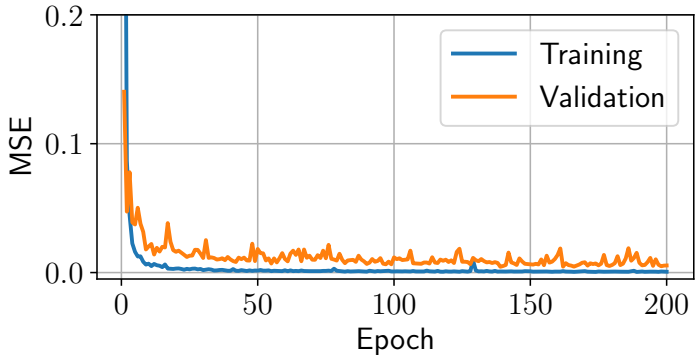


FIGURE 5.3: Training and evaluation losses over time.

(GRU) [74], and functions R_f and R_q as 2-layer fully-connected neural networks with ReLU activation functions. Here, it is important to note that the whole neural network architecture of RouteNet-E (Algorithm 2) constitutes a fully differentiable function, so it is possible to train the model end to end. Hence, all the different functions that shape its internal architecture are jointly optimized during training based on RouteNet-E's inputs (network samples) and outputs (performance metrics).

We use a training dataset with 200,000 samples from the NSFNET and GEANT topologies (100,000 samples each), including a variety of traffic model descriptors (T_i), routing schemes, and queue scheduling configurations – following the descriptions in Section 5.1.2. For the validation and test datasets, we generate 2,000 samples from the GBN topology (1,000 samples for each dataset). We train RouteNet-E for 200 epochs – with 4,000 samples per epoch – and set the Mean Squared Error (MSE) as the loss function, using an Adam optimizer with an initial learning rate of 0.001. Figure 5.3 shows the evolution of the loss during training on delay estimates (for the training and validation samples), which shows stable learning along the whole training process.

5.2 Evaluation

In this section, we evaluate the performance of RouteNet-E in a wide range of relevant scenarios. We seek to understand:

1. Can RouteNet-E model complex traffic models? What is the accuracy when predicting realistic models with strong autocorrelation and heavy tails?

2. Is RouteNet-E able to understand more complex multi-queue scheduling policies? What is the accuracy compared to QT?
3. Is RouteNet-E able to generalize to unseen network configurations and traffic loads? Also, can it generalize to *larger* networks?
4. How fast is RouteNet-E compared to the QT benchmark? Does it allow for real-time operation?

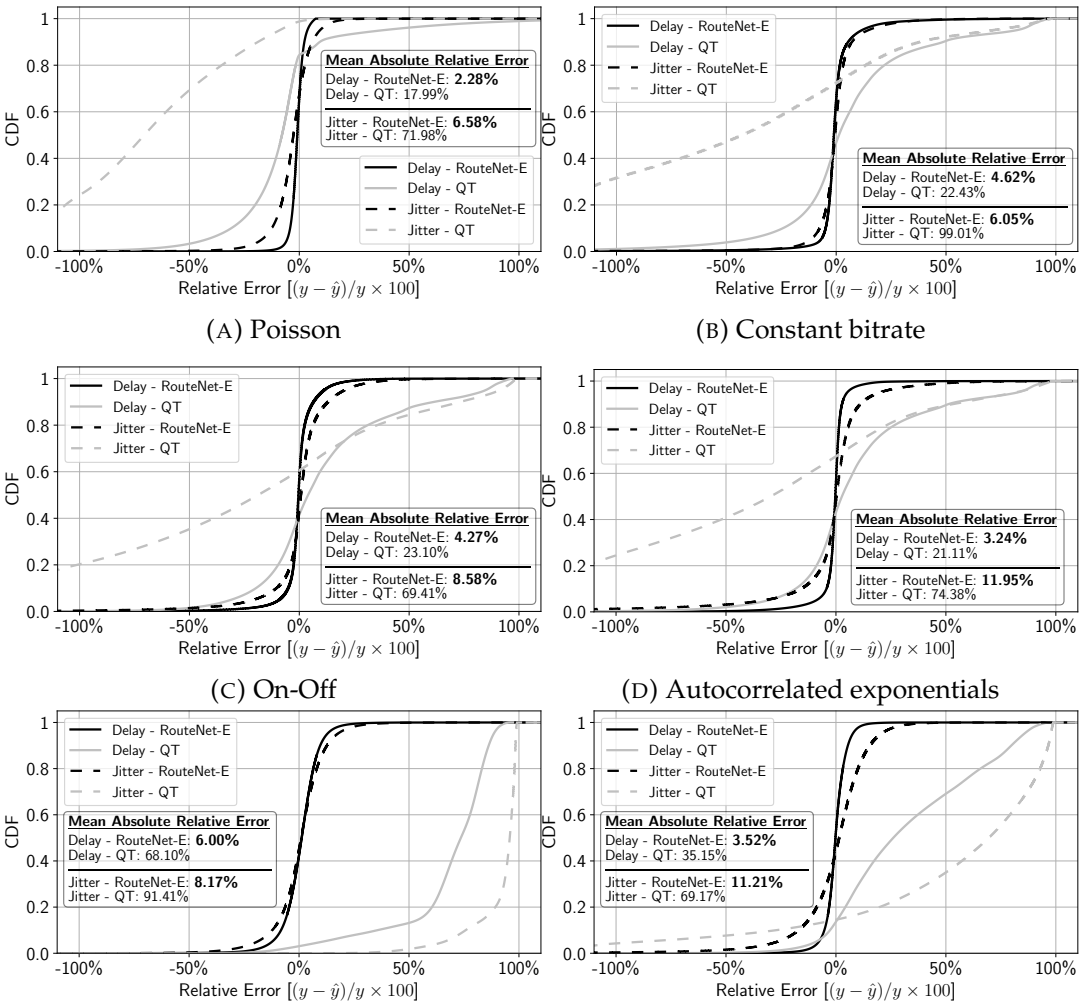
5.2.1 Evaluation Methodology

To analyze the accuracy of RouteNet-E (Sec. 5.1) and benchmark it against the state-of-the-art queuing theory model (Sec. 3.2), we use the following methodology. In all the experiments the ground truth is obtained with a packet-level simulator (see Sec. 5.1.2 for details). Unless noted otherwise, in each evaluation we perform 50k experiments with a random configuration (src-dst routing, traffic intensity, per-interface scheduling policy, and queue length) and compute the mean average delay, jitter, and losses. Then, we compute the error of RouteNet-E's and QT's estimates. For a fair comparison, we use samples of the GBN topology, which is not included during training (see Sec. 5.1.3 for training details). Finally, depending on the experiment we use different traffic models (Sec. 5.1.2) and a wide range of realistic topologies.

5.2.2 Traffic Models

This section focuses on analyzing the accuracy of RouteNet-E in a wide range of traffic models. The experiment is organized such that we add complexity to the traffic model by changing its first and second-order statistics (i.e., variance and autocorrelation). With this, we use challenging models that are good approximations to those seen in Internet links.

Figure 5.4 shows the CDF of the relative error (in %) for all the traffic models under evaluation. We plot the error for the delay and jitter estimates of both, RouteNet-E and QT. As we can observe, RouteNet-E achieves excellent results, producing very accurate estimates of delay and jitter in all traffic models, with a worst-case error below 6% for delay and 12% for jitter (mean absolute relative error).



(A) Poisson

(B) Constant bitrate

(C) On-Off

(D) Autocorrelated exponentials

(E) Modulated exponentials

(F) All traffic models multiplexed

FIGURE 5.4: CDF of the relative error for RouteNet-E and QT with different traffic models. Figures a, b, and c show models with discrete state space. Figures d and e include continuous state space. Each figure also shows numbers of the mean absolute relative error.

As expected, QT results in unacceptable performance in continuous-state traffic models (up to 68% for delay), while it achieves moderate accuracy for discrete-state models. Interestingly, QT shows poor accuracy across all the experiments estimating jitter. The reason for this is that QT assumes independence between queues in the network. Hence, the estimator used for jitter is the sum of the individual delay variance of queues along flow paths, which ignores possible covariance effects between queues.

It is remarkable that RouteNet-E is also accurate even with non-Markovian traffic models (On-Off, Figure 5.4c) and with challenging models that approximate strong autocorrelation (Autocorrelated Exponentials,

Figure 5.4d). For the latter, it has been shown in the literature that the TCP protocol generates traffic with autocorrelation for a finite range of time-scales [101]. In this scenario, RouteNet-E estimates the delay with a mean error of 3.24%.

Figure 5.4e plots the accuracy for the Modulated Exponentials model, this emulates observations found at Internet links [16] by approximating a heavy-tail. In this scenario, RouteNet-E still produces very accurate estimates. It is worth noting that this traffic model could be made even more difficult for QT by increasing both the variance and the autocorrelation factor.

The key to RouteNet-E's performance is that it has been trained for such traffic models. As discussed in Section 5.1, we have parameterized the models and trained the GNN to learn the interaction between the traffic, the queues, and the resulting performance metrics. The experiments depicted in Figure 5.4 show that RouteNet-E can generalize to traffic, providing good accuracy even for traffic models with parameters not seen in training. RouteNet-E is designed to be an extensive model, adding a new traffic model is as simple as pasteurizing it and including it in training.

To showcase this, consider the experiment shown in Figure 5.4f, where we run 100k experiments with samples where each src-dst pair uses a *random traffic model* with random parameters. Effectively, we multiplex all traffic models in a single network topology. As the figure shows, RouteNet-E is able to model this scenario in the presence of complex interactions of various multiplexed traffic models.

5.2.3 Scheduling Policies

With this experiment, we aim to validate that RouteNet-E is able to model the behavior of queues. For this, we use 100k samples of the GBN topology, each router port is configured with three different queues and with a randomly selected scheduling policy (FIFO, WFQ, DRR, SP). For WFQ and DRR, the set of weights is also randomly assigned. Moreover, each src-dst path is assigned a Quality-of-Service class that maps traffic flows to specific queues. In order to provide a fair benchmark with QT, we use only Poisson traffic.

Table 5.2 summarizes the results, which are grouped for various traffic

	Delay			Jitter			Loss		
	Low	Med	High	Low	Med	High	Low	Med	High
RouteNet-E	**2.0%**	**2.2%**	**3.3%**	**4.8%**	**6.2%**	**10.6%**	**12.61%**	**12.7%**	**12.66%**
QT	13.0%	17.3%	25.1%	49.0%	53.2%	59.6%	61.83%	59.3%	57.9%

TABLE 5.2: Mean Absolute Percentage Error for RouteNet-E and the QT-Baseline for the Scheduling Policies experiment.

intensities, from low-loaded to highly-congested scenarios, where the mean packet loss rate is around 3%. As we can observe, RouteNet-E outperforms QT, obtaining highly accurate estimates for all the evaluated metrics.

5.2.4 Generalization to larger topologies

The previous experiments have shown that RouteNet-E achieves remarkable accuracy in performance evaluation under different traffic models (Sec. 5.2.2) as well as complex scheduling policies (Sec. 5.2.3). As we have discussed in Section 3.4, ML-based network models must generalize to unseen and *larger* networks to become a practical solution. In this vein, RouteNet-E was carefully designed to address this challenge (see Sec. 5.1.1 for details).

In this set of experiments, we evaluate RouteNet-E in a wide range of networks considerably larger than the ones seen during training. Specifically, the model has been trained with topologies between 25 and 50 nodes and tested with topologies from 50 to 300 nodes. All these networks have been artificially generated using the Power-Law Out-Degree algorithm described in [54], where the ranges of the α and β parameters have been extrapolated from real-world topologies of the Internet Topology Zoo repository [79]. Link capacities and the generated traffic volume are scaled accordingly.

Figure 5.5 shows how RouteNet-E generalizes to larger topologies not seen in training. Specifically, the boxplots show the absolute relative error with respect to the topology size. As expected, RouteNet-E obtains better accuracy in topologies that are closer to the ones seen during the training phase (50 to 99 nodes), achieving an average error of 4.5% (green line). As the topology size increases, the average error stabilizes to $\approx$ 10%. Note that this value is even lower than the one obtained by the QT model, which achieves a mean error of 12.6% in samples with Poisson traffic (Fig. 5.4a). We could not test larger topologies (>300) in our cluster (180 nodes), as packet-level simulations – used for the ground truth – are sequential in nature, and, with

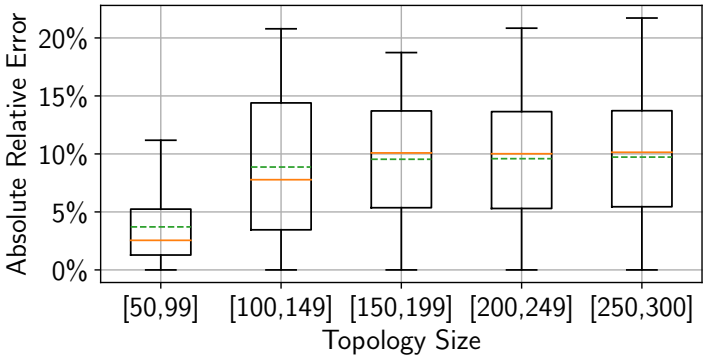


FIGURE 5.5: Absolute relative error vs. topology size.

our traffic configurations, have exponential complexity with respect to the topology size.

Generalization is an open challenge in the field of GNN. As discussed in Sec. 5.1, we have addressed this by using domain-specific knowledge and data augmentation. Particularly, we infer delay/jitter from queues' occupation and apply our scale-independent method to generalize to larger topologies.

5.2.5 Inference Speed

Finally, in this section, we evaluate the inference speed of RouteNet-E. Fast models are especially appealing for network control and management, as they can be deployed in real-time scenarios. For this, we have measured the execution times [Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz] of the experiments in the previous section, for both QT and RouteNet-E. The results (Figure 5.6) show that both models operate in the order of milliseconds. In particular, RouteNet-E goes from a few milliseconds for small topologies to a few hundred for the larger ones.

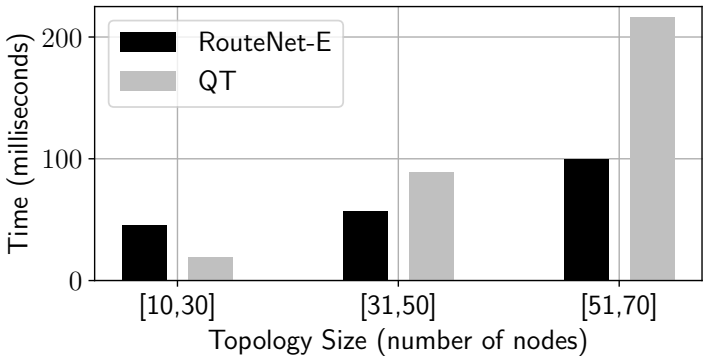


FIGURE 5.6: Execution time vs. topology size.

5.3 Discussion and Concluding remarks

In this chapter, we have presented RouteNet-E, a new tool for network modeling. RouteNet-E has shown remarkable accuracy in all the scenarios, outperforming a state-of-the-art QT model. RouteNet-E also overcomes the main limitation of QT, and it is able to model challenging traffic models. More importantly, the proposed model addresses the main drawback of existing ML-based models, and it is able to provide accurate estimates in larger networks ($\approx$10x).

RouteNet-E provides unprecedented accuracy in network performance evaluation. However, in contrast to QT, it does not help understand the behavior of the network being modeled. The knowledge learned by RouteNet-E during training is not human-understandable. This is a common issue for all ML-based models, and substantial research efforts are being devoted to producing explainable ML models [102]. However, this is still an open research problem.

RouteNet-E's performance enables network optimization, planning, and operation in real-time scenarios. It also represents an open-source extensible model. We hope that the community will use it as a baseline to incorporate additional network components, such as other scheduling policies, traffic models, etc.

Chapter 6

RouteNet-Fermi: Unifying Scheduling, Traffic Models, and Generalization in Network Modeling

In this chapter, we introduce RouteNet-Fermi, a significant advancement beyond its predecessor, RouteNet-Erlang. While RouteNet-Erlang showed incredible capacities in supporting scheduling policies, and traffic models, and demonstrating strong generalization to larger networks, it operated in distinct scenarios for each of these aspects. It fell short of seamlessly integrating traffic models, scheduling policies, and generalization within the same network scenario. RouteNet-F emerges as a novel GNN architecture designed to address and overcome these limitations within a unified model.

RouteNet-F also aligns with the objectives of Queuing Theory, offering precise performance estimates, including delay, jitter, and packet loss, for specified network scenarios (Figure 6.1).

In the subsequent sections, we also benchmark RouteNet-F against MimicNet [53], a state-of-the-art DL-based model, and a cutting-edge queuing theory model. Our evaluation demonstrates the superior performance of RouteNet-F across various scenarios. It achieves an impressive 5.64% error when tested on a dataset with packet traces from a real-world network, an 11% error in a physical testbed evaluation, and a 6.24% error when estimating delay across a large dataset with 1,000 network samples, featuring topologies ranging from 50 to 300 nodes.

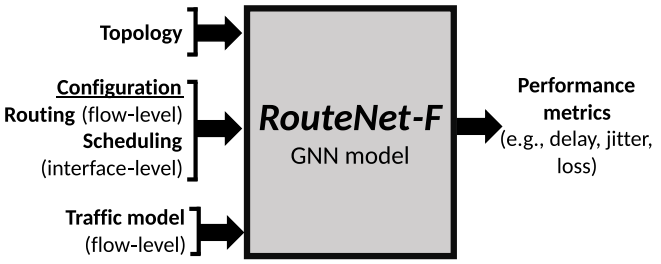


FIGURE 6.1: Black-box representation of RouteNet-F.

