Computer network modeling and root cause analysis with statistical learning

Loïck Bonniot

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Loïck BONNIOT

Computer Network Modeling and Root Cause Analysis with Statistical Learning

Thèse présentée et soutenue à Rennes, le 10 juin 2021
Unité de recherche : IRISA (UMR 6074)

Rapporteur·euse·s avant soutenance :

Chadi BARAKAT	Directeur de recherche, Inria Sophia Antipolis
Isabelle CHRISMENT	Professeure, Université de Lorraine

Composition du Jury :

Présidente :	Élisa FROMONT	Professeure, Université Rennes 1
Examinateur·ice·s :	Chadi BARAKAT	Directeur de recherche, Inria Sophia Antipolis
	Isabelle CHRISMENT	Professeure, Université de Lorraine
	Leonardo LINGUAGLOSSA	Maître de Conférences, Télécom Paris
Dir. de thèse :	François TAÏANI	Professeur, Université Rennes 1
Co-dir. de thèse :	Christoph NEUMANN	Principal Scientist, InterDigital

Invitée :

Sandrine VATON Professeure, IMTA Brest

Acknowledgements

First and foremost, my warmest thanks go to François Taïani and Christoph Neumann, who trusted and guided me during the rewarding adventure of my PhD. Despite challenging times, they allowed me to learn about a lot of different subjects and made possible all the experiments and findings presented in this document. I also thank the ANRT and Technicolor (then InterDigital) for funding this thesis and providing a sane and enjoyable work environment.

Of course, I would like to thank all the members of my PhD committee: Chadi Barakat and Isabelle Chrisment for the time-consuming task of reviewing of my manuscript and their sharp comments; Élisa Fromont, Leonardo Linguaglossa and Sandrine Vaton for their insightful remarks and suggestions made during the defense. Thanks a lot!

Being a CIFRE student, I had the chance to have *two* very inspiring groups of colleagues. I first thank the Home lab at Technicolor and InterDigital (Corentin, François, Jean-Renan, Laurent, Mario, Michel, Nicolas, Pascal, Patrick, Thierry, …): during the pandemy, I have missed our lunch & coffee discussions. I hope to see you all very soon! I also thank all the past and present members of the Inria WIDE team, including of course my fellow doctorate companions: Pierre-Louis, Olivier, Adrien, Louison, Quentin, Alex. I also miss our heated lunch conversations! As a special thank, I am extremely grateful to Virginie and Nelly who helped me navigate in the administrative labyrinth when needed.

Please allow me to digress in French a little bit: le travail de thèse n'a pas toujours été facile, mais j'ai toujours pu compter sur la confiance de mes parents et la bonne humeur de mes amis pour avancer. Merci Ben pour tes relectures et tes yeux de lynx, et bien entendu merci Camille d'être toujours à mes côtés. *Trugarez vras d'an holl !*

Finally, I thank you, the reader of this thesis. I sincerely hope that it will give you interesting insights about the complex problems of network modeling and root cause analysis.

Table of Contents

1 Introduction **9**

1.1 The increasing complexity of computer networks . . . 9
1.2 Data-driven approaches for network modeling . . . 11
1.3 Research challenges and findings . . . 12
1.3.1 Network modeling for fast performance prediction . . . 13
1.3.2 Root Cause Analysis from the edge . . . 14
1.3.3 Leveraging web browsers for network metrics collection . . . 15
1.4 Thesis outline . . . 16

2 Background **17**

2.1 Definitions . . . 19
2.2 Network modeling . . . 22
2.2.1 Modeling from complete network knowledge . . . 22
2.2.2 Modeling from limited knowledge . . . 24
2.3 Root cause analysis . . . 27
2.3.1 RCA from complete network knowledge . . . 28
2.3.2 RCA from limited knowledge . . . 30
2.4 Machine learning . . . 33
2.4.1 Methodology . . . 33
2.4.2 Naive Bayes classifiers . . . 35
2.4.3 Decision tree learning . . . 36
2.4.4 Ensemble learning . . . 37
2.4.5 Artificial Neural Networks . . . 37
2.5 Conclusion . . . 43

3 Computer Network Modeling with Graph Neural Networks **45**

3.1 Context and Motivation . . . 47
3.1.1 ITU's 2020 Graph Neural Network challenge . . . 47
3.1.2 The RouteNet baseline . . . 48

3.2 Our proposal: learn representations for network nodes . . . 50
3.2.1 Message passing over bipartite graphs for dependency modeling . . . 50
3.2.2 Approach A1: dedicated representation with node embeddings . . . 53
3.2.3 Approach A2: node representation through link embeddings . . . 57
3.3 Evaluation with the datasets of the GNN challenge . . . 58
3.3.1 Datasets presentation and challenge results . . . 58
3.3.2 Methodology and implementation details . . . 61
3.3.3 Techniques for optimal training strategy . . . 63
3.3.4 Ablation analysis . . . 65
3.3.5 Visualization of learned representations . . . 69
3.4 Conclusion . . . 72

4 Internet-scale Convolutional Root Cause Analysis 73
4.1 Challenges of RCA from end users . . . 75
4.1.1 Network and service agnosticism . . . 75
4.1.2 Anomaly disentanglement . . . 75
4.1.3 Location agnosticism . . . 76
4.1.4 Root cause extensibility . . . 76
4.2 Convolutional RCA with DiagNet . . . 78
4.2.1 Methodology . . . 78
4.2.2 The DiagNet inference model . . . 80
4.2.3 Non-overlapping convolutions with pooling . . . 81
4.2.4 Tailoring to specific services . . . 83
4.2.5 Fine-grained inference via attention mechanisms . . . 83
4.3 Evaluation with controlled faults . . . 87
4.3.1 Experimental setup . . . 87
4.3.2 Comparison baselines . . . 90
4.3.3 DiagNet evaluation results with controlled faults . . . 91
4.4 Conclusion . . . 99

5 From theory to practice: Root Cause Analysis from web browsers 101
5.1 Data collection from browsers . . . 103
5.1.1 Motivation and related work . . . 103
5.1.2 Architecture of DiagSys . . . 106
5.1.3 QoS measurements provided by landmarks . . . 107

5.1.4 Third-party QoE monitoring from browsers . . . 110
5.2 Insights from our collected dataset . . . 113
5.2.1 Dataset case studies . . . 114
5.2.2 Dataset faults analysis . . . 119
5.2.3 Our proposal: labelling from historical data . . . 120
5.3 Applying DiagNet to DiagSys's dataset . . . 124
5.3.1 Changes to the Root Cause Analysis (RCA) approaches . . . 124
5.3.2 Pinpointing fault families . . . 125
5.3.3 Complete root cause analysis at Internet scale . . . 126
5.3.4 Discussion and remaining open questions . . . 128
5.4 Conclusion . . . 130

6 Conclusion **131**
6.1 Summary and future work . . . 131
6.1.1 Realistic queuing is needed for data-driven modeling of networks . . . 131
6.1.2 The ground-truth problem for accurate RCA models . . . 132
6.2 Outlook: towards greater collaboration between actors . . . 134
6.2.1 Better transparency is needed from service providers . . . 134
6.2.2 The way towards standard measurement APIs . . . 135
6.2.3 Crowdsourcing as a potential solution . . . 137

Résumé en Français **139**

Publications **149**

Acronyms **151**

Bibliography **153**

Chapter 1

Introduction

> "We are all now connected by the Internet, like neurons in a giant brain."
>
> *Stephen Hawking*

The Internet has become the primary communication method for many organizations, individuals and machines. It is powered by a federation of thousands of network operators, collaborating to enable fast and resilient communications between any device on earth. The Internet has in many ways been a resounding success, but it still faces daunting challenges. In particular, with the rise of global Internet demand and multimedia services offering, it is becoming increasingly difficult for operators to ensure good consumer quality of experience. By contrast, the complexity of the network is mostly hidden to end users: troubleshooting connectivity failures remains difficult in practice. Recent advances in the field of Machine Learning have shown that data-driven approaches can help understand complex systems such as the Internet. In this thesis, we show that these approaches can be used for large computer network modeling, even when the available knowledge is scarce.

1.1 The increasing complexity of computer networks

The demand for faster Internet has globally and sharply increased in the last years. According to the "Facts and Figures" from the International Telecommunication Union (ITU) [1], more than 50% of world's population is connected in 2020, compared to less than 30% ten years ago. In particular, 69% of people aged 15–24 have access to the Internet. The number of households with broadband access also doubled from 30% in 2010 to 57% in 2019. While the number of connected peoples grows, the bandwidth each individual requires has also augmented significantly. This evolution is natural given the advances in media technologies allowing cheap capture, processing and display of high-definition content (e.g. images, music, videos, games, ...). For instance, from Cisco's estimations [2], the number of 4K televisions doubled between 2018 and 2020. This trend is set to continue as more and more devices are also being connected to Internet over time: predictions show that more than 13 billion devices may be connected in

2023, including bandwidth-eager devices such as autonomous vehicles or cloud-based surveillance cameras. The Covid-19 crisis further amplified broadband traffic demand due to regional lockdowns and wider interest for teleworking. (As an example, Italy has seen 44% more traffic during the 2020 lockdowns than during the same period in 2019 [1].)

To cope with this rising demand, Internet operators have invested to expand their network: from 2019 to mid-2020, 200 Tbit/sec of additional capacity were globally added according to the ITU, up to an estimated total of 700 Tbit/sec. Since 2018, the global average broadband (resp. mobile) bandwidth of end user connections has climbed at a rate of 20% (resp. 27%) per year, and will probably reach 110 Mbit/sec by 2023 (resp. 44 Mbit/sec) [2]. While this means that more intercontinental cables, local fibers and antennas are being deployed, the strategy used by operators to manage their networks also needed to evolve. One important change is the shift to Software-Defined Networking (SDN), where the network control plane is decoupled from the forwarding (data) plane: this paradigm allows for more adaptability to the dynamic demands of their customers. In [2], 40% of surveyed operators have already switched to such networks, with 55% more operators expecting a deployment within two years. SDN allows in theory to automate most traffic engineering tasks, yielding optimized network configurations, routing and peering policies between operators. In 5G networks (SDN-enabled by design), operators are incentivized to deploy network functions "near the edge" to enable new use cases such as cloud gaming or large-scale telemetry analytics. In practice, this means deploying significant computing resources in datacenters or even antennas near end users. Internet services and applications must also adapt their infrastructure to handle the demand, by deploying distributed systems at multiple world locations for instance. Again, despite a wide range of cloud-services offers, these distributed systems naturally add complexity and new dependencies into the equation. As an example, Netflix has deployed delivery servers in more than 500 locations to cope with the worldwide demand of video streaming [3], allowing the service to become the largest provider of content in France in 2020 [4], ahead of Google and its Youtube video service.

All this added complexity may however backfire and lead to severe outages. On July 17, 2020, while attempting to solve a relatively benign network congestion problem, a Cloudflare engineering team updated a router configuration. However, this configuration change contained an error that redirected the entire Cloudflare traffic through a single network location: this led to the global outage of most of the 12 million websites served by the company for about 30 minutes [5]. Automated systems also may fail spectacularly: on August 30, 2020, CenturyLink/Level3 (one of the biggest network operator according to CAIDA [6]) triggered a

small change to protect a customer from a cyber attack. A chain of automated events led CenturyLink/Level3 to announce a huge number of routing changes to other network operators, shutting down a huge portion of the Internet during almost five hours [7]. In these examples, the relations between the initial root cause and the observed symptoms were not obvious—even to insiders—due to the many dependencies between network components. For external observers such as researchers or end users, troubleshooting and understanding Internet outages is even more complex, as most of these dependencies are hidden (for technical or business reasons). While some public information is available through Internet Exchange Points (IXPs), most of the Internet topology and relations between Internet stakeholders remain difficult to infer [8].

1.2 Data-driven approaches for network modeling

Automation, optimization and decision making software programs are widely used by network operators in an attempt to provide a good quality of service for their customers and reduce operational costs. Historically based on static decision rules designed by experts, these programs may struggle to understand the complex dependencies of modern computer networks. There is however a growing interest for *statistical* and *data-driven* approaches for network design and operation: *machine learning* algorithms leverage the large amounts of data collected by operators to produce relevant *models* of their networks. Yet, applications of machine learning techniques to computer networks and Internet remains difficult in practice because of the *large scale* of the managed networks. These networks are indeed difficult to instrument in order to obtain continuous measurements and up-to-date information. Moreover, with the recent shift to *fog*- and *edge-computing*, more management decisions are automatically taken by equipments with a *limited* knowledge of the full network.

Over the last decade, advances in computational efficiency enabled breakthrough solutions with *neural network* and *deep learning* models. One very popular example is Google's AlphaGo, the first software able to beat professional Go players [9] (2016). Other examples include a wide range of domains, from the historical topic of image analysis to multimedia content synthesis, data-driven translation models or cyber-physical systems analysis for instance. Thanks to their expressivity, these solutions have proven to be better at modeling complex systems and dependencies than most traditional approaches. In particular, given enough data, deep learning models are capable to extract ("learn") complex hidden relationships in the input data, leading to more accurate and generalizable solutions.

In this thesis, we argue that statistical learning can help modeling large computer networks with complex interdependencies, even with limited information about the evaluated networks. We demonstrate this claim by applying recent advances in image and graph processing techniques to computer networks, with data collected from both *simulated* and *real* networks.

Our findings advocate for the integration of more data-driven techniques in the design and management of computer networks. While these networks become increasingly complex and difficult to manage, these techniques hold the potential to improve network performances while reducing downtime risks and operational costs, fundamental metrics for network operators. On the other end, our contributions also show that external observers may use statistical learning to study the characteristics of a computer network despite having little prior information about it. This is very important for a number of actors, including consumers willing to evaluate network operators or third-party research campaigns. While we focus this thesis on computer networks, we believe most contributions can be applied to other domains of interdependent systems with limited changes. For instance, we believe our techniques are likely to be applicable to any cyber-physical system (e.g. smart grids or industrial control systems), social and transport networks or chemical and biological processes.

1.3 Research challenges and findings

In the remainder of this thesis, we study different problems linked to the claim we made in the last section. More specifically, we mainly differentiate problems according to the *amount of available information* usable for statistical learning. Indeed, solutions to the considered problems are actually very different when one has access to the full network topology and internal components' characteristics, compared to when only limited measurements are possible, and the network is considered as a "black box". This differentiation more or less maps to the disparity of knowledge between a network operator (having a relatively good overview of its network) and external observers. As a consequence, the challenges we consider are different according to the amount of available knowledge:

— In a first challenge (Subsection 1.3.1), we consider the case of a network operator willing to estimate network *performance metrics*, knowing the network topology and the characteristics of network components; these topologies and characteristics might be currently deployed or might concern a planned configuration change;

— In Subsection 1.3.2 and Subsection 1.3.3, we ask if the root cause(s) of observed failures can be pinpointed with statistical learning and limited information about a network. A solution to this challenge would allow all Internet stakeholders to *understand* and *troubleshoot* failures, something very difficult today with the lack of available information.

Statistical models are usually constructed *based on* a specific environment (i.e. a computer network), *for* that same environment. However, practical solutions to the aforementioned challenges should remain *generalizable* to any computer network of any size. By proposing extensible and generalizable models *by design*, we ensure that our contributions comply with this requirement.

1.3.1 Network modeling for fast performance prediction

Due to the complex interdependencies between the components of a computer network, it is usually difficult to *predict* how a given network configuration will perfom in practice. This is a major problem for network operators willing to optimize their configurations without introducing problems: small changes in one part of the network could lead to drastic performance changes in other apparently unrelated parts. Historically, operators have used expert knowledge and simulations, but these approaches may be inaccurate and slow. In the context of large-scale dynamic networks, new robust and efficient solutions are needed today.

The Knowledge-Defined Networking (KDN) paradigm [10] has been proposed in this regard. It proposes to leverage the combined capabilities of SDN (facilitated metric collection and network management) and machine learning to *design* and *control* large and complex computer networks efficiently. In KDN, networks need to be modeled from the known network configuration so that automated decisions can be applied. One goal of this modeling is to *predict the network performance given configuration changes* in order to make the best decisions. Inspired by a previous work based on Graph Neural Networks (GNNs) [11], we propose in this thesis to explicitly model the dependencies between network components in multiple bipartite graphs. We apply recent GNN techniques on this dependency model and obtain predictions with very good accuracy for large networks in a few hundreds of milliseconds (by contrast, simulation tools required several *minutes* for the same predictions). In particular, our contribution can handle network components with various *scheduling policies*. We show that it can be *generalized* to different network topologies and configurations and explore its inner workings with visualizations of learned representations.

This work has been performed in the context of a machine learning challenge co-organized by the ITU and the Barcelona Networking Center [12]. We were honored to win this challenge and have published our approach in the following report (at the time of writing, additional submissions are being prepared for SIGCOMM CCR and NeurIPS, see page 149):

> Loïck Bonniot, Christoph Neumann, François Schnitzler, and François Taïani. *Graph Neural Networking Challenge 2020 - Steredeg's solution.* 2020. URL: https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-014-Steredeg/blob/main/report.pdf

1.3.2 Root Cause Analysis from the edge

Broadband and mobile Internet users regularly encounter performance degradations and connectivity issues when accessing services over the Internet. During these incidents, it is difficult for end users to pinpoint the *root causes* behind the observed symptoms: is it due to bad wireless coverage? Service downtime? Device misconfiguration? Internet Service Providers (ISPs) are naturally blamed by users for many perceived degradations, since they are responsible for delivering good Internet connectivity. As a result, their customer support teams are often called upon for symptoms *not actually caused by the ISP*. This leads to additional support costs and damaged customer relationship: TV and Telecom customer services were ranked among the worst in US in 2018 [14].

End users and ISPs would both benefit from an *automated Root Cause Analysis (RCA)* solution, able to pinpoint the most probable origins of failures. Customers would be able to pinpoint failures quickly while avoiding most time-consuming phone calls, leading to greater satisfaction (75% of US citizens from youngest generations prefer written communications in customer service relations [15]); conversely, ISPs would only receive reports for root causes likely to lie within their area of responsibility. Impacted third-party Internet services would also benefit from better customer experience and automatic reports, as even small performance degradations may lead to lower revenues [16, 17].

In this context, the available knowledge is drastically reduced compared to the previous problem (Subsection 1.3.1): when doing RCA from the point of view of one user, it is not possible to have access to the full topology and dependencies of Internet services. To solve this problem, we propose to leverage a combination of crowdsourced measurements and the usage of reference ("landmarks") measurement servers deployed in many Internet locations: by collecting metrics from these vantage points, we show that it is possible to pinpoint many root causes, even without ISP collaboration. We leverage a collected dataset with controlled

failures to build root cause models, and show that a Convolutional Neural Network (CNN) outperforms two alternative solutions derived from typical machine learning algorithms. In particular, our CNN models are *extensible* and support *dynamic* landmark servers by design. This work has been presented in the following paper:

> Loïck Bonniot, Christoph Neumann, and François Taïani. « Towards Internet-Scale Convolutional Root-Cause Analysis with DiagNet ». In: *35th IEEE International Parallel & Distributed Processing Symposium*. 2021. DOI: 10.1109/IPDPS49936.2021.00084. URL: https://hal.archives-ouvertes.fr/hal-02534888

1.3.3 Leveraging web browsers for network metrics collection

The third contribution we present in this thesis is related to the *collection* of metrics relevant for RCA. In this last work, we highlight the need for Quality of Experience (QoE) metrics in a practical RCA framework, and promote *the instrumentation of the execution environment of end user devices* in this regard. More specifically, we argue that web browsers are good platforms for metric collection that can provide accurate estimates of both QoE and network metrics (Quality of Service (QoS) metrics). We propose a set of methods based on browser capabilities, taking into account the security restrictions of modern web browsers.

During more than a year, we deployed and collected metrics using our proposed framework and volunteer end users. We highlighted several interesting findings from the collected dataset, including QoE variations over time, traffic differentiation pinpointing or important Internet routing changes. We also propose a *statistical* approach to pinpoint *faults* and related *root causes* that we observed in our dataset. This effort of data collection is necessary for any data-driven solution: it allowed us to evaluate the RCA techniques we proposed (Subsection 1.3.2) over the Internet with *real* faults and QoE degradations. As such, because our dataset is significantly different from the previous dataset we used with controlled faults, we obtain different results showing that typical machine learning baselines remain interesting for RCA problem. While the overall performance is lower, our study still demonstrates the interest of statistical learning for RCA and raises a number of interesting questions and research avenues for future work in this area. A description of our methodology and early findings from our dataset are presented in the following paper:

> Loïck Bonniot, Christoph Neumann, and François Taïani. « DiagSys: Network and Third-Party Web-Service Monitoring from the Browser's Perspective (Industry Track) ». In: *Proceedings of the 21st International Middleware Conference Industrial*

Track. Association for Computing Machinery, 2020, pp. 16–22. ISBN: 978-1-4503-8201-4. DOI: 10.1145/3429357.3430520. URL: https://hal.archives-ouvertes.fr/hal-02967290

1.4 Thesis outline

This thesis follows the synopsis we described in the previous section with each chapter being written to be self-contained.

— We begin in Chapter 2 by introducing some important definitions about *computer networks*, followed by additional background about network modeling and RCA. In Section 2.4, we also expose some basic notions behind machine learning, including recent neural network developments such as Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs).

— In Chapter 3, we detail the application of generalizable GNNs for performance prediction over network paths. This chapter includes the details of our results over the GNN challenge co-organized by the ITU and the Barcelona Neural Networking Center.

— Chapter 4 presents our second contribution, DiagNet, an extensible neural network model for RCA from end user devices. In particular, we show how DiagNet performs compared to two other models derived from common statistical learning approaches (i.e. an extensible Naive Bayes and an extensible Random Forest classifiers). For our experiment in this chapter, we leverage a dataset collected over a few weeks with injected, controlled faults.

— In an attempt to evaluate how DiagNet performs with *real* Internet faults, we designed and deployed a full measurement platform during more than a year. We present this platform, DiagSys, in Chapter 5. This last contribution shows that building systems with real faults is significantly more challenging than with controlled ones. Yet, we managed to build data-driven RCA models from this dataset, and present our results in this chapter.

— Finally, we conclude this thesis in Chapter 6, where we also draw perspectives and promising ideas for network modeling and RCA services in the next few years.

Chapter 2

Background

Contents

2.1 Definitions . . . 19
2.2 Network modeling . . . 22
2.2.1 Modeling from complete network knowledge . . . 22
2.2.2 Modeling from limited knowledge . . . 24
2.3 Root cause analysis . . . 27
2.3.1 RCA from complete network knowledge . . . 28
2.3.2 RCA from limited knowledge . . . 30
2.4 Machine learning . . . 33
2.4.1 Methodology . . . 33
2.4.2 Naive Bayes classifiers . . . 35
2.4.3 Decision tree learning . . . 36
2.4.4 Ensemble learning . . . 37
2.4.5 Artificial Neural Networks . . . 37
2.5 Conclusion . . . 43

In this chapter, we provide the necessary background needed to understand the subsequent chapters, along with relevant related work. We begin by defining the important concepts used in this thesis (Section 2.1), in particular how we define a *computer network*. We then introduce the *network modeling problem* in Section 2.2. Following the intuition of our introduction (distinguishing approaches depending on full versus limited network knowledge), we propose two different variations of this problem. The first variation corresponds to the *design time* of a network, when one has access to most low-level characteristics of components: in this case, the goal is usually to predict the *behavior* of the network and answer questions such as "What are the expected delays between components?", "Is there any saturated component?" or "What can we do to improve the network?". The second variation is more suited for *operation time* and considers an opposite strategy. From *global* or *macroscopic* measurements and without any prior fine-grained knowledge of the inner-working of the network, can we infer low-level properties of network components? These properties are usually unavailable during

operations, either because they are too costly to obtain or because one party cannot access them for business reasons. Yet, they are critical for a number of use cases.

We explore one of the aforementioned use cases, *root cause analysis*, in Section 2.3. Once a failure or problem has been detected, the goal of root cause analysis is to find the component(s) responsible for the failure. Again, we explore two variations: (1) when one has access to insightful knowledge about the network (e.g. its topology and low-level properties) and (2) when information can only be scarcely obtained, requiring approximations and statistics.

Several contributions of this thesis are based on *machine learning* (Chapters 3 and 4). We provide the intuition and the basic concepts of machine learning in Section 2.4 to help our readers fully grasp these contributions. In particular, we detail the standard *methodology* behind machine learning solutions: dataset collection, model design, training and evaluation. Finally, we present popular models ranging from simple statistical approaches and decision trees to more complex neural network models.

2.1 Definitions

We begin by defining important networking terms and concepts that we will use throughout this thesis.

Computer networks This thesis focuses on computer networks, as formally defined by Andrew S. Tanenbaum in his book "Computer Networks" [20]: "a collection of autonomous computers interconnected by a single technology". More specifically, we are interested in the study of *networks of networks*, called *internetworks* (one obvious example being the Internet). For simplicity, we define every computer within a network as a **node**, whether personal computer, mobile phone, wearable device, dedicated server, smart car, … Two nodes connected by a **link** can communicate by exchanging **messages** using a shared technology (such as Bluetooth Low Energy (BLE), or Internet Protocol (IP) over Ethernet or Wi-Fi [21]). Finally, a **path** is defined as an ordered list of nodes and links and describes a *route* between two nodes that may not be directly connected.

Nodes, links and paths form the basic **components** of every computer network (Figure 2.1). Thanks to their genericity, these concepts can be easily mapped to other kinds of networks (e.g. transportation, social, biological networks).

Useful metrics In practice, computer network links are characterized by a **latency**, a **jitter**, a **loss rate** and a **capacity**. The latency is the average time needed for a message to pass through the link, while the jitter is the average of the *deviation* from that latency. The ratio of lost messages is defined by the loss rate. Finally, the capacity defines the quantity of information that can be transmitted through the link per unit of time: it is usually measured in bits per second. The capacity must not be confused with the *available bandwidth*, the remaining link capacity when a link is already being used. It is important to note that one cannot usually measure *directly* this capacity: we instead measure the *throughput* or "*goodput*" of a

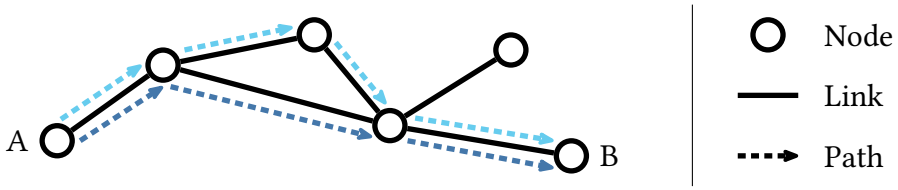


Figure 2.1 – Illustration of a computer network with two possible paths from node A to B.

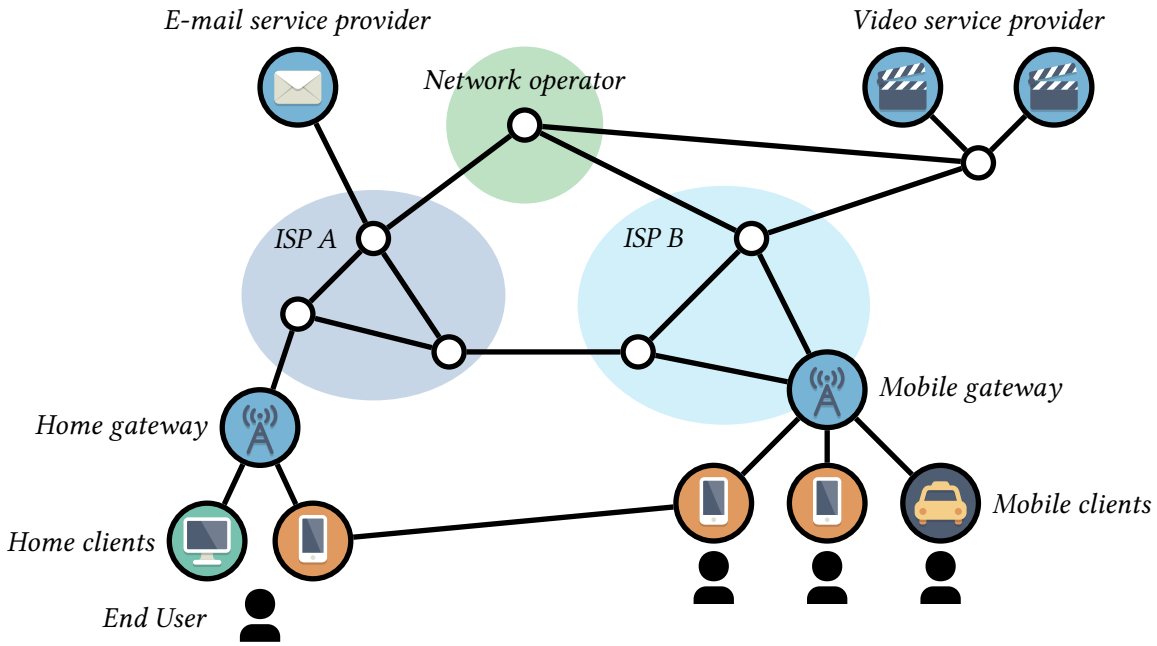


Figure 2.2 – Illustration of roles in a toy Internet-like computer network. End users can use their clients to access e-mail and video services via the gateway provided by their ISPs. Note that two clients can directly communicate, e.g. with BLE.

specific application, that may be lower than the capacity (e.g. due to lower available bandwidth, protocol overheads or retransmissions) and sometimes *higher* (e.g. due to compression and buffering). Similarly, the latency is mainly estimated using Round Trip Time (RTT), a simpler approach than One-Way Time that requires precise clock synchronization. Nodes can be characterized by the additional latency they add when processing messages (modeled by queue or buffer parameters).

Roles of network stakeholders The solutions to computer network problems highly depend on the *assumptions* made about the targeted actors of the networks. Is one solution designed for *network administrators* and needs complete knowledge about network components? Or is it designed to work with little knowledge, thus targeting *customers* of a network?

In order to compare the assumptions made in our contributions to previous work, we define a *generic* set of *roles* based on the Internet architecture [20]. (See Figure 2.2 for an illustration in a toy network topology.) First of all, each **network operator** is responsible for the maintenance of a *part* of a network. We assume that network operators have complete knowledge of nodes and links properties in their area of responsibility. In Internet, we can roughly map

network operators to Autonomous Systems (ASs) [22]. **Internet Service Providers (ISPs)**, are a particular kind of operator: they allow their **customers** to access the global network. We make the distinction between the following two types of customers:

— **Service Providers** deploy computers to the network that offer a dedicated *service*, such as audio or video streaming, file download, e-commerce websites, etc. The deployed computers are usually called "**servers**".

— **End Users**, on the other end, use their computers to access the aforementioned services or communicate with other end users. Their computers are called "**clients**", as opposed to servers (not to be confused with the "customer" term).