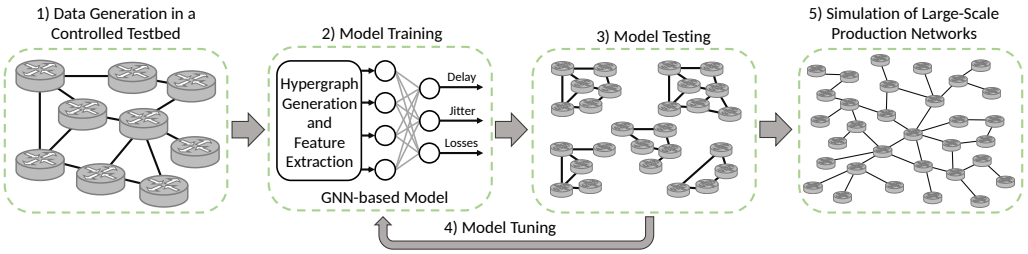


FIGURE 6.2: End-to-end workflow of RouteNet-F. 1) Collection of small-scale observations coming from a controlled environment, 2) Model training, 3) Model testing with various configurations (e.g., routing, scheduling) never seen during the training phase, 4) hyper-parameter tunning to balance highly accurate predictions with performance, 5) Simulation of large-scale production networks. One of the main advantages of RouteNet-F is that usually time-consuming steps like 1), 2), 3), and 4) are all done at small scales and, therefore, are fast as well.

6.1 RouteNet-Fermi

This section describes the internal GNN architecture of RouteNet-Fermi (hereafter referred to as RouteNet-F). This GNN-based model implements a custom three-stage message-passing algorithm that represents key elements for network modeling (e.g., topology, queues, traffic flows). RouteNet-F supports a wide variety of features present in real-world networks, such as multi-queue QoS scheduling policies or complex traffic models.

Figure 6.1 shows a black-box representation of RouteNet-F. The input of this model is a network sample, defined by: a network topology, a routing scheme (flow level), a queuing configuration (interface level), and a set of traffic flows characterized by some parameters. As output, the model produces estimates of relevant performance metrics at a flow-level granularity (e.g., delay, jitter, packet loss). Figure 6.2 shows the end-to-end workflow to train, validate, and use RouteNet-Fermi.

6.1.1 Model Description

RouteNet-F is based on two main design principles: (i) *finding a good representation* of the network components supported by the model (e.g., traffic models, routing, queue scheduling), and (ii) *exploit scale-independent features* of networks to accurately scale to larger networks unseen during training. These two aspects are further discussed in the next two subsections.

6.1.2 Representing network components and their relationships

First, let us define a network as a set of source-destination flows $\mathcal{F} = \{f_i : i \in (1, ..., n_f)\}$, a set of queues on $\mathcal{Q} = \{q_j : j \in (1, ..., n_q)\}$, and a set of links $\mathcal{L} = \{l_k : k \in (1, ..., n_l)\}$. According to the routing configuration, flows follow a source-destination path. Hence, we define flows as sequences of tuples with the queues and links they traverse $f_i = \{(q^{i,1}, l^{i,1}), ..., (q^{i,M}, l^{i,M})\}$, where M is the path length of the flow (number of links). Let us also define $Q_f(q_j)$ and $L_f(l_k)$ as functions that respectively return all the flows passing through a queue q_j or a link l_k. Also, $L_q(l_k)$ is defined as a function that returns the queues $q_{l_k} \in \mathcal{Q}$ injecting traffic into link l_k (i.e., the queues at the output port to which the link is connected).

Following the previous notation, RouteNet-F considers an input graph with three main components: (i) the physical links $\mathcal{L}$ that shape the network topology, (ii) the queues $\mathcal{Q}$ at each output port of network devices, and (iii) the active flows $\mathcal{F}$ in the network, which follow some specific src-dst path (i.e., sequences of queues and links). Traffic in flows is generated from a given traffic model. From this, we can extract three basic principles:

1. The state of flows (e.g., delay, throughput, packet loss) is affected by the state of the queues and links they traverse (e.g., queue/link utilization).
2. The state of queues (e.g., occupation) depends on the state of the flows passing through them (e.g., traffic volume, burstiness).

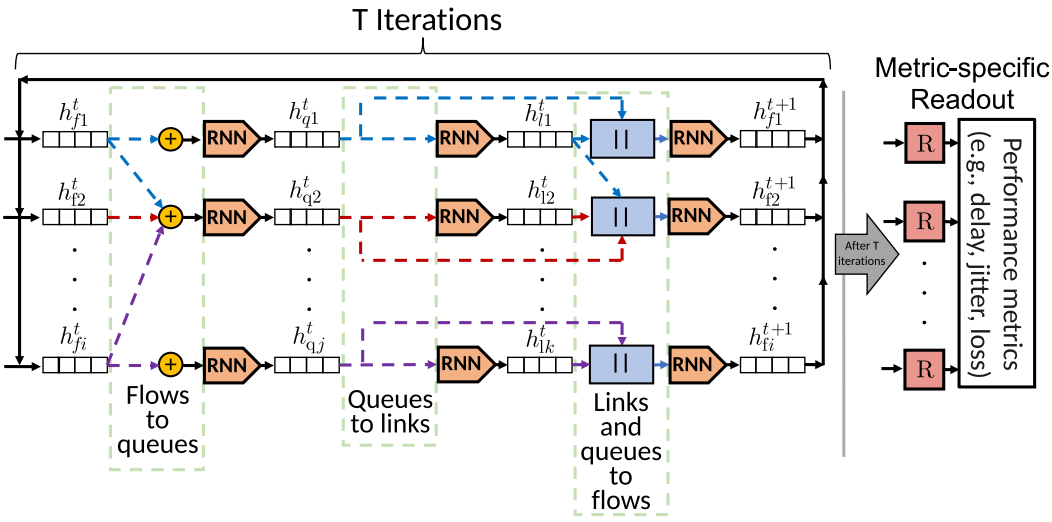


FIGURE 6.3: Schematic representation of RouteNet-F.

3. The state of links (e.g., utilization) depends on the states of the queues that can potentially inject traffic into the link, and the queue scheduling policy applied over these queues (e.g., Strict Priority, Weighted Fair Queuing).

Formally, these principles can be formulated as follows:

$$\boldsymbol{h}_{f_i} = G_f(\boldsymbol{h}_{q^{i,1}}, \boldsymbol{h}_{l^{i,1}}, ..., \boldsymbol{h}_{q^{i,M}}, \boldsymbol{h}_{l^{i,M}}) \tag{6.1}$$

$$\boldsymbol{h}_{q_j} = G_q(\boldsymbol{h}_{f_1}, ..., \boldsymbol{h}_{f_I}), \quad f_i \in Q_f(q_j) \tag{6.2}$$

$$\boldsymbol{h}_{l_k} = G_l(\boldsymbol{h}_{q_1}, ..., \boldsymbol{h}_{q_J}), \quad q_j \in L_q(l_j) \tag{6.3}$$

Where G_f, G_q, and G_l are some unknown functions, and $\boldsymbol{h}_f$, $\boldsymbol{h}_q$ and $\boldsymbol{h}_l$ are latent variables that encode information about the state of flows $\mathcal{F}$, queues $\mathcal{Q}$, and links $\mathcal{L}$ respectively. Note that these principles define a circular dependency between the three network components ($\mathcal{F}$, $\mathcal{Q}$, and $\mathcal{L}$) that must be solved to find latent representations satisfying the equations above.

To solve the circular dependencies defined in Equations (6.1)-(6.3), RouteNet-F implements a three-stage message passing algorithm that combines the states of flows $\mathcal{F}$, queues $\mathcal{Q}$, and links $\mathcal{L}$, and updates them iteratively. Finally, it combines these states to estimate flow-level delays, jitters, and packet loss. Figure 6.3 shows a schematic representation of the internal three-stage message-passing architecture of this model.

Algorithm 3 describes the architecture of RouteNet-F. First, hidden

states $\boldsymbol{h}_f$, $\boldsymbol{h}_q$, and $\boldsymbol{h}_l$ are initialized using the functions HS_f, HS_q, and HS_l respectively (lines 1-3). These functions encode the initial features $\boldsymbol{x}_f$, $\boldsymbol{x}_q$, and $\boldsymbol{x}_l$ into fixed-size vectors that represent feature embeddings. The initial features of flows $\boldsymbol{x}_f$ are defined as an n-element vector that characterizes the flow's traffic. For example, in our case, this vector includes the average traffic volume transmitted in the flow λ, and some specific parameters of the traffic model, such as t_{on} and t_{off} for On-Off traffic distributions or α and β for exponential models. We set the initial features of links $\boldsymbol{x}_l$ as (i) the link load $x_{l_{load}}$, and (ii) the scheduling policy at the output port of the link (FIFO, Strict Priority, Weighted Fair Queuing, or Deficit Round Robin). For the scheduling policy, we use a one-hot encoding. The calculation of the link load $x_{l_{load}}$ is defined in more detail later (Sec. 6.1.3). Lastly, the initial features of queues $\boldsymbol{x}_q$ include: (i) the buffer size, (ii) the queue order/priority level (one-hot encoding), and (iii) the weight (only for Weighted Fair Queuing or Deficit Round Robin configurations).

Once all the hidden states are initialized, the message-passing phase starts. This phase is executed for T iterations (loop from line 4), where T is a configurable parameter of the model. Each message passing iteration is divided into three stages, which represent respectively the message exchanges and updates of the hidden states of flows $\boldsymbol{h}_f$ (lines 5-10), queues $\boldsymbol{h}_q$ (lines 11-14), and links $\boldsymbol{h}_l$ (lines 15-19).

Finally, the loop from line 20 computes the different flow-level performance metrics. Here, function R_{f_d} (line 24) and R_{f_j} (line 28) represent a readout function that is individually applied to the hidden states of flows as they pass through a specific link ($\boldsymbol{h}_{f,l}$). The output of these functions is the average queue occupancy and the delay variation (i.e., jitter) seen by the flow at that link. Note that different flows may experience different queue occupancies and jitter depending on their traffic properties (e.g., traffic volume, burstiness). Lastly, these link-level delay and jitter estimates are combined to compute the final flow-level delay $\hat{y}_{f_d}$ and jitter $\hat{y}_{f_j}$. This calculation is further described in Section 6.1.3. Similarly, R_{f_l} (line 29) is applied to the hidden states of flows h_f to compute the per-flow packet loss rate.

Algorithm 3 Internal architecture of RouteNet-F.

Input: $\mathcal{F}$, $\mathcal{Q}$, $\mathcal{L}$, $\boldsymbol{x}_f$, $\boldsymbol{x}_q$, $\boldsymbol{x}_l$
Output: $\hat{y}_{f_d}$

for each $f \in \mathcal{F}$ **do** $\boldsymbol{h}_f^0 \leftarrow HS_f(\boldsymbol{x}_f)$
for each $q \in \mathcal{Q}$ **do** $\boldsymbol{h}_q^0 \leftarrow HS_q(\boldsymbol{x}_q)$
for each $l \in \mathcal{L}$ **do** $\boldsymbol{h}_l^0 \leftarrow HS_l(\boldsymbol{x}_l)$

for t = 0 to T-1 **do** ▷ Message Passing Phase
  for each $f \in \mathcal{F}$ **do** ▷ Message Passing on Flows
    $\Theta([\cdot,\cdot]) \leftarrow FRNN(\boldsymbol{h}_f^t, [\cdot,\cdot])$ ▷ FRNN Initialization
    for each $(q,l) \in f$ **do**
      $\boldsymbol{h}_{f,l}^t \leftarrow \Theta([\boldsymbol{h}_q^t, \boldsymbol{h}_l^t])$ ▷ Flow: Aggr. and Update
      $\widetilde{m}_{f,q}^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$ ▷ Flow: Message Generation
    $\boldsymbol{h}_f^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$
  for each $q \in \mathcal{Q}$ **do** ▷ Message Passing on Queues
    $M_q^{t+1} \leftarrow \sum_{f \in \mathcal{Q}_f(q)} \widetilde{m}_{f,q}^{t+1}$ ▷ Queue: Aggregation
    $\boldsymbol{h}_q^{t+1} \leftarrow U_q(\boldsymbol{h}_q^t, M_q^{t+1})$ ▷ Queue: Update
    $\widetilde{m}_q^{t+1} \leftarrow \boldsymbol{h}_q^{t+1}$ ▷ Queue: Message Generation
  for each $l \in \mathcal{L}$ **do** ▷ Message Passing on Links
    $\Psi(\cdot) \leftarrow LRNN(\boldsymbol{h}_l^t, \cdot)$ ▷ LRNN Initialization
    for each $q \in L_q(l)$ **do**
      $h_l^t \leftarrow \Psi(\widetilde{m}_q^{t+1})$ ▷ Link: Aggr. and Update
    $\boldsymbol{h}_l^{t+1} \leftarrow h_l^t$

for each $f \in F$ **do** ▷ Flow: Readout
  $\hat{y}_{f_d} = 0$ ▷ Initializing the flow delay
  $\hat{y}_{f_j} = 0$ ▷ Initializing the flow jitter
  for each $(q,l) \in f$ **do**
    $\hat{d}_q = R_{f_d}(\boldsymbol{h}_{f,l}^T)/\boldsymbol{x}_{l_c}$ ▷ Queuing delay
    $\hat{d}_t = \boldsymbol{x}_{f_{ps}}/\boldsymbol{x}_{l_c}$ ▷ Transmission delay
    $\hat{d}_{link} = \hat{d}_q + \hat{d}_t$
    $\hat{y}_{f_d} = \hat{y}_{f_d} + \hat{d}_{link}$ ▷ Sum of link delays along the flow
    $\hat{y}_{f_j} = \hat{y}_{f_j} + R_{f_j}(\boldsymbol{h}_{f,l}^T)/\boldsymbol{x}_{l_c}$ ▷ Sum of link jitters along the flow
  $\hat{y}_{f_l} = R_{f_l}(\boldsymbol{h}_f^T)$ ▷ Packet loss prediction

6.1.3 Scaling to larger networks: scale-independent features

Data-driven models typically need to see edge cases that are uncommon in real-world production networks (e.g., link failures). This means that collecting data directly from production networks requires testing configurations that might break the correct operation of the network. As a result, data-driven network models should be typically trained with data from controlled network testbeds. However, network testbeds are usually much smaller than real networks. In this context, it is essential for our model to effectively scale

to larger networks than those seen during the training phase.

It is well-known that GNN models have an unprecedented capability to generalize over graph-structured data [35, 30]. In the context of scaling to larger graphs, it is also known that GNNs keep good generalization capabilities as long as the spectral properties of graphs are similar to those seen during training [97]. In our particular case, the internal message passing architecture of RouteNet-F generalizes accurately to graphs with similar structures (e.g., a similar number of queues at output ports, or a similar number of flows aggregated in queues). In practice, this means that RouteNet-F should be able to generalize to larger topologies when trained with smaller ones, as long as the networks used for training had the same spectral properties as the larger ones. More details about this can be found in Section 6.2, which presents an empirical evaluation of RouteNet-F's generalization capabilities).

However, scaling to larger networks often entails other aspects beyond the topology size. Two key elements require special attention: link capacities and the range of output variables. First, larger networks naturally have larger link capacities. This in turn results in larger traffic aggregates on core links of the network. Such traffic intensities may fall in ranges not seen in smaller topologies. Second, as the network scales, it inherently has longer end-to-end paths, which result in increased end-to-end path delays. Again, some of these delays may fall outside the range of delays used when training with smaller topologies. These out-of-range parameters require devising mechanisms to effectively scale on them.

Scaling to larger link capacities

In RouteNet-F (Algorithm 3), the most straightforward way to represent the link capacity x_{l_c} would be as an initial feature of the links' hidden states x_l. However, the fact that x_{l_c} would be encoded as a numerical input value would then introduce inherent limitations to scale to larger capacity values. Indeed, scaling to out-of-distribution numerical values is recognized as a generalized limiting factor among all neural network models [98, 99].

Our approach is to exploit particularities from the network domain to find scale-independent representations for link capacities. These representations define link capacities and how they relate to other link-level features

that impact performance (e.g., the aggregated traffic in the link), so they can be used to accurately estimate performance metrics (e.g., delay, jitter, packet loss). Inspired by traditional QT methods, we aim to encode in RouteNet-F the relative ratio between the arrival rates on links (based on the traffic aggregated in the link) and the service times (based on the link capacity). This enables the possibility to infer the output performance metrics of our model from scale-independent values. As a result, instead of directly using the numerical link capacity values, we introduce the *link load* $x_{l_{load}}$ in the initial feature vector of links $\boldsymbol{x}_l$. Particularly, we compute the link load as follows:

$$x_{l_{load}} = \frac{1}{x_{l_c}} \sum_{f \in L_f(l_j)} \lambda_f \tag{6.4}$$

Where λ_f is the average traffic volume of the flows that traverse the link l_j, and x_{l_c} is the link capacity. In other words, we compute the link load as the summation of all the traffic that would traverse the link without considering possible losses and divide it by the link capacity. Then, through the iterative message-passing process, the GNN model updates the load values after estimating the packet loss.

Different output ranges

The previous mechanism enables us to keep scale-independent features along with the message-passing phase of our model (loop lines 4-19 in Alg. 3), while it is still needed to extend the scale independence to the output layer of the model. Note that in larger networks, delay values can vary with respect to those seen during the training in smaller networks. This is because flows can go through links with higher capacities, or because flows can potentially traverse longer paths. This again poses the challenge of generalizing to ranges of delays not seen in the training phase. Equivalently, this also applies to the prediction of flow jitter and packet loss.

To overcome this potential limitation, RouteNet-F infers delays indirectly from the mean queue occupancy on forwarding devices. Based on traditional QT models, RouteNet-F infers the flow delay as a linear combination of the estimated queuing delays (line 24) and the transmission delays after crossing a link (line 25). Note that a potential advantage with respect

to traditional QT models is that the queue occupancy estimates produced by RouteNet-F can be more accurate, especially for autocorrelated and heavy-tail traffic models.

We call the values produced by the R_{f_d} function the *effective queue occupancy*, which is defined as the mean queue occupancy experienced by a given flow f_i as it passes through a specific forwarding device. More precisely, this value is the average number of bits that have to be served on a specific output port before the packets of flow f_i are transmitted. As an example, let us consider the case of packets from a flow with low priority, which are mapped to low-priority queues. If forwarding devices implement a multi-queue Strict Priority scheduling policy, the effective queue occupancy seen by those low-priority packets should include all the bits to be served in the queues with higher priority.

The prediction of this effective queue occupancy — instead of directly predicting delays — helps overcome the practical limitation of producing out-of-range delay values with the readout function R_{f_d}. In this case, the values produced by R_{f_d} are bounded between 0 and the maximum buffer size at the output ports of forwarding devices. Note that the buffer size is a device-specific feature that is independent of the network size.

Lastly, RouteNet-F produces flow-level delay predictions $\hat{y}_{f_d}$ by combining the estimated queuing and transmission delays. The queuing delay $\hat{d}_q$ is indirectly estimated by using the effective queue occupancies (in bits) on queues for a particular flow. Particularly, queue occupancy values are estimated by the readout function $R_{f_d}(\boldsymbol{h}_{f,l}^T)$. Then, they are divided by the capacity of the link connected to the output port x_{l_c} to eventually produce a queuing delay estimate $\hat{d}_q$. Likewise, the transmission delay $\hat{d}_t$ is computed by dividing the mean flow packet size $x_{f_{ps}}$ by the link capacity x_{l_c}. With this, RouteNet-F estimates the delay of a flow after passing through a specific forwarding device and a link ($\hat{d}_{link}$):

$$\hat{d}_q = \frac{R_{f_d}(\boldsymbol{h}_{f,l}^T)}{x_{l_c}} \tag{6.5}$$

$$\hat{d}_t = \frac{x_{f_{ps}}}{x_{l_c}} \tag{6.6}$$

$$\hat{d}_{link} = \hat{d}_q + \hat{d}_t \tag{6.7}$$

Hence, we can compute end-to-end flow delays as the sum of all the link delays $\hat{d}_{link}$ along the flow (loop lines 20-27 in Algorithm 3). Note that the function responsible for computing the effective queue occupancy, $R_{f_d}(\boldsymbol{h}_{f,l}^T)$ takes as input the hidden state of the flow at a specific link $\boldsymbol{h}_{f,l}^T$, instead of directly considering queue states $\boldsymbol{h}_q^T$. This is because each flow can experience a different queuing behavior depending on its properties (e.g., traffic burstiness).

Likewise, jitter estimates $\hat{y}_{f_j}$ are produced by combining the jitter predictions of all links along the flow. These predictions are made by the R_{f_j} function, which takes as input the hidden state of the flow at a specific link $\boldsymbol{h}_{f,l}^T$. Note that we define the jitter as the relative fluctuation with respect to the mean delay, that is, the ratio between the delay variance divided by the flow mean delay.

In the case of packet loss, RouteNet-F makes predictions directly on flows' hidden states h_f^T. We define the packet loss as the relative ratio of packets dropped with respect to the packets transmitted by the source; hence it is a bounded value $\hat{y}_{f_l} \in [0, 1]$. We estimate it with the R_{f_l} function (line 29 in Algorithm 3).

6.1.4 Training and Implementation

We implement RouteNet-Fermi in TensorFlow and it is publicly available at [103].

As in any other ML model, fine-tuning the hyperparameters of RouteNet-F is crucial for achieving optimal performance and accuracy. Two key parameters to consider are the hidden state vectors ($\boldsymbol{h}_f$, $\boldsymbol{h}_q$, $\boldsymbol{h}_l$) and the number of message passing iterations (T). The size of the hidden state vectors determines how much information the model can encode, with larger sizes allowing for more information but also harming performance. Similarly, a larger number of message-passing iterations can help the model reach a higher level of convergence, but at the cost of increased computation. Through grid search experiments, we selected a set of hyperparameters that provide good accuracy while also being efficient. Specifically, we set the size of all the hidden state vectors to 32 elements and T to 8 message-passing iterations.

We implement the functions $FRNN$ (Flow-Level RNN), $LRNN$ (Link-Level RNN), and U_q (Queue Update Function) as Gated Recurrent Units (GRU). Functions HS_f, HS_q, and HS_l are implemented as 2-layer fully-connected neural networks with ReLU activation functions with 32 units each. Similarly, functions R_{f_d}, R_{f_j}, and R_{f_l} are implemented as a 3-layer fully-connected neural networks with ReLU activation function for the hidden layers, and a linear one for the output layer (except for R_{f_l} that uses a Sigmoid activation function). Note that, the whole architecture of RouteNet-F (Algorithm 3) constitutes a fully-differentiable function. This means that it can be trained end-to-end using as input the network samples and as output the different flow-level performance metrics (e.g., delay, jitter, packet loss) as illustrated in the black box diagram of Figure 6.1.

For each set of experiments, we train RouteNet-F during 150 epochs of 2,000 samples each. We set as loss function the Mean Absolute Percentage Error for the delay experiments, the Mean Squared Error for jitter, and the Mean Absolute Error for the packet loss. In all the cases, we use an Adam optimizer with an initial learning rate of 0.001.

6.2 Evaluation

In this section, we evaluate the performance of RouteNet-F in a wide range of relevant scenarios. First, we evaluate RouteNet-F in a variety of scenarios with complex traffic models and scheduling policies and compare it to the state-of-the-art Queuing Theory (QT) benchmark from Section 3.2. In it, the network is modeled as a $M/M/1/b$ system where each queue along a path is treated independently. Note that the baseline, like the majority of QT models, assumes that arrivals to each queue are approximated by a Poisson process and that service times are exponentially distributed. Under these assumptions, the model derives analytical results for queue throughput, delay distributions, and blocking probabilities. A public implementation of the QT model can be found at [104].

Then, we evaluate RouteNet-F's generalization capabilities when evaluated in topologies x30 times larger than the ones seen during training and compare its inference times. Finally, we benchmark RouteNet-F in real-world scenarios with data from a real testbed, with traffic coming from real-world

networks, and compare its performance with MimicNet [53] a state-of-the-art DL-based model.

6.2.1 Performance Analysis

Methodology

In all the experiments (except for subsection 6.2.3 that comes from a real testbed), the ground truth is obtained using a packet-level simulator (Section 6.2.1). Unless specified, in each evaluation we perform 50k experiments with a random configuration (src-dst routing, traffic intensity, per-interface scheduling policy, queue size, and traffic model), and compute the mean average delay, jitter, and packet loss. Then we compute the error of RouteNet-F's and QT's estimates with respect to the results of the packet simulator. For a fair comparison, the evaluation samples have not been used in the training phase of RouteNet-F.

Dataset

We generate our dataset by simulating a wide range of network scenarios with the OMNet++ network simulator, v5.5.1 [8]. An image of the simulator is publicly available and can be found at [105]. Each dataset sample corresponds to a single simulation, and we record the mean delay, jitter, and packet loss for all the flows in the network, as well as queue-level statistics(e.g., mean occupancy, average packet loss, or average packet size). We select the different scenarios to simulate by randomly sampling from the possible values of all the input variables (Table 5.1). The traffic models *autocorrelated exponentials* and *modulated exponentials* reproduce realistic Internet traffic [12, 106]. We define the traffic intensity (TI) as a tunable parameter that defines the overall traffic load in the network scenario. TI represents how congested is the network. In our dataset, it ranges from 400 to 2000 bits per time unit, with 400 being the lowest congested network (0% avg. packet loss) and 2000 a highly congested network ($\approx$3% avg. packet loss).

Traffic Models

This section analyzes the accuracy of RouteNet-F in a wide range of traffic models. The experiment is organized such that, for each traffic model, we add a degree of complexity by changing its first and second-order statistics (i.e., variance and autocorrelation). We start with simple traffic models such as Poisson or Constant Bitrate and end by testing more complex models that are better approximations of traffic seen in Internet links.

	QT				RouteNet-F			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
Poisson	12.6%	**0.001**	**0.017**	0.998	**2.1%**	**0.001**	**0.017**	**0.999**
Deterministic	22.4%	0.715	0.321	0.611	**4.43%**	**0.029**	**0.048**	**0.984**
On-Off	23.1%	0.784	0.363	0.613	**2.90%**	**0.009**	**0.035**	**0.995**
A. Exponentials	21.1%	0.686	0.316	0.618	**2.62%**	**0.010**	**0.030**	**0.994**
M. Exponentials	68.1%	1.10	0.798	0.145	**5.21%**	**0.013**	**0.061**	**0.989**
Mixed	35.1%	0.721	0.430	0.560	**4.71%**	**0.018**	**0.054**	**0.988**

TABLE 6.1: Delay prediction using the QT baseline and RouteNet-F for different traffic models. The error is computed w.r.t. simulation results.

	QT				RouteNet-F			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
Poisson	71.9%	0.013	0.072	0.849	**6.26%**	**0.001**	**0.013**	**0.980**
Deterministic	99.0%	0.057	0.067	-1.86	**7.17%**	**0.001**	**0.008**	**0.924**
On-Off	69.4%	0.057	0.098	0.425	**8.50%**	**0.004**	**0.018**	**0.959**
A. Exponentials	74.3%	0.025	0.067	0.246	**6.29%**	**0.001**	**0.008**	**0.973**
M. Exponentials	91.4%	1.34	0.834	-0.622	**10.3%**	**0.036**	**0.091**	**0.956**
Mixed	69.1%	0.299	0.296	0.025	**9.82%**	**0.007**	**0.034**	**0.974**

TABLE 6.2: Jitter prediction using the QT baseline and RouteNet-F for different traffic models. The error is computed w.r.t. simulation results.

	Poisson		Const. Bitrate		On-Off		A. Exponentials		M. Exponentials		Multiplexed	
	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2
QT	1.0%	0.97	11%	0.64	9.5%	0.63	10%	0.65	9.5%	0.08	4.6%	0.50
RouteNet-F	**0.3%**	**0.99**	**1.0%**	**0.99**	**1.0%**	**0.99**	**1.2%**	**0.99**	**1.1%**	**0.98**	**0.50%**	**0.99**

TABLE 6.3: Packet Loss evaluation - Mean Absolute Error and Coefficient of Determination (R^2) of QT and RouteNet-F for the different traffic models.

Tables 6.1 and 6.2 show the errors of the delay and jitter for both, RouteNet-F and QT, with respect to the values obtained using the simulator. We can see that RouteNet-F achieves excellent accuracy results, producing very accurate estimates of delay and jitter in all traffic models: the worst cases are 5.21% and 10.40% for delay and jitter, respectively.

As expected, the estimates of the QT model are unacceptable in continuous-state traffic models, e.g. almost 70% for modulated exponentials. On the other hand, it achieves moderate accuracy for discrete-state models (Poisson, Deterministic, and On-Off). It is also noticeable how the QT model shows poor accuracy across all the jitter estimations. The main reason for this is that QT assumes independence between queues in the network. Hence, the estimator used to compute the jitter is the sum of the individual delay variance of queues along the flow's paths.

For non-Markovian traffic models (e.g., On-Off), RouteNet-F produces accurate estimates, as well as for more challenging models that implement strong autocorrelation (Autocorrelated Exponentials) and heavy-tail distributions (Modulated Exponentials). These models are important since they approximate real traffic generated by TCP [101], similar to that found at Internet links [16]. Also, notice that this traffic model could be made even more difficult for QT by increasing both the variance and the autocorrelation factor.

We add a final scenario (Mixed) where each src-dst pair generates traffic by randomly selecting one of the five available traffic models, and using random parameters for these models. In other words, we multiplex all traffic models in a single network topology. As the table shows, RouteNet-F shows good accuracy not only when modeling individual traffic models, but also when they are mixed across links in the network. It is worth noting that although RouteNet-F shows excellent accuracy for arbitrary parameterizations of these 6 traffic models, it cannot generalize to new traffic models not introduced during the training phase.

Finally, Table 6.3 shows the different metrics for the packet loss ratio. Since there are some paths where the packet loss ratio is zero, we provide the Mean Absolute Error and the Coefficient of Determination (R^2). The packet loss ratio is measured as the percentage of packets dropped w.r.t. packets sent, that is why the MAE is expressed in % units. In this particular case, it is noticeable how the QT baseline works well in the scenario with the Poisson traffic model. However, the accuracy decreases remarkably in more complex

scenarios. On the other hand, RouteNet-F obtains a high accuracy with a worst-case MAE of 1.2% and $R^2 \geq 0.98$.

Scheduling

This section aims to validate if RouteNet-F is capable of modeling the behavior of queues in the presence of several scheduling policies. For this purpose, we train the model using samples with mixed queue scheduling policies across nodes in the GEANT and NSFNET topologies. Then, we evaluate the model on samples of the GBN topology (unseen during training). In this experiment, each router port implements three different queues with a randomly selected scheduling policy for the queues: (i) First In, First Out (FIFO), (ii) Weighted Fair Queueing (WFQ), (iii) Deficit Round Robin (DRR), and (iv) Strict Priority (SP). For WFQ and DRR, the set of weights is also randomly selected. Furthermore, each flow has been assigned a Quality-of-Service class that maps it to a specific queue depending on the flow priority. To provide a fair benchmark with QT, in this experiment we use only Poisson traffic.

	Delay			Jitter		
	Low	Medium	High	Low	Medium	High
QT	13.0%	17.3%	25.1%	49.0%	53.2%	59.6%
RouteNet-F	**0.80%**	**2.60%**	**7.31%**	**3.95%**	**5.77%**	**14.8%**

TABLE 6.4: Delay and jitter evaluation - Mean Absolute Percentage Error of QT and RouteNet-F in the presence of Scheduling Policies for low, medium, and high traffic intensity.

Table 6.4 shows the Mean Average Percentage Error (MAPE) of the delay and jitter for three different traffic intensities: from low-loaded to highly-congested scenarios. According to [78] the average packet loss on the Internet is around 2%-3%. Based on this, in the highly-congested scenarios, the mean packet loss rate is around 3% which we believe represents a wide range of realistic network scenarios. We can see that RouteNet-F outperforms the QT benchmark in both metrics (delay and jitter), obtaining a MAPE of 3.57% for delay and 8.17% for jitter after averaging the results over the three traffic intensities.

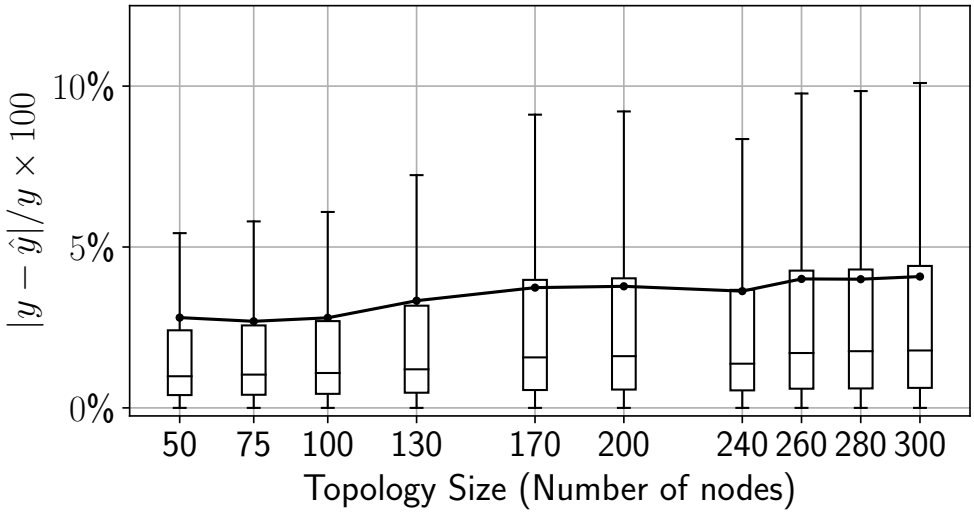


FIGURE 6.4: Scaling with mixed traffic models and scheduling policies - Mean Absolute Relative Error of delay predictions vs. topology size, including different traffic models and queue scheduling configurations. The model was trained on a dataset with 10,000 samples from networks of 5 to 10 nodes.

Similarly, Table 6.5 presents the results for packet loss. Again, RouteNet-F outperforms the QT benchmark showing an MAE of 0.7% and an average R^2 close to 0.99.

	Low		Medium		High	
	MAE	R^2	MAE	R^2	MAE	R^2
QT	4.55%	-0.05	9.22%	0.00	10.6%	0.29
RouteNet-F	**0.2%**	**0.97**	**0.10%**	**0.99**	**0.40%**	**0.99**

TABLE 6.5: Packet loss evaluation - Mean Absolute Error and Coefficient of Determination (R^2) of QT and RouteNet-F in the presence of Scheduling Policies for low, medium, and high traffic intensity.

6.2.2 Generalization and Scalability

Generalization to larger networks

The previous experiments show that RouteNet-F achieves remarkable accuracy when tested in scenarios with different traffic models (Section 6.2.1) and different scheduling policies (Section 6.2.1) with topologies never seen in training. As previously discussed in Section 3.4, data-driven network models must generalize to larger networks than those seen during training to become a practical solution.

In this section, we evaluate RouteNet-F in a wide variety of networks significantly larger than the ones seen during the training phase. We generate a training set with 10,000 samples from networks of only 5 to 10 nodes. Following the process described in Sections 6.1.1 and 6.2.1, for each flow, we randomly assign a traffic model, and for each router port, we assign an arbitrary queue scheduling configuration. We evaluate the accuracy of RouteNet-F in topologies from 50 to 300 nodes, configuring the traffic models and the scheduling policies accordingly to the descriptions of Sections 6.1.1 and 6.2.1. In contrast to previous experiments, all these networks have been synthetically generated using the Power-Law Out-Degree algorithm described in [54], where the ranges of α and β parameters have been extrapolated from real-world topologies of the Internet Topology Zoo repository [79]. Link capacities and generated traffic volumes are scaled accordingly.

Figure 6.4 shows the MAPE of the delay predictions made by RouteNet-F. We can observe that the proposed model obtains a worst-case error of $\approx$8% for samples of networks with 300 nodes. This shows how RouteNet-F is capable of generalizing to networks 30x larger than those seen during training, even when introducing various traffic models and queue scheduling policies along the network. This is due to the capability of this model to effectively learn the underlying relationships between flows, links, and queues in the scenarios seen during training, and the posterior ability to exploit this learned knowledge in new scenarios not seen before.