Keep in mind that this model is an attempt to *simplify* and *generalize* internetworks, complex and decentralized systems. For instance, large service providers may act as network operators to optimize their costs and quality of service. Conversely, ISPs usually provide other useful services such as mailbox or website hosting. Yet we believe our role classification fit with existing network models while allowing easy comparison of solutions.

2.2 Network modeling

Modeling a computer network is essential to *understand* and *predict* its performances [23, 24, 25]. There are many ways to accomplish this task and we are going to present existing methods in two subsections. In a first subsection, we assume that we solve the problem for a *network operator* with full knowledge about the network we want to model. In this case, modeling methods are typically used during the *design* of the network. By contrast, the second subsection discusses methods working with *limited knowledge*: these methods are best suited during the operation of a network, or when we solve the problem for *non-operators* (e.g. end users).

2.2.1 Modeling from complete network knowledge

We present methods for network modeling when the network *topology* and most component characteristics are known, that can be classified in four broad categories: queuing theory, petri nets, discrete events simulation and network emulation. We discuss each in turn in the following.

Queuing Theory The first model of a telecommunication network with *queues* was proposed by Agner K. Erlang in 1917 [26, 27] to estimate average waiting times in automated telephone exchanges. Queuing theory makes it possible to analytically predict expected queue behavior according to the arrival rate of "customers", the distribution of service time and the number of "servers". Multiple queues can be *linked* together in *queuing networks* [28]. This theoretical framework has been mapped to computer networks by Leonard Kleinrock [29], leading to the development of ARPANET (Internet's ancestor). However, it has been recognized that analytical computations become quickly intractable, especially for queuing policies more complex than "first-in, first-out". In particular, most queuing networks assume *independence* between the arrival rates of queues, an assumption that does *not* hold in practice [30]. Recent works rely on rate correlation and approximation to analyze relatively small queuing networks [31, 32, 33, 34].

Petri nets Introduced in 1962, Petri nets [35] augment automata and queuing theories by modeling *dependencies* explicitly. They are defined by a graph with two types of vertices: *places* and *transitions*. The important constraint is that vertices' types *alternate* in a bipartite graph: no place (resp. transition) is linked directly to another place (resp. transition). Places

can hold any number of *tokens* (e.g. communication packets). After being defined, a Petri net can be executed by firing transitions, allowing tokens to *flow* within the network.

As defined by Carl Petri, these networks cannot accurately model queuing and delay present in computer networks. A large body of works have therefore studied *timed* Petri nets [36], a variant better suited in this case, with many successful applications [37]. As an additional variant, *queuing* Petri nets [38] were proposed to model queuing delays of real networks [39]. Ultimately, these extended Petri nets can be translated to *Markov chains*, allowing the analytical analysis of network properties similarly to queuing networks, with similar tractability limitations.

Discrete events simulation We have seen that realistic computer networks cannot be analytically solved because of the intractable number of states they may exhibit. Yet, one can resort to *simulation* to predict network performance. In particular, discrete event simulation [40] is a practical approach taken by the most popular computer network simulators, including NS-3 [41] and OMNeT++ [42] among others [25]. Within network simulators, events such as "message received at node x" are processed one after another: the simulator *jumps* from events to events since nothing is supposed to happen between consecutive events. Concurrent events are avoided by using stochastic durations, and it becomes possible to measure the performance of a computer network at any point in time, even for complex networks. For instance, NS-3 provides models for Linux networking stacks and for wireless communication such as Wi-Fi[21]. Despite extensive optimization, simulators remain however limited by the underlying computing power available; the cost of simulation increases with the size of the evaluated network and the number of events, i.e. the number of exchanged messages. Care should further be taken when setting simulation parameters, as small errors can lead to unrealistic results [43]. As we shall discuss in Chapter 3, OMNeT++ has for instance been used to create a dataset of computer network performances given many different configurations. While using this simulator is slow in practice (several minutes of computation are needed per configuration), the produced dataset can be extensively used to design and validate other data-driven approaches.

Emulation A more recent alternative to simulation is *emulation*: network stacks of *real* computers are used to exchange *real* messages. The challenge in emulation is to *replicate* a network model by emulating node and link characteristics. As an example, the Linux kernel provides `netem`, a set of tools to add artificial latency, loss or bandwidth shaping to received

or sent messages. The MiniNet emulator [44] leverages `netem` and *process namespaces* to emulate an arbitrary computer network on a single host. Recent systems such as Kollaps [45] or AdvantEdge [46] can build networks spanning multiple hosts thanks to container technologies and Kubernetes instrumentation [47]. Wireless emulation is possible using dedicated testbeds [48], although being less practical than simulation in most cases.

Emulation presents many benefits: it makes it possible to evaluate existing applications in an environment close to that of a real network with low overhead, reduces computation costs significantly and provides an acceptable level of reproducibility [49, 50]. However, some limitations remain [51]. First, the emulated network is limited by the capacity of the *underlying physical network* between emulation computers—e.g. it is not possible to emulate a link with 1 msec of latency over a physical link with 5 msec of latency. Furthermore, concurrent flows of messages may be problematic when modeling limited link capacities.

More recently, several works have proposed to model computer networks using machine learning techniques and quite notably neural networks [11]. We return to this field in Subsection 2.4.5.

2.2.2 Modeling from limited knowledge

During the operation of a network, the available knowledge about a computer network can be a lot scarcer than during its design. This may be due to operational constraints (e.g. inability to monitor *every* link of a network) or to the *interplay* between the different areas of responsibility. In Internet, no ISP has a *complete* knowledge of every component of the whole network; even managing all characteristics of components in their own ASs is known to be a difficult problem [52]. This is even more complex for *end users* who can only observe behaviors from the far *edge* of the computer network. We now review several works that attempt to *discover* more information about a network and *infer* components' characteristics from global observations.

Network discovery using probes One frequent issue when dealing with large networks is the absence of a *known topology*. The main solution to gather information about a network topology is to send *probes* to different nodes and analyze the *responses* to these probes [53]. The `traceroute` tool for instance is widely used to explore IP networks, leveraging packets' Time To Live (TTL). When traversing intermediate nodes on a path from the traceroute's source to its destination, the TTL header of an IP packet is decreased by one. If it reaches zero, the node

is expected to send a special Internet Control Messaging Protocol (ICMP) "TTL exceeded" error back to the initial packet sender. The idea of `traceroute` is to send successive packets while increasing the TTL: in theory, the source will receive one ICMP error per intermediate node, allowing readers to identify links within the probed path. This technique has been exploited for full topology inference [54], but its accuracy may suffer from missing responses and voluntary obfuscation from intermediate nodes [55, 56]. Such methods remain an active area of research, recent works have for instance proposed to exploit parallel probes and stochastic sampling to considerably improve the efficiency of traceroute measures [8].

Service dependencies discovery Nodes in a computation network rely on other nodes (servers) providing *services*. As such, it is very important for a network operator to understand the *relationships* (dependencies) between nodes so the operator can optimize the topology and characteristics of its network. Another use case for dependency discovery is the *root cause analysis of failures*, that consists in determining the origin(s) of encountered faults (we return to this use case in Section 2.3). This is not a trivial problem, and several methods have been proposed (Table 2.1), using messages' metadata [57, 58, 60], packet traces [59, 61, 62] or controlled perturbations [63]. As an example, Sherlock [59] collects packet traces from intermediate nodes and estimates the *conditional probability* of accessing one service A (dependency) a short time before serving B (dependent). After basic filtering, this conditional probability yields the *dependency probability* between A and B, and is later used for root cause analysis.

Table 2.1 – Selected methods for services dependency inference.

Reference	Year	Short description
Pinpoint [57]	2002	J2EE communication layer to add tracing in queries.
Tulip [58]	2003	Tracing log written in packets by traversed Internet routers.
Sherlock [59]	2007	Packet traces are extracted from routers and later correlated.
X-Trace [60]	2007	Tasks tagged using metadata.
Orion [61]	2008	Correlation of delays extracted from packet traces.
NetMedic [62]	2009	Instrumented Windows sockets and firewall.
WebProphet [63]	2010	Perturbated queries using a controlled proxy.

Network tomography Moshe Y. Vardi formalized the problem of network tomography in 1996 [64, 65]. The goal in tomography is to estimate a set of parameters $\mathbf{x}$ from a set of observations $\mathbf{y}$, knowing a routing matrix $\mathbf{A}$ in the following equation:

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x} \tag{2.1}$$

For instance, in the "Origin-Destination" estimation problem [64], one estimates the traffic for every path in the network ($\mathbf{x}$) (every origin-destination pair) from reports of traffic in every link $\mathbf{y}$. This is essentially an *inverse problem* that can be solved using iterative algorithms such as least squares solvers. Another problem is the estimation of link-level characteristics such as latency and loss ($\mathbf{x}$) from end-to-end path measurements ($\mathbf{y}$). Multicast probes can be used to solve this problem [66], but they are usually not usable in most real computer networks where unicast probes are preferred [67, 68]. In topology identification, the $\mathbf{A}$ routing matrix can also be inferred from a *forest of possible topologies* [69, 70].

However, in most cases $\mathbf{A}$ is rank-deficient, leading to multiple solutions [65, 71]. One way to ease the resolution of this kind of equations is to identify which network characteristics can be estimated ("identified") and place monitors accordingly within a computer network [72, 73, 74]. Other approaches leverage additional statistical knowledge from links [67, 69, 75, 76]. More recently, several methods have been proposed to take into account the effects of *failures* during tomography [77, 78, 79].

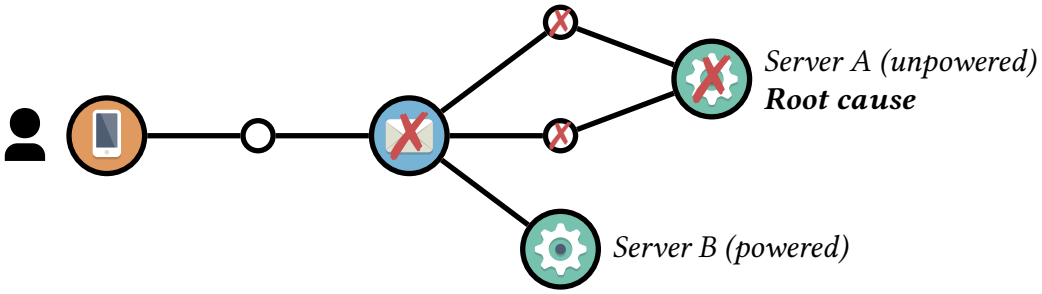


Figure 2.3 – Root Cause Analysis (RCA) example: one end user is unable to access their email service due to the power failure of "Server A" (the root cause). Other components relying on this server appear faulty (e.g. the email service).

2.3 Root cause analysis

One famous use case of the models we described in the previous section is *the ability to understand failures*. More specifically, we focus our research on the localization of the *root causes* of failures. We believe failure detection and RCA are two distinct problems with different solutions: detection methods leverage outlier detection to flag anomalies, while RCA tries to find the *anomalies' origin(s)* after their detection.

Figure 2.3 illustrates the challenge of RCA in a simple setup. In this example, one user encounters fatal errors when trying to access their email. This user may assume that the whole email service is faulty, or that the links to the service are not working properly. An ideal RCA system would pinpoint the *origin* of the failure (usually termed the "fault" [80]). Here, the origin is a power outage of a critical backend server, making the email service unavailable via a *chain of dependencies*. By contrast, fault detectors would highlight problems in all "crossed" components (✗), making troubleshooting more difficult.

Obtaining the data in practice The type of RCA methods that can be applied largely depends on the type and amount of information that is available; this in turn depends on how the data and knowledge is obtained in practice. ISPs can use their internal knowledge [81] or SDN metrics [82, 83, 84] to correctly build dependency graphs between network components. In most datacenters, the computer network topology is redundant and hierarchical in Clos-like schemes (e.g. in Facebook [85] or Azure [86], servers are interconnected in a 3-layers scheme with redundant links between layers). This choice allows further optimizations [86, 87, 88] and distribution of the RCA between network components. For instance, routers can *vote* for suspected faulty components in their *rack* [85, 89]. Without peeking inside ASs or datacenters,

end users can still extract *some* network knowledge from publicly-available data. Traceroutes, public Border Gateway Protocol (BGP) feeds and IXP databases have been successfully used for outage detection and localization in [90, 91]. Passive traffic monitoring is also possible using Internet background radiation [92]: variations in the traffic coming to unallocated IP prefixes can hint at serious global outages. Yet, with the scarcity of data regarding internal topologies and ASs peering, greater collaboration between end users and ISPs is required for fine-grained RCA [93].

We now review the available RCA solutions according to the amount of available network knowledge.

2.3.1 RCA from complete network knowledge

Many RCA methods (Table 2.2) require extensive knowledge about the analyzed computer network. Early attempts proposed to use *binary* network tomography: with this approach, a network component is either *nominal* or *faulty*. The goal is then to infer which components are *faulty* from end-to-end measurements on network paths [98, 99]. This is equivalent to solving a *Minimum hitting set* problem, which is known to be NP-hard. Heuristics are thereby used to obtain a result in practical time [95, 100, 101, 102, 103, 104]. As an example, the authors of Tomo [95] propose an iterative greedy algorithm: as long as there exist *unexplained* failures, Tomo computes a score for every link l as *the number of unexplained failures that can be explained by* l *being faulty*. It then selects the link having the highest score and loops again. This simple approach fails with dynamic networks and transient failures, making it difficult to use for RCA in practice. Later works model *probabilities of failure* instead of binary values [105, 106].

In the same vein, Bayesian networks have been used for RCA [59, 94, 97, 107, 108, 109]. These networks—sometimes known as *belief* networks—are designed to model the *conditional probabilities* of dependent random variables in a directed acyclic graph. However, exact inference in Bayesian networks is also a NP-hard problem [107]: methods differ in the *modeling* of the computer network to the Bayesian network and in the approximations they make. In a first example, SCORE [96, 110] models the *observations* and *possible causes* in a bipartite graph. It then uses Occam's Razor to select the simplest explanation from the set of possible causes, using the same score computation as Tomo [95]. In another example, Sherlock (presented in Section 2.2) creates a detailed Bayesian network from detected dependencies [59]. It then tests *every* possible root cause by computing a plausibility score, while assuming that at most k components are faulty. This assumption is quite reasonable for $k \leq 3$ and reduces

Table 2.2 – Summary of RCA methods based on dependency modeling. The presented methods are designed for different scales (− small network, + enterprise, ++ datacenter/ISP, + + + Internet) and propose several exploration and ranking strategies. Sorted by date.

Method	**Scale**	**Model**	**Exploration**	**Ranking**
Shrink [94]	++	Bipartite	Complete (3 causes max)	Most probable
Tomo / NetDiagnoser [95]	++	Bipartite	Greedy while unexplained failures remain	Maximizes intersection with unexplained failures
Sherlock [59]	−	Multi level	Complete (3 causes max)	Most probable
NetMedic [62]	−	Multi level	N/A	Abnormality from known states
SCORE [96]	++	Bipartite	Greedy while unexplained failures remain	Prefers simpler explanations
Gestalt [97]	+	3 levels	Partial (3 causes max)	Most probable
Facebook [85]	++	Topology-based	N/A	Outlier detection
Kepler [90]	+ + +	BGP / IXP	N/A	Fraction of unreachable paths
007 [89]	++	Traceroute	Greedy with score threshold	Number of votes per link
SDNProbe [83]	++	Bipartite	Hopcroft-Karp	Hypothesis testing
Deepview [86]	++	Bipartite	Lasso Regression	Hypothesis testing
Zeno [84]	+	Provenance	N/A	Hypothesis testing

the complexity for n components from $O(2^n)$ to $O(n^k)$. Finally, Gestalt [97] is a hybrid solution that classifies and builds on previous work: it optimizes Sherlock by ignoring non-likely combinations of failures, giving better results in a fraction of the complete exploration time.

2.3.2 RCA from limited knowledge

Network operators and end users may not have access to the wealth of information required by the methods we described in previous subsection. However, they may want to understand *why* the computer network *"does not work"* on their end, and if they can do something to fix the problem on their own. Internet speed checks are the go-to services many end users visit when looking for root causes: these services allow to quickly test connectivity towards *measurement servers*, returning average RTT and throughput. (Speedtest [111] alone announced more than 800 million tests from 220 million unique devices in 2020 [112].) We now present two families of methods in this space, the first one being based on *predefined rules*, and the second on *crowdsourcing*.

Diagnosing using predefined rules This is the typical approach to pinpoint failures causes. The intuition is that *automated tests* are run in sequence, each test being associated with a *potential root cause*. One failed test (or failed group of tests) may point the end user to the root cause explaining the fault. We say that the tests follow *predefined rules*: these rules are usually determined by experts in networking [113, 114, 115, 116], helped by machine learning algorithms in some proposals [117, 118, 119, 120, 121].

Most ISPs provide methods for self-diagnostic, either via a dedicated service in their home gateway, or as a step-by-step documentation on their websites. At the time of writing, the French ISP Bouygues Telecom provides a set of "Diagnostic" pages in its fiber gateway's web server, with live information about traffic, ICMP, Transport Control Protocol (TCP) and Domain Name Service (DNS) latencies. However, these services are best-suited for tech-savvy end users who know *where* to find them and *how to interpret the results*; and only target frequent problems and misconfiguration scenarios in *home* networks. In the literature, Netalyzr [114], Fathom [116] and FireLog [122] proposed diagnostic tools that also exploit rules and are based on web browser extensions. Web browsers can run on most kinds of computers (including mobile platforms), allowing most end users to run reliable RCA. However, modern browsers show technical and security limitations that prevent the execution of some predefined rules (such as measuring *low-level* statistics of network communications). We revisit this point in Chapter 5, where we propose a measurement platform tailored for modern web browsers.

Pooling knowledge from a crowd Another approach for root cause analysis is to pool the data collected from *multiple* vantage points [123] and find discrepancies using statistical methods. This is the basic idea behind the DownDetector website [124], that monitors social media (the "crowd") for complaints about web services. When a threshold of complaints is reached in a given time window, a service is marked as "degraded" or "down". One of the earliest proposal, PeerPressure [125, 126], allows the exchange of *Windows registry configuration keys* between end users. It then estimates the probability that each configuration key is the root cause of a specific fault using a simple Bayesian estimator. Statistical aggregation can also be performed for computer networks' metrics [113, 127, 128, 129, 130, 131] to detect and pinpoint root causes. In this case, metrics are often exchanged between end users in the same *geographical* or *topological* regions using a central or distributed store [113, 132]. Popular in peer-to-peer communications, Distributed Hash Tables (DHTs) allow end users to access information indexed by "hashes" and hosted by other end users [133]. As an example, DYSWIS [129] publishes metrics at different granularities in a DHT, from home network-level to AS-level. This allows end users to compare their metrics to their neighbors' ones.

Historical data can also be used to get more knowledge about a specific fault [16, 62, 134, 135, 136, 137]. End users can regularly measure relevant metrics and build historical *time series* of these metrics. The assumption made is that if the *distribution* of a time series changes, then it is worth looking at the monitored metrics for potential root causes. This problem is framed as *change point detection and localization* and can be solved statistically with CUSUM (CUmulative SUM control chart) [138]. For instance, FChains [135] and Callegari et al. [136] use CUSUM to detect and localize failures in cloud systems and web browsers, respectively. In LOUD [137], dependencies between time series are inferred using the Granger causality test [139]; the authors demonstrated good localization of root causes by finding *central* nodes in the inferred dependency graph.

These *crowdsourcing* methods give coarser RCA than methods described in Subsection 2.3.1, but can easily scale to thousands and even millions of vantage points. They have been widely used to compare ISPs' performances [140] and detect large-scale outages [92], without the need for central services. However, they suffer from a number of important limitations. Their accuracy highly depends on the data shared by peers (or stored in local time series): if a peer is not identified correctly in the network topology, his data might be incorrectly analyzed [141]. Worse, peers could *alter* the shared data to make RCA useless, and could even spy on sensitive troubleshooting information [126]. Time series require regular measurements over time and are sensible to network dynamicity—this is particularly relevant for *mobile end users*.

The quest for ground truth Many of the presented approaches rely on available *ground truth* for some observed failures: these examples allow to design and evaluate a RCA method efficiently. This is especially true for *supervised machine learning*, as detailed in the next section. While ground truth is easy to obtain in most controlled environments (e.g. labs or datacenters networks), it is *much harder* to obtain reliable information at Internet scale: network operators often *hide* root causes for business or security reasons. Public status pages, network experts' mailing lists and social media sometimes contain relevant information, but collecting this data is often a manual and laborious task [90, 127].

2.4 Machine learning

We mentioned the interest for machine learning in network modeling and RCA. In this section, we describe the general *methodology* behind machine learning approaches [142] along with some practical examples. This short primer is intended to provide our readers with all the necessary tools to understand the contributions proposed in the next chapters.

2.4.1 Methodology

Machine learning denotes a broad range of techniques within the even wider field of Artificial Intelligence (AI). At a high level, a *machine learning algorithm* can be used to build predictive **models** related to some **dataset**. The goal of any said algorithm is to *automatically* improve the built models according to a **performance** metric. After the initial **training** phase (Figure 2.4), a model can be used to solve a task based on new data (this is the **inference** phase).

The importance of data As we just described, machine learning algorithms *need* data during both training and inference. We say that the data is stored as *samples* in a *dataset*, usually partitioned in the following sets (see Figure 2.4 for a visual representation):

- **Training set**: usually the largest, it is used during the training phase. In practice, each sample in this set is read and used by the algorithm in sequence or multiple times.
- **Validation set**: this set is used to confirm that the trained model is able to perform well on unknown data. It is possible to choose model hyperparameters such that the performance is the best on the validation set. If the model's performance is good on the training set, but poor on the validation set, we say that the model **overfits**: it becomes specialized on the training set and is not able to generalize to more data.
- **Test set** (also called **evaluation set**): after having chosen the best hyperparameters — thanks to the validation set—and trained a model accordingly, one can evaluate a model using the test set. Since this final set was *not* used during model design and training, it should best reflect the behavior of the final model with "fresh" data. In machine learning challenges, this set is *not revealed* to participants, such that it is possible to compare models objectively.

One common assumption is that all sets follow the *same* probability distribution, but this cannot be guaranteed in practice: newly-provided data may shift to values unexplored during

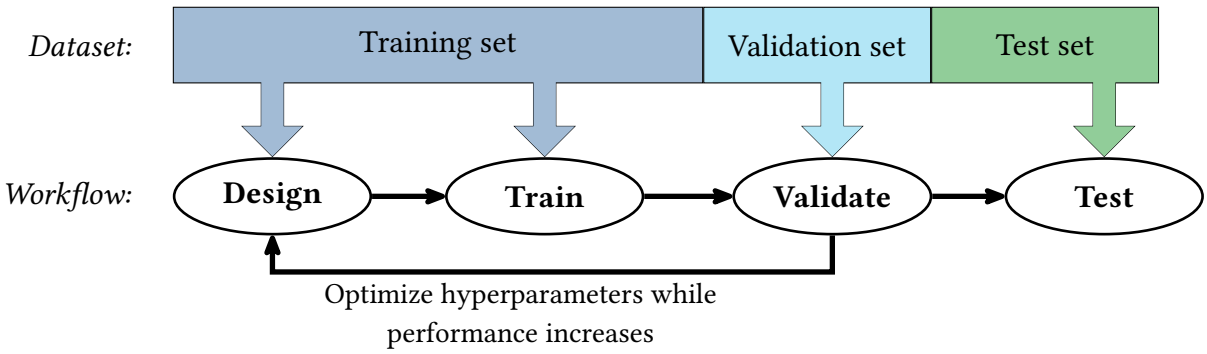


Figure 2.4 – High level overview of the iterative machine learning workflow. Each step must use the correct set of the input data to avoid overfitting.

training. Fortunately, recent algorithms are able to relax this assumption: we exploit this ability in Chapter 3, we train a machine learning model on two network topologies, then we validate and test this model on two *other, untrained* network topologies. We take a similar approach in Chapter 4, where we feed data coming from *vantage points unseen during training* in a network RCA model. These two examples clearly violate the aforementioned assumption, yet allow us to build more *generic* models. A word of caution before continuing: there has been much confusion between validation and test sets in the literature, probably because test data can easily *become* validation data if it is used during model design.

Defining model performance The choice of a specific performance metric for a machine learning algorithm is driven by the *task* the trained model is expected to solve. We distinguish two main classes of tasks: *supervised* and *unsupervised* learning. In supervised learning, each sample in the dataset is associated with a **label**. This label represents the *expected output* that the model should return when given the sample's inputs (also called **features**). The performance can then be defined from the average error between the *expected* label (from the dataset) and the *predicted* one (from the model's outputs).

— In regression tasks, the label is a numerical value: the task is to *predict* that value given some inputs (e.g. "predict the latency over a link given network characteristics"). Common error metrics include Mean Absolute Error (MAE) and Mean Square Error (MSE).

— In classification tasks, labels are categorical classes that the model is expected to predict for every sample (e.g. "predict if a given link is *nominal* (class 1) or *faulty* (class 2) given network characteristics"). In this case, the performance error is often measured by cross entropy, recall or precision metrics [142].

However, as we underline in Chapter 5, labels may be unavailable. It may be because it is too costly to label every sample, or because some critical information cannot be collected. This problem is tackled by unsupervised approaches, which seeks to identify *patterns* in the training data. Data clustering and anomaly detection (mentioned in Section 2.3) are two examples of unsupervised tasks. With no label available, it is difficult to assert the performance of unsupervised algorithms. In most cases, a small number of samples are labelled in the validation and test sets, allowing for empirical evaluation. This is often a manual process involving laborious information collection and cross-referencing, as underlined in Subsection 2.3.2 for RCA within computer networks.

We now provide some examples of machine learning algorithms that have been successfully applied to computer networks in recent studies.

2.4.2 Naive Bayes classifiers

Naive Bayes is one of the simplest family of classifiers, based on the Bayes theorem [143]. Each sample in the dataset is associated with features $\mathbf{x} = (x_i)_{i\in\{1,\dots,n\}}$ and a class C_k among K classes. For every class k, we want to estimate the *conditional probability* that a sample belongs to class k given its features $\mathbf{x}$. Using the Bayes theorem:

$$P\left(C_k \mid \mathbf{x}\right) = \frac{P\left(C_k\right) \cdot P\left(\mathbf{x} \mid C_k\right)}{P\left(\mathbf{x}\right)} \tag{2.2}$$

Now comes the *naive* part: we assume that features x_i are mutually independent. (This assumption is often not verified, but naive Bayes classifiers remain surprisingly robust in practice [143].) When comparing classes, we ignore the denominator in Equation 2.2 since it does not depend on C_k. We can now estimate the most probable class $\hat{k}$ as:

$$\hat{k} = \underset{k\in\{1,\dots,K\}}{\operatorname{argmax}} P\left(C_k\right) \prod_{i=1}^{n} P\left(x_i \mid C_k\right) \tag{2.3}$$

The prior probability for class C_k can be empirically estimated as the ratio of samples having this class in the training data. The probability distributions of features are usually modelled with a normal distribution, but it is possible to build more expressive models with Kernel Density Estimation (KDE) [144, 145]. Given a set of possible features' values for a class C_k, KDE uses a weighted sum of *kernels* to estimate the distribution of observed values. Kernels are non-negative functions estimating the probability distribution *locally*; the standard normal density function is often used in practice. Naive Bayes classifiers usually require few

parameters, even with KDE; this means that they can be trained and stored very efficiently. In computer networks, they have been used for fault detection [146, 147, 148] and even network tomography with beCAUSe [104].

2.4.3 Decision tree learning

Many machine learning algorithms have been proposed to build *decision tree models* [149]. These models are made of a succession of tests ("branches") leading to final predictions ("leaves"). One can apply a decision tree model to a sample by following the branches according to the results of the tests: at every branch, the value of one sample feature is tested. For instance, one test A could check if a feature is lower than 10; if true then the next test would be B, otherwise if would be C. At the bottom of the tree, leaves contain a class or a numerical value for classification or regression problems, respectively. The main advantage of decision trees is their *interpretability*: following the tests made in branches, it is easy to understand the reason behind a prediction.

Since constructing an optimal decision tree is known to be NP-complete, most learning algorithms recursively find the *best test*, or *best split*: at every branch, the algorithm estimates what would be the best test to *split* the training dataset between different predictions. The quality of a test is estimated using a *loss function* on created child branches: for every class C_k, at child i, let $P_i(C_k)$ be the proportion of samples having class C_k as label. The loss at i can be estimated using the two popular formulas:

$$H_i = 1 - \sum_{C_k} P_i\left(C_k\right)^2 \qquad \text{(Gini impurity)}$$

$$H_i = -\sum_{C_k} P_i\left(C_k\right) \log\left(P_i\left(C_k\right)\right) \qquad \text{(Entropy)}$$

Similarly, we can use MAE and MSE as loss functions for regression trees. It is possible to split the dataset until all leaves contain only one class (or one value); however this could lead to very large trees and serious overfitting. Most algorithms propose *stop criterions* such as "minimum number of training samples per leaf" or "maximum tree depth" to avoid this issue. Among the learning algorithms, C4.5 [150] and CART [151] have been widely used in the literature of computer networks. For instance, Ebay and Microsoft use decision trees to locate failures in their datacenters [119, 152]; mobile streaming issues are diagnosed by a C4.5

tree in [120]; NetPrints [118] propose configuration troubleshooting using interpretable trees and TCP congestion control algorithms are automatically inferred by decision trees in [153].

2.4.4 Ensemble learning

It is possible to *combine* multiple models to improve machine learning predictions. For classification problems, one can take the class predicted by the *majority* of models, while the *average* prediction can be used in regression problems. Perhaps the most popular example of ensemble learning is the *random forest* approach [154]. With random forests, many simple decision trees are trained on *random* subsets of the training dataset; during inference, the majority prediction over all the small trees is chosen as output. This process is called "bagging" (for "bootstrap aggregating") and can be applied with other families of models.

"Boosting" [155] is an alternative ensemble method: new models are *successively* trained on samples with invalid predictions from previous models. If the performance function is differentiable, gradient boosting [156] can be used to optimize how new models are constructed. Finally, "Stacking" [157] uses a "meta-model" trained to *combine* the output of other models into the final prediction.

These simple approaches help to fight overfitting and works surprisingly well in practice: ensemble methods usually exhibit the best performance in machine learning challenges [158, 159]. This trend has been verified in many recent computer network problems, including fault detection and localization [119, 147, 148, 160, 161, 162]. Empirically, ensemble output should be *at least* as performant as the best model's output for a given sample: errors of individual models are cancelled on average. We confirm this intuition by averaging multiple machine learning models in Chapter 3 and Chapter 4. One major drawback is that it is harder to *understand* the prediction of an ensemble model (compared to decision trees).