Note that in this and the previous section, we have not tested any other baseline (e.g. RNN, QT) since they already fail in other *relevant* scenarios.

Despite generalization is an open challenge in the field of Deep Learning (as previously discussed in Section 3.4) by using a custom GNN-based architecture and domain expert knowledge, RouteNet-F shows strong capabilities to generalize to considerably larger networks than the ones seen during training.

Few-shot Learning

The performance of DL models is often determined by the quantity and quality of the training data used. Here, it is important to consider that the process of collecting and labeling large amounts of data can be very costly and,

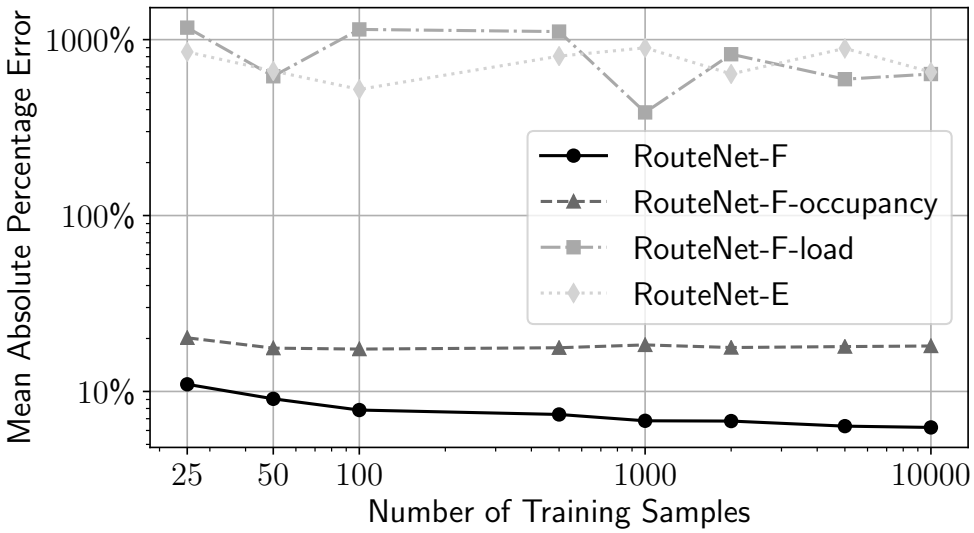


FIGURE 6.5: Delay evaluation - Mean Absolute Percentage Error vs Number of Training Samples for the different versions of RouteNet-F and RouteNet-E.

sometimes, infeasible. In the field of computer networks, such data collection implies generating and storing data from costly network infrastructures. This can be generally a very expensive and time-consuming process. An alternative is the use of packet-level simulators, which are also very expensive in terms of computational cost. In this section, we evaluate the accuracy of RouteNet-F when trained with a very limited number of samples (i.e., few-shot learning). This may be very helpful to dramatically reduce the cost of generating the datasets and reduce the carbon footprint of training the model.

To do so, we train the model by randomly selecting 25, 50, 100, 1,000, 2,000, 5,000, and 10,000 samples from the previous training dataset (Section 6.2.2). Note that the topologies used for training range from 5 to 10 nodes, while the evaluation is done over samples of networks from 50 to 300 nodes. Figure 6.5 shows the MAPE with respect to the number of training samples used (see only results of *RouteNet-F*). Interestingly, when trained with only 25 samples, the model shows an average error of 11%. As the number of samples increases, RouteNet-F obtains slightly better accuracy, achieving an error of 6.24% when trained with 10,000 samples.

Ablation test

We aim to analyze which features of RouteNet-F have more impact on its accuracy. For this purpose, we perform an ablation test, by considering

four models where we remove different features. The first one (labeled as *RouteNet-F*) is the complete model used in the previous experiments, as it is previously described in Section 6.1. Second, in *RouteNet-F-occupancy* we remove the link load as an input feature (see Sec. 6.1.3), and replace it directly with the link capacity value as an initial feature of links $\boldsymbol{x}_l$. Third, *RouteNet-F-load* predicts the flow-level delay using directly the hidden state of flows ($\boldsymbol{h}_f$), instead of predicting the effective queue occupancy of flows at a specific link ($\boldsymbol{h}_{f,l}$) and then adding up all the estimated link-level delays (see Sec. 6.1.3). Finally, we use *RouteNet-E* [106] as a reference, which is the previous version of RouteNet-F, without any of the aforementioned features. Figure 6.5 shows the results obtained in this experiment. We can see that predicting the delay as the sum of link-level delays along flows (*RouteNet-F-occupancy*) seems to have the largest impact on the accuracy of the model, achieving an error of 17.63%. In addition, the results suggest that using the link load as input instead of the capacity (*RouteNet-F-load*) does not have a significant impact on the model's accuracy. However, when we combine this feature with the one of *RouteNet-F-occupancy* we see a slight improvement. Particularly, in *RouteNet-F*, which implements the two features, we can see that the prediction error decreases to 6.24%.

In this experiment, it is observed that the previous version of the model (*RouteNet-E*) exhibits poor accuracy. This is likely because RouteNet-E was not designed to support scalability levels of 30x larger networks. Additionally, RouteNet-E assumes uniform delays for all flows traversing the same queue and link. In contrast, RouteNet-F takes into consideration that flows may experience different delays when traversing the same queue and link, depending on their traffic model (e.g., traffic burstiness).

Scalability: Training and Inference time

Models that are capable of producing fast estimations are particularly interesting for network control and management applications, as they can be deployed in real-time scenarios. In this section, we evaluate the inference time of both RouteNet-F and the QT baseline. To this end, we measure the inference times of the experiments of the general evaluation (Sec. 6.2.2). Table 6.6 shows that both models operate in the order of milliseconds for topologies lower than 110 nodes. Particularly, QT performs better for smaller topologies

	Topology Size					
	10	30	50	70	90	110
RouteNet-F	48.03 ms	76.25 ms	110.5 ms	285.6 ms	455.3 ms	613.3 ms
QT	28.01 ms	49.19 ms	131.68 ms	326.5 ms	662.15 ms	962.5ms

TABLE 6.6: Inference time vs. topology size for RouteNet-F and the QT baseline.

(<50 nodes). However, as the size of the topology increases, RouteNet-F is faster.

Finally, note that one difference between analytical models (e.g., QT) and DL-based models (e.g., RouteNet-F) is that the last need a training process that may be taken into consideration when comparing these times. These training times depend greatly on various factors like the size of the training dataset, the hyperparameters, the used hardware, etc. In our particular case, the model that obtained the higher accuracy was trained during 20 epochs of 2,500 samples each, lasting for about 2h30min.

In this experiment, we used one CPU [AMD Ryzen 9 3950X @ 3.5 GHz]. However, an advantage of DL-based models is that they can be easily parallelizable using hardware-specific solutions (e.g., GPU), thus reducing considerably their execution times in production.

6.2.3 Benchmarking of RouteNet-F

Testbed

In previous sections, we examined how RouteNet-F can be applied in different network scenarios. This section evaluates its performance in a real-world scenario using real-world hardware and synthetic traffic. For this, we set up a physical network testbed, as shown in Figure 6.6. This testbed includes (i) 8 Huawei NetEngine 8000 M1A routers, (ii) 2 Huawei S5732-H48UM 2CC 5G Bundle switches, and (iii) 4 servers. Two of the servers are used to generate traffic using the TRex traffic generator, and the other two are used for capturing and analyzing traffic with the PF_RING software.

To train and test the model, we generated 1,000 samples with realistic network topologies (with a maximum of 8 nodes) and different routing,

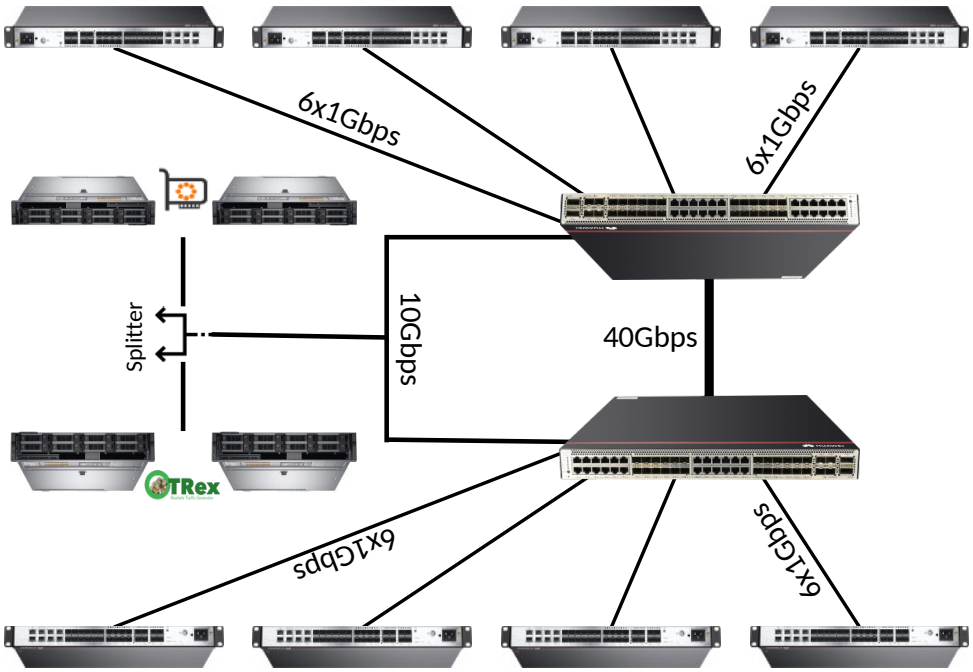


FIGURE 6.6: Schematic representation of the network testbed.

queueing, and traffic configurations. Out of these 1,000 samples, we randomly selected 800 samples for training the model and used the remaining 200 samples for testing. Note that, the algorithm 2 described in Section 6.1.1 has been slightly modified to add the state of the queues of the switches found in the topology. This only affects the initialization process, and not the message-passing architecture itself.

Table 6.7 shows the results of RouteNet-F. As can be seen, RouteNet-F obtains a remarkable performance (11% MAPE) which is in line with the previous results when compared with simulated data.

Real Traffic

In our previous experiments, we tested RouteNet-F using synthetic-simulated traffic. Now, we want to see how well this model performs when applied to actual traffic data.

For this, we used real-world traffic data from the SNDlib library [82] and combined it with packet inter-arrival times from a recent snapshot of the MAWI repository (Sample point 2022/09) [83]. We then scale the inter-arrival times to match the values in the traffic data. Additionally, we used a distribution of source-destination flows from a real internet service provider [84] to

map flows to different ToS classes. Our dataset includes 4 real-world network topologies, including one previously used (GEANT) and three new topologies that the model has never seen before (ABILENE, NOBEL-GBN, and GERMANY50). For this experiment, we leveraged the knowledge obtained from previous versions and used a previous checkpoint, fine-tuning it using 200 samples of the GEANT topology.

Table 6.7 shows the results of RouteNet-F. We can see that RouteNet-F achieves a remarkable performance (5.67% MAPE) when tested using real-world traffic data. Again, these results are close to the simulator and testbed ones.

	MAPE	MSE	MAE	R^2
Testbed	11.0%	6.12×10^{-5}	0.0007	0.869
Real Traffic	5.67%	1.66×10^{-5}	0.0003	0.877

TABLE 6.7: Delay prediction using RouteNet-F for the testbed and the real traffic traces experiments.

State-of-the-Art

This section aims to compare the accuracy of RouteNet-F against MimicNet [53]. MimicNet is a DL-based model which combines discrete event simulators with Deep Neural Networks (DNN). MimicNet takes advantage of the accuracy of discrete packet-event simulators to generate data of a small network which then is used to train an RNN-based estimator known as a "mimic". Finally, MimicNet composes several of those mimics to perform predictions of much bigger larger networks. This makes MimicNet an alternative to a discrete packet-event simulator as it reduces the cost of simulation for large networks, providing an accurate estimation of the per-packet level distributions. However, as stated in the MimicNet paper, the main limitation of this strategy is that it only works for FatTree topologies.

Following the parameters and the specifications described in [107] we generate a dataset containing three different topologies (FatTree16, FatTree64, and FatTree128) and compute the average RTT.

Table 6.8 shows the results for both RouteNet-F and MimicNet. For RouteNet-F we show the same metrics as before and the Normalized Wasserstein Distance (W_1). Contrary to RouteNet, MimicNet does not directly compare the average RTT of the packets aggregated per path. Instead, it computes the distance of both distributions (the predicted and the real one).

Note that in this scenario, RouteNet-F achieves an outstanding accuracy not only when compared with MimicNet, but also when compared with a state-of-the-art QT model (Table 6.1). This is mainly because, in previous experiments, we explored a wide variety of scenarios containing from low to high (3%) packet loss ratios, while in this particular scenario, the packet loss rate is about 0.3%.

Finally, as shown in section 6.2.2, RouteNet-F is capable of predicting the performance metrics in the order of milliseconds. In contrast, MimicNet ranges from minutes for the smaller topologies to hours for the larger ones. This difference is mainly because, while RouteNet-F focuses on predicting the different performance metrics aggregated by flows, MimicNet predicts those metrics at a packet level.

	MimicNet	RouteNet-F				
	W_1	MAPE	MSE	MAE	R^2	W_1
FatTree16	0.0080	**0.37%**	**1.31x10^{-10}**	**5.33x10^{-6}**	**0.999**	**0.0018**
FatTree64	0.0122	**0.44%**	**1.70x10^{-10}**	**7.15x10^{-6}**	**0.999**	**0.0026**
FatTree128	0.0172	**0.67%**	**3.40x10^{-10}**	**1.20x10^{-5}**	**0.998**	**0.0060**

TABLE 6.8: Average RTT prediction using MimicNet and RouteNet-F for different FatTree topologies. The error is computed w.r.t. simulation results.

6.3 Discussion and Concluding Remarks

In this chapter, we have presented RouteNet-Fermi, a custom GNN model designed for network performance analysis. This model supports a wide range of configuration parameters related to routing, queue scheduling, and traffic models while being able to accurately model networks 30 times larger than the ones seen during training. In our evaluation, we have shown that the proposed model outperforms a state-of-the-art queuing theory model, especially in scenarios with complex and realistic traffic models. At the same

time, RouteNet-Fermi achieves comparable accuracy with respect to computationally expensive packet-level simulators (MAPE $\approx$ 6.24%) while exhibiting considerably lower inference times (on the order of milliseconds in networks of 100 nodes). Finally, we validated RouteNet-F in a wide variety of real-world scenarios including a testbed and real-world traffic traces.

Chapter 7

ST-RouteNet: Adding the Temporal Dimension

The models presented in previous chapters have demonstrated remarkable accuracy in modeling network performance metrics, even extending their capabilities to generalize across diverse network topologies, configurations, and traffic loads. However, a noteworthy limitation of these models is their treatment of network traffic as "Traffic Matrices". This simplification implies that they only consider aggregated bandwidth over a specific time interval between a source and destination pair. While this assumption suffices for certain use cases, it imposes constraints because the most pertinent description of network traffic is in the form of "flows."

In network operations, flows represent a vital concept. A flow is essentially a set of packets that share common characteristics. A commonly used aggregation approach in practical networks involves 5-tuple flows, comprising packets with the same protocol (TCP or UDP), source and destination IP addresses, as well as source and destination ports. Flows play an important role in networking because applications inherently operate with them, and network optimization usually revolves around providing varying levels of quality of service to different flows. Past research efforts, such as [108], have suggested working at the flow level for optimizing user quality of experience (QoE) in video streaming, while other works like [17] and [109] focus on flow routing optimization in carrier-grade networks. Additionally, solutions like [110] aim to enhance flow completion in data centers. It's worth noting that several spatiotemporal Graph Neural Networks (GNNs) have been proposed in the past, such as [111] and [112]. However, these models do not specifically target the unique challenges presented by the networking domain.

Operating at the flow level is challenging, flows are dynamic and have a finite duration that ranges from a few packets (typically ms) to tens of thousands of packets (e.g., backup session) and spans days. In networks, flows are concurrent and at a given instant of time, millions of flows can be active at the same time. As a consequence, modeling flows requires understanding the time dimension and as a consequence, modeling how the state of the network changes over time and as flows are created and destroyed.

In this Chapter, we present a new GNN-based model that is able to understand and model flows. This network model works in the time domain and supports the creation and destruction of flows, as well as flows that dynamically change their characteristics. With this, it is able to provide per-flow metrics (delay and jitter) based on the input flow dynamics as well as network configuration (routing, topology, etc.).

7.1 Network Scenario

We define a flow as an aggregation of packets that have some -loose- common characteristics. Our model supports arbitrary aggregation of packets into flows, for instance, 5-tuple flows. In practice, the main limitation of the aggregation level used with our model depends on the capabilities of the monitoring infrastructure deployed at the network. Operating with fine-grained flows can be computationally expensive [113].

Each flow is defined as a starting time and duration, a source and destination node, and a traffic model. Optionally, it can also include additional information such as ports, protocol, etc. The traffic model describes the inter-arrival time distribution of the packets as well as their packet size. Our model supports arbitrary traffic models, including non-Poisson arrivals with long tails and auto-correlation (Sec. 3.1.1).

With respect to the temporal dimension consider figure 7.1. We define a time-bin as a stationary period for the entire network, this includes all the flows as well as network configuration and topology. Changes in the network state are considered in a new time-bin. Examples of changes in the network state are the creation or destruction of a flow, a flow changing its traffic model or traffic model parameters (increase rate), link failures, or a change in the routing policy.

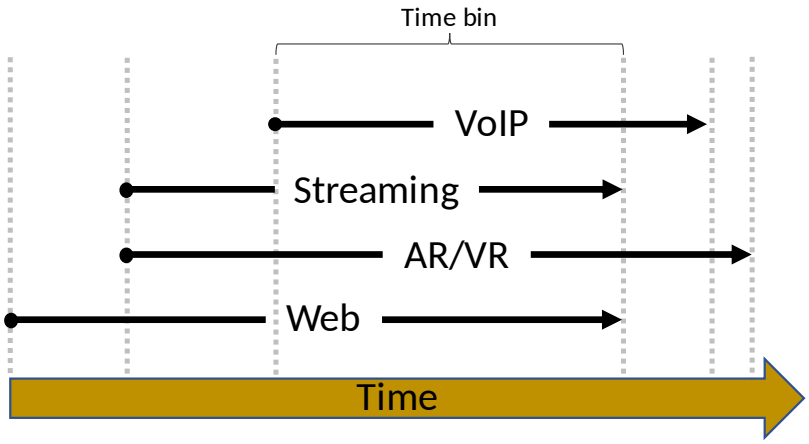


FIGURE 7.1: Representation of the temporal dimension. Whenever there is a change in the network state (e.g., the creation of a flow), a new time-bin is established.

Informally, the network model produces an inference per time-bin, computing per-flow, and per-bin delay and jitter. To produce an estimate, it uses the current state of the network for that time-bin (traffic models, topology, routing, etc.) as well as the previous time-bin state of the network. Also, the network model is suited to work with dynamically changing topologies. This not only means that the model is capable of handling different topologies but also capable of handling topology changes like link failures or network upgrades.

7.2 Flow-Aware Network Model

This section describes how ST-RouteNet works, a novel GNN-based solution tailored to accurately model the behavior of real network infrastructures at a flow-granularity level. Particularly, ST-RouteNet describes a new network modeling architecture where the different key elements for network modeling (e.g., forwarding devices, links, flows) exchange messages of their state to the ones they are related with (e.g., via routing).

Specifically, ST-RouteNet (Fig. 7.2) takes as input (i) a given network configuration (topology, link capacities, and routing) (ii) the per-flow level parameters (iii) the previous bin network state, and produces as output (i) the current network state that will be later used as input for the next time-bin (i) the performance per-flow metrics according to the network state (per-flow mean delay and jitter).

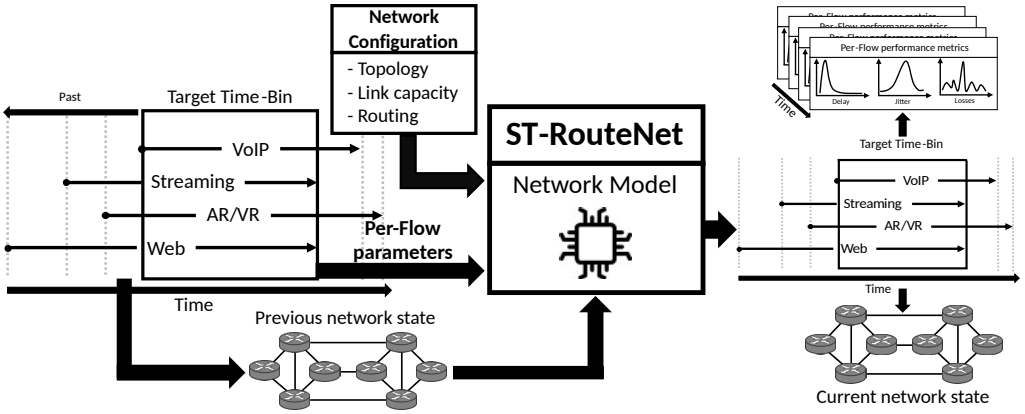


FIGURE 7.2: ST-RouteNet diagram. ST-RouteNet takes as inputs the network configuration (e.g., topology, link capacities, routing), the target per-flow parameters, and the previous network state. ST-RouteNet outputs the per-bin and per-flow performance metrics (delay and jitter) and the current network state.

One of the central ideas of ST-RouteNet is that it encodes the state of the network resulting from the previous time bin, and this state is used to accurately infer the per-flow performance statistics of the current bin. Since a computer network can be understood as a queuing system, we expect this state to be the state of the link/queues.

7.2.1 Model description

Following the notation used in previous Chapters, a computer network can be represented by a set of links $L = \{l_i : i \in (1, ..., n_l)\}$, a set of source-destination flows $F = \{f_i : i \in (1, ..., n_f)\}$, and the routing configuration defined as a set of source-destination path that flows follow. Hence, we define flows as a sequence of the links they traverse $f_i = \{l_{F_l(f_i,0)}, l_{F_l(f_i,1)}, ..., l_{F_l(f_i,|f_i|)})\}$, where the function $F_l(f_i, j)$ return the index of the j-th link along the path of flow f_i.

Using this notation, ST-RouteNet considers three main inputs: (i) the physical links L defined by the network topology, (ii) the active flows F in the network in a specific time-bin, including the source-destination path and the traffic model, and (iii) the network state of the previous time-bin.

The main assumption behind ST-RouteNet is that information at the flow level (e.g., delay) and the link level (e.g., link delay, loss rate, link utilization) can be encoded as vectors of numbers of a given size. Based on this,

the main intuition behind this architecture is:

1. The state of flows is affected by the state of the links they traverse.
2. The state of links depends on the states of the flows that go through them.
3. The initial state of the network depends on the previous time-bin network state.

Formally, we define the state of a flow as a hidden vector h_f of a given size. In a similar way, the state of a link is also defined by a hidden vector h_l. Knowing this, the principles described above can be formally formulated as follows:

$$h_{f_k} = g_f(h_{l_{f_k(0)}}, h_{l_{k(1)}}, ..., h_{l_{f_k(|f_k|)}}) \tag{7.1}$$

$$h_{l_j} = g_l(h_{f_1}, ..., h_{f_m}), \quad l_j \in f_k, k = 1, ..., i \tag{7.2}$$

Where g_f and g_l are some unknown functions that the model needs to learn. Note that a direct approximation of functions g_f and g_l is not possible due to the circular dependencies found between the two components (L and F).

We expect that h_f and h_l encode important information like the per-flow throughput/losses or the per-link utilization. However, we do not make any assumptions and let the model learn directly from the data that is fed into the model.

The architecture of ST-RouteNet (see Algorithm 4) is specifically designed to deal with those circular dependencies via an exchange of messages between links and flows. ST-RouteNet receives as input the initial link and flow features (x_l and x_f respectively), and the network state computed in the previous time-bin ($h_l^{T^{(b-1)}}$). In our particular case, x_l is defined as (i) the link capacity and (ii) the buffer size. On the other side, the initial flow features x_f are the features that describe the behavior of the flow.

First, in Algorithm 4, the hidden states h_l and h_f are initialized (line 1 - 2) using the initial features described before. In the h_l case, note that the feature vector (x_l) is concatenated with the previous network state ($h_l^{T^{(b-1)}}$). If we are in the first time-bin, the $h_l^{T^{(b-1)}}$ vector is encoded as an array of 0.

Algorithm 4 Internal architecture of ST-RouteNet

Input: F, L, x_f, x_l, $h_l^{T^{(b-1)}}$
Output: h_l^T, h_f^T, $\hat{y}_f$

1: **for each** $l \in L$ **do** $h_l^0 \leftarrow [x_l, 0...0]$
2: **for each** $f \in F$ **do** $h_f^0 \leftarrow [h_l^{T^{(b-1)}}, x_f, 0...0]$
3: **for** t = 0 to T-1 **do** ▷ Message Passing Phase
4: **for each** $f \in F$ **do** ▷ Message Passing on Flows
5: **for each** $l \in f$ **do**
6: $h_f^t \leftarrow FRNN(h_f^t, h_l^t)$ ▷ Flow: Aggr. and Update
7: $\widetilde{m}_{f,l}^{t+1} \leftarrow h_f^t$ ▷ Flow: Message Generation
8: $h_f^{t+1} \leftarrow h_f^t$
9: **for each** $l \in L$ **do** ▷ Message Passing on Links
10: $M_l^{t+1} \leftarrow \sum_{f \in L_f(l)} \widetilde{m}_{f,l}^{t+1}$ ▷ Link: Aggregation
11: $h_l^{t+1} \leftarrow U_l(h_l^t, M_l^{t+1})$ ▷ Link: Update
12: $\widetilde{m}_l^{t+1} \leftarrow h_l^{t+1}$ ▷ Link: Message Generation
13: $\hat{y}_f \leftarrow R_f(h_f^T)$ ▷ Readout phase

Once the initial states are initialized a message-passing phase starts. This message passing phase is executed iteratively during T times (line 3). This process is done to ensure that the state of all the elements has converged. Each message passing iteration can be divided into two stages where the different elements receive information about the elements they are related with, then, this information is aggregated and used to update their state ($FRNN$ and U_l) and finally, create the new message that will be sent. Particularly, the flow aggregation, update, and message creation are described in lines 4 - 8, and the link ones in lines 9 - 12.

Finally, the readout phase is executed (line 13). In it, the function R_f is responsible for producing the per-flow level metrics.

7.3 Experimental evaluation

To train, validate, and test ST-RouteNet we use as ground truth a packet-level network simulator (OMNeT++ v5.5.1 [8]), where network samples are labeled with performance metrics, including the flows mean delay and jitter over time. In order to generate the dataset, for each sample, we randomly select a combination of input features (flow duration, traffic model, topology, and routing) according to the descriptions below.

7.3.1 Flow Configuration

In our experiments, traffic is generated using five different models that range from a Poisson generation process to more realistic (non-Poisson) traffic generation [16]. The different implementation details of these models are defined in the previous Section 3.1.1).

The creation of each flow is based on a Poisson process. The flow duration is based on the distributions described in [114]. All flows have a maximum duration of 1000s which represents 99.5% of all the flows measured in [114]. Note that the flow-creation time and the average traffic per flow have been manually configured to produce low to high congestion levels (to a maximum $\approx$ 3% of packet loss) similarly to what has been experimentally measured [115].

7.3.2 Topologies

To train and test ST-RouteNet, we used three different real-world topologies that have already been used in previous works [77, 70]. Specifically, to train we used NSFNET [56], and GEANT [57] topologies. Then, we validate the accuracy of the model in GBN [58].

7.3.3 Baselines

To assess the accuracy of ST-RouteNet, we subjected it to a benchmark against two distinct models. The first benchmark is a stateless Queueing Theory model (s-QT), which is based on the methodology presented in [77]. This s-QT model characterizes the system as a series of finite $M/M/1/b$ queues linked together. It's important to note that the s-QT model doesn't take into account the state of the network from the previous time-bin.

The second benchmark employs a Gated Recurrent Unit (GRU), which is constructed using a Recurrent Neural Network (RNN) [116]. In this scenario, the different flows are modeled in a similar manner to ST-RouteNet, where flows are represented as sequences of links determined by the routing configuration. The primary distinction between the GRU baseline and ST-RouteNet lies in their approach to modeling. While GRU treats each path

as a sequence of links without consistently employing the same states, ST-RouteNet iterates over all the elements of the network's states, updating their states during each iteration.

7.3.4 Training and evaluation

We implemented ST-RouteNet as well as the baselines using Tensorflow. In total, ST-RouteNet was trained using 120,000 samples (NSFNET and GEANT) for training and evaluated using 60,000 (GBN) more samples. More information about the topologies used can be found in section 7.3.2.

In our experiments, we use a hidden vector size of 32 for both h_l and h_f. In our particular case, x_l is defined as (i) the link capacity and (ii) the buffer size. On the other side, the initial flow features x_f are defined by the parameters of each distribution described in section 3.1.1 (e.g., on and off times, α, λ), the total number of packets, and the generated traffic. Note that, since we are working with synthetic data, the flow features are initialized with the parameters used to create those distributions. However, in a real-world scenario, these features could be learned directly from the data using a specialized module. The total number of iterations (T) is 8.

The functions found in Algorithm 4 are implemented as follows: $FRNN$ (line 6) and U_l (line 11) as Gated Recurrent Units (GRU) [74], and the function R_f (line 13) as a 2-layer fully-connected neural network with ReLU activation functions. It is important to note that the architecture of ST-RouteNet (Algorithm 4) has been specifically designed to be differentiable in order to train the model end to end. Hence, all the different functions that shape its internal architecture are jointly optimized during training based on ST-RouteNet's inputs (network samples) and outputs (per-flow performance metrics).

During the training, we selected as the loss function the Mean Squared Error (MSE). This loss function is minimized using an Adam optimizer with an initial learning rate of 10^{-3}.

Table 7.1 shows a summary of the delay and jitter experiments for the GBN topology (never seen during training). We provide 4 metrics: the Mean Absolute Percentage Error (MAPE), the Mean Squared Error (MSE), the Mean Absolute Error (MAE), and the Coefficient of Determination (R^2).

	Delay				Jitter			
	MAPE	MSE	MAE	R^2	MAPE	MSE	MAE	R^2
s-QT	35%	0.72	0.43	0.56	69%	0.29	0.29	0.02
GRU	50%	1.15	0.54	0.29	137%	0.16	0.22	0.45
ST-RouteNet	**3.5%**	**0.006**	**0.035**	**0.995**	**11.2%**	**0.004**	**0.032**	**0.983**

TABLE 7.1: Performance comparison for NSFNET and GEANT networks seen during training.

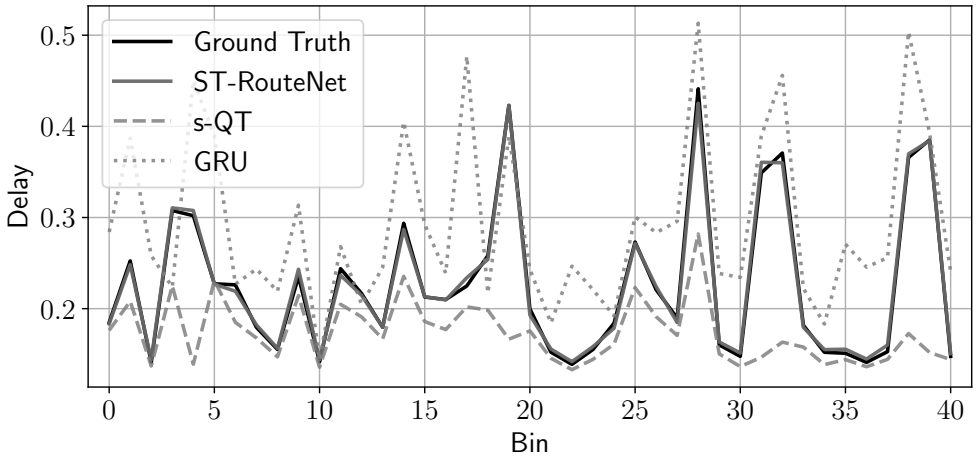


FIGURE 7.3: Delay prediction of one randomly selected flow of the GBN topology.

As can be seen in Table 7.1, ST-RouteNet clearly outperforms all the benchmarks. This high accuracy shows how the described model is able to generalize to unseen topologies, routings, and per-flow characteristics.

Figure 7.3 shows an example of the time series for a randomly selected flow. As can be seen, the network model is able to react to the dynamic behavior of the flows, providing accurate estimates of the delay experienced by each flow. In contrast, both baselines seem to provide pretty good estimates when the delay of a time-bin is close to 0.1. However, in peak delays where the network is more saturated, they clearly fail with s-QT underestimating and GRU overestimating the delay. Note that in this experiment only flow-related features are changed over time (e.g., packet rate, distribution, flow parameters). However, the model also accepts changes in the topology (e.g., link capacities, link failures, routing).

In order to analyze how the residuals are distributed we plot them in Figure 7.4. The figure shows the Probability Density Function (PDF) of the Relative Error of the three models. As the figure shows, ST-RouteNet produces estimates with the error centered around 0. It looks like, while

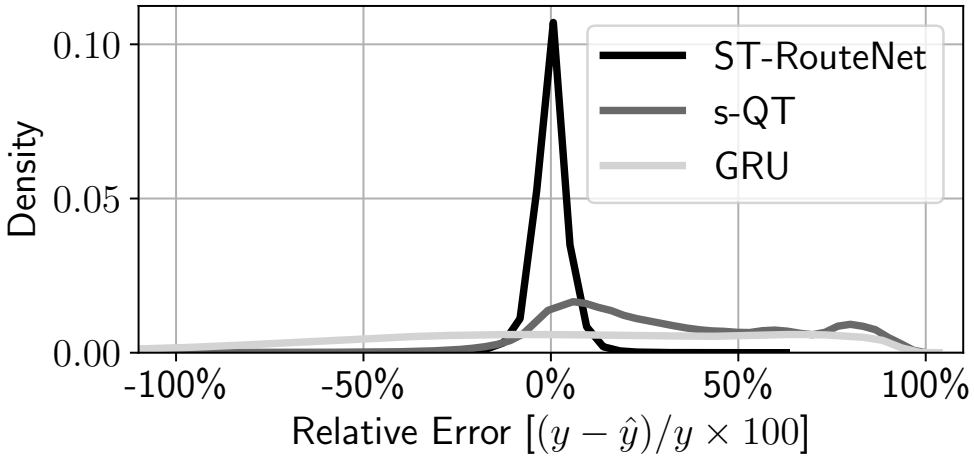


FIGURE 7.4: PDF of the Relative Error reported for the delay prediction.

GRU shows poor performance, the s-QT model undershoots the true values. This is in line with previous experimental results [77], the key insight is that queuing theory assumes Markovian arrivals while in practice, flows have highly autocorrelated traffic models which result in bursts of packets that increase the queue weighting time.

7.4 Discussion and Concluding Remarks

In this Chapter, we introduced ST-RouteNet, a modification of RouteNet's family architectures designed to model network behavior with a focus on flows and the temporal dimension. We addressed a crucial limitation in existing models that primarily consider network traffic as "Traffic Matrices," which aggregate bandwidth over time between source-destination pairs. While this approach has its merits, it can be constraining since the more meaningful representation of network traffic often involves "flows."