2.4.5 Artificial Neural Networks

One of the earliest model of brain neurons interactions is Franck Rosenblatt's *perceptron* [163]. A perceptron can be described as a binary linear classifier: it iteratively tries to

find the parameters of a hyperplane allowing the *separation* of two classes. For n input features, this hyperplane is defined by *weights* $\mathbf{w} \in \mathbb{R}^n$ and *bias* $b \in \mathbb{R}$ such that the predicted class $\hat{y_i}$ of a sample with features $\mathbf{x}_i$ is:

$$\hat{y_i} = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i + b > 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Weights are iteratively updated for every sample i in the training dataset, as shown in Equation 2.5. One important parameter of the training process is the *learning rate* $\eta \in]0, 1]$: large learning rates speed up the training but make the weights less stable.

$$\forall j \in \{1, \ldots, n\} : w'_j = w_j + \eta \left(y_i - \hat{y}_i\right) x_{i,j} \quad \text{and} \quad b' = b + \eta \left(y_i - \hat{y}_i\right) \tag{2.5}$$

It has been proven that perceptrons converge if the training set is *linearly separable*. However, most datasets are not: this led to the development of more complex models such as Support Vector Machines (SVMs) and Multilayer Perceptrons (MLPs), the latter more widely known as "fully-connected neural networks". In a MLP, multiple perceptrons with *non-linear activation functions* are used to create a sequence of *layers* of any dimension. The first layer corresponds to sample features while the last layer contains the model output(s). Intermediate layers are often called "hidden layers" and can be of any size—the number of layers and their sizes becoming the main hyperparameters of the MLP. Activation functions for hidden layers can be as simple as the Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$. In contrast, since the last layer directly maps to the model's output(s), its activation function is problem-dependent: for classification tasks with K classes, the softmax function is often used to normalize the K outputs to a *probability distribution*.

$$\sigma : \mathbb{R}^K \mapsto \mathbb{R}^K : \sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=0}^{K} e^{x_j}} \qquad \text{(Softmax activation)}$$

A perceptron transfers the information from a layer L_t of size u to a layer L_{t+1} of size v using its weights ("parameters") matrix $\mathbf{w}^t \in \mathbb{R}^{u \times v}$. Compared to Equation 2.4, matrix multiplication is used instead of the dot product between a perceptron's inputs and $\mathbf{w}^t$ (see Figure 2.5 for an illustration on a neural network with a single hidden layer). The training process is similar, albeit with many more weights to *learn*: for each training sample, the algorithm must compute the *gradient* of every weight value with respect to the task's loss function. Backpropagation [164] optimizes this process: it starts by computing the gradients of the *last* perceptron

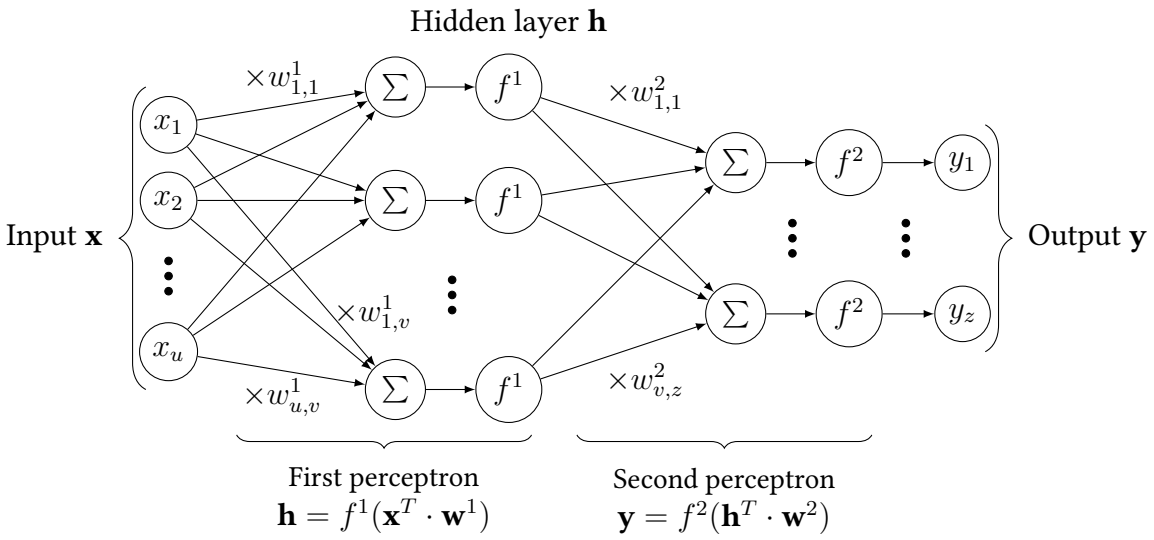


Figure 2.5 – A simple MLP with one hidden layer. Values are *propagated* through the neural network using matrix multiplications between inputs ($\mathbf{x}$ then $\mathbf{h}$) and weights ($\mathbf{w}^k$). Activation functions are denoted f^k.

and then *backpropagates* to the first using the chain rule. Weights are then updated according to their gradients and the chosen learning rate η, a process known as "Stochastic Gradient Descent". (Using the full dataset in each iteration being impractical, random samples are selected instead, making the "Stochastic" part of the process.) Other optimization methods and dynamic learning rate schedulers have been proposed to speed up the training process and avoid "local loss minima". Originally designed to run on costly dedicated hardware using potentiometers as weights and motors for updates, artificial neural networks can now run on commodity Control Processing Units (CPUs) and Graphical Processing Units (GPUs). We now explore several variants of MLPs along with their applications.

CNNs With the rise of available computing power, it has become possible to train neural networks with high-dimensional data. Images are an example of such data: it is easy to map a $n \times m$ pixels image with c color channels to a $n \times m \times c$ input vector. Convolutional Neural Networks (CNNs) [165] have initially been proposed to recognize written numbers in small images: through a number of *convolutional* layers, CNNs can recognize *patterns* in the input image. One convolutional layer works by convoluting small *filters* (or *kernels*) over the 2-dimension space of the image. The main advantage of this approach is that it requires way fewer parameters than fully-connected layers, reducing overfitting. Pooling layers can also further reduce the number of parameters in hidden layers by "merging neighbor pixels" using

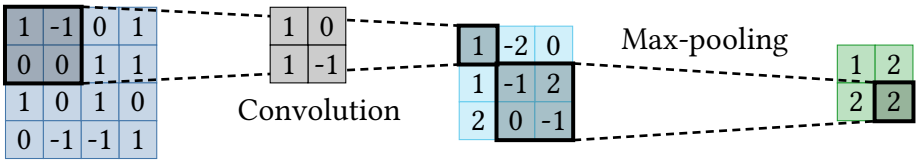


Figure 2.6 – Example of a 2×2 convolution followed by a 2×2 max-pooling operation. The convolutional and pooling kernels "slide" over the inputs to construct the outputs from left to right, top to bottom.

e.g. the average or maximum values. Since the same filters are applied multiple times to cover an image, CNNs are more robust to *translations*, and are better at pattern detection.

Figure 2.6 shows a toy CNN example with as input (left) a 4×4 matrix. A two-dimensional input is used in practice for images; a third dimension can also be used to represent colors (e.g. for red, green and blue channels). During the 2×2 convolution, we sequentially compute dot products between 2×2 "windows" over the input and the parameterized kernel $K = \left(\begin{smallmatrix} 1 & 0 \\ 1 & -1 \end{smallmatrix}\right)$, learned during the training phase. As shown in the figure, the top-left output is computed by taking the top-left window in the input (Equation 2.6). The top-center output is then computed (Equation 2.7) by "sliding" the window one cell to the right: this is a convolution with *overlapping* windows, leading to an output of size 3×3.

$$\text{vec}\left(\left(\begin{smallmatrix} 1 & -1 \\ 0 & 0 \end{smallmatrix}\right)\right) \cdot \text{vec}\,(K) = 1 \times 1 + (-1) \times 0 + 0 \times 1 + 0 \times (-1) = 1 \tag{2.6}$$

$$\text{vec}\left(\left(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right)\right) \cdot \text{vec}\,(K) = (-1) \times 1 + 0 \times 0 + 0 \times 1 + 1 \times (-1) = -2 \tag{2.7}$$

Pooling operations helps to reduce the number of dimensions in a CNN: in Figure 2.6, we apply a 2×2 "Max-pooling" operation using a similar sliding window. For instance, the final bottom-right output is computed my taking the maximum value over the bottom-right window: $\max\left(\left(\begin{smallmatrix} -1 & 2 \\ 0 & -1 \end{smallmatrix}\right)\right) = 2$.

While early convolution layers detect *simple* patterns (such as horizontal or vertical lines), deeper layers are activated by increasingly complex patterns (circles or even human faces). It is worth noting that best-performing CNNs are extremely *deep*: the ResNet model [166] won the ILSVRC2015 image recognition challenge with more than 150 hidden layers. In classification tasks, final layers are usually just fully-connected layers trained with a specific dataset. This leads to an interesting property: it is usually possible to change the classes a model can predict by *only retraining the last layers of a network*. Intuitively, convolutional layers extract relevant features from the raw input while fully-connected layers specialize to the requested prediction.

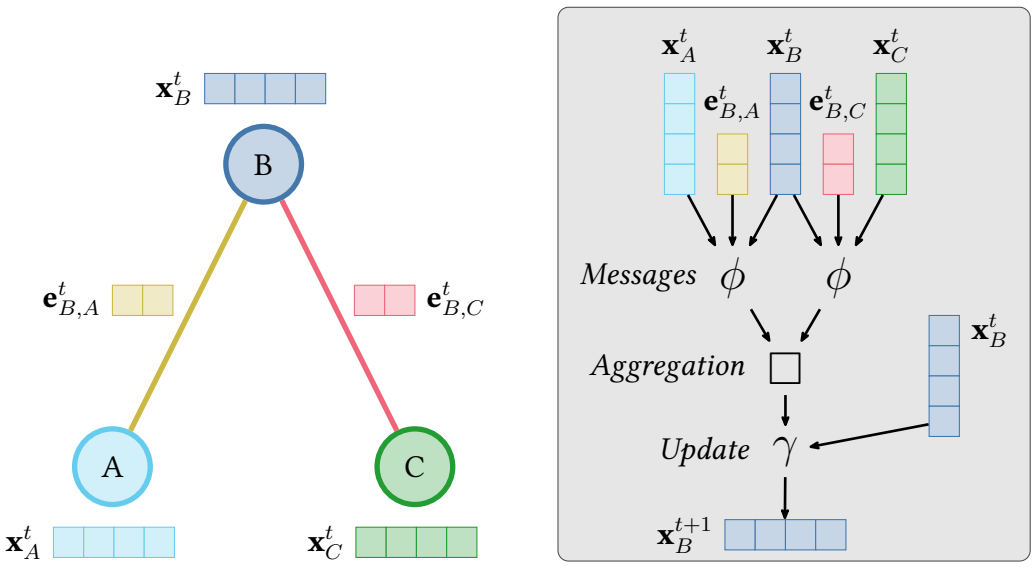


Figure 2.7 – Left: simple graph with associated embeddings. Right: using GNN message passing (Equation 2.8), the procedure to update B's representation given its neighbors A and C.

Despite an overwhelming adoption in image analysis, there has also been some interest for CNNs in anomaly detection [147, 167].

RNNs Recurrent Neural Networks (RNNs) [168] are best suited when the model's input is made of a *sequence* of values. This is for instance the case in time-series, audio files or text sentences. Compared to previous architectures, recurrent layers maintain an *internal state*; this is critical to remember the *context* between different values in the input sequence. Long Short-Term Memory (LSTM) [169] is the most used recurrent layer. It has been successfully applied in automated translation, speech recognition and synthesis, time-series prediction, anomaly detection, etc. Combining the best of both worlds, ConvLSTM layers [170] add *convolutions* to RNNs, enhancing support for more complex inputs such as sequences of radar probes or movies.

GNNs More recently, Graph Neural Networks (GNNs) were introduced to fully exploit the topology of *graph inputs* [171]. This family is particularly interesting to model physical networks (city roads, power grids, molecules, etc.) but also dependency networks (e.g. social graphs). Of course, computer networks can be easily mapped to a graph: GNNs were used for routing optimization [172], edge anomaly detection [173] and SDN modeling [11]. In many

popular GNNs, message passing [174] is used to *propagate* the information signal between vertices. At each layer t, each vertex of a GNN graph is associated with a *hidden states* $\mathbf{x}_i^t \in \mathbb{R}^u$ (also called *embeddings* or *representations*). Graph edges are also associated with embeddings $\mathbf{e}_{i,j}^t \in \mathbb{R}^v$. For every edge and vertex, embeddings for the next layer are computed using previous embeddings *and* neighbor's embeddings. We illustrate how one can update embeddings of a 3-vertices graph in Figure 2.7. For every neighbor of the target vertex (here A and C), we compute messages using ϕ. Then, we aggregate the messages with $\square$ and update the B's representation using γ. More generally, embeddings are updated with Equation 2.8 where:

- $\phi : \mathbb{R}^{u \times u \times v} \mapsto \mathbb{R}^w$ is the *message* function;
- $\gamma : \mathbb{R}^{u \times w} \mapsto \mathbb{R}^u$ is the *vertex update* function;
- $\square$ is an *aggregation function* that can take any number of arguments (e.g. sum, average);
- $\mathcal{N}(i)$ returns the set of indices of vertex i's neighbors;
- γ and ϕ are usually (simple) neural networks.

$$\mathbf{x}_i^{t+1} = \gamma\Big(\mathbf{x}_i^t \, , \, \square_{j \in \mathcal{N}(i)} \, \phi \left(\mathbf{x}_i^t \, , \, \mathbf{x}_j^t \, , \, \mathbf{e}_{i,j}^t \right) \Big) \tag{2.8}$$

This message passing method can be adapted to any graph topology thanks to the aggregation operator $\square$. (Note that the used GNN topology is derived from the studied network topology, but is not necessarily the same.) Trained weights are only defined in γ and ϕ: any existing training algorithm can be used to optimize these weights. Again, fully-connected layers may be used to *readout* information from constructed embeddings [11, 174] and output the requested prediction at edge, vertex or graph level.

2.5 Conclusion

At this point, we have just touched the vast subjects of computer network modeling, root cause analysis and the fascinating problems and methods of machine learning. There have been many proposals for computer network modeling, which we have categorized depending on the amount of information available. In this thesis, we argue that statistical (machine) learning can be very useful in the modeling of large computer networks with complex interdependencies. To support this claim, we propose three contributions, each of which approaches the network modeling problem from a different perspective.

In Chapter 3, we start by reviewing how GNNs can be used for network path performance estimation. Nowadays, simulation and emulation are the most practical approaches to estimate the performance of a network knowing its components' characteristics. However, these methods are costly and may not scale to large networks. By contrast, we show that this first contribution allows accurate simulation of networks with dozens of nodes in *only a few milliseconds*. We detail in particular how we won the first place of a GNN challenge co-organized by the ITU and the Barcelona Neural Networking Center.

Our second contribution (Chapter 4) removes the assumption of complete network knowledge we made previously. It moves to a situation in which some ground truth about faults and failures is available, but the exact makeup of the network is not fully understood. More precisely, we rely on network emulation with `netem` to artificially inject *failures* into a controlled testbed, allowing us to build a dataset of nominal and faulty network metrics involving diverse cloud providers on multiple continents. We use several machine learning models (a Naive Bayes and a Random Forest classifiers, along with an original CNN: DiagNet) on this dataset in an attempt to model a large scale computer network from the viewpoint of end users. We specialize these models for RCA: by crowdsourcing active measurements from multiple end users to various vantage points, they provide automated dependency inference and diagnosis without needing ISPs collaboration. Compared to typical network tomography, our solutions do not need precise information about a network's topology: they leverage the *diversity* of vantage points and output combinations of *location* and *fault family* as probable root causes. We designed our three solutions to be *extensible*, and show the superiority of the DiagNet CNN in its capacity to accept *new features unseen during training*.

In Chapter 5, our goal is to evaluate how the solutions we described in Chapter 4 can be applied using *real* measurements and third-party failures. To this end, we first describe how we built a web browser extension and a measurement platform (DiagSys) to collect real network

metrics from clients. Despite modern security restrictions, this new extension is compatible with web browsers supporting the WebExtension Application Programming Interface (API). We collected more than one year of network measurements thanks to the volunteers that participated in our experiment. We present interesting insights from this dataset and finally evaluate how DiagNet performs over it. Compared to the previous contributions, the main challenge was the *labelling* of training samples since no ground truth was available for most observed failures. Aided by our collected dataset, we nonetheless propose a simple statistical method to label samples with probable root causes as a combination of fault family and network location.

Chapter 3

Computer Network Modeling with Graph Neural Networks

Contents

3.1 Context and Motivation 47
3.1.1 ITU's 2020 Graph Neural Network challenge 47
3.1.2 The RouteNet baseline 48
3.2 Our proposal: learn representations for network nodes 50
3.2.1 Message passing over bipartite graphs for dependency modeling . . 50
3.2.2 Approach A1: dedicated representation with node embeddings . . . 53
3.2.3 Approach A2: node representation through link embeddings 57
3.3 Evaluation with the datasets of the GNN challenge 58
3.3.1 Datasets presentation and challenge results 58
3.3.2 Methodology and implementation details 61
3.3.3 Techniques for optimal training strategy 63
3.3.4 Ablation analysis 65
3.3.5 Visualization of learned representations 69
3.4 Conclusion 72

Network operators design computer networks to provide good connectivity to their customers. As such, they need reliable methods to predict the expected performance even before deploying the networks' physical links and appliances or modifying Software-Defined Networking (SDN) configurations. As we discussed in the previous chapter, there exists two families of approaches to model a computer network. Network emulation tries to replicate the modeled network topology over an existing network. With enough available hardware resources, emulation can be very useful to evaluate how the modeled network would perform in practice. Simulation methods are more flexible, allowing us to model networks with arbitrary characteristics; this is very useful when the emulation platform is not large enough. However, given the complexity of network components' interactions, simulation tools often demand intensive resources or require to simplify the network model drastically. This is problematic in

the context of large dynamic networks, where the traffic flows continuously change over time: operators need to review and adapt the network topology according to these changes, while seeking to minimize the impact over the network. Network designers must also be able to test alternative network configurations in a short time. A performant network modeling tool is thus required to quickly evaluate how the updated network topology would perform with live network characteristics.

In this chapter, we explore how the detailed performance of a large network model can be predicted without relying on costly simulations. We extend on early work in this area [11], and explore how Graph Neural Networks (GNNs) [171] (Subsection 2.4.5) can be trained to produce performance predictions in a fraction of the time required by a simulation tool. We first detail the motivation behind this work in Section 3.1, along with a presentation of an early GNN solution, RouteNet. We then explain in Section 3.2 how we extend the initial RouteNet approach to map all the components of a computer network to a GNN. In Section 3.3, we present the results of our contributions, including the details of our participation to the GNN Challenge 2020 [175], co-organized by the Barcelona Neural Networking Center and the International Telecommunication Union (ITU) [12]. We were honored to win this challenge with a first approach (A1) leading to an average relative error of just 1.53% over the provided evaluation dataset. Using a second approach (A2), we obtained a lower relative error of just 1.15% after the conclusion of the challenge. We also provide in Section 3.3 an ablation study and additional visualizations of learned representations in an attempt to fully characterize our contributions.

This work has been done with the collaboration of François Schnitzler (InterDigital) and people from the Barcelona Neural Networking Center. We are especially thankful to José Suárez-Varela for his availability during the challenge.

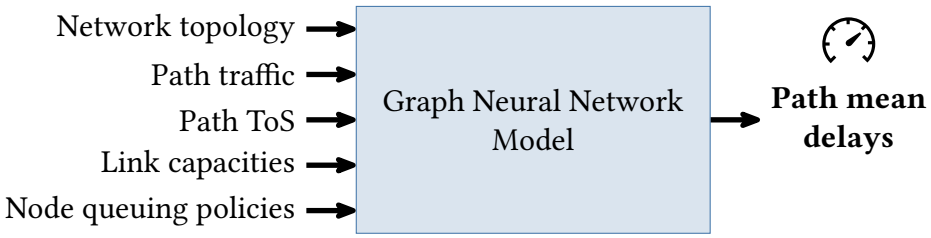


Figure 3.1 – Illustration of the GNN challenge task: predict delays for network paths from a given configuration.

3.1 Context and Motivation

The Barcelona Neural Networking Center organized the GNN Challenge 2020 [175] as part of the "AI/ML in 5G" Challenge from the ITU [12]. We start by describing the challenge, then introduce RouteNet, the baseline model that was provided by the organizers as a starting point.

3.1.1 ITU's 2020 Graph Neural Network challenge

In the SDN context, Rusek et al. [11] showed that GNNs [171] are particularly promising for computer network modeling. This recent family of machine learning models is well suited to capture complex interactions between network components. Actually, it has been successfully used in a wide range of applications, from the analysis of unknown chemical compounds [174] to traffic prediction in road networks [176]. One important specificity of GNNs is that they can be adapted for *any* network topology after the initial training phase.

The ITU challenge has been proposed in this context. The task was to provide a GNN model that, knowing a number of network characteristics (i.e. topology, link capacities, path Type of Service (ToS) and traffic, and node queuing/scheduling policies), could predict *path performance* (Figure 3.1). In particular, we were asked to build a model that predicts the *average delay* over every path, taking into account queuing delays at intermediate nodes. A few rules were stated: the solutions must use the provided training dataset only, and must be based on artificial neural networks trained "from scratch". In particular, it was not possible to use network simulation techniques.

To evaluate the solutions objectively, the challenge's authors provided an evaluation dataset (test dataset, see Subsection 3.3.1) with missing path performance metrics. This evaluation dataset is based on an *unknown* topology, not present in the provided training and validation datasets. This choice makes it possible to evaluate how solutions *generalize* to different topologies without retraining, a critical property to use the solutions in practice. Using their

solution, each challenge team had to submit the missing path delays on a dedicated website; the website then computed the Mean Absolute Percentage Error (MAPE) against the secret, expected metrics (Equation 3.1). (Note that this score function is valid because expected delays are always positive.) To avoid excessive numbers of attempts, each team could only submit 20 propositions.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\text{expected}_i - \text{predicted}_i}{\text{expected}_i}\right| \tag{3.1}$$

3.1.2 The RouteNet baseline

RouteNet [11] is a message passing GNN (see Subsection 2.4.5) supporting two types of embeddings, one for *paths* and one for *links*. Knowing network characteristics, RouteNet can predict average path delays, jitter and loss—important metrics to assert network performance—making it a good challenge baseline. We now briefly explain the intuition behind RouteNet: in the next section, we will formalize and extend this intuition in the presentation of our own approach. A computer network topology is mapped to the GNN as follows. First, one vertex of type "link" is created for every link in the network topology. Then, *for every path*, one "path" vertex is added and is linked to every "link" vertices that are part of the path. With this approach, the resulting GNN is essentially a bipartite graph underlining the circular dependencies between links and paths. (See next section for an illustration of this bipartite mapping.)

Embeddings are initialized with corresponding path and link features. *Messages* are then iteratively exchanged to update these embeddings: first, path embeddings are updated from the *ordered* embeddings of its member links. (Internally, this first update function uses a RNN to take path ordering into account.) Then, RouteNet does the opposite: every link embedding is updated by aggregating the embeddings of all the paths that use this link. To represent increasing *load* on links, this aggregation is done by a global sum function, followed by a simple perceptron. These two circular updates are executed $T = 8$ times for *every sample*, both during training *and* inference. Given the complex interdependencies between paths and links, this method mimics an approximated fixed-point iteration. Message passing GNNs departs from the typical *layers* of neural networks but the approach is actually similar: successive non-linear operations lead to new hidden states for paths and links. The path hidden states are then processed in a standard MLP to *readout* the expected performance metrics (i.e. average path delay).

Limitations As shown in the baseline paper, RouteNet supports *arbitrary network topology* and can generalize well to other topologies. Still, this algorithm was initially designed for a simpler problem without path ToS and node queuing policies. As such, it ignores these critical features and lead to poor results in the datasets provided for the challenge. For the evaluation dataset, the challenge organizers announced a baseline MSE of over 46% and a MAPE over 100% [1]. We re-implemented the algorithm in PyTorch [177] and after extensive tweaking obtained a MAPE below 8% (more on that later in Subsection 3.3.4).

1. Numbers from challenge's mailing list

3.2 Our proposal: learn representations for network nodes

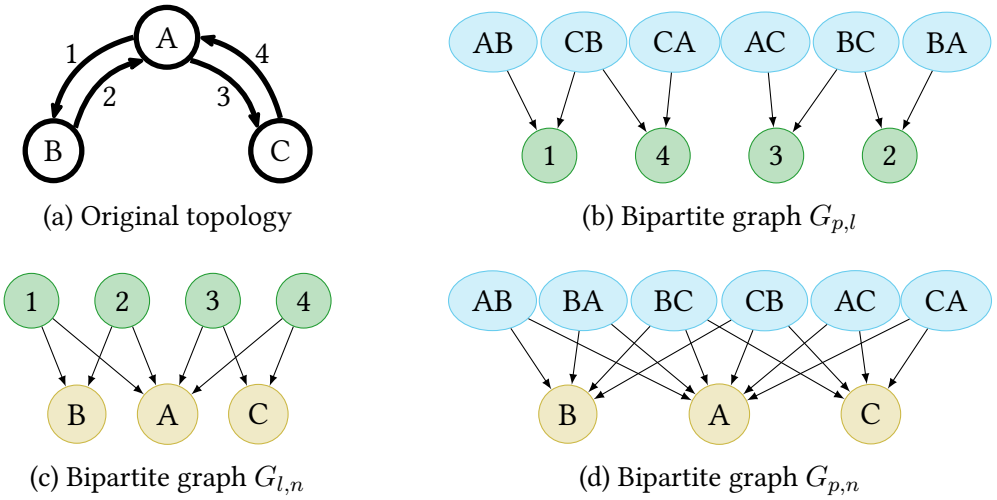
Noting the limitations of RouteNet, we propose two different approaches to learn representations mapped to network nodes. In a first approach (A1), we add *dedicated* embeddings for nodes. These new embeddings contribute to the updates of link and path embeddings in an enhanced version of the baseline algorithm. This first approach was proposed during the GNN challenge and allowed our team to win the competition. We then present a second approach (A2) that embeds node features *within* link features. Despite being simpler, we found after the challenge that this alternative approach further reduced average errors. We start this section by the common ground between the two approaches, explaining how interdependencies are *modeled* and *propagated* between paths, links and nodes.

3.2.1 Message passing over bipartite graphs for dependency modeling

We recall that RouteNet models link-path dependencies using a bipartite graph. Directed bipartite graphs joining types A and B are denoted by $G_{A,B}$; as an example, the graph modeling the dependencies between paths and nodes in RouteNet is $G_{p,l}$. Since we want to add embeddings for network nodes (approach A1), we need additional dependency graphs to model the node-path *and* the node-link interdependencies. Therefore, we construct $G_{p,l}$, $G_{l,n}$ and $G_{p,n}$, three bipartite graphs as follows:

- $G_{p,l}$ connects each path to the links it uses,
  it is essentially the bipartite graph proposed in RouteNet;
- Defined similarly, $G_{p,n}$ connects each path to the nodes it traverses;
- Finally, $G_{l,n}$ connects each link to its *starting* and *ending* nodes.

As an illustration, we show a toy computer network topology in Figure 3.2, along with the corresponding three bipartite graphs. For clarity, this network topology only holds three nodes (A, B and C), four links (1 to 4) and six paths (AB, AC, etc.). Routing is trivial in this topology, since all paths must be routed through node A. As an example, the BC path is made up of links 2 and 3: $G_{p,l}$ has therefore an edge between "BC" and "2" vertices and an edge between "BC" and "3". Link 2 starts at node B and ends at node A, hence $G_{l,n}$ connects "2" with "B" as well as "2" with "A". To conclude the illustration, note that messages in path BC are routed as B $\mapsto$ A $\mapsto$ C. Therefore, three corresponding edges are added in $G_{p,n}$: (BC, A); (BC, B) and (BC, C).



(a) Original topology

(b) Bipartite graph $G_{p,l}$

(c) Bipartite graph $G_{l,n}$

(d) Bipartite graph $G_{p,n}$

Figure 3.2 – Example of a basic topology leading to the three presented bipartite graphs. Nodes are denoted by a single letter, links by a number, paths by two letters (XY meaning path from X to Y).

Let us define $\mathcal{N}_{A,B}(i)$ as the *forward neighbors* of vertex i in the bipartite graph $G_{A,B}$. More formally, $\mathcal{N}_{A,B}(i) = \{j \in \text{vertices}\,(G_{A,B}) \,|\, (j,i) \in \text{edges}\,(G_{A,B})\}$. Conversely, we define $\tilde{\mathcal{N}}_{A,B}(i)$ as the indices j of vertices where (i,j) is an edge in $G_{A,B}$. This $\mathcal{N}$ function (resp. $\tilde{\mathcal{N}}$) can be used to immediately identify dependencies. From the previous paragraph, we have $\mathcal{N}_{p,l}\,(3) = \{\text{AC}, \text{BC}\}$: this means link 3 supports both paths AC and BC. Conversely, the performance of path BC is affected by links 2 and 3, hence $\tilde{\mathcal{N}}_{p,l}\,(\text{BC}) = (2,3)$. Note that this last neighbor list can be *ordered* to capture additional domain information since links are ordered within each path. Node A is part of *every link*: $\mathcal{N}_{l,n}\,(\text{A}) = \{1,2,3,4\}$. Finally, we can also provide a neighbor ordering from path-node dependencies, e.g. $\tilde{\mathcal{N}}_{p,n}\,(\text{BC}) = (\text{B}, \text{A}, \text{C})$.

Some additional notations and definitions

We follow the GNN notations defined in our background chapter (Subsection 2.4.5 in particular). First, we recall in Equation 3.2 a simplified version of the message passing formula. This formula describes how embeddings $\mathbf{x}^{t+1}$ are updated from their previous value $\mathbf{x}^t$ and an aggregation $\square$ of *neighbors' embeddings*. Compared to the original message passing formula, we omit $\mathbf{x}_j^t$ and $\mathbf{e}_{i,j}^t$ arguments in ϕ to reduce its complexity since GNN edges have no embeddings in our approach.

$$\mathbf{x}_i^{t+1} = \gamma\Big(\mathbf{x}_i^t \,,\, \square_{j \in \mathcal{N}_{A,B}(i)}\, \phi\,(\,\mathbf{x}_j^t\,)\,\Big) \tag{3.2}$$

In the following, we will use two different aggregation functions $\square$: one for *unordered* inputs and one that is better suited for *ordered* inputs. In typical GNNs, neighbors are not ordered: the aggregation function can be as simple as a sum of input embeddings. Some proposals normalize this sum, for example by dividing by the degrees of the involved vertices—this helps when some vertices have many more neighbors than others. We take another approach and propose *multiple* permutation-invariant aggregation functions (e.g. sum, average or maximum), all combined through concatenation (denoted $\|$).

$$\text{Agg} : \mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \mapsto (\ \text{avg}\,(\mathcal{X})\ \|\ \text{sum}\,(\mathcal{X})\ \|\ \min(\mathcal{X})\ \|\ \max(\mathcal{X})\ \|\ \text{var}\,(\mathcal{X})\ ) \qquad \text{(Unordered aggregation)}$$

That being said, one could provide the ordering of neighbors for richer aggregation. For instance, neighbor vertices could be ordered by properties such as distance or chemical interactions. In the challenge case, links are naturally ordered by their position within a path. We take a similar approach than RouteNet and use RNNs to aggregate ordered sequences of embeddings.

$$\text{OrdAgg} : \mathcal{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \mapsto \text{RNN}\,(\mathcal{X}) \qquad \text{(Ordered aggregation)}$$

In the remaining of this chapter, γ and ϕ are single-layer perceptrons followed by a non-linear activation function. We denote the embeddings states for paths, links and nodes as $\mathbf{p}$, $\mathbf{l}$ and $\mathbf{n}$ respectively. Intuitively, the latter embeddings $\mathbf{n}$ are important to model the nodes' *load* based on paths' traffic and Type of Service (ToS). However, they introduce significantly more parameters to learn and might not be needed if there is no interactions between all the queues on the same node. In the following subsections we explore two approaches: one with node embeddings $\mathbf{n}$ (A1) and one without (A2).