We evaluated ST-RouteNet using a comprehensive set of experiments with synthetic data generated using a packet-level network simulator. ST-RouteNet significantly outperformed benchmark models, such as stateless Queueing Theory (s-QT) and a Gated Recurrent Unit (GRU), in predicting per-flow performance metrics. Its high accuracy was particularly evident when applied to unseen network topologies and conditions.

This chapter showcases how ST-RouteNet's unique architecture and

focus on flows in the temporal domain provide an effective solution for network performance modeling and optimization. With its ability to understand and adapt to the dynamic nature of real-world networks, ST-RouteNet opens new possibilities for enhancing network performance in scenarios where flows are dynamic and changing.

Chapter 8

Network Performance Digital Twins

The concept of a Digital Twin has gained significant traction across various industries, from aeronautics [117] to logistics [118]. In the fast-paced world of modern industry, Digital Twins have emerged as powerful tools with the potential to revolutionize many sectors. By creating virtual replicas of physical systems, Digital Twins provide a unique opportunity to enhance understanding, optimize performance, and streamline decision-making processes.

In the realm of networking, the Network Digital Twin (NDT) has gained recognition as a valuable concept. An NDT is essentially a virtual replica of a physical network, preserving one or more essential characteristics of the original network. Different types of NDTs exist, with the Network Performance Digital Twin (NPDT) being one of the most common [119, 120]. The NPDT, is specifically designed to predict performance metrics, such as the average per-path delay, for a given input network, which includes both the network's topology and routing configuration.

The NPDT, like other forms of NDTs, plays a critical role in understanding, optimizing, and enhancing the performance of real-world networks. It serves as a digital counterpart that helps network operators make informed decisions, monitor network behavior, and proactively address potential issues [119, 120]. As technology and networks continue to evolve, NPDTs are poised to play an increasingly significant role in network management and optimization.

Hence, an NPDT can be extremely useful in network management scenarios. By leveraging the NPDT, industry professionals can safely analyze

the impact of potential scenarios and configurations without any disruption to the real network. It enables them to address critical questions such as determining the maximum number of link failures before Service Level Agreements (SLAs) are compromised or identifying the optimal network hardware upgrade within a specified budget. This capability is commonly referred to as what-if analysis. Another interesting use case is network optimization. By coupling the NPDT with an optimization algorithm [121], network administrators can evaluate the performance of various candidate network configurations. Hence, the NPDT is a powerful tool in the decision-making process, making it easier to explore different network optimization strategies to achieve the expected performance outcomes.

In other words, it is possible to view an NPDT as an advanced network model that supports a wide range of network configurations and can quickly produce performance estimations. Traditionally, network administrators have relied on network modeling tools that present some limitations when it comes to constructing an NPDT. The most commonly used tools include analytical queuing-theoretic models and packet-level simulators. As previously discussed in Chapter 3, Queuing theory (QT) models are fast and scalable, making them suitable for large-scale simulations. However, their accuracy diminishes when applied to non-Poisson traffic models. On the other hand, packet-level simulators offer excellent accuracy and can support any traffic model, by means of supplying a packet trace. Nonetheless, their simulation time is often too extensive for near real-time operations, making them impractical for certain scenarios.

These limitations require the development of novel approaches and techniques that strike a balance between speed, accuracy, and flexibility in order to construct a reliable and efficient NPDT. Recent advances in Machine Learning (ML) offer promising solutions in this regard. ML models have the capability to capture and implement arbitrary nonlinear relationships when appropriately trained. In the context of a Network Performance Digital Twin, this involves training a model with a wide range of network data, e.g. pairs of `(network configuration, performance values)`. Once the model is trained, it can accurately predict the performance values of the input networks. The prediction process is usually fast, in the order of ms. Hence, ML techniques offer a compelling balance between accuracy and speed, paving the way for an NPDT that can support the use cases we mentioned previously.

In this Chapter, we discuss the viability of the RouteNet family models (Chapters 4, 5, and 6) as a possible architecture of a Network Performance Digital Twin (NPDT), and provide a comprehensive exploration of the requirements of this kind of Digital Twins.

8.1 Network Performance Digital Twins

The concept of a Network Digital Twin (NDT) is fundamentally rooted in the creation of a virtual counterpart for a physical network. However, it's important to note that NDTs are not one-size-fits-all; they can take on various forms and implementations to suit the specific needs of different use cases. This adaptability allows NDTs to serve a wide range of scenarios and requirements.

NDTs can come in different types, each with its own scope and focus. For instance, a Network Performance Digital Twin (NPDT) primarily deals with predicting and optimizing network performance metrics. Other types of NDTs may focus on different aspects of network modeling and management, depending on the objectives of the use case.

Furthermore, the complexity of an NDT's implementation can vary significantly based on the requirements of the specific use case. Some NDTs may be relatively simple, providing a basic virtual representation of the network, while others can be highly sophisticated, incorporating intricate details and modeling capabilities. The degree of complexity is tailored to match the complexity of the physical network and the level of detail required for accurate modeling and analysis.

Among these use cases, in this Chapter, we focus on a Network Performance Digital Twin (NPDT). In a nutshell, an NPDT is an advanced network model. Its inputs are a wide range of network configuration parameters, and the outputs are key performance metrics of the input network. More specifically, the inputs are the network topology, traffic matrix, the nature of the input traffic (either a traffic model of each flow, or a packet trace), the routing configuration, and the scheduling configuration (queues per port and scheduling algorithm). Regarding the outputs, the most interesting ones are common performance metrics used by network engineers, namely average delay, jitter, and packet loss, both for individual links and end-to-end paths.

A key property of an NPDT is the wide range of input and output variables, that allows it to replicate different network configurations, as well as calculate different metrics.

8.1.1 Architecture

In a practical operational setting, the NPDT is intricately linked with the control and management plane of an actual physical network (as depicted in Figure 8.1). This interconnection enables the NPDT to be effectively configured to function as a virtual replica of the physical network. Such integration and configuration empower the NPDT to be used in a multitude of critical use cases, ranging from network optimization and efficient network provisioning to expedited troubleshooting and diagnostics, among various others. This close connection between the NPDT and the real network's control and management plane amplifies its utility as a powerful tool for enhancing network performance, making informed decisions, and ensuring seamless network operations.

The control plane maintains communication with various network elements through a set of interfaces, which may conform to established standards or operate in a non-standardized manner:

Administrator Interface

Ideally, a standardized interface is employed to articulate network service requisites and optimization objectives. This interface also serves as a conduit for connections to external network management applications, facilitating seamless integration with other management tools and systems.

Network Digital Twin Interface

This interface is specifically designed to enable the submission of prospective network configurations to the Network Performance Digital Twin and retrieve the corresponding performance parameters associated with these configurations.

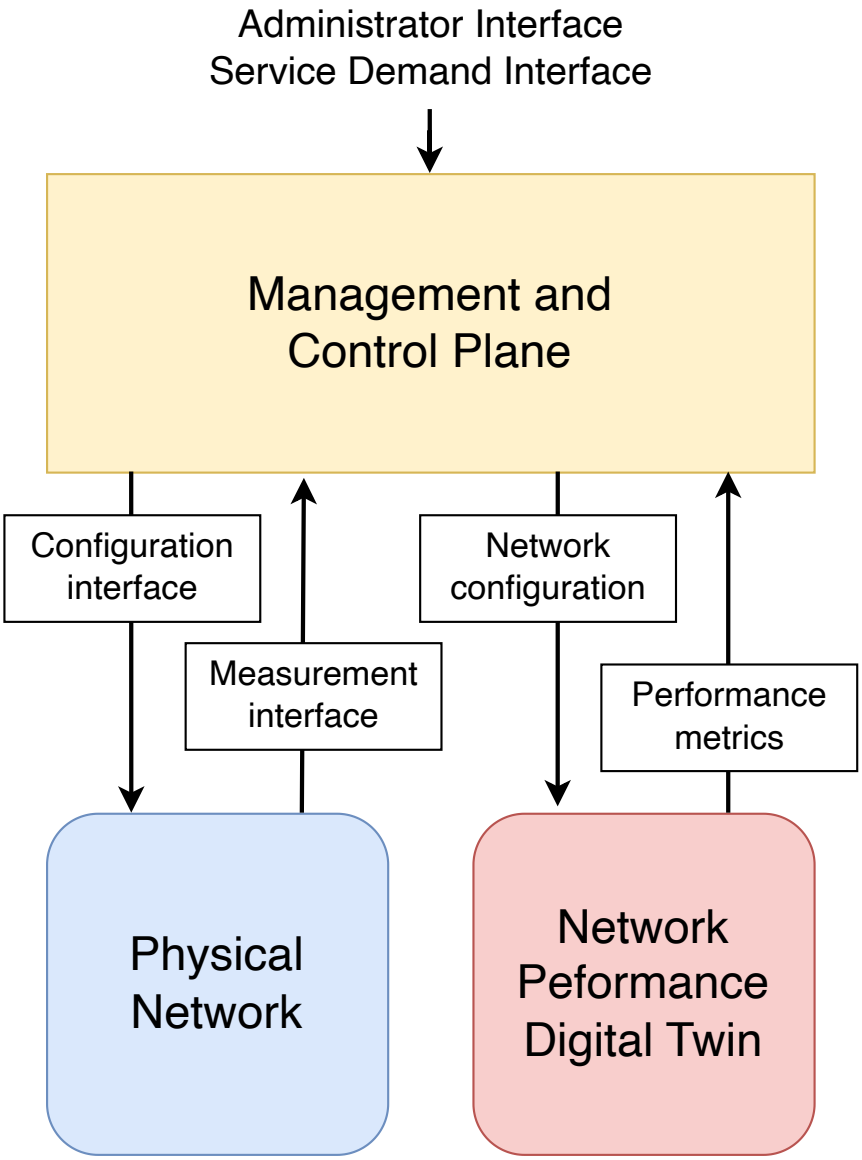


FIGURE 8.1: Architecture of an NPDT-assisted network, with the interfaces between the NPDT and the Management and Control Plane.

Network Interface

This interface relies on common or standardized protocols for the configuration of the physical network infrastructure. It encompasses a range of configuration protocols, including but not limited to NETCONF and PCE, as well as measurement protocols such as Netflow and SNMP. These protocols are leveraged for the configuration of network devices and the retrieval of measurement data from the physical network components.

8.1.2 Requirements

To fulfill its role effectively and support a wide variety of network management scenarios, an NPDT must adhere to the following stringent requirements:

- Speed: The NPDT should deliver fast predictions, with response times on the order of milliseconds. This rapid processing is critical for optimization scenarios that necessitate the evaluation of numerous network configurations within tight timeframes.
- Accuracy: The predictions generated by the NPDT must exhibit minimal error when compared to ground truth data. High accuracy, with deviations well below a defined threshold, is essential to ensure the NPDT's practicality and reliability in real-world network environments.
- Generalization: The NPDT should possess the ability to support networks with arbitrarily large and intricate topologies seamlessly. Its capacity to generalize is critical to accommodate the ever-expanding scale and complexity of modern networks.
- Diverse Input Support: The NPDT should be versatile, and capable of accepting a diverse range of input variables and configurations. This encompasses a variety of routing configurations, scheduling schemes (e.g., FIFO, Weighted Fair Queueing, and Deficit Round Robin), arbitrary network topologies, traffic matrices, traffic models (e.g., Constant Bitrate, Poisson, On/Off), and parameters specific to certain network types (e.g., optical networks). This adaptability ensures that the NPDT can adapt to the unique characteristics of different network scenarios.
- User Accessibility: The NPDT should feature a user-friendly interface tailored to the needs of network engineers and administrators. This interface should leverage well-established and widely adopted networking standards and conventions, ensuring ease of use and familiarity for industry professionals. Additionally, the NPDT's output metrics should be readily comprehensible to network engineers, and it may include confidence values to gauge the reliability of its estimations. These features collectively enhance the NPDT's accessibility and utility within the networking community.

8.2 Technologies for Network Performance Digital Twins

As we mentioned previously, we can think of an NPDT as an advanced network model. Hence, we can find a wide range of technologies in the state of the art that can potentially satisfy the requirements of an NPDT. In this section, we discuss the advantages and drawbacks of the most relevant technologies and how they compare with the requirements. Table 8.1 presents a summary of these technologies.

Requirement	Fast	Accurate	Generalization	Diff. inputs	Low cost
Analytical models	✓	∗	✓	✓	✓
Packet simulators		✓	✓	✓	
Emulators	✓	∗	✓	✓	
Testbeds	✓	✓	∗	∗	
Traditional Neural Networks	✓				✓
Graph Neural Networks	✓	✓	✓	✓	✓

TABLE 8.1: Requirements vs. candidate technologies to implement an NPDT. ∗ stands for partially

8.2.1 Analytical models

Analytical models, particularly those founded on Queuing Theory, exhibit strengths in addressing several of the NPDT requirements. They can fulfill the need for speed and accuracy, allowing for efficient performance calculations across various network topologies, routing strategies, and scheduling configurations.

However, a noteworthy drawback of analytical models is their reliance on strong assumptions regarding the traffic's arrival process, typically assuming Poisson arrivals. This assumption doesn't align with the complex and dynamic nature of real-world packet arrival processes, and as such, these models may fall short of capturing the intricacies of actual network behavior. This limitation can impact their applicability to practical network scenarios where traffic patterns are diverse and often exhibit non-Poissonian characteristics.

8.2.2 Packet-level simulators

Packet-level simulators are extensively employed for performance evaluation and algorithm development, and they excel in terms of accuracy and versatility. They can faithfully replicate diverse network scenarios, including complex topologies, routing configurations, and scheduling algorithms. However, a critical limitation of utilizing packet-level simulators as an NPDT is the time required for simulation. This time tends to increase linearly with the number of packets to process, which can severely restrict their ability to simulate high-speed links effectively. This limitation impacts their suitability for scenarios involving networks with significant data throughput and high-speed connections, where timely predictions are crucial.

8.2.3 Emulators

Emulators are a valuable option for NPDT applications due to their ability to virtualize the target network, making them capable of supporting various input configurations and network types while delivering estimations within a reasonable timeframe. However, it's important to note that the performance of emulators is ultimately constrained by the capabilities of the underlying hardware. As a result, if emulators are employed to replicate large and complex networks, their emulation time and cost may escalate significantly, potentially exceeding practical limits for near real-time operations. Consequently, emulators can be a viable choice when applied judiciously to scenarios where the computational and financial resources required for emulation remain within acceptable bounds.

8.2.4 Testbeds

Testbeds offer the notable advantage of unparalleled accuracy, primarily because they operate within a real network environment. Depending on the specific testbed configuration, they can provide support for a diverse array of network configurations and topologies while also generating performance estimations with considerable speed. However, it's essential to recognize that testbeds come with significant costs, both in financial terms and in the expenditure of energy and human resources. These costs can be substantial, and as

such, testbeds are typically employed in scenarios where the benefits of their exceptional accuracy outweigh the associated resource investments.

8.2.5 Traditional Neural Networks

Traditional Neural Networks like Multilayer Perceptrons and Recurrent Neural Networks are known for their speed, ease of construction, and flexibility in handling various types of input data. However, they do have some limitations when applied to computer networks, particularly when dealing with factors like link failures. While these networks can predict network performance with a high degree of accuracy in many cases, they struggle when confronted with network changes such as link failures. This limitation is mainly attributed to their inability to fully grasp the complex relationships between different elements within the network graph, which is essential for adapting to network changes and disruptions.

8.2.6 Graph Neural Networks

As discussed in previous Chapters, Graph Neural Networks represent a class of neural networks that are purpose-built to operate on graph-structured data. They are characterized by their ability to accept graph structures as input. This innate compatibility with graph data makes GNNs particularly well-suited for computer networks. By utilizing GNNs, it becomes possible to effectively encode and process all the intricate relationships within a network, enabling them to adapt to various network topologies and configurations.

In conclusion, Graph Neural Networks present a promising candidate for the implementation of a Network Performance Digital Twin. The evidence presented in Chapters 4, 5, and 6 indicates that the RouteNet GNN-based family of models are capable of providing rapid performance estimates, can accommodate network graphs of varying sizes, and possess the ability to comprehend the intricate relationships between different elements within a network. This combination of speed, adaptability, and network understanding makes these models a compelling choice for implementing a Network Performance Digital Twin.

8.3 Implementation

Our NPDT has two main components: the User Interface (UI) and the GNN model (RouteNet-F Chapter 6). The UI is a web-based application built with Vue.js[1] and G6 Graph Visualization Engine[2]. It is a user-friendly interface for collecting input data of the target network to be simulated. The UI communicates with a flask-based web server that is responsible for converting the collected input data into a suitable format and forwarding the target network scenario to the GNN model. RouteNet-F, implemented using TensorFlow, processes the received data and calculates the predicted performance values for the given network configuration. Once the GNN model completes the prediction process, it returns the result to the UI. The UI then presents the performance estimates to the user, both graphically and numerically, as a matrix of source-destination delays.

8.3.1 User Interface

The primary objective of the UI in the proposed NPDT system is to provide an interactive and user-friendly platform for end users, including network administrators and engineers. The UI aims to overcome the potential knowledge gap by enabling users without extensive technical expertise to easily utilize the NPDT. It also aims to simplify the process of entering parameters into the GNN model, reducing the likelihood of errors and streamlining the overall user experience.

The UI consists of three main components, as shown in Figure 8.2, and is publicly available[3]:

- **Network Diagram:** This component displays a map featuring the network's nodes and edges. Users have the option to choose between two well-known topologies (NSFNET and EARN) or create a custom topology. Each edge is color-coded based on its load, while the node sizes are proportional to the amount of traffic they process. This visual representation provides users with an intuitive overview of the network structure.

[1] `https://vuejs.org/`
[2] `https://g6.antv.antgroup.com/en/`
[3] `http://gnn-frontend.cba.upc.edu/`

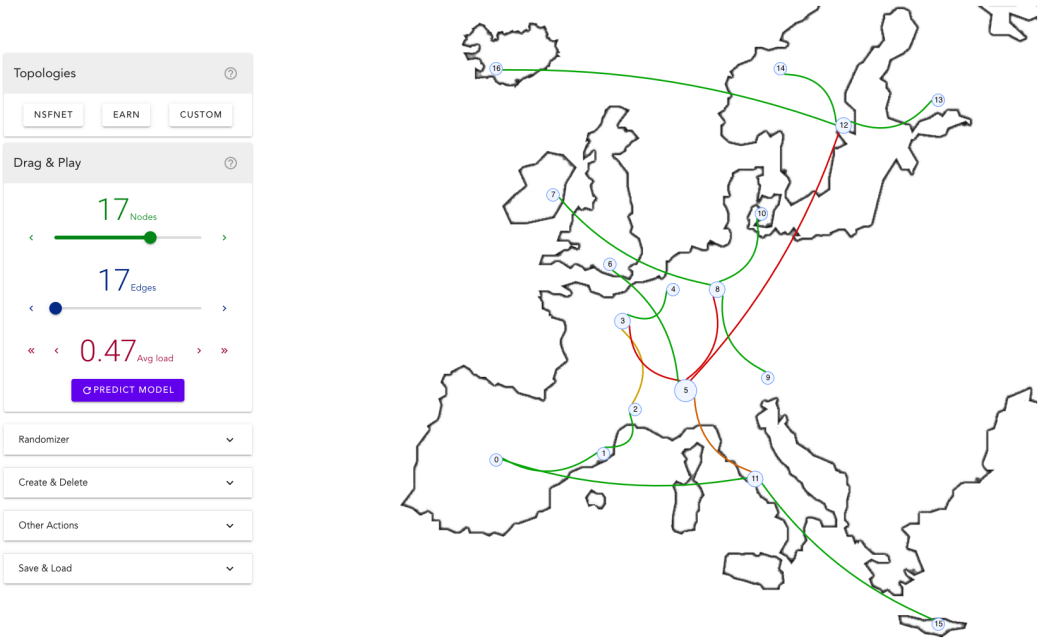


FIGURE 8.2: User controls and graphical representation of the input network, showing the EARN network.

- **Input Controls:** The UI includes a set of controls that enable users to adjust various input parameters. These controls allow increasing or decreasing the number of nodes, edges, and average network load, among other parameters. By interacting with these controls, users can easily customize the input scenario of the NPDT.

- **Delay Estimation Matrix:** The UI presents a matrix that displays the estimated average link delay between all nodes in the network. This matrix provides a concise and organized representation of the delay estimations, enabling users to quickly analyze network performance.

Moreover, the UI provides users with additional controls that allow assigning random capacities to links, random traffic intensities to nodes, and selecting different routing policies. Furthermore, users can edit individual values of the traffic matrix, link capacity, or weight of each link for more fine-grained customization.

8.3.2 RouteNet-F Interface

The interface between the front-end and the back-end is a standard REST API, based on the Flask framework[4]. The back end is implemented following the Model-View-Controller (MVC) pattern, and it translates the information entered by the user in the front end to three matrices that represent key network parameters. These matrices serve as the inputs to the GNN model implemented in TensorFlow, as described in Chapter 6. The three matrices generated by the back-end represent (i) the traffic flow between different nodes in the network (Traffic Matrix) (ii) information about the routing paths followed by the network flows (Routing Matrix), and (iii) the topology of the network and the capacity of each of its links (Capacity Matrix).

8.3.3 Features

RouteNet-F, as a part of the RouteNet GNN-based family of models, possesses a compelling set of features that make it a strong candidate for the implementation of an NPDT. These features include:

- **Fast Performance Estimates:** RouteNet-F is capable of providing performance estimates in small timescales. This speed is crucial for real-time network management scenarios and optimization tasks.
- **Generalization and scalability to larger networks:** It can accept network graphs of different sizes, allowing it to understand and replicate the relationships between different elements in the network. This scalability is vital for accommodating networks of varying complexities and sizes.
- **Routing:** The routing configuration is defined as a series of networking devices that form a path, allowing the model to emulate diverse routing scenarios. Hence, it can support different well-known routing protocols such as OSPF, IS-IS, BGP, LISP, SRv6, OpenFlow, and others.
- **Traffic Models:** The model accepts various traffic models, including Poisson, Constant Bitrate (CBR), On-Off, Autocorrelated Exponentials, and Modulated Exponentials, or a combination of any of the previous. These models capture different traffic patterns and behaviors.

[4] `https://flask.palletsprojects.com/en/2.3.x/`

- **Traffic Matrix:** The amount of traffic that flows from one network device to another network device. It is represented as an NxN matrix where N is the total number of network devices.
- **Scheduling policy:** Each router can be configured with an arbitrary number of output ports, with one of the following queuing configurations: First-in-First-Out (FIFO), Weighted Fair Queuing (WFQ), Deficit Round Robin (DRR), and Strict Priority (SP).

These features collectively position RouteNet-F as an ideal choice for building a Network Performance Digital Twin (NPDT), as demonstrated in Chapters 4, 5, and 6. Its combination of speed, scalability, variety of inputs, and generalization capabilities equips it to effectively replicate and assess networks with diverse characteristics and configurations, making it a valuable tool for network management and optimization.

8.4 Use Cases

The Network Performance Digital Twin (NPDT) offers practical utility in various network management scenarios, addressing real-world challenges and facilitating informed decision-making. Here, we delve into some key use cases of the NPDT:

8.4.1 What-if scenarios

One of the primary applications of the NPDT is in conducting "what-if" scenarios. This involves analyzing the potential impact of hypothetical scenarios and configurations on the network without affecting the real network. The NPDT allows network administrators to assess various scenarios, such as:

- Predicting the impact on network performance when incorporating a new company or a large number of employees.
- Estimating when the network might reach its capacity due to organic user growth.

- Determining the optimal network hardware upgrade within a specified budget.
- Assessing the effect of traffic spikes and evaluating strategies to reduce packet loss.

The NPDT's user-friendly interface simplifies the evaluation of these scenarios. Network administrators define the network configuration within the web application, request predictions, and receive immediate results.

8.4.2 Network Optimization

The NPDT is a valuable tool for network optimization. It can be paired with optimization algorithms to achieve specific objectives. In this scenario, the NPDT provides performance estimates for different network configurations, and the optimizer aims to meet optimization objectives. These objectives may include constraints like maintaining path delays below a certain threshold or limiting link utilization. A wide range of optimization algorithms can be applied, from Integer Linear Programming (ILP) to genetic algorithms or even modern approaches based on Deep Reinforcement Learning.

Once the optimizer finds the best network configuration, it can be applied to the actual network through standard configuration interfaces.

Figure 8.3 illustrates the role of the NPDT in a network optimization scenario. This combination of prediction and optimization empowers network administrators to make data-driven decisions and fine-tune their networks for optimal performance.

8.5 Discussion and Concluding Remarks

In this Chapter, we have discussed the potential of the RouteNet family of models as a possible implementation of a Network Performance Digital Twin designed to predict the performance of diverse network scenarios. We equipped the NPDT with a user-friendly graphical interface, making it accessible for end-users to input network configurations and analyze results rapidly. We've also elaborated on the interfaces through which the NPDT

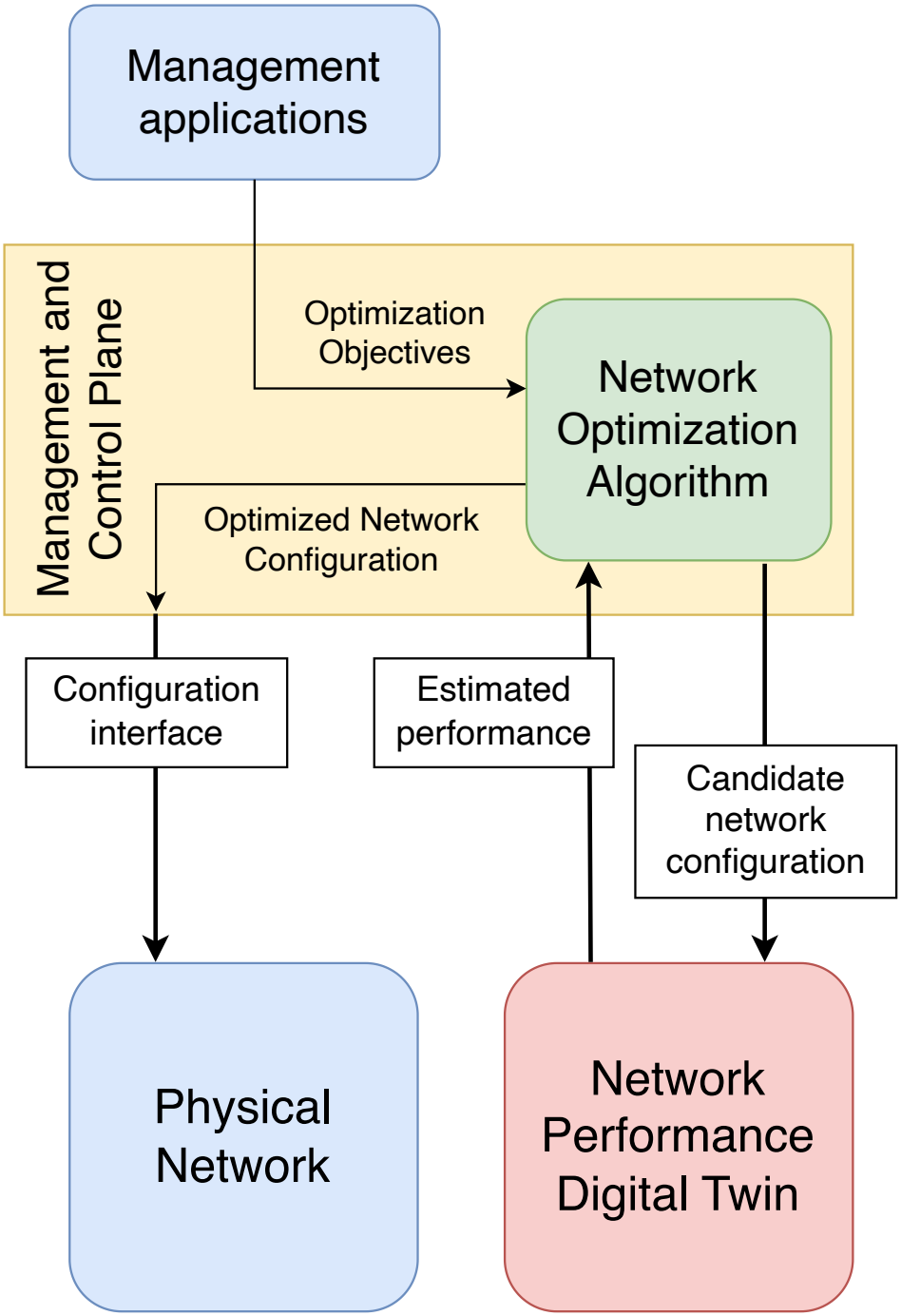


FIGURE 8.3: Role of the NPDT in a network optimization scenario.

interacts with the management and control planes of physical networks, establishing its integration within real network environments.

Furthermore, we have highlighted two significant use cases of the NPDT: "What-If" analysis, which serves as a tool for assessing the potential impact of new configurations on network performance, and network optimization. Network optimization involves leveraging the NPDT as a model

for optimization algorithms, and aligning network configurations with defined objectives. By presenting the NPDT's capabilities, interfaces, and practical applications, we underscore its potential as a valuable asset for network management and decision-making processes.

Chapter 9

Conclusions and Future Work

In this thesis, we have embarked on a journey through the dynamic landscape of network modeling, a basis for the design, evaluation, and optimization of computer networks. The foundation of network modeling, established in the early days of networking, has proven indispensable for a numerous number of applications, including protocol design, performance assessment, and network planning. The prevalent techniques of analytical modeling grounded in Queuing Theory (QT) and packet-level simulators have been the go-to of this field, facilitating substantial progress in understanding network behavior.

However, as computer networks continue to evolve, their growing complexity and diverse traffic characteristics have unveiled inherent limitations in classical modeling methodologies. These limitations have become particularly pronounced in scenarios where advanced network modeling techniques are essential.

This thesis has underscored a fundamental principle: the efficiency of network optimization is intricately linked to the capacity to create accurate and rapidly responsive network models. While conventional discrete event simulation (DES) methodologies have offered a degree of accuracy, they often fall short due to their computational demands, particularly when confronted with large-scale networks and realistic traffic volumes. Furthermore, these modeling techniques can prove inadequate for online network optimization applications due to stringent time constraints. It has thus become increasingly evident that state-of-the-art modeling techniques must adapt to meet the exacting requirements of contemporary packet-switched networks. The limitations imposed by Queuing Theory, which predicates network behavior

on Poisson traffic assumptions, are increasingly apparent as network behaviors exhibit strong autocorrelation and heavy-tailed distributions, rendering these classical models insufficient to capture the complexities of real-world networks.

In this dissertation, we have delved into the realm of network modeling, exploring innovative solutions to address significant challenges in the field of computer networks. We have presented three distinct models, namely RouteNet-D, RouteNet-E, and RouteNet-F, each designed to tackle specific problems within the current state of the art.

RouteNet-D laid the foundation by introducing the fundamental challenge of incorporating scheduling policies into network modeling. We recognized the critical role of scheduling in network performance and developed a model capable of addressing this element. It paved the way for a more comprehensive approach to network modeling, emphasizing the importance of capturing the intricacies of scheduling policies.

Building upon RouteNet-D's foundational principles, RouteNet-E expanded its scope to tackle scheduling, traffic models, scalability, and generalization to larger networks. This journey revealed a crucial distinction between analytical models, which rely on theory and need no training, and DL models, which demand a diverse training dataset for accurate predictions. Generating such a dataset from production networks, experiencing rare performance issues, is problematic. Controlled testbeds emerged as a practical solution, enabling the replication of diverse network scenarios. RouteNet-E was crafted to meet this challenge, showcasing its adaptability and the ability to provide accurate predictions on networks significantly larger in scale than the ones encountered during training.

RouteNet-F marked a significant milestone by unifying the elements of scheduling, traffic models, and generalization to larger networks, offering a comprehensive solution to network modeling. Its capabilities in providing rapid performance estimates, accommodating diverse network configurations, and adapting to the complex relationships between network elements make it an ideal choice for implementing a network model.

Building on the foundation of RouteNet-F, we explored the integration of the temporal dimension into network modeling. Recognizing the dynamic nature of real-world networks, we considered how network models

can adapt to changes over time. By incorporating temporal aspects, we presented an innovative approach to enhance the accuracy and practicality of network modeling.

In the final chapters, we showcased the practical application of these models by implementing a Network Performance Digital Twin (NPDT). The NPDT serves as a powerful tool in network management scenarios, facilitating "what-if" analysis and network optimization. It empowers network administrators to assess the impact of potential scenarios and make informed decisions, all while ensuring minimal disruption to the actual network.

Throughout this thesis, the exploration of network modeling has paved the way for the emergence of several works that directly engage with the challenges in this domain, drawing inspiration from the concepts and methodologies developed herein. Noteworthy among these works is xNet [122], which capitalizes on Graph Neural Networks (GNNs) to comprehensively model complex attributes of computer networks. By learning the state transition function between time steps, xNet achieves a holistic fine-grained prediction trajectory. In a parallel vein, initiatives like MimicNet [53] and DeepQueueNet [107] employ Deep Learning (DL) models to accelerate network event simulators. The first one, focuses on accelerating specific aspects of the simulation process by the use of LSTMs, concentrating on per-packet delay prediction. Similarly, DeepQueueNet takes advantage of Transformers to generate a scalable and generalizable network performance estimator with packet-level visibility.

In conclusion, the journey through these chapters highlights the significance of network modeling as an indispensable tool in understanding, optimizing, and enhancing the performance of real-world networks. The RouteNet-family models offer innovative solutions to address the challenges within network modeling, ultimately paving the way for more informed and efficient network management. As the landscape of computer networks continues to evolve, these models and the insights gained from this thesis are poised to play an increasingly critical role in network modeling and optimization.

Before concluding, our ambition with this dissertation is twofold. We do not only expect that the contributions presented in this thesis will represent significant advancements towards the achievement of the long-desired network models but also will open up new research paths to be explored.

We summarize below some avenues of research that may be interesting to investigate in the future:

- Enhanced Traffic Representation: This thesis has demonstrated the capability of the RouteNet-family models in comprehending various traffic models commonly observed in real-world networks. However, these models have primarily been formulated based on mathematical parameters, and the real-world scenario often deviates from this idealized mathematical representation. Currently, RouteNet processes input primarily using mathematical traffic model parameters, which may not always capture the nuanced complexities of actual packet traces. A promising trajectory for future research involves extending the model's capabilities to interpret real packet traces. This transition from a purely parameter-driven approach to one grounded in the analysis of authentic packet-level data could significantly enhance the model's fidelity and applicability in practical network modeling.

- Packet-Level Visibility: The current RouteNet-family models offer valuable insights into network behavior, particularly in terms of predefined performance metrics such as delay, jitter, and packet losses. However, they operate at a higher level of abstraction, mapping network facts to these predetermined metrics. What they presently lack is the capability to delve into the fine-grained details of individual devices or specific data flows within the network. A promising avenue for future exploration is the development of mechanisms within RouteNet to provide packet-level visibility. This means enabling the models to offer granular, packet-level statistics, empowering network analysts and administrators with deeper insights into network dynamics and performance on a per-packet basis.

Bibliography

[1] Leonard Kleinrock. "Analytic and simulation methods in computer network design". In: *Proceedings of the May 5-7, 1970, spring joint computer conference*. 1970, pp. 569–579.

[2] Thomas G Robertazzi. *Computer networks and systems: queueing theory and performance evaluation*. Springer Science & Business Media, 2000.

[3] Richard M Fujimoto et al. "Large-scale network simulation: how big? how fast?" In: *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* IEEE. 2003, pp. 116–123.

[4] Paul Almasan, Miquel Ferriol-Galmés, et al. "Digital twin network: Opportunities and challenges". In: *arXiv preprint arXiv:2201.01144* (2022).