Message passing in detail

Equipped with these notations, we can now apply the definition of message passing layers to our problem (Equation 3.2). Building on our previous example, we update link embeddings from dependent path embedding: in other words, compute $\mathbf{l}^{t+1}$ from its previous value $\mathbf{l}^t$ and *relevant* $\mathbf{p}^t$. This operation is illustrated in Figure 3.3 for "link n.3". We first find the paths that link n.3 supports: from previous paragraph, we know that these paths are AC and BC. For every path found, we compute a *message* with the ϕ function; we then *aggregate* all these messages to a single vector using an aggregation function (we chose "Agg", defined in previous

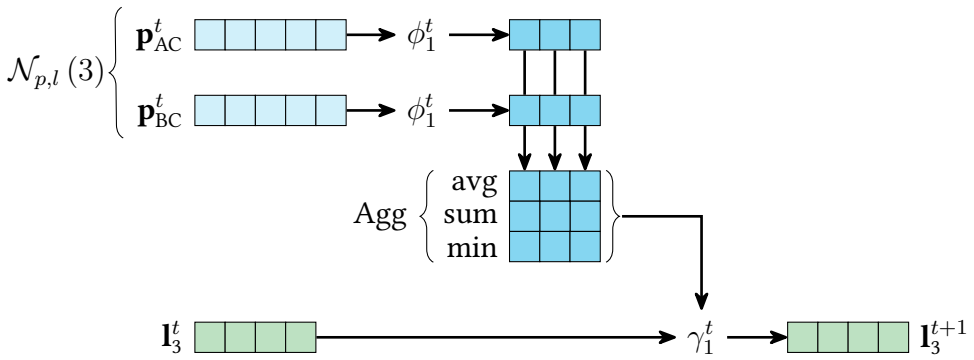


Figure 3.3 – Updating the embedding of link n.3 with neighbor path hidden states (from our toy topology). We can see how "messages" are generated and aggregated to update the link embedding. This is equivalent to $\mathbf{l}_3^{t+1} = \gamma_1^t \left(\mathbf{l}_3^t \, , \, \text{Agg}_{j \in \mathcal{N}_{p,l}(3)} \, \phi_1^t \left(\mathbf{p}_j^t \right) \right)$.

subsection). Finally, we use the *update* function γ to compute the new embedding from its previous value and the aggregation result.

3.2.2 Approach A1: dedicated representation with node embeddings

In this first approach, we propose to add a third embedding type for nodes. We thereby must solve the dependencies between three kinds of embeddings: for paths, links and nodes. We propose to solve the interdependencies between network components by propagating the information signal from path embeddings to link embeddings to node embeddings. This strategy is directly derived from the interdependencies one might intuitively expect between these network components: paths traffic have a direct impact on links, in turn having a direct impact on nodes. However, congested nodes may *cap* their throughput over links, later impacting path performance metrics. Thus, it is also important to propagate dependencies the other way around: from nodes to links to paths.

We show the interactions between embeddings in Figure 3.4, with the detailed algorithm available in Algorithm 1. First, we initialize embeddings to their related features: path embeddings are initialized with path features along with an additional padding to fit the embedding dimension. The same operation is applied for links and nodes—this corresponds to step ① in Figure 3.4 and lines 1 to 3 in Algorithm 1. We then execute a first message passing layer to update link embeddings using the $G_{p,l}$ bipartite graph and the associated dependencies $\mathcal{N}_{p,l}$ (step ②, line 5). In a similar way, we update node embeddings in step ③ (line 6) using $G_{l,n}$ and messages from freshly-updated link embeddings. We then propagate the information signal "backward" from nodes to links to paths using the inverse dependencies $\tilde{\mathcal{N}}_{l,n}$ and $\tilde{\mathcal{N}}_{p,l}$. This

Algorithm 1: High-level pseudo-code of proposed GNN approach A1

Input: Features $\mathbf{F}_p, \mathbf{F}_l, \mathbf{F}_n$; neighbors $\mathcal{N}$ and $\tilde{\mathcal{N}}$; trainable functions ϕ, γ and FC.
Output: Readout performances of every path

▷ *State initialization, padding with zeroes*

$\forall i \in \mathbf{p} : \mathbf{p}_i \leftarrow [\mathbf{F}_{p,i}, 0, \ldots, 0]$

$\forall i \in \mathbf{l} : \mathbf{l}_i \leftarrow [\mathbf{F}_{l,i}, 0, \ldots, 0]$

$\forall i \in \mathbf{n} : \mathbf{n}_i \leftarrow [\mathbf{F}_{n,i}, 0, \ldots, 0]$

for $t \leftarrow 1$ **to** T **do**

▷ *Update link embeddings from path embeddings*

$\forall i \in \mathbf{l} : \mathbf{l}_i \leftarrow \gamma_1^t \left(\mathbf{l}_i, \text{Agg}_{j \in \mathcal{N}_{p,l}(i)} \, \phi_1^t \left(\mathbf{p}_j\right)\right)$

▷ *Update node embeddings from link embeddings*

$\forall i \in \mathbf{n} : \mathbf{n}_i \leftarrow \gamma_2^t \left(\mathbf{n}_i, \text{Agg}_{j \in \mathcal{N}_{l,n}(i)} \, \phi_2^t \left(\mathbf{l}_j\right)\right)$

▷ *Update link embeddings from node embeddings ("backward operation")*

$\forall i \in \mathbf{l} : \mathbf{l}_i \leftarrow \gamma_3^t \left(\mathbf{l}_i, \text{Agg}_{j \in \tilde{\mathcal{N}}_{l,n}(i)} \, \phi_3^t \left(\mathbf{n}_j\right)\right)$

▷ *Update path embeddings from link embeddings, two aggregations*

$\forall i \in \mathbf{p} : \mathbf{p}'_i \leftarrow \gamma_4^t \left(\mathbf{p}_i, \text{Agg}_{j \in \tilde{\mathcal{N}}_{p,l}(i)} \, \phi_4^t \left(\mathbf{l}_j\right)\right), \mathbf{p}''_i \leftarrow \gamma_5^t \left(\mathbf{p}_i, \text{OrdAgg}^t_{j \in \tilde{\mathcal{N}}_{p,l}(i)} \, \phi_5^t \left(\mathbf{l}_j\right)\right)$

$\mathbf{p} \leftarrow FC_0 \left(\mathbf{p}' \parallel \mathbf{p}''\right)$

▷ *Bonus message passing from node to path embeddings*

$\forall i \in \mathbf{p} : \mathbf{p}'_i \leftarrow \gamma_6^{T+1} \left(\mathbf{p}_i, \text{OrdAgg}^{T+1}_{j \in \tilde{\mathcal{N}}_{p,n}(i)} \, \phi_6^{T+1} \left(\mathbf{n}_j\right)\right)$

▷ *Start readout from path embeddings and include features back*

$\mathbf{r}_0 \leftarrow \left(\mathbf{p} \parallel \mathbf{p}' \parallel \mathbf{F}_p\right)$

$\mathbf{r}_1 \leftarrow \text{Activation} \left(FC_1 \left(\mathbf{r}_0\right)\right)$

$\mathbf{r}_2 \leftarrow \text{Activation} \left(FC_2 \left(\mathbf{r}_1\right)\right)$

$\mathbf{r}_3 \leftarrow \text{Activation} \left(FC_3 \left(\mathbf{r}_2\right)\right)$

return $FC_4 \left(\mathbf{r}_3\right)$

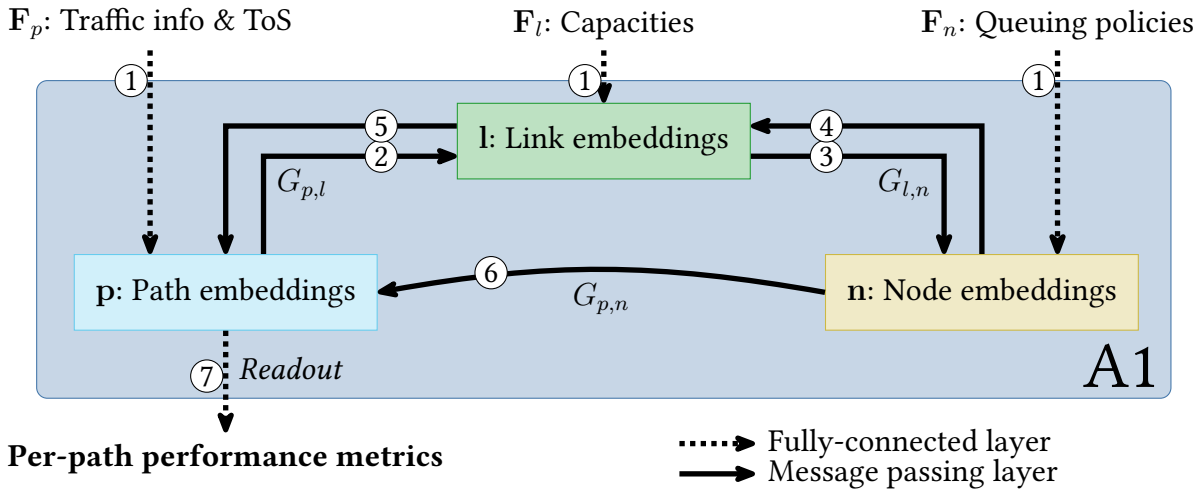


Figure 3.4 – Visual representation of approach A1, proposed for the GNN challenge. First, embeddings are initialized from network features ①. Using message passing operations over the bipartite GNN graphs $G_{p,l}$ and $G_{l,n}$, embeddings are updated through steps ② to ⑤, which are all repeated T times. Path embeddings are finally updated with $G_{p,n}$ ⑥ and we readout the predicted average delay for each path ⑦ using a MLP.

corresponds to steps ④ and ⑤, lines 7 and 8. For the last message passing from links to paths, we leverage the ordering of links within paths and exploit *two aggregations* (line 9).

At this point, all embeddings have been updated at least once. However, only the *direct* dependencies have been modeled and propagated by the four successive message passing layers. To uncover potential cyclic and indirect dependencies, we employ the same strategy as RouteNet which solves this problem by *repeating the message passing cycle* T times (lines 5 to 9). Each new iteration propagates the interdependencies one step further, allowing non-trivial dependencies between network components to be discovered by the algorithm. This approach has been previously used in message passing GNNs as a fixed-point iteration approximation. Notice the chosen indices and exponents in the formulas of Algorithm 1: as an example in line 5, γ_1^t and ϕ_1^t respectively denote the update (resp. message) functions indexed by 1 at iteration t. This means that each iteration ("GNN layer") holds its *own set of trainable function, i.e. trainable parameters* allowing embedding convergence. While RouteNet used $T = 8$, we found that $T = 3$ was a good tradeoff between accuracy and computational complexity. With an average of three neighbors, one network component could reach up to $3^{4\times T}$ other components (i.e. more than enough for embeddings convergence). (In the interest

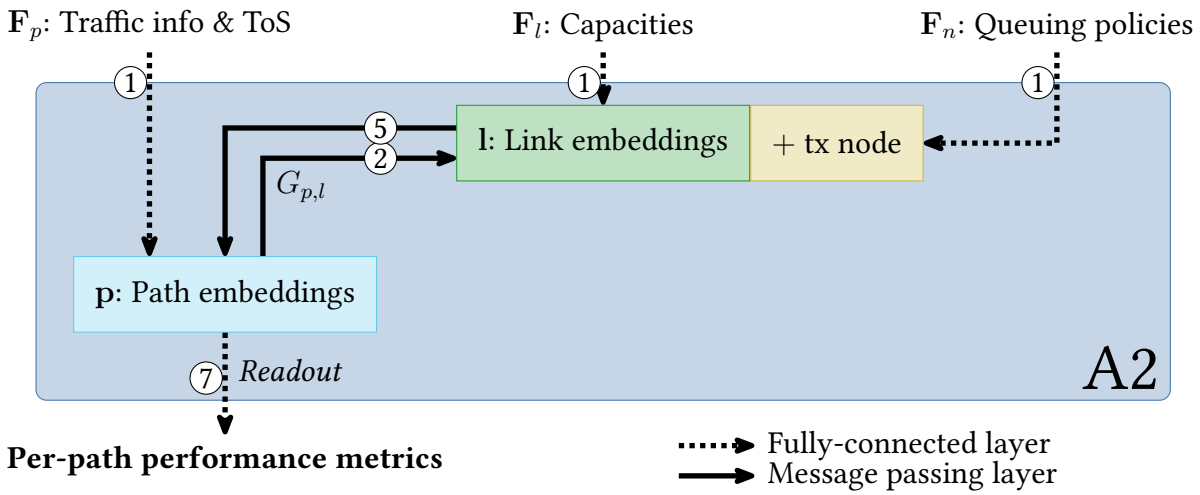


Figure 3.5 – Visual representation of approach A2, studied after the GNN challenge. Compared to approach A1 (Figure 3.4), link embeddings are completed with the *transmitting node's features* $\mathbf{F}_n$. Steps ③④⑥ are not needed in this approach.

of clarity, we omit the indices of embeddings in the algorithm since we *overwrite* the previous embedding values during updates.)

After having performed these T iterations, we use the third bipartite graph $G_{p,n}$ to compute alternative path embeddings $\mathbf{p}'$ from dependent nodes in step ⑥, line 10. This last message passing operation again exploits the *ordering* of nodes within each path thanks to the OrdAgg aggregation function. Our assumption is that this last operation allows the model to leverage the indirect dependency between paths and nodes, without using links as intermediaries.

Finally comes the "Readout" part of our GNN (step ⑦, line 11). We recall that we want to extract performance metrics from paths; so we need to read out information directly from the path embeddings $\mathbf{p}$ and $\mathbf{p}'$. Path features $\mathbf{F}_p$ may have been diluted by the successive message passing operations. To ensure that they are directly mapped to the model's output, we add them in the readout inputs (line 11). Similar to RouteNet's readout, we use a MLP with three hidden layers (denoted FC for "fully-connected") and non-linear activation functions (lines 11 to 15). It is worth noting that this full algorithm is applied both during *training* and *inference*, *for every data sample*.

3.2.3 Approach A2: node representation through link embeddings

We studied this second approach after the completion of the challenge. It is a *simplified* version of approach A1, where node embeddings are not modeled explicitly, but rather "mixed" in link embeddings. In the initial network model, each link i is associated with a *transmitting* node $\text{tx}(i)$ and a *receiving* node $\text{rx}(i)$. We thereby initialize the link embeddings with the relevant link features *and* the transmitting node's features. (We assume that network queues are only modeled for *outbound* messages, hence we ignore receiving node's features in this approach.)

$$\forall i \in \mathbf{l} : \mathbf{l}_i \leftarrow \left[\mathbf{F}_{l,i}, \mathbf{F}_{n,\text{tx}(i)}, 0, \ldots, 0\right] \qquad \text{(Replaces line 2 in Algorithm 1)}$$

Note that here we actually model one set of queues *for every link*. This departs from the approach A1, where queues were *shared* between all outgoing links of every node thanks to dedicated node embeddings. We note that this method has been used by the second and third teams of the GNN challenge (Salzburg Research and Fraunhofer HHI). Given the good results of both teams, we were curious to test their proposal with our own choices of neural network architecture and training hyperparameters.

3.3 Evaluation with the datasets of the GNN challenge

We evaluated our contributions using the datasets provided in the context of the GNN challenge. During the challenge, we only used our first approach A1. We begin by presenting the provided datasets along with a short review of the scores we obtained during the challenge with this approach (Subsection 3.3.1). Then, we detail in Subsection 3.3.2 and Subsection 3.3.3 the methodology and learning techniques we used to reach these results. In an attempt to characterize the effects of our contributions, we provide an ablation analysis in Subsection 3.3.4 including results for our second approach A2. Finally, we explore some visualizations of learned embeddings in Subsection 3.3.5.

3.3.1 Datasets presentation and challenge results

Following the standard methodology of machine learning methods (Subsection 2.4.1), we had access to three different datasets: one for training, one for validation and finally one for evaluation. In these datasets, each sample contains the *results* of a single OMNeT++ network simulation over a known topology with specific component characteristics. The training dataset contains samples from *two different topologies*: one from the historical National Science Foundation Network (NSFNET) with 14 nodes and 21 links, and one from the larger Gigabit European Advanced Network Technology 2 (GÉANT2) network containing 24 nodes and 37 links. The German Backbone Network (GBN)'s topology (17 nodes and 26 links) is used in the validation dataset. Finally, the evaluation set provides samples with an *unknown* topology having 19 nodes and 31 links.

In the simulation model used to create the samples, links have a limited capacity but *zero latency* and they do not drop messages. However, OMNeT++ simulates *message queues* at every node: outbound messages are delayed until there is enough link bandwidth available to send them to the next node. When a packet is put in a full queue, or is delayed for more than 20 simulation time units, it is *dropped*. Of course, real computer networks have links with positive latency. The approach taken in this challenge is nevertheless interesting because it can simulate buffering and congestion problems quite accurately.

Each sample contains the following information:

— The network topology, including path routing information;

— $\mathbf{F}_p$, the features associated with every path, including the amount of generated traffic and the ToS;

— $\mathbf{F}_l$, relevant features for links (in practice, the only feature was the link capacity);

— And $\mathbf{F}_n$, node features (i.e. queue policies and optional queue weights).

Traffic generation One important simulation parameter is the *traffic matrix*, representing the frequency of messages in each *network path*. This frequency is defined by the λ parameters of Poisson processes, the expected number of emitted messages in one time unit. (Message sizes are randomly chosen from a bimodal distribution.) In each sample, there is one path for every source-destination pair of nodes (e.g. 14×13 paths for NSFNET). Additionally, each path has a randomly-selected ToS (0, 1, or 2, to be combined with queuing policies in nodes). Since nodes are not directly connected to every other nodes, a *routing matrix* is also provided: it is used by intermediate nodes to forward a message to its final destination via specific links. Traffic and routing matrices are both randomly generated for every sample, yielding many unique network configurations.

Queuing policies Compared to the initial RouteNet study [11], the authors introduced path ToS and node queueing policies. In the simulator, nodes pick the next messages they send to each link from three distinct first-in first-out queues. The following policies are used:

— Messages are picked from queues with highest priorities with the **Strict Priority** policy;

— With **Weighted Fair Queuing (WFQ)**, queues are given a portion of the link capacity;

— Finally, **Deficit Round Robin (DRR)** is proposed as a computationally efficient approximation of Weighted Fair Queuing.

These three proposed policies correspond to widely used scheduling policies, which are for instance available in the Linux kernel. With the variety of traffic distribution and queuing strategies, we believe the proposed simulation model can replicate the behavior of *real* computer network routers. For every generated network configuration, the OMNeT++ simulation measures performance metrics on every path, including the average bandwidth, delay, jitter and loss rate. A configuration simulation stops when these metrics reach a stationary state: in practice, one dataset sample takes from 30 seconds to more than ten minutes of simulation time—according to provided data.

Challenge results After hyperparameter tuning (see Subsection 3.3.2), we obtained from our best model (based on A1) a MAPE of 1.66% on the evaluation dataset after 750 000 training samples. We detail the chosen hyperparameters in Table 3.1. We would like to highlight that embeddings are significantly larger than the original RouteNet embeddings (400 vs. 8). This

Table 3.1 – Hyperparameters resulting in best model during the challenge.

Batch size	8
Embeddings size	400 for paths, links and nodes ($\mathbf{p}, \mathbf{l}, \mathbf{n}$)
RNN size	same as embeddings (400)
$\mathbf{FC_1}$ size	512
$\mathbf{FC_2}$ size	256
$\mathbf{FC_3}$ size	256
T	3
Activation	Leaky Rectified Linear Unit
Total params	11 465 185

Table 3.2 – Summary of hyperparameters used in models for our best submission.

Approach	Loss function	Hidden state size	$\mathbf{FC_1}$ size	$\mathbf{FC_2}$ size	$\mathbf{FC_3}$ size	T
A1	MAPE	400	512	256	256	3
A1	MSE	400	512	256	256	3
A1	MAPE	300	128	128	256	3
A1	MSE	300	128	128	256	3

leads to more expressive models, but also required more computations and storage capacity: while RouteNet needs less than a day for training, our best model required around four days.

Following the successful approaches in previous machine learning challenges, we tried different ensemble methods to further improve our score. With the GNN challenge, the most successful approach was *ensemble averaging*. We trained different models with slightly modified hyperparameters. While each of these models had an error percentage higher than 1.66% on the evaluation set, *averaging their output* actually led to a *lower error*. This can be expected: combining multiple models generally leads to reduced bias and variance among the results. Our best solution used the *harmonic mean* of the output of four models, leading to a MAPE of 1.53% (see Table 3.2). The advantage of this power mean is that it favors small values—just like the MAPE—reducing the error in our delay predictions. We also tried stacking as another ensemble method, without that much success. (Note that the second team from Salzburg Research obtained a MAPE of 1.95%; this last trick was not really significant compared to our other contributions but could have helped win the challenge.)

3.3.2 Methodology and implementation details

During the few months of the challenge, we tried a lot of ideas and tested them against the provided validation dataset. We started small by just adding the node embeddings and a few additional message passing layers, then tried different GNN flavors and gradually increased embedding and hidden layers dimensions. As in many challenges, this incremental process relied on trial and error: we discarded ideas that did not result in a better model after a fixed training time—giving a better score on the validation dataset. Of course, due to computation and time constraints, it was not possible to try *every* combination of options we had. Hyperparameters were chosen using intuition, domain knowledge and microbenchmarks in an incremental process.

Implementation with PyTorch We implemented our algorithm from scratch using PyTorch 1.4 [177] and the PyTorch Geometric module [178]. This module contains numerous primitives for message passing GNNs: in particular, it provides good abstractions and GPU-accelerated aggregation functions for graphs. To fully exploit PyTorch Geometric, we had to use a single tensor containing all the embeddings for paths, links and nodes: this required some additional padding and concatenation operations compared to Algorithm 1. Our algorithm is implemented in under 500 lines of python code, with double as much for data preprocessing and evaluation [2].

Features preprocessing Input features exhibit large differences between them: for instance, path traffic vary from 40 to 2000 while links capacity vary from 10 000 to 100 000. Hence, we standardize continuous features by removing the mean and scaling to unit variance— this classic preprocessing alone allowed us to greatly improve on the RouteNet baseline. Furthermore, node features only have a limited set of values: there are only three possible queuing policies and five combinations of weights for two of these policies, leading to a small total of 11 possible combinations. We encode these 11 combinations to a 4-dimensional embedding (not to be mistaken with the *network components embeddings* $\mathbf{p}, \mathbf{l}, \mathbf{n}$): this "input embedding" corresponds to the output of a single-layer perceptron optimized during the training process. We apply another input embedding for the paths ToS, converting the three possible values in a dimension of size 4.

2. Please contact Christoph Neumann via email (firstname.lastname@interdigital.com) to request access to the source code.

Training duration We ran the algorithm on an internal computing grid with Nvidia Tesla M60, P100 and V100 GPUs. Each of these GPUs contains between 2048 and 5120 CUDA cores and either 8 or 16 GB of memory. During hyperparameters search, we limited the training to 200 000 training samples. This corresponds to half of the provided training samples, but allowed us to maintain reasonable training durations over the shared computing grid (below one day on average). One week before the end of the challenge, we focused on what we believed was the best architecture, and trained it with 750 000 samples (2 epochs). With the chosen hyperparameters (Subsection 3.3.1), this took around four training days at a speed of 2.7 samples per second on Tesla M60 (to be compared with 4.8 samples per second on P100).

Some remarks on discarded ideas For completeness and future research, we briefly describe the ideas that did not result in better scores in our microbenchmarks.

- We tried to use the *same message and update functions* for all T iterations in the algorithm. We found that this was better to use *different* functions for every iteration (thereby different sets of trainable parameters).
- The training dataset includes samples from two different network topologies. We tried to use only one of these topologies at a time: while the training scores were actually *greatly improved*, it was clear that the produced models were unable to generalize to other network topologies with poor results on the validation set.
- Following the RouteNet approach, we tried to add dropout layers in different locations of our algorithm. These layers are useful to avoid overfitting and can be further used for posterior Bayesian estimation. In our algorithm, it seemed that dropout operations were not useful.
- We tried to *augment* the training dataset, a technique widely used in the literature. In particular, we added random gaussian noise to features and *removed* random nodes, links and paths from network topologies used for training to increase sample diversity. This led to poor results over the validation samples, probably because our augmentation technique introduced too much error in expected performance metrics.
- Finally, we tried to apply several transfer-learning techniques. Our intuition was that we could train a first model on all available performance metrics (including metrics non evaluated in the challenge). Then, we would use the learned parameters of this first model to train a *second* model optimized for path delay prediction. However, this only led to poor results in our evaluations, so we decided to discard this research path.

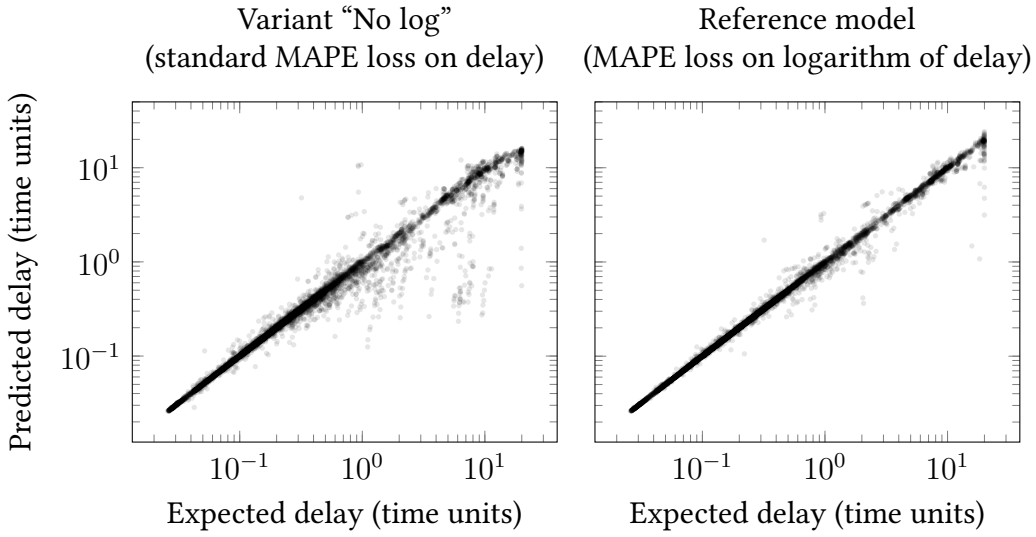


Figure 3.6 – Predicted vs. Expected delay for some paths (each dot is a prediction).

3.3.3 Techniques for optimal training strategy

To obtain good models from a machine learning algorithm, it is necessary to provide a good training strategy. We now review two techniques that allowed us to *significantly* improve our score during the challenge.

Using the logarithm in the loss function The challenge uses the MAPE error to evaluate solutions against expected average delays. This metric tends to favor small predictions: predicting 1 when the expected result is 5 leads to an error of 80%, while predicting 5 when the expected result is 1 leads to an error of 400%. It seemed important to train our models on this loss function, but this implies some practical considerations. First, it is not possible to normalize or standardize the expected model output (a zero value would lead to a division by zero and negative values do not make sense for MAPE). Then, expected delays range from 0.0075 to 20 simulation time units, with most values staying below 1. To ensure numerical stability (and since we are interested in *relative* differences in this challenge) we trained our models over the *logarithm* of the expected average delays distribution: $d'_i = \log(d_i + 1) > 0$, were d_i is the actual delay to be predicted and d'_i the transformed value we use for training.

We review the effect of this contribution in Figure 3.6, where we compare the predictions of two models based on the same algorithm (A1). The first model (left) has been trained using original delays, while the second (right) used the logarithm method we just detailed. We can clearly see that the left model is biased towards *low delay predictions*; this is not surprising

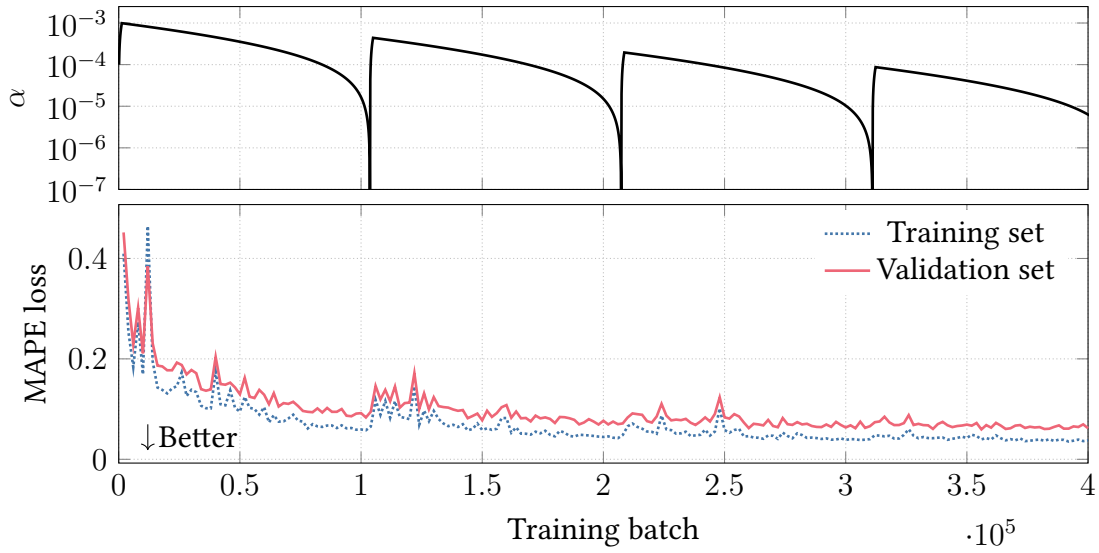


Figure 3.7 – Adam's α value and MAPE loss during the first batches of training. We use PyTorch's Cyclic Learning Rate scheduler.

given the definition of the MAPE loss. Applying the logarithm function in the loss (right plot) reduced this undesired bias substantially, because "over"-predictions are less penalized: taking the previous example, predicting 5 instead of 1 would result in a loss of 159% (instead of 400%). (Note: we also tried to train over the MSE loss, but as expected this led to poor scores with respect to the MAPE evaluation.)