[5] Huan Nguyen et al. "Digital twin for 5G and beyond". In: *IEEE Communications Magazine* 59.2 (2021), pp. 10–15.

[6] Seyedali Mirjalili. "Genetic algorithm". In: *Evolutionary algorithms and neural networks*. Springer, 2019, pp. 43–55.

[7] George F Riley and Thomas R Henderson. "The ns-3 network simulator". In: *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[8] András Varga. "Discrete event simulation system". In: *European Simulation Multiconference (ESM)*. 2001, pp. 1–7.

[9] Robert B Cooper. "Queueing theory". In: *Proceedings of the ACM'81 conference*. 1981, pp. 119–122.

[10] Zhiyuan Xu et al. "Experience-driven networking: A deep reinforcement learning based approach". In: *IEEE INFOCOM*. 2018, pp. 1871–1879.

[11] Asad Arfeen, Krzysztof Pawlikowski, et al. "The role of the Weibull distribution in modelling traffic in Internet access and backbone core networks". In: *Journal of network and computer applications* 141 (2019), pp. 1–22.

[12] Thomas Karagiannis et al. "A nonstationary Poisson view of Internet traffic". In: *IEEE INFOCOM 2004*. Vol. 3. IEEE. 2004, pp. 1558–1569.

[13] Thomas Karagiannis, Mart Molle, and Michalis Faloutsos. "Long-range dependence ten years of Internet traffic modeling". In: *IEEE internet computing* 8.5 (2004), pp. 57–64.

[14] Edward Kresch and Sarvesh Kulkarni. "A poisson based bursty model of internet traffic". In: *IEEE International Conference on Computer and Information Technology*. 2011, pp. 255–260.

[15] Vern Paxson and Sally Floyd. "Wide area traffic: the failure of Poisson modeling". In: *IEEE/ACM Transactions on networking* 3.3 (1995), pp. 226–244.

[16] J Popoola and RA Ipinyomi. "Empirical performance of weibull self-similar tele-traffic model". In: *International Journal of Engineering and Applied Sciences* 4.8 (2017), p. 257389.

[17] Renaud Hartert et al. "A declarative and expressive approach to control forwarding paths in carrier-grade networks". In: *ACM SIGCOMM computer communication review* 45.4 (2015), pp. 15–28.

[18] Liang Guo and Ibrahim Matta. "Search space reduction in QoS routing". In: *Computer Networks* 41.1 (2003), pp. 73–88.

[19] Mathieu Jadin et al. "CG4SR: Near optimal traffic engineering for segment routing with column generation". In: *IEEE INFOCOM*. 2019, pp. 1333–1341.

[20] Randeep Bhatia et al. "Optimized network traffic engineering using segment routing". In: *IEEE INFOCOM*. 2015, pp. 657–665.

[21] Jochen W Guck, Van Bemten, et al. "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation". In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 388–415.

[22] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[23] Yann LeCun et al. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[24] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021). (Accelerated article preview). DOI: 10.1038/s41586-021-03819-2.

[25] Albert Mestres et al. "Understanding the modeling of computer network delays using neural networks". In: *ACM SIGCOMM BigDAMA workshop*. 2018, pp. 46–52.

[26] Mowei Wang et al. "Machine learning for networking: Workflow, advances and opportunities". In: *Ieee Network* 32.2 (2017), pp. 92–99.

[27] Franco Scarselli, Marco Gori, et al. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80.

[28] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *arXiv preprint arXiv:1704.01212* (2017).

[29] Peter Battaglia et al. "Interaction networks for learning about objects, relations and physics". In: *Advances in neural information processing systems*. 2016, pp. 4502–4510.

[30] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *arXiv preprint arXiv:1812.08434* (2018).

[31] Oliver Lange and Luis Perez. *Traffic prediction with advanced Graph Neural Networks*. 2020. URL: https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks (visited on 07/26/2021).

[32] Amina Al-Sawaai et al. "Performance Evaluation of Weighted Fair Queuing System Using Matrix Geometric Method". In: *IFIP NETWORKING*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 66–78. ISBN: 978-3-642-01399-7.

[33] Asha Seth Kapadia, Mohammad Fasihullah Kazmi, and A Cameron Mitchell. "Analysis of a finite capacity non preemptive priority queue". In: *Computers & operations research* 11.3 (1984), pp. 337–343.

[34] Ilkka Norros. "A storage model with self-similar input". In: *Queueing systems* 16.3 (1994), pp. 387–396.

[35] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[36] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[37] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[38] David Raposo et al. "Discovering objects and their relations from entangled scene representations". In: *arXiv preprint arXiv:1702.05068* (2017).

[39] Xiaolong Wang et al. "Non-local neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7794–7803.

[40] Manzil Zaheer et al. "Deep sets". In: *Advances in neural information processing systems*. 2017, pp. 3391–3401.

[41] Diego Kreutz et al. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2014), pp. 14–76.

[42] Albert Mestres et al. "Knowledge-defined networking". In: *ACM SIGCOMM Computer Communication Review* 47.3 (2017), pp. 2–10.

[43] Raouf Boutaba et al. "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities". In: *Journal of Internet Services and Applications* 9.1 (2018), p. 16.

[44] Krzysztof Rusek and Piotr Chołda. "Message-passing neural networks learn little's law". In: *IEEE Communications Letters* 23.2 (2018), pp. 274–277.

[45] Fabien Geyer and Georg Carle. "Learning and generating distributed routing protocols using graph-based deep learning". In: *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. 2018, pp. 40–45.

[46] Liang Lu et al. "Ranking attack graphs with graph neural networks". In: *International Conference on Information Security Practice and Experience*. Springer. 2009, pp. 345–359.

[47] Fabien Geyer. "Performance evaluation of network topologies using graph-based deep learning". In: *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. 2017, pp. 20–27.

[48] Fabien Geyer and Steffen Bondorf. "DeepTMA: Predicting effective contention models for network calculus using graph neural networks". In: *IEEE INFOCOM*. 2019, pp. 1009–1017.

[49] Fabien Geyer and Georg Carle. "The case for a network calculus heuristic: Using insights from data for tighter bounds". In: *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 2. IEEE. 2018, pp. 43–48.

[50] Krzysztof Rusek, José Suárez-Varela, et al. "Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN". In: *ACM Symposium on SDN Research*. 2019, pp. 140–151.

[51] José Suárez-Varela et al. "Challenging the generalization capabilities of Graph Neural Networks for network modeling". In: *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. 2019, pp. 114–115.

[52] Elias Weingartner, Hendrik Vom Lehn, and Klaus Wehrle. "A performance comparison of recent network simulators". In: *2009 IEEE International Conference on Communications*. IEEE. 2009, pp. 1–5.

[53] Qizhen Zhang et al. "MimicNet: fast performance estimates for data center networks with machine learning". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 287–304.

[54] Christopher R Palmer and J Greg Steffan. "Generating network topologies that obey power laws". In: *IEEE Global Telecommunications Conference (GLOBECOM)*. Vol. 1. 2000, pp. 434–438.

[55] R.B. Nelsen. *An Introduction to Copulas*. Lecture notes in statistics. Springer, 1999. ISBN: 9780387986234.

[56] Xiaojun Hei, Jun Zhang, et al. "Wavelength converter placement in least-load-routing-based optical networks using genetic algorithms". In: *Journal of Optical Networking* 3.5 (2004), pp. 363–378.

[57] Fernando Barreto et al. "Fast emergency paths schema to overcome transient link failures in ospf routing". In: *arXiv preprint arXiv:1204.2465* (2012).

[58] João Pedro, João Santos, and João Pires. "Performance evaluation of integrated OTN/DWDM networks with single-stage multiplexing of optical channel data units". In: *International Conference on Transparent Optical Networks*. 2011, pp. 1–4.

[59] Giovanni Giambene. *Queuing theory and telecommunications*. Vol. 585. Springer, 2014.

[60] Frank P. Kelly. *Reversibility and Stochastic Networks*. Cambridge University Press, 2011, p. 230. ISBN: 1107401151.

[61] Forest Baskett et al. “Open, Closed, and Mixed Networks of Queues with Different Classes of Customers”. In: *J. ACM* 22.2 (Apr. 1975), 248–260. ISSN: 0004-5411. DOI: 10.1145/321879.321887. URL: https://doi.org/10.1145/321879.321887.

[62] William J. Stewart. “Numerical Methods for Computing Stationary Distributions of Finite Irreducible Markov Chains”. In: *Computational Probability*. Ed. by Winfried K. Grassmann. Boston, MA: Springer US, 2000, pp. 81–111. ISBN: 978-1-4757-4828-4. DOI: 10.1007/978-1-4757-4828-4_4. URL: https://doi.org/10.1007/978-1-4757-4828-4_4.

[63] Masaaki Kijima. *Markov Processes for Stochastic Modeling*. Stochastic Modeling Series. New York, NY: Springer-Science+Business Media, B.V., 1997.

[64] Sankar K Pal and Sushmita Mitra. “Multilayer perceptron, fuzzy sets, and classification”. In: *IEEE Transactions on neural networks* 3.5 (1992), pp. 683–697.

[65] Nargess Sadeghzadeh et al. “An MLP neural network for time delay prediction in networked control systems”. In: *2008 Chinese Control and Decision Conference*. IEEE. 2008, pp. 5314–5318.

[66] Tomáš Mikolov, Stefan Kombrink, et al. “Extensions of recurrent neural network language model”. In: *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2011, pp. 5528–5531.

[67] Salem Belhaj and Moncef Tagina. “Modeling and Prediction of the Internet End-to-end Delay using Recurrent Neural Networks.” In: *J. Networks* 4.6 (2009), pp. 528–535.

[68] AysŞe Rumeysa Mohammed et al. “Machine learning and deep learning based traffic classification and prediction in software defined networking”. In: *2019 IEEE International Symposium on Measurements & Networking (M&N)*. IEEE. 2019, pp. 1–6.

[69] Sheraz Naseer et al. “Enhanced network anomaly detection based on deep neural networks”. In: *IEEE access* 6 (2018), pp. 48231–48246.

[70] José Suárez-Varela, Paul Almasan, Miquel Ferriol-Galmés, et al. “Graph Neural Networks for Communication Networks: Context, Use Cases and Opportunities”. In: *IEEE Network* (2022).

[71] Amina Al-Sawaai, Irfan Awan, and Rod Fretwell. "Performance evaluation of weighted fair queuing system using matrix geometric method". In: *International Conference on Research in Networking*. Springer. 2009, pp. 66–78.

[72] Jiaqi Ma, Junwei Deng, and Qiaozhu Mei. "Subgroup generalization and fairness of graph neural networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 1048–1061.

[73] Gilad Yehudai et al. "From local structures to size generalization in graph neural networks". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11975–11986.

[74] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: (2014).

[75] Günter Klambauer et al. "Self-normalizing neural networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. –.

[76] Miquel Ferriol-Galmés. *Building a Digital Twin for Network Optimization using Graph Neural Networks*. `https://github.com/BNN-UPC/Papers/wiki/TwinNet`. 2021.

[77] Krzysztof Rusek et al. "RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2260–2270.

[78] Sally Floyd and Vern Paxson. "Difficulties in simulating the Internet". In: *IEEE/ACm Transactions on Networking* 9.4 (2001), pp. 392–403.

[79] Simon Knight et al. "The internet topology zoo". In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1765–1775.

[80] Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. "Random graphs with arbitrary degree distributions and their applications". In: *Physical review E* 64.2 (2001), p. 026118.

[81] Alexei Botchkarev. "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006* (2018).

[82] Sebastian Orlowski, Roland Wessäly, et al. "SNDlib 1.0—Survivable network design library". In: *Networks: An International Journal* 55.3 (2010), pp. 276–286. URL: `http://sndlib.zib.de`.

[83] Romain Fontugne, Pierre Borgnat, et al. "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *ACM CoNEXT '10*. Philadelphia, PA, Dec. 2010, pp. –.

[84] Stefan Schnitter et al. "Quality-of-service class specific traffic matrices in IP/MPLS networks". In: *ACM Internet Measurement Conference*. 2007, pp. 253–258.

[85] Akyildiz et al. "A roadmap for traffic engineering in SDN-OpenFlow networks". In: *Computer Networks* 71 (2014), pp. 1–30.

[86] Frode Sorensen Yiannis Yiakoumis Nick McKeown. *Network Tokens*. Internet-Draft draft-yiakoumis-network-tokens-01. Work in Progress. IETF, June 2020. 28 pp.

[87] Zhenjie Yang et al. "Software-defined wide area network (SD-WAN): Architecture, advances and opportunities". In: *International Conference on Computer Communication and Networks (ICCCN)*. 2019, pp. 1–9.

[88] Haijun Zhang et al. "Network slicing based 5G and future mobile networks: mobility, resource management, and challenges". In: *IEEE communications magazine* 55.8 (2017), pp. 138–145.

[89] Xiaomin Li, Di Li, et al. "A review of industrial wireless networks in the context of industry 4.0". In: *Wireless networks* 23.1 (2017), pp. 23–41.

[90] Meryem Simsek et al. "5G-enabled tactile internet". In: *IEEE Journal on Selected Areas in Communications* 34.3 (2016), pp. 460–473.

[91] Rajesh Gupta et al. "Tactile internet and its applications in 5G era: A comprehensive review". In: *International Journal of Communication Systems* 32.14 (2019), e3981.

[92] Zhe Chen et al. "NEW IP Framework and Protocol for Future Applications". In: *IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–5.

[93] Richard Li. "Towards a new Internet for the year 2030 and beyond". In: *Annual ITU IMT-2020/5G Workshop Demo Day*. 2018, pp. 1–21.

[94] Changhoon Kim et al. "In-band network telemetry via programmable dataplanes". In: *ACM SIGCOMM, Posters and Demos*. 2015, pp. –.

[95] Minlan Yu, Lavanya Jose, and Rui Miao. "Software Defined Traffic Measurement with OpenSketch". In: *Symposium on Networked Systems Design and Implementation (NSDI)*. 2013, pp. 29–42.

[96] Steven Gay, Pierre Schaus, and Stefano Vissicchio. "Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms". In: *arXiv preprint arXiv:1710.08665* (2017).

[97] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. "Graph neural networks and the transferability of graph neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020).

[98] Logan Engstrom et al. "Exploring the landscape of spatial robustness". In: *International Conference on Machine Learning*. 2019, pp. 1802–1811.

[99] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks". In: *IEEE Transactions on Evolutionary Computation* 23.5 (2019), pp. 828–841.

[100] Miquel Ferriol-Galmés et al. *RouteNet-Erlang*. `https://github.com/BNN-UPC/Papers/wiki/RouteNet_Erlang`. 2021.

[101] Daniel R Figueiredo et al. "On the autocorrelation structure of TCP traffic". In: *Computer Networks* 40.3 (2002), pp. 339–361.

[102] Zili Meng et al. "Interpreting Deep Learning-Based Networking Systems". In: *ACM SIGCOMM*. 2020, pp. 154–171.

[103] Miquel Ferriol-Galmés et al. *RouteNet-Fermi*. `https://github.com/BNN-UPC/Papers/wiki/RouteNet_Fermi`. 2022.

[104] Krzysztof Rusek. *Queuinx*. `https://github.com/krzysztofrusek/queuinx`. 2023.

[105] Miquel Ferriol-Galmés Albert López et al. *BNNetSimulator*. 2023. URL: `https://github.com/BNN-UPC/BNNetSimulator`.

[106] Miquel Ferriol-Galmés et al. "Routenet-erlang: A graph neural network for network performance evaluation". In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE. 2022, pp. 2018–2027.

[107] Qingqing Yang, Xi Peng, et al. "DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, 441–457. ISBN: 9781450394208. DOI: 10.1145/3544216.3544248. URL: https://doi.org/10.1145/3544216.3544248.

[108] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. "Neural adaptive video streaming with pensieve". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 197–210.

[109] Long Qu, Maurice Khabbaz, and Chadi Assi. "Reliability-aware service chaining in carrier-grade softwarized networks". In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 558–573.

[110] David Zats et al. "DeTail: Reducing the flow completion time tail in datacenter networks". In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 2012, pp. 139–150.

[111] Elvin Isufi and Gabriele Mazzola. "Graph-time convolutional neural networks". In: *2021 IEEE Data Science and Learning Workshop (DSLW)*. IEEE. 2021, pp. 1–6.

[112] Rongzhou Huang et al. "LSGCN: Long Short-Term Traffic Prediction with Graph Convolutional Networks." In: *IJCAI*. 2020, pp. 2355–2361.

[113] Alessandro D'Alconzo et al. "A survey on big data for network traffic monitoring and analysis". In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 800–813.

[114] Myung-Sup Kim, Young J Won, and James W Hong. "Characteristic analysis of internet traffic from the perspective of flows". In: *Computer Communications* 29.10 (2006), pp. 1639–1652.

[115] Simon Bauer et al. "On the evolution of internet flow characteristics". In: *Proceedings of the Applied Networking Research Workshop*. 2021, pp. 29–35.

[116] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[117] Minglan Xiong and Huawei Wang. "Digital twin applications in aviation industry: A review". In: *The International Journal of Advanced Manufacturing Technology* 121.9-10 (2022), pp. 5677–5692.

[118] Ahmed Zainul Abideen et al. "Digital twin integrated reinforced learning in supply chain and logistics". In: *Logistics* 5.4 (2021), p. 84.

[119] Yiwen Wu, Ke Zhang, and Yan Zhang. "Digital twin networks: A survey". In: *IEEE Internet of Things Journal* 8.18 (2021), pp. 13789–13804.

[120] Paul Almasan et al. "Network digital twin: Context, enabling technologies, and opportunities". In: *IEEE Communications Magazine* 60.11 (2022), pp. 22–27.

[121] Jennifer Rexford. "Route optimization in IP networks". In: *Handbook of Optimization in Telecommunications* (2006), pp. 679–700.

[122] Mowei Wang, Linbo Hui, Yong Cui, et al. "xNet: Improving Expressiveness and Granularity for Network Modeling with Graph Neural Networks". In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE. 2022, pp. 2028–2037.