Cyclic learning rate scheduling The learning rate of the training procedure is a major factor to ensure the models actually *converge* to better parameters. Rates that are too small slow down training more than necessary, while higher rates may introduce instability in the results. To avoid these issues, we leveraged the Adam optimizer [179] with its default parameters. Briefly speaking, this optimizer works by maintaining an *adaptive* learning rate for every trainable parameter, updating this learning rate according to observed variance and a configurable step size α [179]. One popular strategy is to vary the learning rate (resp. α) in repeating cycles [180]; first increasing it up to a defined threshold, then decreasing it over the training batches. To this aim, we used two scheduling policies provided by PyTorch:

1. From sample 0 to 500 000, α cycles between a minimum "base" value of 1×10^{-8} and a maximum of 1×10^{-3}, with changes performed every 128 samples. The growth phase spans over 1280 samples while the reduction phase lasts over 102 400 samples, using an exponential decay factor of 0.999;

2. From sample 500 000 onward: the same hyperparameters are used except a base value of 1×10^{-10} and a maximum value of 1×10^{-4}.

We depict the evolution of α and the observed loss during the training process in Figure 3.7. The chosen scheduling policy allowed our models to reach lower error values, compared to monotonically-decreasing schedulers. Intuitively, this policy avoids local minima by *"jumping"* to unexplored regions in the parameters space. This is clearly visible around batch 100 000: after a slow decrease, the learning-rate jumps back to around 5×10^{-4}. The training loss then increases for about 50 000 batches before reaching lower values. We note that the loss over the validation dataset follows the loss over the training dataset with a reasonable difference, showing no sign of overfitting.

3.3.4 Ablation analysis

Our GNN approaches include several contributions. To have a better understanding of the effect of each of these contributions, we provide an *ablation analysis* in this section. The method is as follows: first, we create variants of the *most complex* approach A1 (presented in Section 3.2), each variant having one component *disabled* or slightly modified. Then, we train these variants *with similar hyperparameters* and compare their results over a reference model based on A1. We also compare the variants to the alternative approach A2.

Variants

— **Baseline8 and Baseline400:** These variants are based on the provided RouteNet model. We re-implemented RouteNet in our PyTorch framework and used the same learning hyperparameters than our reference model. (Adam optimizer over the MAPE loss with logarithm applied, using a cyclic learning rate scheduling.) While the RouteNet model initially used embeddings of size 8 (Baseline8), we also implemented a Baseline400 variant with embeddings of size 400. In Baseline400, hyperparameters for readout layers match A1's hyperparameters (Table 3.1).

— **No log:** A simple variant that does not use logarithmic targets in the loss function. We still standardize features, but use the "raw" target delays from 0 to 20 time units.

— **Sum agg:** To evaluate the effect of our proposed non-ordered aggregation "Agg", we replace it with a simpler version that only computes the global sum of its inputs. Note that in this variant, the sum is applied on the 400 dimensions of the input embeddings. In the reference model, embeddings are first reduced to dimension 80 through a perceptron

Table 3.3 – Comparison of ablated variants over the evaluation dataset after training with one million samples. MAPE scores are indicated for the full evaluation dataset (column "all") and for large expected delays only (delays over 5 time units, column ">5").

Variant	# Parameters	MAPE (all)	MAPE (>5)	MSE	R^2
Reference A1	11 465 185	1.63%	4.23%	0.078	0.983
Approach A2	7 702 305	1.15%	3.63%	0.072	0.984
Baseline8	13 089	7.37%	27.1%	0.840	0.817
Baseline400	2 095 561	4.81%	20.8%	0.808	0.824
No log	11 465 185	1.61%	4.36%	0.134	0.971
Sum agg	13 005 025	1.52%	4.14%	0.082	0.982
No path-node	9 817 185	3.38%	4.83%	0.165	0.964

to allow for the concatenation of the five operators. With "Sum agg", this perceptron is kept as an additional hidden layer of size 400. This variant has thereby slightly more parameters than the reference model.

— **No path-node:** In this variant, we simply remove the message passing step from nodes to paths (step ⑥, line 10). Since the corresponding message passing functions do not need to be learned, the overall model size is reduced.

Results

We trained each variant over one million samples (250 000 more than during the challenge). When doing this analysis, and since the challenge was over, we had access to the expected delays in the evaluation dataset and were able to evaluate the variants over this dataset. (Note that the results are *extremely similar* for both training and validation sets, indicating no sign of overfitting.) We report the MAPE, the MSE and the coefficient of determination R^2 [181] in Table 3.3. This last coefficient indicates how much of the prediction variance is explained by the model: a model returning a constant path delay may have good MAPE and MSE, yet will obtain a maximum R^2 of zero. Conversely, a model fitting perfectly well the data will have a coefficient of determination equals to one. Surprisingly, many variants have a *lower MAPE* than the model we submitted during the challenge. This is especially true for the approach A2 with only 1.15% of average relative error. We now give the lessons we learned in this study.

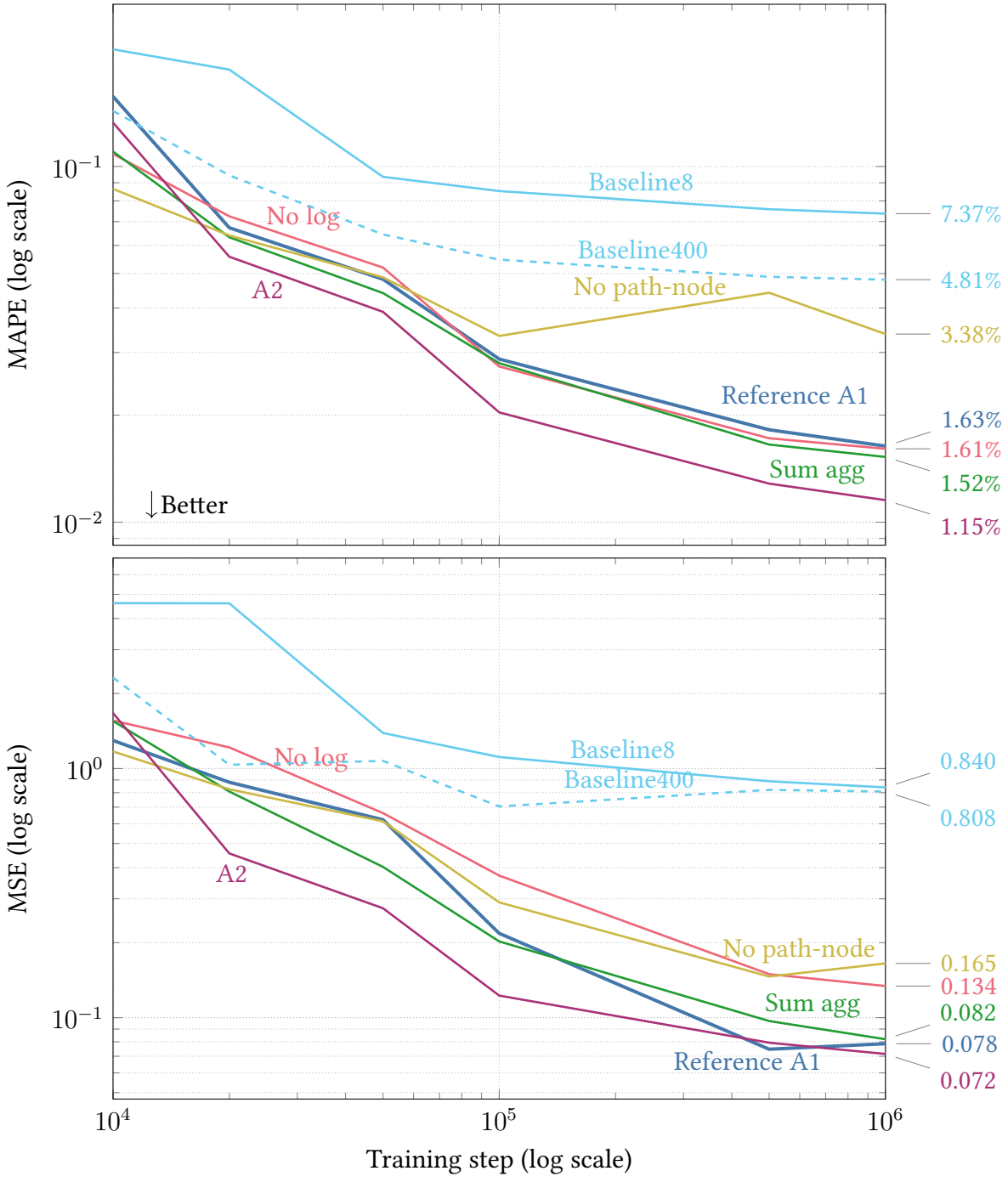


Figure 3.8 – Comparison of ablation variants' performances while increasing training samples. Surprisingly, many variants perform actually better than the reference model.

The baseline can be improved with an optimized learning strategy

By standardizing features and using an advanced optimizer and scheduler, we managed to obtain a MAPE of just 7.37% for the RouteNet baseline. This result is quite impressive, since this baseline is unable to use node features (queuing policies and weights). Adding more parameters to RouteNet speeds up the training (Figure 3.8) and leads to even better results (4.81%). Our results match the findings of the third team of the challenge from Fraunhofer HHI: by optimizing RouteNet hyperparameters, they were able to score 5.42% during the challenge.

Simpler aggregations might be better in practice

The "Sum agg" variant stays head to head with our reference on both MAPE and MSE scores. This seems to indicate that single aggregations work better than our proposed composite aggregation with five global functions (Agg). Actually, this is not really surprising in this dataset: delays are *additive* over links, hence sums are sufficient. It is highly likely that if we wanted to compute other metrics (such as loss rates), other aggregation functions would be best-suited. Moreover, in this variant the complete embeddings of size 400 can be summed. In our reference model, only one fifth of the embeddings are summed (80) due to the computation of other aggregation functions (avg, min, ...).

"Logarithmic delays" lead to less biased results

During our reference model optimization, we found that using the logarithm of the expected delay led to better MAPE scores. However, this is not entirely clear from our ablation study: the "No log" variant actually has slightly better MAPE score (0.02% difference). When looking at MSE and R^2, we see that our reference model still performs better. This confirms that the MAPE loss incentivizes models to favor *low delays*: despite a slightly worse MAPE score overall, logarithms help providing better MSE and R^2 along with better MAPE for high expected path delays.

The need for direct message passing

Just removing the direct path-node message passing in "No path-node" variant has a huge impact in the final scores. This clearly confirms the need for such direct layer between nodes and paths in our reference model. However, removing node embeddings and the related message passing operations (approach A2) actually give the better results in our analysis, both

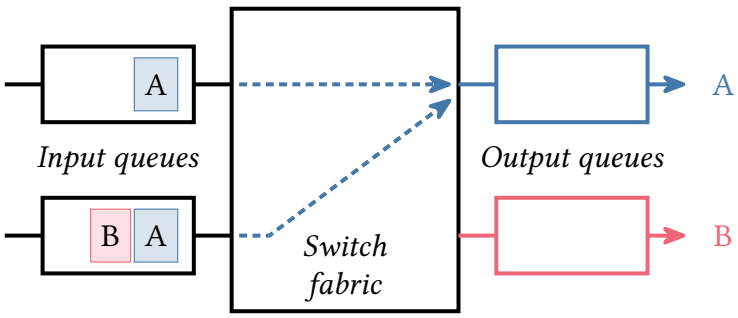


Figure 3.9 – Representation of a realistic router with both input and output queues. In this example, there is some contention between two packets routed to the "A" output link. Additionally, despite empty output queues, a packet to link "B" is "head-of-line"-blocked in the second input queue.

for MAPE, MSE and R^2. This trend is visible early in the training process (after just 20 000 samples, Figure 3.8). We have a couple of plausible explanations for that behavior.

— The challenge organizers confirmed that the simulation used to generate the dataset models one set of queues for *every outgoing port*. Node embeddings were introduced to let the model *store* some coupling between the queues of a node, for instance due to some shared resource (CPU, memory, etc.). Figure 3.9 shows an illustration of a standard router with both input and output queues [182]: in some cases, packets may be delayed despite empty output queues. In this dataset, this additional node-level status is *actually not needed* since queues are only modeled for *output* ports. This certainly makes the approach A2 more relevant.

— Updates in A2 are only being made between path and link embeddings. There are fewer updates through message passing than in the reference model, and no "intermediate" update between paths and nodes. As underlined by the poor performance of "No path-node", direct updates are essential. Future work could focus on learning functions that take several types of embeddings as input at once; for instance by computing node embeddings from related link embeddings *and* path embeddings at the same time.

3.3.5 Visualization of learned representations

Each path's predicted delay is extracted from the corresponding path embedding using the readout fully-connected layers. It is thus interesting to analyze the *content* of these learned representations to verify the model's ability to extract relevant features to our specific problem. To do so in an understandable way, we project the high-dimensional embeddings (i.e.

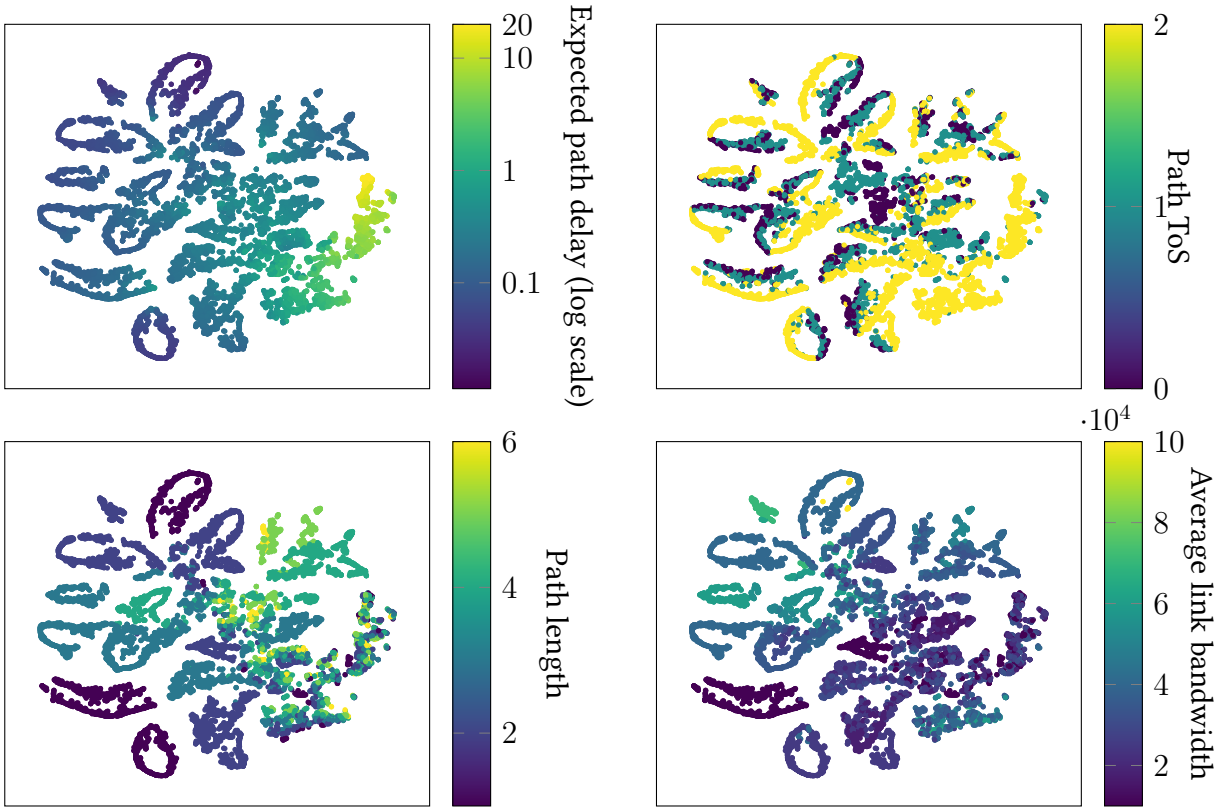


Figure 3.10 – 2-components t-SNE visualization of path embeddings, colored by expected path delay (top left), path type of service (top right), path length in number of links (bottom left) and average link bandwidth in paths (bottom right). Given the visible clusters, we can say that these characteristics are correctly embedded in embeddings.

400 dimensions) to an interpretable 2-dimensional space with *t-distributed Stochastic Neighbor Embedding (t-SNE)*, a dimension reduction technique commonly used to visualize high-dimensional representations. This technique has for instance been used to visualize embeddings of video game screens [183] in the literature. In the following, we use the best model we obtained so far (using approach A2).

Figure 3.10 shows the result of a t-SNE projection over validation samples, with different coloring maps for relevant features in the case of the delay-prediction problem (each point represents a path). In the top left plot, we color the paths according to the *expected* average delay and we can clearly see a smooth color gradient between low and high latencies, hinting that the embeddings hold enough information to give good delay predictions. The top right plot hints that paths with high delays are in part correlated with paths having a ToS of 2. This

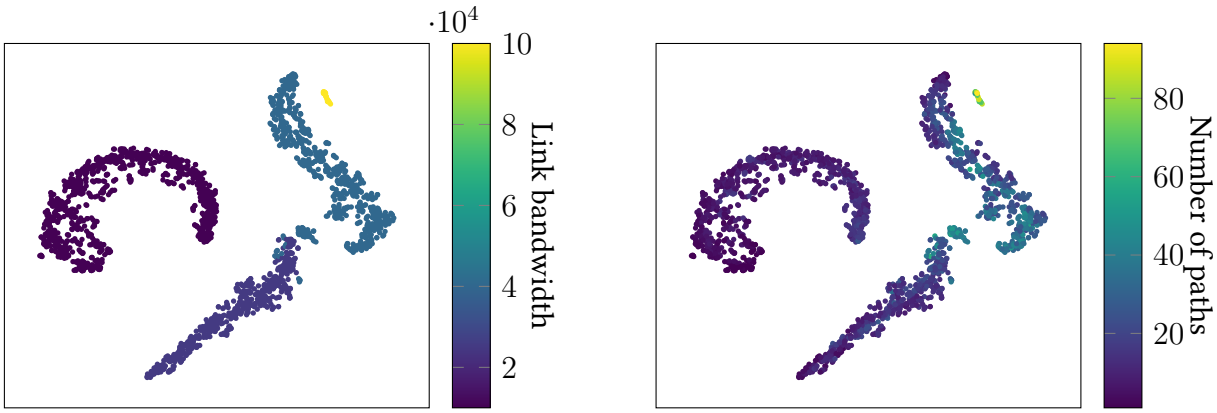


Figure 3.11 – t-SNE visualization of link embeddings colored by their bandwidth (left) and by the number of paths that use each link (right).

is not surprising: this ToS has the lowest weights for WFQ and DRR queuing policies in the provided dataset, hence paths with ToS 2 have less priority than other paths. Bottom plots show that the hidden states also contain information about *links making up each path*. We can clearly observe that paths are grouped by their *lengths* in the bottom left plot and by the average of their link bandwidths in the right plot, hinting that message passing layers are very useful to extract important signals. It is worth noting that groups exhibit a clear partitioning of ToS values.

Results are similar for the embeddings of links, as shown in Figure 3.11. On the left plot, we can confirm that information about link bandwidths is preserved despite numerous message passing updates and aggregations. On the right plot, we depict the number of paths passing through every link. We can see that links are correctly discriminated between low- and high-traffic, which is expected for the path delay estimation task.

3.4 Conclusion

In this chapter, we have presented and evaluated advanced techniques for computer network modeling using Graph Neural Networks (GNNs). In particular, we contributed a new model architecture (A1), using several bipartite graphs in message passing layers to model component interdependencies in computer network knowledge. After hyperparameter optimization, we obtained a machine learning model fitted for the task of path delay prediction with very good generalization to unknown network topologies. We highlighted in an ablation study that a simpler approach (A2) can actually give better results than our reference architecture (A1) on the evaluation dataset. The main difference between the two approaches is that A2 does not model dependencies between network queues on the same node explicitly. While these dependencies were not actually needed in the simulation data we used for the evaluation, we expect that A1 would perform better on more realistic deployments with shared resources between network queues. Moreover, while there are still many unknowns in this specific problem, our feeling is that message passing layers can still be improved to compute embeddings more directly, i.e. by avoiding *intermediate* message passing steps as in our architecture. We believe nonetheless that our contributions can both be applied to a wide range of network modeling problems. Path delay prediction is just an example of what could be predicted in a fraction of the computing time required by network simulators: our models roughly require 500 msec to compute delays over a full topology while simulators would require tens of minutes. As such, the proposed methods in this chapter could be integrated in network simulators (and emulators) for more efficient computations, at the cost of some prediction accuracy. A similar approach has recently been proposed for the simulation of integrated circuits [184].

Our analysis also highlights the clear benefits of ablation studies. When designing machine learning algorithms, it is common that one adds more components than actually needed: as shown in Subsection 3.3.4, removing one contribution at a time can lead to surprising insights. Of course, these studies can only show the results over a *specific dataset*: there is no "best" neural network architecture that can be fitted to every problem. It would be interesting to evaluate our contributions over more diverse and realistic network deployments, especially to evaluate the contribution of node embeddings proposed in A1. We revisit the application of machine learning techniques to real networks in Chapter 5.

This work has been done in the context of a machine learning challenge, where we won the first rank with an average relative error of just 1.53%. The second team from Salzburg Research had a score of 1.95% while the organizers proposed a solution with 3.88% of error [185].

Chapter 4

Internet-scale Convolutional Root Cause Analysis

Contents

4.1 Challenges of RCA from end users 75
4.1.1 Network and service agnosticism 75
4.1.2 Anomaly disentanglement 75
4.1.3 Location agnosticism 76
4.1.4 Root cause extensibility 76
4.2 Convolutional RCA with DiagNet 78
4.2.1 Methodology 78
4.2.2 The DiagNet inference model 80
4.2.3 Non-overlapping convolutions with pooling 81
4.2.4 Tailoring to specific services 83
4.2.5 Fine-grained inference via attention mechanisms 83
4.3 Evaluation with controlled faults 87
4.3.1 Experimental setup 87
4.3.2 Comparison baselines 90
4.3.3 DiagNet evaluation results with controlled faults 91
4.4 Conclusion 99

In the previous chapter we have explored how we can model arbitrary computer networks using Graph Neural Networks (GNNs) and extract relevant performance metrics. While the presented approach yields good results, it requires a very detailed description of the modeled network: its topology of course, but also the inner characteristics of network components. In the problem we discussed in Chapter 3, this included link capacities and node queuing policies for instance. We focus in this chapter on Root Cause Analysis (RCA) of user-perceived problems occurring in *large* networks such as the Internet. In these networks, detailed information regarding the topology or components is typically not available or difficult to obtain. This is especially true for *end users*, who have a very limited view of the computer network:

they often just know how to communicate with their ISP's gateway. As such, and considering the large size of the considered networks, the techniques we presented in Chapter 3 cannot be applied.

In particular, there are several challenges that must be tackled when observing the network from end users (which we return to in more detail in Section 4.1). First, as we just mentioned, RCA methods should work without needing advanced details about the analyzed network and the troubleshooted services. Second, they should be able to discriminate between potential root causes and other unrelated anomalies. Finally, they should be easily extensible, both in terms of end users, targeted services and root causes diversities.

To overcome these challenges, we propose DiagNet in Section 4.2, a distributed platform for the RCA of Internet-based services. DiagNet exploits novel convolutional techniques derived from machine learning and image processing [165, 186, 187]. It applies these techniques to data collected from both 1) end user devices and 2) opportunistically deployed probing servers that we named *landmarks*. By relying on a neural network for root cause inference, DiagNet does not require any knowledge of the low-level network fabric that connects its target services (such as routers, switches, or peering policies), and can ingest new types of network measurements without the need for retraining. Doing so, DiagNet can pinpoint root causes in locations of the Internet it never encountered before, and can easily be adapted to different types of online services with very little retraining. The principles behind DiagNet are further not limited to end user problems, and generalize easily beyond Browser-based services, to distributed business-to-business APIs.

We carefully evaluate DiagNet in Section 4.3 by injecting controlled faults into a deployment of realistic geodistributed services and clients. Our experiment involves four cloud providers in ten world regions, interdependent services, and emulated end users running in automated browsers. We compare DiagNet against state-of-the-art inference methods and show that it consistently overperforms its competitors in a dynamic context —which is typical of today's Internet services. It also delivers close to ideal performances in a static setting. In our evaluation, DiagNet yields an overall Recall@1 of up to 73.9%.

4.1 Challenges of RCA from end users

Measurements taken from end user devices are inexpensive, but the information they provide is limited. To infer some accurate diagnosis from this information, we argue that a RCA system working at Internet-scale should meet four key challenges: (i) *network and service agnosticism*, (ii) *anomaly disentanglement*, (iii) *location agnosticism* and (iv) *root cause extensibility*. We discuss each of them in turn in the following.

4.1.1 Network and service agnosticism

Internet-services rely on a wide variety of systems, sub-services, and networks to function properly. This includes data centers, cloud-providers, Content Delivery Networks (CDNs), along with diverse operators' networks. The underlying network topologies and distributed architectures of these systems are complex, continuously evolving and often unknown. We argue that an Internet-scale root cause analysis method should not assume any prior knowledge regarding the architecture of its targeted services (e.g. in terms of the cloud regions they are deployed in, or the CDNs used), or regarding the network topology on which they execute (e.g. in terms of peering-points), a property that we call *network and service agnosticism*. This largely departs from common root cause analysis relying on network tomography [95, 98, 99], bespoke methods for data centers [88, 89] and Software-Defined-Networking [83, 188], as detailed in Section 2.3.

Root cause analysis requires however some location information to pinpoint the area (e.g. cloud region, point of presence, AS) in which a root cause is likely to be located. Our choice in DiagNet is to rely on *landmark servers* to provide this location information while eschewing a precise knowledge of the underlying network. Landmark servers are easy to deploy, cheap to run and maintain, and can provide a good overview of the network health provided they are present in multiple and diverse vantage points. The intuition is that if there is a sufficiently wide deployment of landmarks, some of these landmarks will be located in the topological vicinity of targeted services, or in the path towards them, thus offering indications on the location and family of the incident impacting a user.

4.1.2 Anomaly disentanglement

By providing measurements from all over the Internet, landmark servers are bound to record a constant stream of anomalies (a drop in bandwidth here, a high latency there). Most of

these anomalies will however be unrelated to a particular end user's problem with a particular service: long delays between Paris and Brussels are unlikely to explain why a Korean student cannot access her university's web-site. An internet-scale root-cause analysis service should therefore be capable of separating spurious outliers from actual causes when analyzing an incident, a property we have dubbed *anomaly disentanglement*.

Disentangling real causes from coincidental effects is however far from trivial without any detailed knowledge of a service's internal organization (the fact that the Korean web-site does not use any resource in Europe for instance), or of the network topologies it executes on (packets circulating in Korea do not normally go through any Paris-Brussels link). Our intuition in DiagNet is that a *learned inference model* should be able to autonomously discover these hidden relationships, by inferring the internal dependencies within a service and within the network from past observations.

4.1.3 Location agnosticism

Historically, diagnostic tools operating on Wide-Area Networks have exploited the precise location of every user device (also called *client* in the following) accessing the service to pinpoint failures accurately [113], both at a geographical (from neighborhood to country) and topological level (from subnet to ISP). Obtaining such detailed data from every end-device can be difficult and even undesirable as users might refuse to share their location out of privacy concerns.

In DiagNet, we propose to circumvent the need for precise location information altogether, and argue instead that a root cause analysis model should be *location-agnostic*: the same single model should apply to every end user device that participates in the root-cause analysis service. However, we believe it is acceptable to have distinct models for distinct services, since they are definitely less numerous than possible client profiles while being possibly very diverse. We show in Section 4.2 how this level of expressiveness can be achieved by revisiting multi-layer perceptrons and convolutions in the context of network diagnosis.

4.1.4 Root cause extensibility

Because we make very few assumptions on the underlying network, and do not use detailed location information of end users, the granularity of the measurements we obtain is closely related to the deployment of our landmark servers. Many factors can however alter

the *availability* of these landmarks (e.g. failures, maintenance or saturated capacity). Conversely, if the system contains a very high number of landmarks, individual clients cannot be expected to probe every landmark in order to keep measurement overheads low. As an extreme example, it would require at least 99,000 landmark servers to cover every autonomous system [1], a number clearly too high for comprehensive probing.

To address these issues of scale and dynamicity, we require our generic RCA system to be *extensible*: trained models should be able to consume measurements from a varying number of landmarks, depending on their availability at a given time. This property allows for easy maintenance of the landmarks fleet. Since the location of a plausible root cause is directly inferred from landmarks, the better landmarks cover the Internet the more precise the resulting inference can be expected to be. A *root cause extensible* model should still however provide accurate results even when only a subset of landmarks is available. This implies a number of choices in the design of our proposal to avoid frequent model retraining.

1. Data from Regional Internet Registries as of January 1, 2021

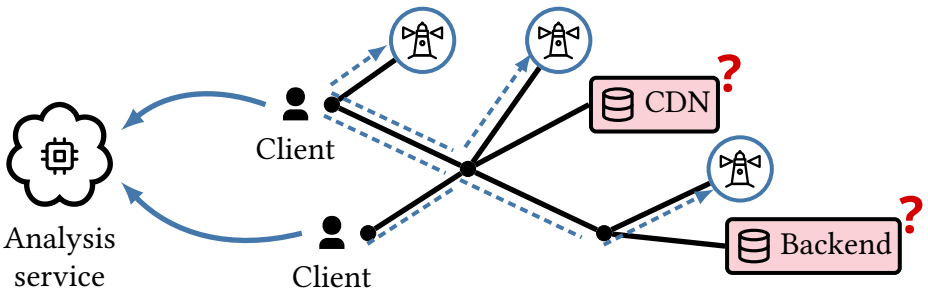


Figure 4.1 – Toy example of a topology for an online-service relying on a CDN and a backend server. Clients can evaluate links (solid lines) by actively probing landmark servers (dashed lines). Probes are sent to a root cause analysis service, which builds and provides the root cause inference model.

4.2 Convolutional RCA with DiagNet

We present an overview of DiagNet's workflow over a toy network in Figure 4.1. In this topology, we distinguish between clients (operated by end users), landmark servers (operated by any network operator) and online-service components (here a CDN and a backend). During a RCA, end users try to pinpoint the faulty network component(s) within all possible network links *and* service components. However, they lack accurate information about the network and service topologies. In an attempt to solve this issue, clients probe landmarks for network metrics, and then send the collected metrics to a centralized RCA service. We now describe the design we propose to realize such a RCA service for Internet-scale networks.

4.2.1 Methodology

We assume that clients continuously probe landmarks, obtaining a stream of network metrics. These metrics are combined with ground-truth information about faults to train a root-cause inference model using machine learning techniques. In this chapter, we assume that we have access to this ground-truth information during model training through controlled fault injection. (We will relax this assumption in Chapter 5, where we apply DiagNet to real online services.) This inference model is then provided to clients through the *analysis service*, and used to diagnose failures of the online services consumed by clients. More concretely, each client produces measurements by actively probing ℓ landmark servers, implemented as a set of stateless public services that can be provided by different ISPs or third parties, similarly to the

Speedtest global network [2]. In our implementation, we leverage modern web browser capabilities to fetch TCP statistics, latency and throughput information from the landmark servers (for a total of k metrics per landmark server and per client), to which we add some *local system features* measured on the clients themselves (e.g. client CPU and memory load). We opted to collect only this limited set of metrics in our prototype, but additional metrics could easily be added. More details are given in Chapter 5 about our implementation.

Notations The measures collected by a client c_i form a vector of m measures $\mathbf{x}_i = (x_{i,j})_{1 \leq j \leq m} \in \mathbb{R}^m$ ($k \times \ell$ measures from the ℓ landmark servers, plus the local client metrics). They provide the *features* that are fed into the RCA service. (In the following we may use the terms *measure* and *feature* interchangeably.) We assume clients also collect the QoE perceived by their users through a binary indicator, that records whether a user is experiencing a problem or not for a given service. This QoE information might be manually provided by users, or automatically estimated. It can be as simple as a page load time or can rely on a method that calculates it [189]. From that data, and assuming that a client c_i is encountering some QoE degradation, DiagNet processes the vector $\mathbf{x}_i$ of measures collected by c_i, and outputs a list of probable root causes using its learned inference model, ranked according to their likely impact on the incident being diagnosed.

Relation between landmarks and root causes In DiagNet, a root cause diagnosis combines a (possibly coarse-grained) *location* with a *fault family*, e.g. "abnormal jitter within the Amazon Web Services (AWS) US east coast region" or "high latency within local network". In practice, we use individual landmarks to represent remote locations (e.g. a landmark deployed in AWS's east coast region will represent this region), and equate fault families with the network and local metrics we collect (e.g. "upload bandwidth", or "CPU load"), as these metrics capture relatively well the behavior of the infrastructure on which services execute. As a result, the space of possible root causes of an incident is precisely that of the features we collect: $\ell \times k$ features represent remote root-causes, made up of a remote location (landmark) with a network metric, while the local features capture local root causes located at the client's end of the network.

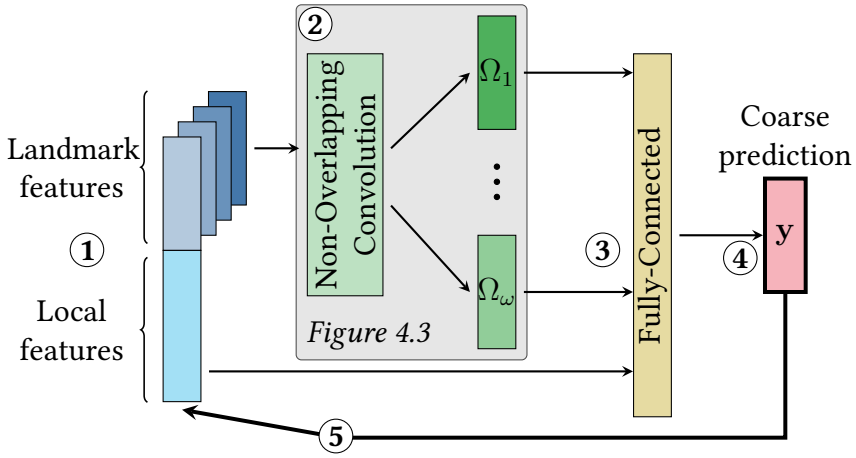


Figure 4.2 – Architecture of DiagNet. ① Landmark features are first separated from local features and ② fed in the LandPooling layer with multiple parallel global pooling operations. ③ A hidden fully-connected layer is applied after concatenating the LandPooling output with local features. ④ The coarse fault prediction is obtained by applying a series of non-linearities. ⑤ Finally, an attention model is applied on the coarse prediction to return to the feature space and propose a fine-grained fault localization.

4.2.2 The DiagNet inference model

At the core of DiagNet's analysis service lies its inference model, which we implement using a combination of neural network techniques (notably non-overlapping convolutions and pooling layers), and an attention mechanism [187, 190, 191], a technique derived from image-based classifiers that is able to relate a prediction to its inputs.

DiagNet's inference model exploits the fact that the root causes we seek to predict correspond to the set of features it consumes. This connection between inputs and outputs allows us to decompose the prediction process in two simpler steps: In a first step, we only predict the *family* of encountered fault(s) without any information on their locations (what we call a *coarse prediction* in the following). The number of fault families c is fixed (corresponding to network and local metrics), and the resulting prediction is a small-size vector $\mathbf{y} \in [0,1]^c$ of probabilities. We use the following fault families in our prototype: *nominal* (non-faulty); *uplink latency* for gateway malfunctions; *remote link latency*, *link jitter*, *link loss* and *link download/upload bandwidth* for end-to-end issues not related to the local uplink; and *local load* for client device overload. This set of fault families covers all problems generally investigated in the networking literature that can be linked to metrics obtained by users. As c (the dimension of coarse predictions) is low ($c \ll m$), we can build accurate inference models for fault families

2. `https://www.speedtest.net/speedtest-servers`

with a reasonably low number of ground-truth samples. Intuitively, coarse predictions help us disentangle features showing hidden relationships. For instance, a high TCP latency can lead to a degraded throughput [192]: in that case, the coarse prediction should return the latency as the root cause of the problem ("remote link latency"), rather the bandwidth ("link download bandwidth" despite the low throughput observed.

In a second step, DiagNet uses the vector $\mathbf{y} \in [0,1]^c$ of predicted coarse predictions to return to the input feature space of dimension m and locate the fault, in effect equating the final predicted classes with the space of input features. We use an *attention mechanism* for this step, a machine learning technique that is able to compute the *influence* of each input feature in the coarse model's prediction, usually without the need for any additional training.

The global architecture of DiagNet's inference engine is depicted in Figure 4.2. The coarse prediction phase involves the steps of ① separating landmark features from local features, ② processing the landmark features with a specific type of *convolutional neural network* (detailed in Subsection 4.2.3), ③④ processing all features with a fully-connected network and obtaining the final coarse prediction (detailed in Subsection 4.2.4). The second phase involves the step of ⑤ returning back to the input features via attention mechanisms (Subsection 4.2.5).

4.2.3 Non-overlapping convolutions with pooling

In image analysis, CNNs have been used with considerable success to classify images (see Subsection 2.4.5). Their convolutional layers extract patterns over multiple pixels by applying small filters over each pixel and its neighbors. We borrowed this idea of *pattern extraction* to extract *common patterns* between different landmarks, with some notable differences.

First, in contrast with image pixels, we want to combine measures of different nature (linked to "fault families", such as latency and bandwidth). For a landmark λ, and a client c_i, we note $\mathbf{x}_i[\lambda] \in \mathbb{R}^k$ the vector of measures (e.g. RTT, throughput) recorded by c_i w.r.t the landmark λ. For example, $x_{i,1}[\lambda]$ might store the RTT from c_i to λ, and $x_{i,2}[\lambda]$ the throughput. In this first phase, we seek to extract recurring patterns from each landmark in isolation. To this aim, we apply a set of f *non-overlapping* convolutions to each client/landmark measure vector $\mathbf{x}_i[\lambda]$. These convolutions are commonly parameterized by a kernel $\mathbf{K} \in \mathbb{R}^{f \times k}$ and a bias $\mathbf{b} \in \mathbb{R}^f$. Formally:

$$\forall \lambda \in \{1, \ldots, \ell\}, \mathbf{F}[\lambda] = \mathbf{K} \cdot \mathbf{x}_i[\lambda] + \mathbf{b} \tag{4.1}$$

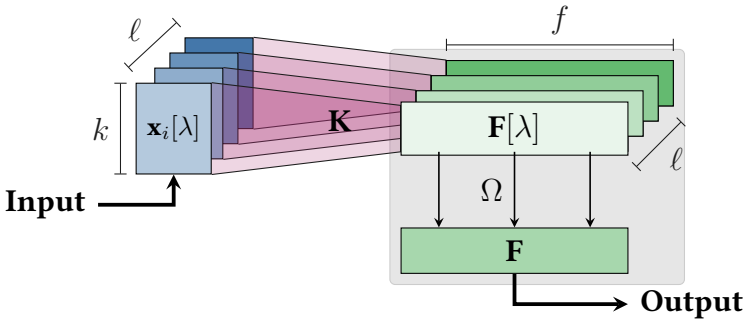


Figure 4.3 – Overview of the non-overlapping convolutional layer with pooling (LandPooling). For each landmark λ, the k features of that landmark $\mathbf{x}_i[\lambda]$ are transformed to a new feature space $\mathbf{F}[\lambda]$ of size f through a shared kernel $\mathbf{K}$. To return a fixed-size output of size f, the results for the ℓ landmarks are combined through a global Ω function, such as maximum, average or others.

At this stage, the $\ell \times k$ landmark features have been projected into a new feature space of dimension f (the number of filters). Since the $\mathbf{K}$ and $\mathbf{b}$ parameters are shared for every landmark, we believe that *common patterns between landmarks* can be learned: our model shall hopefully extract useful information about the underlying network architecture. Nevertheless, it is still required to return a vector whose size is independent of the number of available landmarks. We thus leverage global pooling layers [186], a popular mechanism to support variable-size inputs and ensure good generalization in image analysis. In our case, we apply a global function Ω on every landmark's convolution feature element-wise:

$$\mathbf{F} = \bigop{\Omega}_{\lambda=1}^{\ell} \mathbf{K} \cdot \mathbf{x}_i\left[\lambda\right] + \mathbf{b}, \mathbf{F} \in \mathbb{R}^f \tag{4.2}$$

We define this process as a new kind of neural network layer and call it "LandPooling" by reference to landmarks. An illustration of this landmark-flattening process is depicted in Figure 4.3. We note that any commutative function that can be applied with a generic number of arguments can be chosen for Ω: this is actually very similar to the GNN aggregation functions we described in Section 3.2.1. We explored several combinations of hyperparameters and kept the best configuration, listed in Table 4.1.

Table 4.1 – Notations and hyperparameters in Chapter 4

ℓ	Total number of landmarks (10)
f	Number of convolutional filters (24)
k	Number of features per landmark (5)
m	Number of features per sample ($\ell \times k +$ local features $= 55$)
c	Number of coarse fault families (7)
Ω	Global pooling operations (min, max, avg, variance, p10, …, p90)
$\mathbf{x}_i$	$= (x_{i,j})_{1 \leq j \leq m}$, input sample of client c_i
$\mathbf{y}_i$	$= (y_{i,j})_{1 \leq j \leq c}$, coarse predictions for client c_i
$\hat{\gamma}_i$	$= (\hat{\gamma}_{i,j})_{1 \leq j \leq m}$, predicted features usefulness for c_i
2 hidden layers $(512 \times 1), (128 \times 1)$ with ReLUs; optimizer: SGD with Nesterov (learning rate = 0.05, decay = 0.001)	
Auxiliary model: Random Forest (Gini impurity criterion, 50 estimators, maximum depth $= 10$)	

4.2.4 Tailoring to specific services

Similarly to typical classification tasks relying on convolutional architectures, we add a Multilayer Perceptron (MLP) after the LandPooling mechanism presented in the previous subsection. The main purpose of these additional layers is to increase the expressivity of DiagNet, by permitting a non-linear combination of the results of the global pooling and the local features resulting in coarse fault predictions. As illustrated in Figure 4.2, this MLP ("Fully-Connected layers") accepts multiple inputs: the global pooling functions $\Omega_1, \ldots, \Omega_\omega$ along with the "local features" that are independent of available landmarks. This additional expressivity is necessary to model the dependencies between services and input features.

At this point, DiagNet uses one single *general* set of final fully-connected layers to diagnose multiple services. However, such *general* model could demonstrate irregular performance if the set of monitored services is very diverse: not all Internet services have the same network requirements and dependencies. For example, while the latency is critical in multiplayer games, it might intuitively not be the case for video streaming systems where the available bandwidth is usually the bottleneck. It is thus possible to build one *specialized* DiagNet model per service to improve its accuracy, by learning a dedicated set of fully-connected layers for that service. We detail and evaluate this property in Section 4.3.3.

4.2.5 Fine-grained inference via attention mechanisms

To offer a fully extensible model, we need a mechanism to evaluate the *importance* of each input feature (each possible root cause) in the coarse-grained fault prediction. There exist

techniques to directly evaluate such importance in simple models (e.g. with decision trees), but it is well-known that this kind of *attention evaluation* is non-trivial for neural networks. While some generic techniques are applicable to any black-box model including ours [190], we instead propose to compute the *gradients* of the coarse predictions with respect to the input features. This method has already been tested in image analysis with great success [187, 191], and takes advantage of the fact that we can observe the internal weights and architecture of the coarse model ("white-box setup"). Given a coarse prediction $\mathbf{y} = (y_j)_{1 \leq j \leq c} \in \mathbb{R}^c$ (step ④ of Figure 4.2), we first compute the *ideal label vector* $\mathbf{y}^\star$ that would have been given during the training for the input sample. (For readability, and without ambiguity since we are now working on a single sample i, we removed the i indices of all notations.)

$$\forall j \in \{1, \ldots, c\}, y_j^\star = \begin{cases} 1 & \text{if} \max(\mathbf{y}) = y_j \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

We use $L^\star(\mathbf{y}) = -\sum_{j=1}^{c} y_j^\star \log y_j = -\log y_{\arg\max(\mathbf{y})}$ the cross-entropy loss that is minimal for this ideal label vector. By applying a single backpropagation step—as done during the training phase— and thanks to the complete knowledge of the coarse model architecture, we can compute the *gradient* of this loss function with respect to the input features. We make here the assumption that each partial derivative $\nabla_j = \frac{\partial L^\star}{\partial x_j}$ represents the *usefulness* of each feature j. It must be normalized according to the absolute value of ∇_j to account for both positive and negative derivatives.

$$\hat{\gamma}_{i,j} = \frac{|\nabla_j|}{\sum_k |\nabla_k|} \tag{4.4}$$

In our early experiments, we observed that the attention mechanism (Equation 4.4) used alone as a predictor of root causes gave inaccurate results. This is because a "pure" gradient-based backpropagation does not fully exploit the information provided by the MLP (④ in Figure 4.2). To overcome this problem, we give a bonus to the most relevant root causes that belong to the *same family fault* as the most probable coarse cause returned by the coarse prediction. For instance, if the model predicts a *remote link latency* problem, we use this hint to increase the predicted usefulness of every latency-related feature while penalizing other features. The weighting mechanism is detailed in Algorithm 2.

Given a coarse prediction vector $\mathbf{y}$, the algorithm first selects a set of features p related to the most significant class in $\mathbf{y}$ (in practice of the same family) at line 2. In our implementation, we manually assign each feature to a coarse class. Then, a ratio w is computed between the

Algorithm 2: Multi-label score weighting using coarse predictions

Input: Predictions $\hat{\gamma}$ and coarse predictions $\mathbf{y}$
Output: Tuned predictions $\hat{\gamma}'$

▷ *Isolate the best coarse prediction*
1 $\phi \leftarrow \arg\max(\mathbf{y})$
2 $p \leftarrow \{$*indices of features with same family as* $\phi\}$

▷ *Compute the relative weight*
3 $w \leftarrow \frac{y_\phi}{\sum y_i}$
4 $s \leftarrow \sum_{j \in p} \hat{\gamma}_j$
5 **if** $s = 0 \vee s = 1$ **then** ▷ *Extreme case*
6 $\quad \hat{\gamma}' \leftarrow \hat{\gamma}$
7 **else**
8 $\quad$ **foreach** $j \in p$ **do** $\hat{\gamma}_j{}' \leftarrow \hat{\gamma}_j \frac{w}{s}$ ▷ *Bonus*
9 $\quad$ **foreach** $j \notin p$ **do** $\hat{\gamma}_j{}' \leftarrow \hat{\gamma}_j \frac{1-w}{1-s}$ ▷ *Penalty*

model's confidence in its coarse prediction and the sum s of the usefulness of related features (line 4). The tuned $\hat{\gamma}'$ are computed in line 8 and line 9. By construction, Algorithm 2 always returns a probability distribution vector.

Ensemble model averaging

The architecture of DiagNet is designed to naturally extend to new landmarks without retraining. As a result, however, it loses information compared to more direct methods such as random forests (Subsection 2.4.4). To further boost our solution, and reap the benefits of both worlds, we use *ensemble model averaging* as a last optimization step, a popular method to combine multiple specialized models [193]. (Again, this is similar to what we have done during the GNN challenge in Chapter 3.) We average the tuned attention predictions with another prediction from an *auxiliary* model, designed to be simpler and specialized in *known root causes*. We chose a random forest approach as our auxiliary model and give more insights about this choice in the next section.

We briefly formalize this last optimization. Let $\mathcal{U}$ be the set of unknown landmark's features, not seen during training. Let $\hat{\gamma}'$ and $\hat{\alpha}$ be the prediction obtained from the tuned attention mechanism and the auxiliary model, respectively. We define $w_\mathcal{U}$, the probability that the

root cause is explained by an unknown landmark's features, as predicted by the tuned attention mechanism. Observing that $w_{\mathcal{U}} \in [0, 1]$ by definition, we compute the final prediction of DiagNet after model averaging using:

$$w_{\mathcal{U}}\, \hat{\gamma}' \;+\; (1 - w_{\mathcal{U}})\, \hat{\alpha} \quad \text{with} \quad w_{\mathcal{U}} = \sum_{j \in \mathcal{U}} \hat{\gamma}'_j \tag{4.5}$$

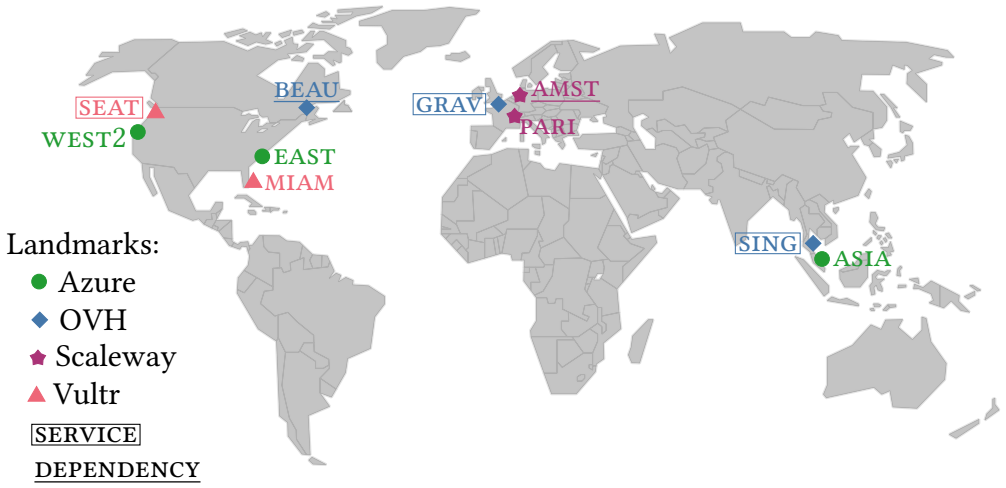


Figure 4.4 – Locations of landmarks and services in our multi-cloud experimental deployment. We deployed emulated clients in every location (region). The East, Grav and Seat landmarks were hidden during training.

4.3 Evaluation with controlled faults

To evaluate DiagNet, we deployed a multi-cloud geo-distributed network of clients, online services, and landmark servers. In this section, we present our evaluation methodology and introduce baselines offering similar properties as DiagNet.

4.3.1 Experimental setup

Deployment In order to train and evaluate the root cause analysis models, we deploy one landmark and multiple clients in each of the ten regions listed in Figure 4.4. Three of these regions (grav, seat, sing) also host mock-up online services to evaluate the QoE with diverse setups (Table 4.2). Some services only require a single HTML file, while others download resources from distant regions. (Recall that the nature of individual services, and hence the relations between regions and services are hidden during model training.) Region locations are chosen to benefit from both the diversity of a worldwide multi-cloud deployment and the proximity of co-located regions for fault localization. At the time of writing, our experimental pipeline was made of roughly 5000 lines of Python and Go code. (We used Tensorflow 1.13.1 for neural network training [194].)

Table 4.2 – Online services used in experiments.

Service	Description
1. single	Static HTML page with no dependency
2. script.far	Requires a JS file in BEAU
3. script.cdn	Requires a JS file from nearest region
4. image.local	Loads a 5MB image from same server (using the same HTTP connection)
5. image.far	Loads a 5MB image from BEAU
6. image.cdn	Loads a 5MB image from nearest region

Table 4.3 – Local and landmark features

Type	Name	Collection method
Local	CPU load	Obtained from the operating system
Local	Avail. memory	Obtained from the operating system
Local	Total memory	Obtained from the operating system
Local	Disk I/O	Obtained from the operating system
Local	Edge RTT	Average ping to first traceroute hop
Landmark	Landmark RTT	Average of WebSocket pings
Landmark	Jitter	Deviation of WebSocket pings
Landmark	Download throughput	Measured with large HTTP GET
Landmark	Upload throughput	Measured with large HTTP POST
Landmark	Loss rate	Extracted with `getsockopt`

Table 4.4 – Relation between features and coarse fault families

Fault family	Associated fault family
Latency	Landmark RTT
Jitter	Landmark jitter
Download bandwidth	Landmark download throughput
Upload bandwidth	Landmark upload throughput
Loss rate	Landmark loss rate
Uplink	Edge RTT
Load	(Other local features)

Landmark features Live network metrics are obtained by querying each landmark through web endpoints (Table 4.3). To estimate download and upload bandwidths, we measure the duration of large GET and POST HTTP requests. We avoid the typical overhead of HTTP requests for RTT estimation by upgrading the connection to WebSocket. Finally, we use the `getsockopt` linux syscall on each landmark server to make raw TCP statistics available to landmarks' clients. We mainly extract the ratio of reordered and retransmitted packets from these statistics. For more details about the collection of landmark features, please refer to Subsection 5.1.3.

Fault injection Clients are implemented using automated Chromium browsers that periodically fetch network features from landmarks and visit mockup services to evaluate their QoE from timings returned by the `window.performance` API. Each client also periodically measures the RTT to its local network gateway. We inject artificial network faults into each cloud region using Linux `tc` Network Emulator rules, a realistic and popular emulation method for reproducible experiments. QoE information is then used to flag samples as "nominal" or "faulty" with the (known) root-cause ground-truth as class label for model training. As presented in Subsection 4.2.4, we first train a *general* DiagNet model based on eight mockup services, and then build a *specialized* model for each service by retraining only the last fully-connected layers. All the scores presented in this evaluation section are computed using the specialized models. We give more details about this approach in Section 4.3.3, along with an evaluation of training cost.

Root cause extensibility We train and test root cause models on two different sets of landmarks to assess the extensibility capabilities. For all experiments in this section, three landmarks are "hidden" during training: EAST, GRAV and SEAT, named *new landmarks* and denoted by $\star$ in this chapter. We chose these landmarks due to their immediate proximity to the mockup services and several injected faults, and limit the availability of their features to model evaluation only. They are opposed to *known landmarks* (the remaining seven). In doing so, we reduce the quality of the measures available to training, and make faults located close to the hidden landmarks particularly hard to detect, as neither these faults, nor the measures they impact most are used to train the models.

Dataset We ran our experiment during the last two weeks of December 2019, using different hours of day and days of week to ensure a large coverage of traffic and congestion patterns

between cloud providers. 213 000 of "nominal" samples along with 30 000 "faulty" samples were collected during our experiment. 80% of each kind of samples are used for training and validation, while the remaining 20% are reserved for evaluation. We injected the following six families of faults into regions involving services (Seat, Beau, Grav, Amst and Sing), leading to diverse fault scenarios:

1. Download bandwidth shaping (capped at 8 Mbit/sec)
2. Additional service latency (50 msec)
3. Additional gateway latency (50 msec)
4. Additional jitter (up to 100 msec)
5. Increased packet loss (8%)
6. Large CPU stress (mainly impacts Chromium's navigation in clients)

Faults are uniformly distributed between regions and families to avoid bias towards more frequent root causes. In some scenarios, we inject multiple faults at the same time, but at most one fault is the real root cause for QoE degradation in a given region. In many cases, we observed that the QoE is not degraded despite the injected fault(s): for instance, the QoE of a small HTML website is not affected by shaped bandwidth or CPU stress. We thereby flag these samples as "nominal" despite the injected fault.

As explained just above, three landmarks out of ten are hidden during training: samples with faults at these landmarks are forced to appear only in the evaluation set. Of these samples with "hidden faults", a fraction does not exhibit any QoE degradation, and are therefore flagged as "nominal". At the end, 23% of the testing samples with degraded QoE involve root causes in *hidden regions*, that are not seen at training time.

4.3.2 Comparison baselines

We propose two baselines that use common classification or outlier detection models and offer the same three key properties as DiagNet, namely location and topology agnosticism, along with root cause extensibility.

e-Random Forest: Extensible Random Forest Classifier

To train an extensible random forest (Subsection 2.4.4), we naively set the features dimension to the maximum possible size, and we set to zero the missing landmarks values in each sample. (This is a common approach when some decision trees' inputs are not available.) We

also add a special "unknown" output class, selected when the given sample is classified as "nominal". We evenly redistribute the score obtained for this special class to every other class: this allows non-trained faults to have a non-null score in the final prediction. This model is used as the auxiliary model in the ensemble averaging optimization presented in Subsection 4.2.5.

E-Naive Bayes: Extensible Naive Bayes Classifier

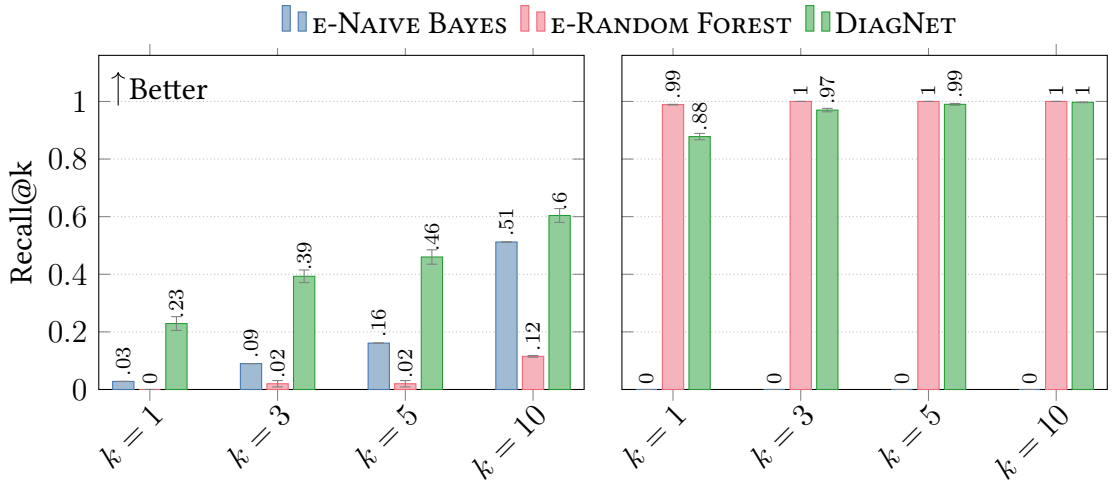
We propose another approach for extensibility, based on the merger of several probability distributions and Naive Bayes classifiers (Subsection 2.4.2). Compared to standard Naive Bayes approaches, it is highly probable that one particular root cause C_k has not been seen during the training phase, and the prior probability for class k is unknown. Thus, we define the prior probability of each class C_k as $P(C_k) = 1$ for every root cause. This also has the positive side-effect of canceling bias with unbalanced datasets. Then, we use KDE [144] to estimate likelihood probabilities $P(x_{i,j} \mid C_k)$. In contrast with the more common Gaussian model, the KDE greatly increases the expressivity of this baseline model. Finally, we build *generic aggregate likelihoods* for unknown features or new classes. For each measure family t collected in landmarks (such as uplink latency or download bandwidth), we build a *generic* likelihood $P(x_{i,t} \mid C_t)$, defined as the KDE of the union of all measures *for every landmark available during training*. This generic likelihood is then used when no specific likelihood is available for a given feature or class.

4.3.3 DiagNet evaluation results with controlled faults

Recall evaluation with the Recall@k metric

The final goal of root cause analysis is to return a *ranked list of probable causes*. We use the Recall@k metric for model evaluation: for a set of known real causes and a ranking method, the Recall@k is the number of correctly predicted causes *within the first* $k \geq 1$ *causes* divided by the total number of actual causes. A high recall would demonstrate that a method of ranking (model) can be useful to users, being able to quickly pinpoint the real root cause of a QoE degradation among a set of possible causes. In our setup, we argue that it is acceptable to return the expected cause within the first $k \leq 5$ predictions from 55 possible root causes.

Figure 4.5 shows the Recall@k for two types of fault: faults injected near *new landmarks* in (a), and faults injected near *known landmarks* in (b). (As a reminder, new landmarks' features are hidden during training.) DiagNet offers the best recalls for faults near new landmarks (a),



(a) New landmarks (b) Known landmarks

Figure 4.5 – Recall@k for failures near new and known landmarks, for different values of k. DiagNet consistently overperforms its competitors on new landmarks, while delivering close to ideal performances on known ones. By comparison e-Random Forest works perfectly for known landmarks, but degrades starkly on new ones, while e-Naive Bayes offers reasonable performance with new landmarks but is lost on known landmarks.

thanks to its attention mechanism that fully exploits the information coming from the new features without additional training. Our proposal also shows close to ideal results for faults injected near known landmarks (b), thanks to the "hybrid" mode of operation offered by ensemble averaging (Subsection 4.2.5). The combined Recall@1 for DiagNet (including faults near known and new landmarks) is 73.9%, a very good score given the high number of probable root causes. By contrast, e-Random Forest works perfectly for known landmarks, but its recall degrades dramatically in the case of new landmarks; the described extensible random forest model essentially gives *completely random predictions* in this second case. It reaches nonetheless a combined recall of 77%. Conversely, e-Naive Bayes shows extremely poor results for known landmarks, with its best score reached for high values of k in (a); its combined recall is only 5.3%. This is due to a severe bias towards new features that systematically get high prediction scores *even for known failure types*.

Figure 4.6 presents each recall per family of fault and per location. We clearly see e-Naive Bayes' bias towards some fault families and new landmarks grav and seat. DiagNet is the only model showing good recalls for every family of fault near both known and new landmarks regions.

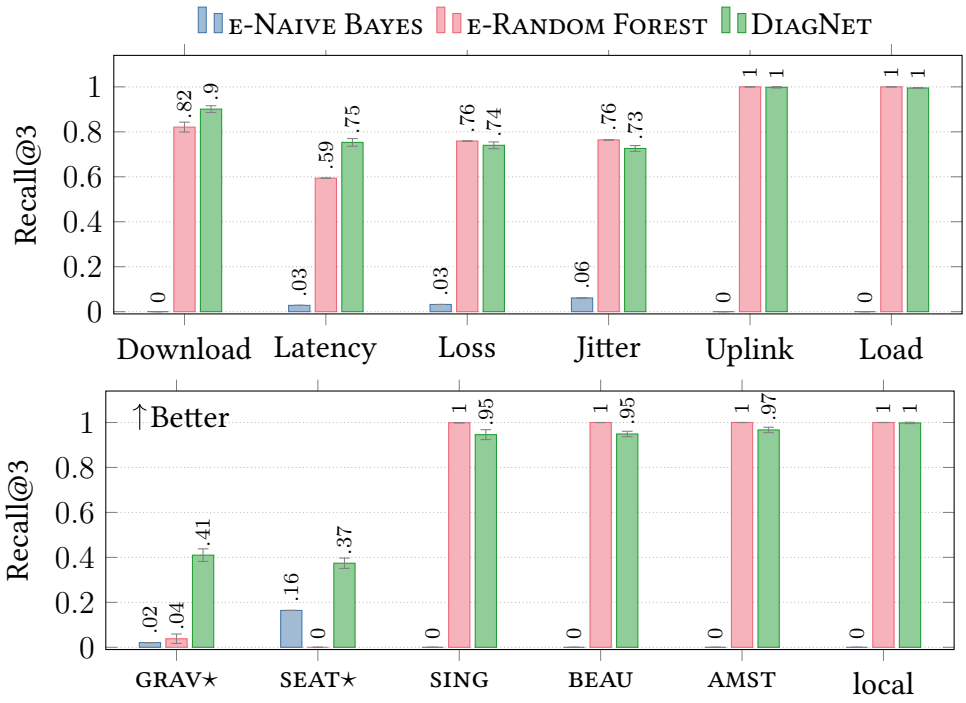


Figure 4.6 – Recall per fault family (top) and fault region (bottom). Regions hidden during training are indicated with a star ⋆. Again, e-Random Forest gives best results for known landmarks, but DiagNet is the only solution able to adapt to the different scenarios, with close to optimal results for local faults.

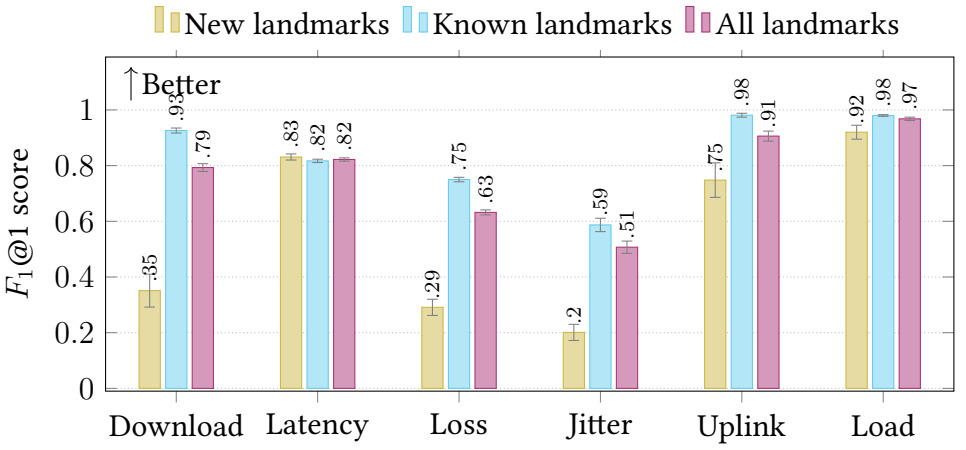


Figure 4.7 – Performance of the coarse classifier, with an overall accuracy of 0.70 ± 0.013 for faults near new landmarks and 0.85 ± 0.005 for faults near known landmarks (ratio of correct predictions over evaluated samples).

Diving into the performance of the coarse classifier

To shed light on the results just presented, and validate the design of DiagNet's convolutional approach, we evaluated the F_1 score of DiagNet's coarse classifier (corresponding to step ④ in Figure 4.2) for each fault family (Table 4.4).

$$F_1 = \frac{\text{True positives}}{\text{True positives} + \frac{1}{2}\left(\text{False positives} + \text{False negatives}\right)} \tag{4.6}$$

As explained in Subsection 4.2.5, the output of this coarse classifier is used by the attention mechanism and eventually averaged with the random forest classifier to provide the final classification. Thereby, we expect this coarse classifier to be critical in the final root cause ranking. Figure 4.7 presents separately the results for samples impacted by faults near known landmarks and near unknown landmarks. As expected, samples with faults close to known landmarks are overall better classified than samples with faults close to unknown landmarks. The F_1 scores also show that some fault families are easier to classify than others (Latency, Uplink and Load). Overall scores demonstrate the value of DiagNet's convolutional neural network.

Effect of client diversity

To validate the *location agnosticism* property of DiagNet, we gradually increase the location *diversity* of participating clients. (Put differently, we vary the *number of regions with active clients submitting samples*.) The results of that experiment are shown in Figure 4.8, with the aggregate Recall@5 for all families of faults near newly-introduced landmarks. For completeness, we note that we measured the Recall@5 for every possible combination of active clients to eliminate potential discrepancies between configurations. The key take-away is that DiagNet is able to deliver the best predictions for all scenarios of client diversity, showing great stability. Our results hint that DiagNet is truly able to distinguish between dissimilar clients (e.g. clients in America vs. Asia or Europe).

In contrast, the e-Naive Bayes model prefers to handle few regions at a time. This is explained by the *KDE merge* process of this baseline: with more diverse clients, merged KDEs are "flattened" and converge to uniform distributions, biasing the model towards unknown features as seen in Figure 4.5 and Figure 4.6. e-Random Forest is less sensitive to client diversity, with only a slight recall increase—probably due to the larger number of available training samples.

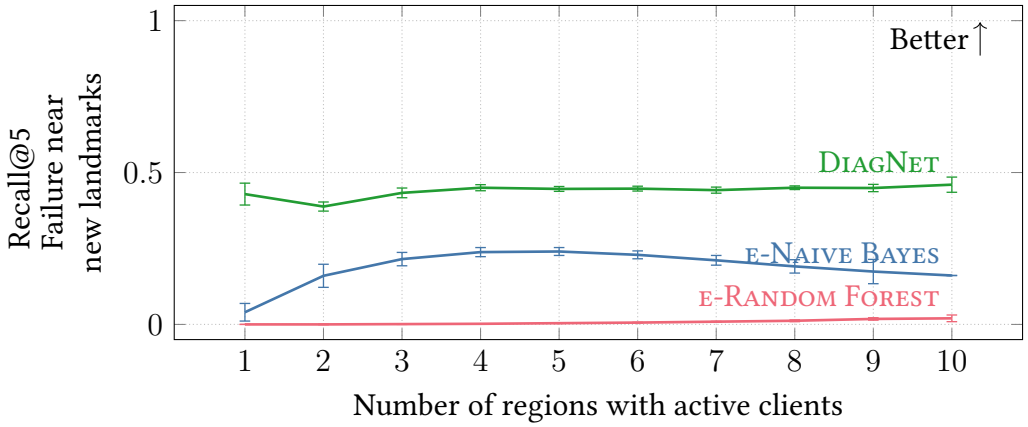


Figure 4.8 – Comparison of models' performance for new landmarks with increasing diversity of clients (we modified the number of regions with active clients). DiagNet can scale very well for landmarks unseen during training.

Accuracy with simultaneous faults

The critical task of a root cause model is to find the *correct* root cause when multiple simultaneous anomalies are detected. DiagNet uses specialized models *for each web service*: some services might be impacted by a given anomaly, when other services might not. To evaluate this property, we simultaneously injected two latency faults near the Beau and Grav regions and quantified the number of predictions towards each region. Figure 4.9 gives the detailed results for DiagNet's general model (a) and for the specialized model of each service (b). The results for the general model are quite poor, with a confusion between Beau and Grav regions and a lot of other faults predicted. Specialized models provide sharper predictions, with a recall of 76% for the latency root cause near Beau, 28% for the latency root cause near Grav—a region unseen during training—and 71% when both faults are actually root causes. This analysis confirms the benefit to specialize models for each monitored service.

Training cost of new service models

To remove the need for the complete retraining of DiagNet when new online services are being added, we assume that the weights learned in the non-overlapping convolution are shared between services, as they extract global network features. We also assume that the final layers of DiagNet capture the specific "behavior" of each analyzed service. We now give the details of the DiagNet learning procedure, that has been used in the whole evaluation section and is based on these assumptions. We first train a *general* model on a subset of eight

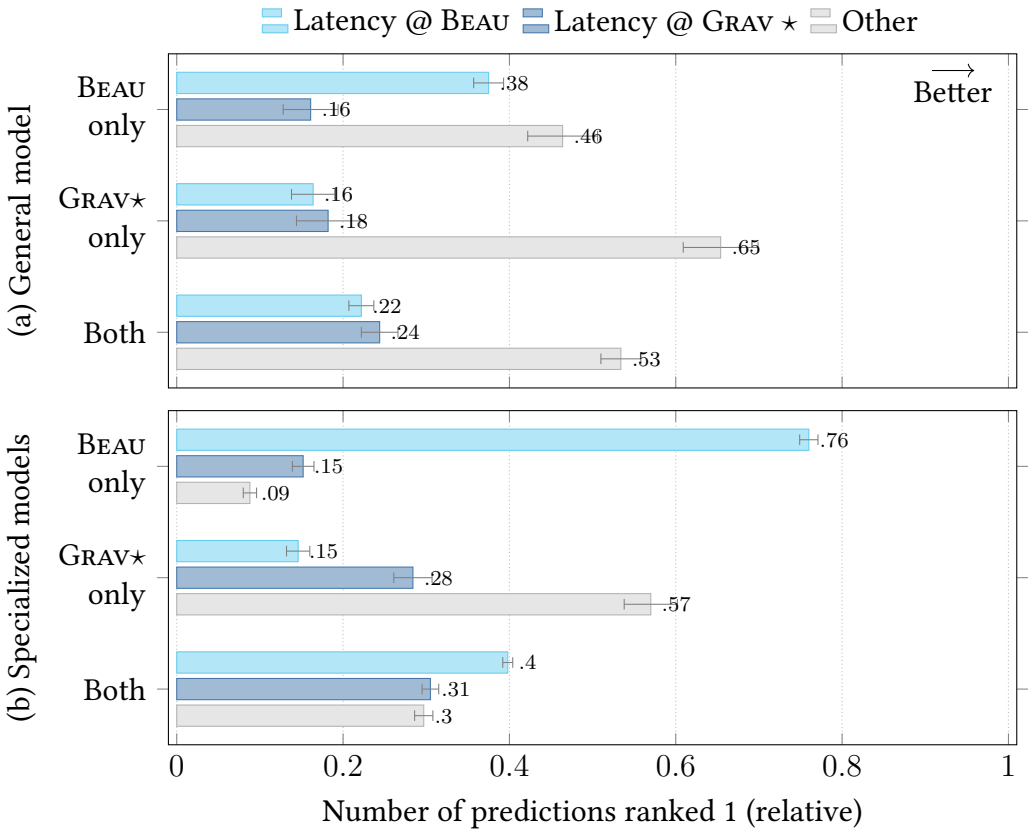


Figure 4.9 – Predicted root causes for general and specialized DiagNet models, with simultaneous injected faults (latency near the Beau and Grav regions). The y-axis shows the *relevant* root cause at the origin of the QoE degradation on the evaluated service (this is the "control" variable). Grav metrics were not used during training (⋆). Overall, the specialized models (lower chart) deliver better predictions for all three combinations of relevant faults.

initial services, taking the union of services' problems as the expected model output. Then, we freeze the weights of the non-overlapping convolution, and optimize the weights of the final layer for each of a set of additional services, not contained in the original set. This leads to one *specialized* model per additional service. With the hyperparameters from Table 4.1, the general model must learn 215 312 parameters while specialized models hold 65 664 trainable parameters. The remaining 149 648 parameters are set to their value in the general model.

Learning losses on training and validation sets are plotted in Figure 4.10 through learning epochs, for the general model and for a small subset of service models. We consider that the training is done when the validation loss is no longer decreasing (an indication of overfitting). Although the training time on the general model is higher (around 20 learning epochs), service models converge in less than five epochs on average. This indicates that specialized service models per service are easy to learn once one global model exists. The similarity between training and validation losses also hints that the specialized models offer good generalization. We note that while the general model can be trained with a subset of services and landmarks, it can later be generalized to more landmarks and services with minimal re-training time: on a commodity laptop using the CPU only, it takes 32 seconds to train the general model and 4 seconds to train each service model. Furthermore, root causes are inferred in 45 ms on average.

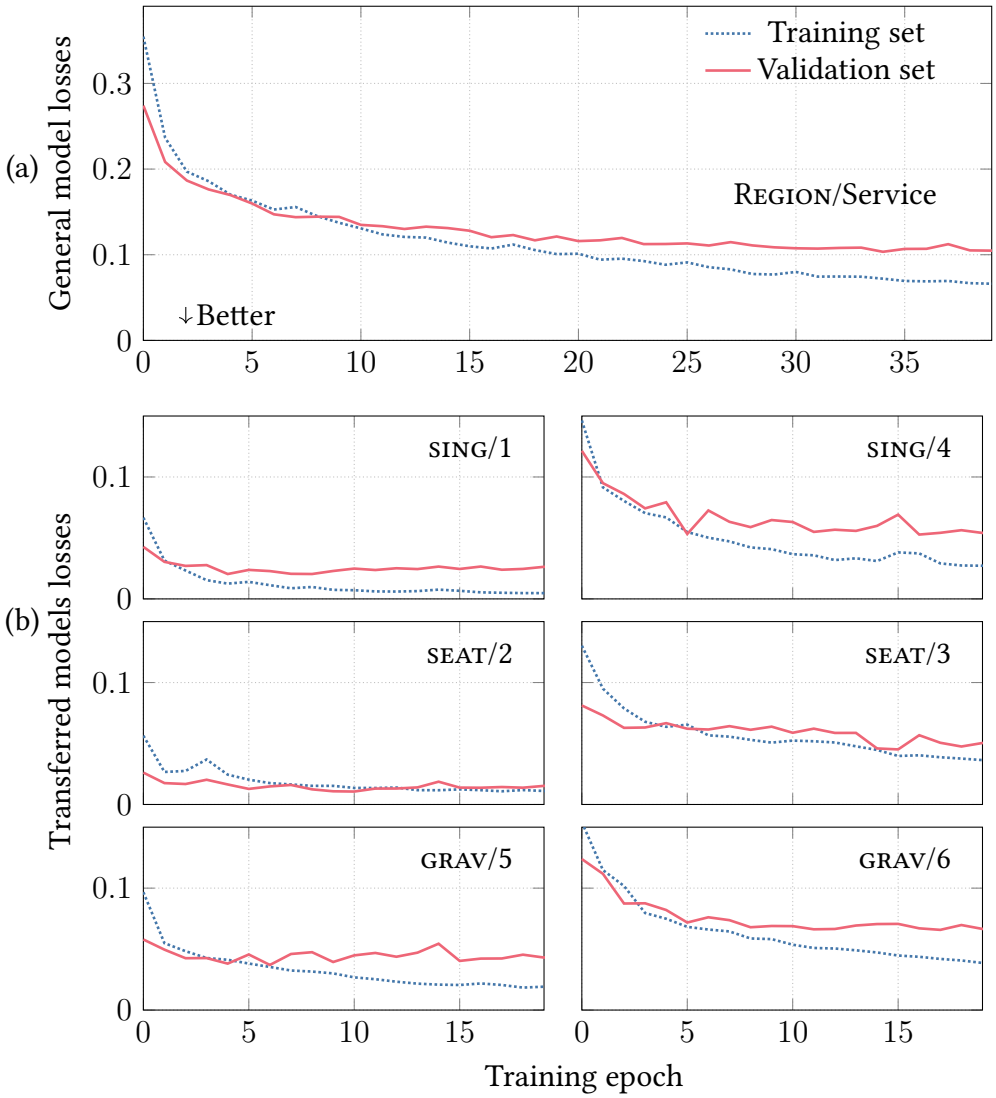


Figure 4.10 – Evaluation of model transferability: after building a general model on eight services chosen uniformly at random (a), it is possible to build specialized models for other services while freezing convolutional kernels (b). A relatively low loss rate is quickly reached for most services.

4.4 Conclusion

Root Cause Analysis at the scale of the Internet is recognized as a hard problem given the decentralized design of the global network. In this chapter, we presented DiagNet, a generic and extensible RCA method based on active landmark probing. DiagNet does not depend on any prior knowledge of either the network topology or targeted services, which makes it practical for clients having a very limited view of the Internet topology past their ISP gateway. The inference model of DiagNet leverages a special kind of Convolutional Neural Network (CNN) specially crafted for metric pattern recognition. It also uses a simple attention mechanism using backpropagation gradients, along with several other optimizations (multi-label score weighting and ensemble model averaging).

We evaluated our proposal on a geodistributed platform with controlled injected faults, and compared it to two baselines. While we demonstrated that Random Forest models are very insightful when diagnosing in a *static* setting, their accuracy starkly decreases when landmarks (hence input features) are selected at *inference* time. Similarly, the other extensible Naive Bayes classifier baseline was unable to cope with varying inputs. By contrast, DiagNet shows good results in *all* scenarios, i.e. it can diagnose local and remote failures in static and dynamic network settings, even with very diverse participating clients from across the globe. This clearly highlights the benefits of CNNs for networking applications, a technique that is mostly known for its applications in image analysis and classification.

While we believe that our proposal answers the Internet-scale RCA challenges we listed in Section 4.1, this is only a first step towards a fully-fledged RCA system. Our evaluation was actually limited in size and service diversity: this allowed us to control the experiments, but it might have missed critical observations applicable in larger computer networks. For instance, we had access to fault ground-truth during the evaluation, but how can we access and verify *real* faults' labels in practice? Can DiagNet scale to thousands of clients, landmarks and diagnosed services? We attempt to answer these questions in the next chapter.

Chapter 5

From theory to practice: Root Cause Analysis from web browsers

Contents

5.1 Data collection from browsers 103
5.1.1 Motivation and related work 103
5.1.2 Architecture of DiagSys 106
5.1.3 QoS measurements provided by landmarks 107
5.1.4 Third-party QoE monitoring from browsers 110
5.2 Insights from our collected dataset 113
5.2.1 Dataset case studies 114
5.2.2 Dataset faults analysis 119
5.2.3 Our proposal: labelling from historical data 120
5.3 Applying DiagNet to DiagSys's dataset 124
5.3.1 Changes to the RCA approaches 124
5.3.2 Pinpointing fault families 125
5.3.3 Complete root cause analysis at Internet scale 126
5.3.4 Discussion and remaining open questions 128
5.4 Conclusion 130

To provide insightful reports, Root Cause Analysis (RCA) solutions should ideally reflect the Quality of Experience (QoE) perceived by end users when they use online services such as websites and web APIs. Because QoE problems are often explained by causes near end users [121, 195, 196], many RCA solutions have been implemented at the network's edge, by taking the viewpoint of either the home gateway, the browser, or by using dedicated tools running on clients [114, 116, 197]. Although the location of the measuring probes within the network is critical, we believe that the device used (PC, smartphone …) and the *execution environment* (i.e. the web browser or a dedicated application) are also essential to correctly capture a user's QoE. As others before us, we therefore argue that any RCA solution should ideally include parts that directly run on users' devices to provide accurate QoE measurements.

This user-facing software should however be particularly *easy* to deploy and use, remain non-intrusive and incur a minimal network overhead.

With this goal in mind, we explore in this chapter how *modern web browsers* can be used to take RCA measurements while adhering to the above principles, and present the year-long dataset we have collected using the resulting measurement platform with the help of volunteers. Most on-line services used by end users run within browsers, where measurement scripts can be easily deployed using JavaScript with little to no user interaction. Moreover, web browsers make it easy to target a wide range of devices, from PCs to smartphones through gaming consoles. They however come with a number of technical and security restrictions that limit measurement capabilities, especially in modern web browsers with limited access to low-level network primitives.

We begin by detailing how one can still obtain relevant RCA and QoE metrics from modern browsers in Section 5.1. In practice, we follow the path of Chapter 4 and collect relevant network metrics using cooperating *landmark servers*. We complement these network metrics with *service probes* to estimate users' QoE without requiring the cooperation of third-party online services. We also use landmarks to provide additional insights by actively probing other landmarks and third-party services with automated web browsers. (This helps to make up for the intermittent connectivity of end users in practice.) Using these methods, we have built a dataset of more than two million measurement samples spanning over more than a year. In Section 5.2, we illustrate the interest of this dataset with some selected case studies, demonstrating that web browsers can indeed provide a rich platform for live metric collection.

Finally, Section 5.3 explores how DiagNet (presented in Chapter 4) perform on this collected dataset. Our main challenge in this work was to *infer* the root causes behind real Internet failures: ground truth remains difficult to obtain in practice, especially when few end users are impacted. To cope with the general unavailability of ground truth, we propose a statistical model to label our measurement samples based on the collected RCA and QoE metrics. In particular, our model leverages outlier detectors over complete historical data to label most probable root causes. DiagNet is then evaluated with *no* access to this historical data. Although perfectible, we believe this approach offers a promising basis for further work and highlights the experimental challenges that come with the use of live Internet-scale measurements.

5.1 Data collection from browsers

In order to build RCA systems targeted at end users, it is important to collect network and QoE metrics close to their home network and computer devices. We first review available techniques for such metric collection and then detail DiagSys, a collection framework designed for recent web browsers.

5.1.1 Motivation and related work

For network operators, it is relatively easy to deploy servers within the Internet infrastructure to measure inter-AS network metrics. Yet, it has been shown that most connectivity issues are due to the last links towards end user devices [121, 195, 196, 198]. This is not surprising: these last links are often shared between ISP customers, may use old technologies such as Digital Subscriber Line (DSL) and are easily affected by the environment (weather, road works, etc.). As an example, during Covid-19 lockdowns, it has been measured that last-mile links suffered from higher congestion levels than before [199], suggesting that these links lack capacity in general. Wi-Fi [21] is also widely used to connect devices to Internet in home setups, yet Sunderasan et al. have shown that most setups exhibit poor performance due to unoptimized access point placement [195]. Monitoring last-mile links is the reason why early works have focused on capturing network metrics from *within* home networks. Home gateways [200, 201] are particularly interesting here: they can be managed by the ISP without user intervention, are always online and provide a good overview of the whole home network. SamKnows [202] and BISmark [203] are two examples of academic deployments of instrumented home gateways. They allowed participants to run numerous active and passive measurements that led to new insights from home networks. The RIPE Atlas [204] deployment takes another approach: it leverages custom hardware installed in volunteers' home networks to provide worldwide measurement campaigns to researchers.

While home network measurement probes can capture last-mile metrics for residential users, they cannot measure *mobile traffic* over broadband networks. Moreover, they can only *infer* QoE from network metrics (or "QoS metrics") as the behavior of final applications remains hidden due to use of cryptography in web traffic. Some methods have been proposed—for instance inferring live video traffic QoE [205, 206]—but they remain limited to specific services and cannot generalize to all web services. With the limited available computing power, it is difficult to *emulate* web services in measurement hardware. As an example, Mirage [207] (deployed in both SamKnows and BISmark) parses web pages and download their resources

Table 5.1 – Summary of metric collection tools deployed in home networks.

Method	**Type**	**Location**	**Metrics**	**Status**
Glasnot [115]	Java	Browser	QoE	Unsupported
Wehe [214]	Software	Mobile	QoE	Supported
Acqua [215]	Software	Mobile	QoS, QoE	Supported
Netalyzr [114]	Java	Browser	QoS	Unsupported
Fathom [116]	XPCOM Ext.	Browser	QoS	Unsupported
Mirage [207]	Software	Gateway	QoS, QoE	Retired
ATLAS [204]	Software	Network	QoS	Supported
HomeNet [208]	Software	Device	QoS	Retired
HoBBIT [209]	Software	Device	QoS	Retired
HostView [210]	Software	Device	QoE	Retired
NDT [197]	JavaScript	Browser	QoS	Supported
DiagSys	JavaScript Ext.	Browser	QoS, QoE	Supported

to estimate the real QoE; but it gives inaccurate estimations for dynamic web services using JavaScript. To lift these limitations, it has been proposed to install native measurement services directly *within* end user devices: this makes it possible to take advantage of the additional computing power and the proximity to applications of user devices. For instance, HomeNet [208] and HoBBIT [209] provided Java and Qt measurement applications, respectively. This choice offers coverage for a large range of operating systems. System applications further provide access to low-level network primitives, allowing for fine-grained QoE estimation[87, 140, 195, 196, 210, 211, 212, 213]. Yet, these software programs require *manual installation* from end users and their deployment remain challenging in practice, requiring support for multiple operating systems including mobile operating systems, leading to important maintenance costs and introducing additional security and privacy risks, etc.

For all these reasons, measurements based on web browsers have gained popularity over the last decade: most Internet services used by end users run within browsers (including mobile ones), where measurement scripts can be deployed with little to no user interaction. Most proposals leverage JavaScript APIs to measure the QoS of the connection [197], and several metrics have been proposed to estimate the web QoE [189, 216]. As an example, Da Hora et al. [189] propose to monitor the *placements* of objects within webpages to detect *when* the visible objects stop loading, yielding the recognized Above The Fold (ATF) QoE metric. In 2010, Glasnot [115] used a Java applet to detect traffic differentiation from within browsers; this choice allowed users to measure differentiation in a real web environment. The same approach

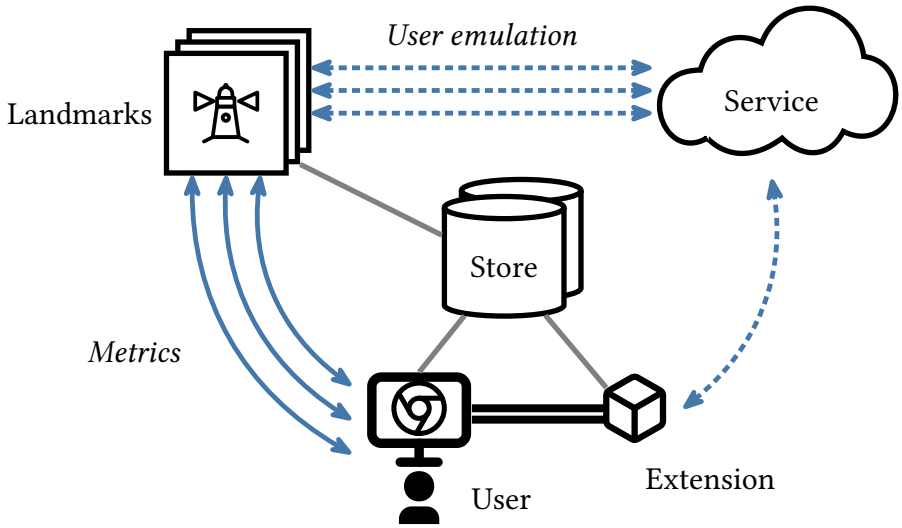


Figure 5.1 – Overview of DiagSys. Landmark metrics are fetched using a user's browser, while service probes can only run through a browser extension or client emulation in landmark servers (dashed lines). A distributed datastore is used to collect experiment samples.

has been taken by Netalyzr [114] for its measurement and debugging service. Fathom [116] further simplifies the measurement setup by providing a Firefox XPCOM extension, that does not require any Java runtime. In modern browser versions, Java applets and XPCOM extensions are no longer available, having been replaced by the WebExtension API standard [217]. While this recent API offers researchers a secure way to interact with the web browser, it is inherently more restricted than its predecessors. For example, it is no longer possible to *execute* system commands nor to send *custom* packets over the network. Yet, this API continues to make it possible to *estimate* low-level QoS metrics from browsers, i.e. packet loss rates from application throughput [218, 219] or congestion detection from requests timing [121]. Other approaches include using mobile applications: Wehe [214] is for instance the mobile successor of Glasnot, while the Acqua mobile application [215] uses machine learning models to *infer* the expected QoE to popular services from QoS measurements. Yet, these applications remain limited to Android and iOS smartphones. We compare the main approaches in Table 5.1, along with DiagSys, our own framework for browser-based metric collection (with optional WebExtension), which we present in this chapter.

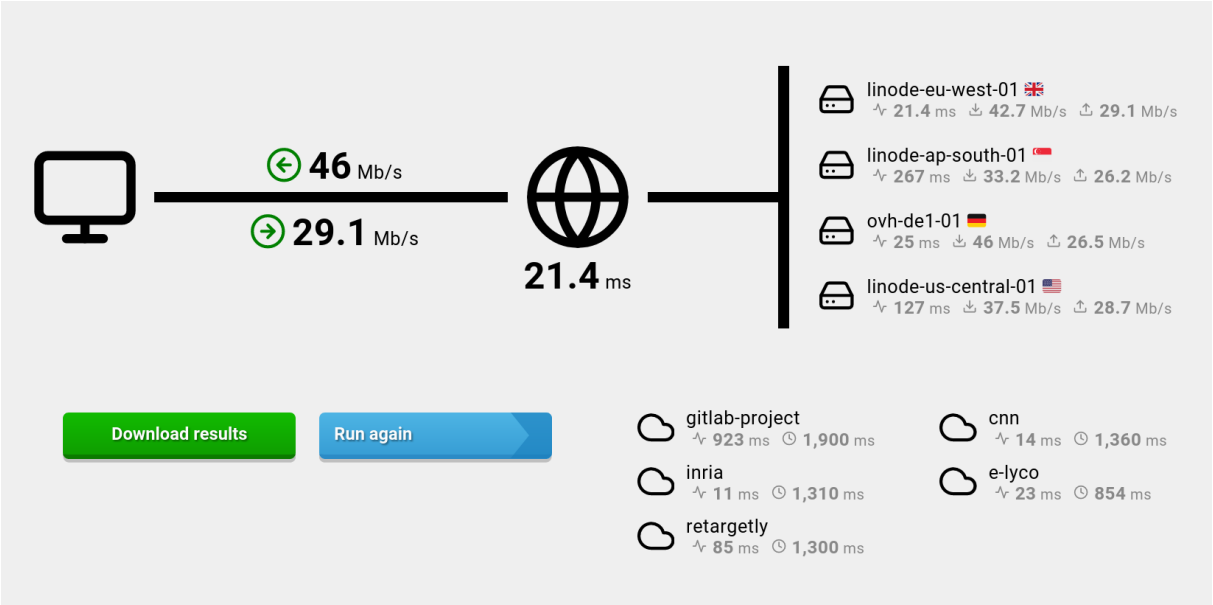


Figure 5.2 – Screenshot of a DiagSys report with measurements to four landmarks and five third-party services (the browser extension enabled in this example, enabling QoE measurements to five third parties).

5.1.2 Architecture of DiagSys

The DiagSys framework leverages several components to enable metric collection from web browsers (Figure 5.1). First, it relies on a fleet of landmark servers, acting as reference points and providing network metrics to web clients. Landmarks are self-contained stateless public HTTP servers that can be provided by different ISPs, cloud providers or other third parties in exchange of measurement analytics. (These servers the same we used for metric collection in our DiagNet RCA proposal, presented in Chapter 4.) End users can probe the landmark servers through any web browser with JavaScript capabilities. We provide two methods to estimate the QoE of third-party services:

- Landmark servers run headless chrome browsers to evaluate the performance of services, relaying this information to end users. This is relevant for landmarks *near* end users, speaking in terms of topological proximity; but the measured QoE can actually be very different from the users' real QoE.
- To allow third-party QoE measurements from web browsers, we provide an *optional* WebExtension that end users can install in their browser. We detail why this extension is necessary in Subsection 5.1.4.

A distributed datastore provides references to landmarks and services, along with long-term storage for the collected metrics. Since DiagSys has no strong consistency requirements, we use CouchDB for asynchronous multi-master replication to provide a highly-available, low-latency datastore.

5.1.3 QoS measurements provided by landmarks

DiagSys implements browser-based probing in JavaScript, which can be incorporated into any webpage (an example user interface is shown in Figure 5.2). We assume that JavaScript is enabled in user browsers to allow custom logic to be executed (this is the default). We begin by describing the measurement endpoints served by landmarks, then detail how landmark perform measurements to these endpoints.

Landmark measurement endpoints

In our design, a landmark server does not make any assumption about the underlying layers under the HTTP application layer: it is possible to serve clients using legacy HTTP/1 over the TCP transport, to more recent clients requiring HTTP/3 over User Datagram Protocol (UDP) transport. Landmarks provide the following endpoints for metric collection:

- **`/ping`** This endpoint first upgrades the HTTP connection to WebSocket and respond immediately with an empty message for each message sent by a client.
- **`/download`** Clients can download uncompressed random binary data with a single GET query. We use a multithreaded pseudo-random number generator to provide the maximum possible throughput server-side and actually measure the network limit. A waiting queue is also used to limit to only one download at a time and avoid concurrency between clients. This design allows us to reliably measure download speeds up to 8 Gb/s with recent commodity hardware and cloud servers. Clients discard the first chunk of data to avoid delay introduced by the waiting queue, and can download chunks for a maximum of five seconds.
- **`/upload`** Clients can upload random binary data using a single POST query. Again, we rely on a waiting queue to avoid client contention.
- **`/conn`** While a client cannot extract transport layer statistics from the available JavaScript functions, landmarks can provide their own statistics to clients. If the HTTP connection is supported by a TCP socket (the general case), we use the `getsockopt` Linux syscall

on landmark servers to obtain raw TCP statistics, containing among others the number of retransmissions and the minimum RTT measured by TCP. When available, we also return the congestion control algorithm used by the server, along with the set of statistics for supported congestion control algorithms. Thanks to the HTTP/1.1 Keepalive feature, TCP connections are not reset between HTTP calls: one client can thereby retrieve the full TCP statistics after having performed the download and upload tests for in-depth insights on its connectivity towards the landmark.

— **`/traceroute`** We also provide two endpoints on each landmark to 1) start a traceroute from a landmark server to a client public IP and 2) retrieve the result of this traceroute a few seconds later. This allows a client to start a traceroute asynchronously without blocking while waiting for the response. To accurately detect Network Address Translations (NATs) and Equal-cost multi-path (ECMP) routes, we use dublin-traceroute, a variant of the popular paris-traceroute [220]. We complete the intermediate hops found with their Domain Name Service pointer record (usually containing relevant operation and location information).

Latency measurement

Recent browsers (in our case Firefox version 76 and Chromium version 83) expose a standardized JavaScript API (the Resource Timing interface [221]) to extract each HTTP request's delays. This makes it possible to retrieve the connection, wait and download delays with millisecond precision. However, for privacy considerations, the W3C recommendation states that these delays can only be available programmatically if the requested resource stays on the same origin (subdomain) or a suitable `Timing-Allow-Origin` response header is provided by the server. For raw network round-trip measurement, another option is to rely on WebSockets [222, 223]. A simple HTTP request is usually accompanied by a text header of more than 100 bytes, forcing the server to download and parse it. Compared to this scheme, an empty WebSocket message has only an overhead of six bytes [131]. A third option would be to use WebRTC data channels to measure round-trip times without the TCP overhead. The main issue with that last option is that data channels may require strong permissions from users, such as microphone or webcam access: this would be questionable for a latency-measurement tool to request such permissions. We thereby rely on multiple *empty* WebSocket messages (**`/ping`** endpoint) to estimate the average latency to landmarks (and the associated jitter).

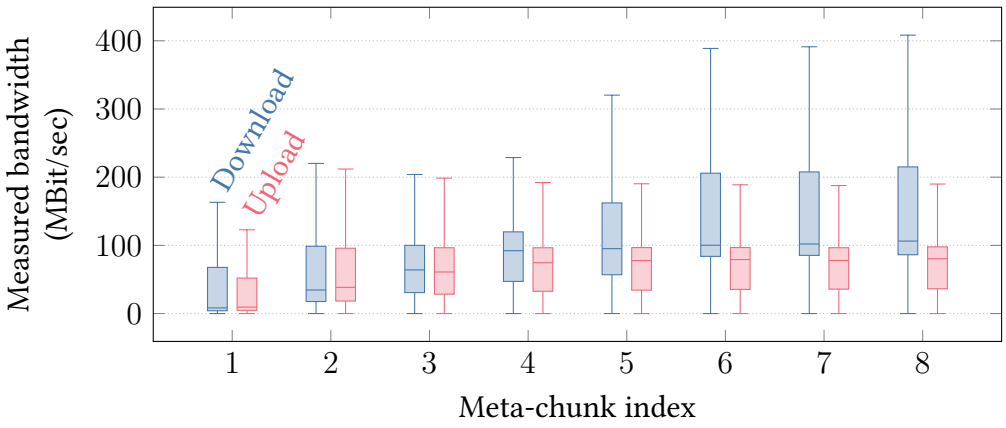


Figure 5.3 – Distribution of measured download and upload bandwidth per meta-chunk from the whole DiagSys dataset. Each box denotes the Interquartile Range (IQR) and the median, while whiskers denote the Q1 $-$ 1.5 IQR and Q3 $+$ 1.5 IQR interval. We can clearly see the effect of TCP slow-start.

Bandwidth measurement

To estimate the available network bandwidth, clients send one HTTP request to sampled landmarks and measure its throughput. This minimizes the overhead mentioned in the last paragraph. More specifically, we exploit the fact that clients can download (resp. upload) uncompressed random binary data with single GET (resp. POST) HTTP queries. While test data is generated server-side for **`/download`** measurements, the main challenge is to limit the overhead imposed by JavaScript browser-side for **`/upload`** measurements. In practice, a pseudo-random number generator is too costly to use, so clients only generate 1024 bytes of pseudo-random data and repeat them until the desired sample size is reached.

Clients and servers retrieve the result of an HTTP request chunk by chunk. Chunk sizes are unpredictable, but are usually a few kilobytes worth of data: it is possible to store chunks' size and the absolute time at which they are received. To standardize measurements, we aggregate chunks into eight "meta-chunks" (each having the same number of chunks) and compute the total time taken to download each meta-chunk. The final bandwidth is computed by averaging meta-chunks after removing the fastest and slowest meta-chunks. This method soothes the variations of measured throughput and avoid the bias caused by TCP's slow-start and the bursts caused by browser and system buffers. In Figure 5.3, we show from collected data that early meta-chunks underestimate the available bandwidth measured by later chunks. Clients can directly obtain chunk details for download, but they cannot observe how data is split in

chunks during upload: landmarks are therefore responsible for computing uploaded meta-chunks and reporting their transmission time to clients.

The special case of CDNs

Content Delivery Networks (CDNs) are widely used as the public-facing component of many web service [224]: they cache static resources and relay requests to "origin" servers. The main advantages of such architecture are two-fold. First, a CDN can redirect clients to the closest Point of Presence (PoP), thus lowering latencies. Second, many CDNs propose security features to protect the origin server from abnormal traffic, such as Distributed Denial of Service attacks or ill-formed requests. DiagSys covers common CDN PoPs by leveraging their caching mechanisms to serve as *degraded* landmarks. The basic idea is to host two files on a controlled origin server: an empty file (for degraded latency measurement) and a random file of known size (we use 8MB, for throughput measurement). A CDN can be configured to cache the files *indefinitely*: any client accessing one file will obtain it directly from the PoP selected by the CDN service. Our assumption is that the chosen PoP only depends on a client's location, and will remain the same when downloading a resource from a landmark as when using an actual web service. We deployed this strategy in Cloudflare and get the selected PoP from the CF-RAY header.

5.1.4 Third-party QoE monitoring from browsers

The techniques we have just described to obtain network and QoE measurements from a browser cannot in general be directly implemented as a script in a page, due to the security restrictions imposed by modern browser standards. In particular, to protect users from Cross-Site Scripting (XSS) vulnerabilities, recent browsers block requests to third-party origins by default. These vulnerabilities happen when a script sends a malicious request to a domain ("origin") different from the domain it was downloaded from, leading to impersonation for instance. Cross-origin requests are still possible by using the Cross-Origin Resource Sharing (CORS) mechanism: third-parties accepting to *receive* such requests can add special headers to their HTTP responses to disable some browsers restrictions. This makes it difficult for a webpage to probe third-party services' QoE: every response not having CORS headers (the default) will be blocked. More specifically, when the DiagSys measurement script (with domain X) tries to fetch the pages of a third-party service (with a different domain Y), the responses from domain Y will in general not include the CORS header and thus be blocked

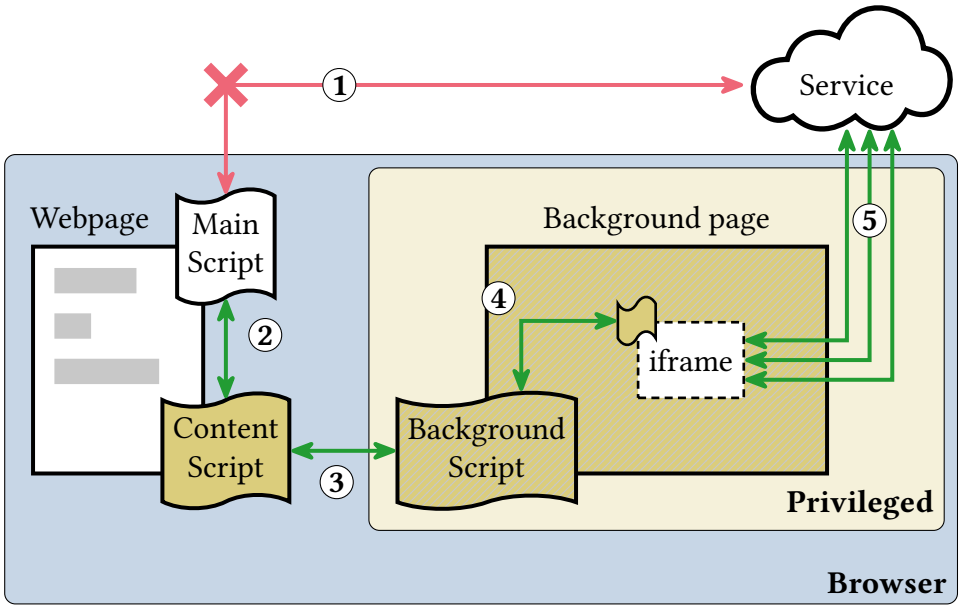


Figure 5.4 – Third-party QoE estimation via a browser extension: due to the default cross-origin policy ①, a webpage cannot directly fetch a third-party service's resources. However, it is possible to communicate with the background script of an extension *via* injected content scripts (② and ③). The background script can create background iframes ④ that are allowed to load the resources of any service ⑤.

by the browser. Similarly, one webpage can create tabs and iframes pointing to third-party services, but accessing the properties of these resources is restricted by browsers.

To circumvent these CORS restrictions, we have implemented the complete version of our measurement software as a browser extension, based on the *WebExtension API* [217]. (At the time of writing, this is the standard method for building extensions for Mozilla Firefox, Google Chrome and Microsoft Edge.) We recall that the installation of this extension is optional: we also provide a more limited script that can be included in a standard webpage to obtain a service QoE estimation *as seen from landmarks* (Figure 5.1). With the appropriate permissions set, a WebExtension can *intercept* and *modify* web requests sent to any third-party service. A first solution to disable cross-origin security would be to insert the CORS headers in every response. This would however open a major security hole in the browser's security model, as there is no standard method to add these headers only to requests originating from trusted sources.

Our solution is depicted in Figure 5.4. We first inject a WebExtension *content script* into trusted webpages (e.g. DiagSys's homepage). The content script registers itself with the *main webpage script* ② and relays messages from the main script to the extension's *background script*

③. In this setup, the communication between each script is secured by safe *Messaging APIs* provided in WebExtension. When the main script requests a QoE check to a specific service, the background script creates an iframe in the extension background page ④ (this operation is invisible to users). Iframes are used to *fully load a service in the background*, from the initial HTML document to the very last resource load. Another *content script* is injected in background iframes to obtain the resources timings and send them back to the main script (via the reverse path ④ → ③ → ②). The requests originating from our extension's background iframes can be identified using their unforgeable `originUrl`. We can thereby safely update CORS HTTP headers of responses corresponding to these requests ⑤. More specifically, we remove the `X-Frame-Options` and `Content-Security-Policy` headers to allow the loading of the third-party resources from iframes, and we set `Timing-Allow-Origin` to `*` to enable precise timings measurements. In particular, we obtain an estimation of the QoE of a third-party service by measuring the corresponding Page Load Time (PLT). Note that some services detect that they are being loaded from iframes and decide to stop loading or to take ownership of the parent frame (i.e. the extension background page). Thankfully, iframes can be *sandboxed* with a limited set of features which avoids losing control of the background page. For volunteers that enabled this option, our browser extension run periodic measurements (every 15 minutes) in the background, helping in the construction of dataset with continuous measurements.

Table 5.2 – Extract of third-party services monitored by DiagSys. For comparison with the first services of this list, we deployed some landmarks in Linode and OVH datacenters.

Name	Category	Hosting provider
Changelog	Software	Linode
Retargetly	Software	Linode
Stadiamaps	Mapping	Linode
E-lyco	Education	OVH
Sofoot	Sport	OVH
Systran	Translation	OVH
Gitlab	Software	Cloudflare
Mozilla	Software	Cloudflare
Zoom	Communication	AWS
CNN	News	Fastly
Le Monde	News	Fastly
Bison futé	Transport	N/A
Discord	Communication	N/A
Github	Software	N/A
Inria	Research	N/A
Parcoursup	Education	N/A
RATP	Transport	N/A
SolarLowTech	News	N/A
Walmart	E-commerce	N/A
Wikipedia	Education	N/A

5.2 Insights from our collected dataset

DiagSys ran continuously from October 2019 to March 2021. It allowed the collection of more than two millions measurement samples containing around five million probes to 26 landmark servers and 20 diverse third-party services, listed in Table 5.2 (one probe results from the interaction with one landmark or one service while one sample may contain several probes). More than 300 unique end users have provided measurements, and among them around 50 have installed our browser extension and produced periodic background measurements. Participants were recruited through community mailing lists, with the guarantee that no personally identifying information or browsing history was collected. We recall that landmarks emulate additional users to provide supplementary and more continuous measurement samples. We first present some case studies extracted from this early dataset, then characterize the faults we observed in practice.

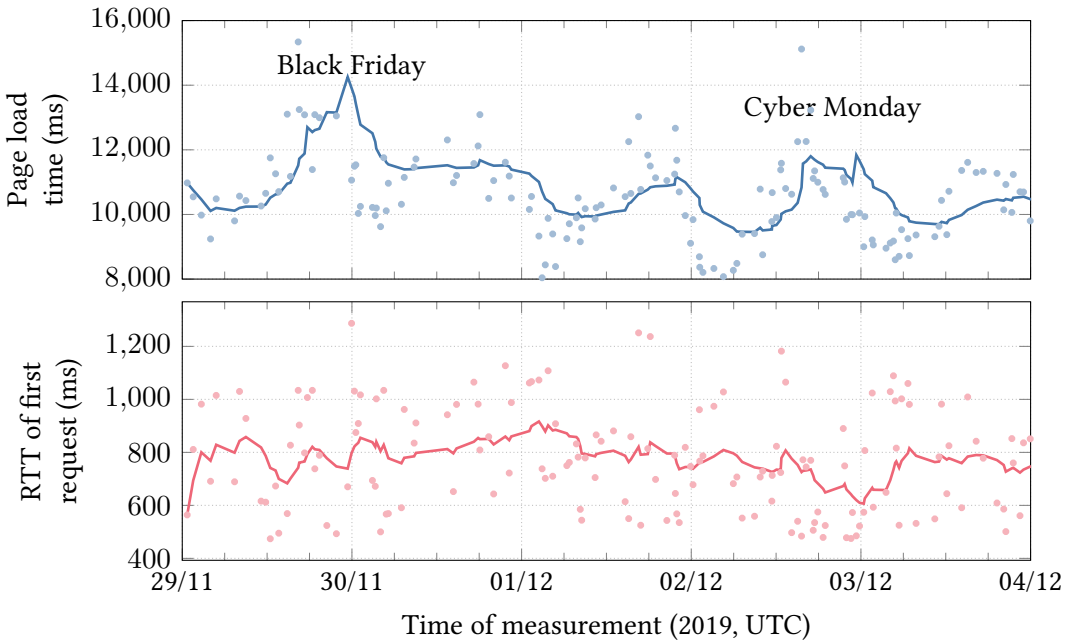


Figure 5.5 – `walmart.com` page load time and RTT of first HTTP request, as measured by DiagSys. We observe increases in page load time during Black Friday and Cyber Monday, in contrast with first request timings.

5.2.1 Dataset case studies

The collected dataset contains detailed information captured at a fine temporal granularity from a diverse set of viewpoints. This combination of diversity and good temporal resolution makes it possible to identify problems and phenomena such as varying PLTs, regional differences, PoP selection for CDNs or route changes.

Monitoring page load times

Users who run the DiagSys extension provided periodic measures of the PLT of selected third-party services. This can be used to estimate a web service's QoE and detect local and global perturbations. As an example, Figure 5.5 plots the PLT of `walmart.com` during the 2019 Black Friday with visible slowdown periods during expected traffic peaks. (The measurements were taken by a landmark in Paris.) We find that measuring PLTs provides more insight than just measuring the first request's RTT, as depicted in the lower part of Figure 5.5: PLT accounts for *every* remote resource, including scripts and medias from other third-parties. Similar highly-correlated patterns have been observed for different landmarks and users, with different amplitudes. This demonstrates the benefit of full browser emulation in landmark

Table 5.3 – Resources fetched by cnn.com for different regions (May 18, 2020 16:35 UTC)

Region	Europe	USA	Japan
HTML Body Size (bytes)			
Compressed	156 908	156 911	156 910
Uncompressed	1 132 658	1 132 658	1 132 658
Number of loaded resources			
style	20	19	19
script	28	61	61
query	20	53	54
iframe	3	13	14
media	7	48	57
total	**78**	**194**	**205**

servers, as provided by DiagSys, in contrast to only using unitary and simple measurements such as pings.

Highlighting regional differences

The user diversity of DiagSys makes it possible to spot differences in content served by third-party services to different visitors. Table 5.3 shows the number of unique resources fetched by three landmarks around the globe with identical configuration measuring `cnn.com` around the same time. We notice that the European landmark loads *far fewer resources* than its peers, despite receiving the same HTML page (assumed from the identical uncompressed body size of each response). When we look at the difference in loaded resources, we find that non-European visitors load more content related to analytics and ad tracking.

Impact of user mobility

Many users are mobile and use multiple methods to connect their devices to the Internet (wired, cellular, Wi-Fi, …) [225]. As a result, measurement samples from one user can be very diverse across time. We evaluate this diversity by using mobility ground truth of a specific volunteer that uses both Wi-Fi and wired connections as provided by their ISP. As expected, we observe clear differences in measured throughput between wired and wireless modes. More surprisingly, we note that some landmarks needed to retransmit *around 10% of packets* with wired connection, compared to zero retransmissions with Wi-Fi. We use BBR as the default

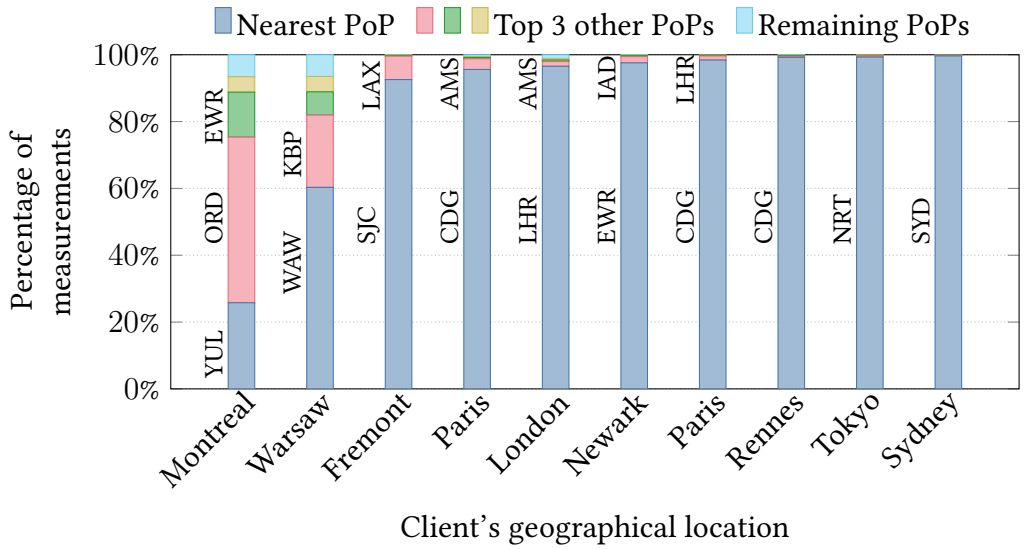


Figure 5.6 – Repartition of Cloudflare PoPs for various clients, sorted by percentage of measurements for nearest PoP. The following IATA codes are used by Cloudflare: AMS Amsterdam, CDG Paris, EWR Newark, IAD Washington, KBP Kyiv, LAX Los Angeles, LHR London, NRT Tokyo, ORD Chicago, SJC San Jose, SYD Sydney, WAW Warsaw, YUL Montreal.

TCP congestion control algorithm in landmarks, and this is certainly the reason why we are observing this behavior, as previously studied by Cao et al. [226].

Monitoring CDN performance

We measured the diversity of Cloudflare PoPs chosen for each user, and found that most users always reach the CDN network from the same PoP as illustrated in Figure 5.6 (we recall that the selected PoP is added in every HTTP response's header). However, we found that PoPs were much more dynamic for some regions, and we take as an example one landmark located in Warsaw's OVH datacenter. While most (66%) of HTTP responses were served from Warsaw's PoP (Figure 5.7) with a median latency of 8ms, the remaining responses were served from either Kyiv with twice that latency and even Moscow with a median latency of 50ms. Because this observation spans over two months of measurements taken from a static landmark, it is possible that this behavior is due to some load-balancing mechanism or non-optimal configuration.

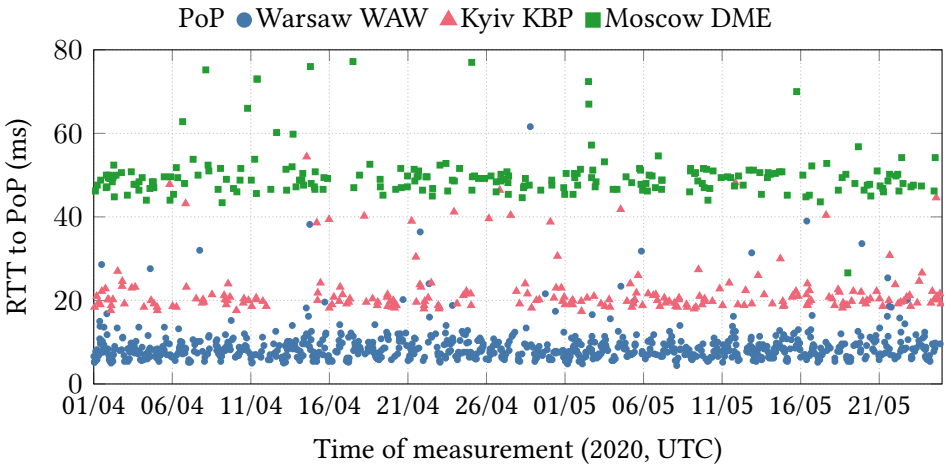


Figure 5.7 – RTT between a landmark in Warsaw and Cloudflare. Three different PoPs are regularly serving traffic with up to $5\times$ more latency from Moscow than from Warsaw.

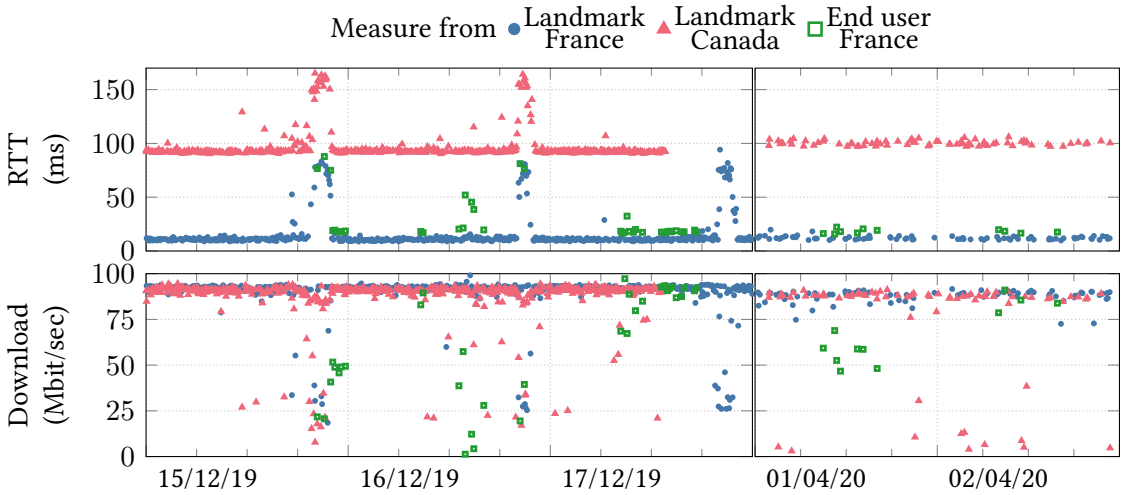


Figure 5.8 – Evolution of RTT and download throughput from two landmarks to a home-network landmark in France. There is a pattern of anomalies during evenings in the first time frame, probably due to congested link.

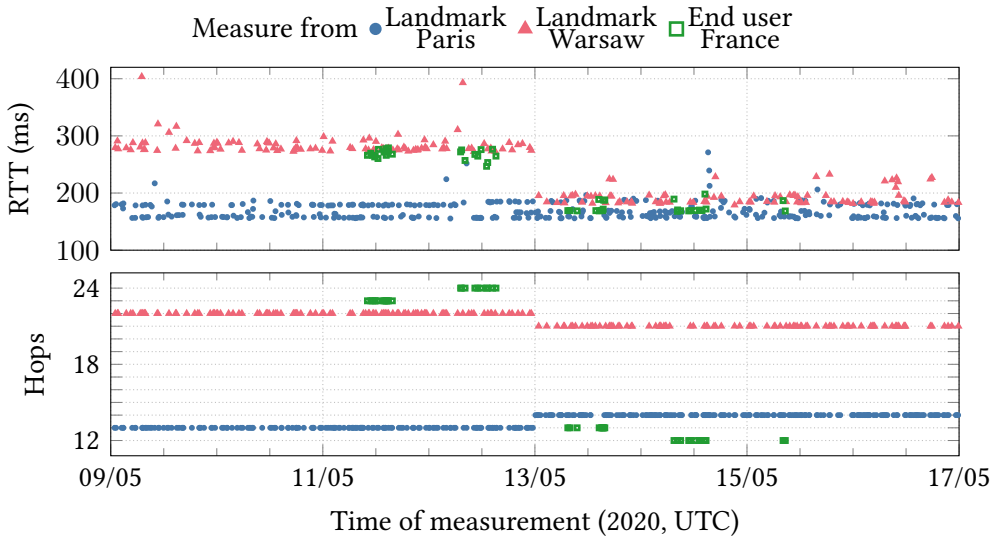


Figure 5.9 – RTT and number of hops in traceroute to a landmark hosted in Vultr Singapore region. We can see a change in routing strategy on 13/05/2020 at midnight with an immediate decrease of RTT for some regions.

Effects of network load and routing

DiagSys does not provide any information about network topology and BGP announcements. Yet, the collective knowledge gathered from users and landmarks is sufficient to detect and analyze changes in Internet paths and links, overcoming the opacity of ISPs networks. As a first example, we study the performance of a landmark hosted in a home network served by the French ISP "Free". Figure 5.8 shows the RTT and download throughput of this landmark as measured from two other landmarks in France and Canada and one end user from France. Measures from end users are sparser and noisier: this is expected, as their devices are not powered continuously and may have less reliable network connections. During the first time frame, we clearly see anomalies during evenings: the landmark's host confirmed that he encountered QoE degradation, which suggests that the root cause came from an overloaded link in the Free network. After a few months, the anomalies disappeared (second time frame in Figure 5.8). In a second example (Figure 5.9), we detected an important routing change between some users and a landmark in Singapore. The RTT to Singapore measured from Warsaw dropped by 30%, with one less hop in the reverse traceroute. Looking at the traceroute details, we discovered that the traffic was redirected from NTT (AS 2914) to GTT (AS 3257)—two Tier 1 networks. We used BGPlay [227], a routing history visualizer and confirmed this finding. The observations are similar for an end user in France, but no change is noticeable for another

landmark in Paris with already good performance before May 13. It would have been difficult to detect this routing change from within browsers using BGP announcements alone.

5.2.2 Dataset faults analysis

Clients and landmarks report any error they encounter when trying to reach third-party services and other landmarks: over the collected dataset, about 0.6% of measurements to landmarks and 1% of probes sent to third-party services resulted in failures. More specifically, a measurement is marked as "failed" when HTTP queries have not succeeded within a specific timeout. This timeout is arbitrarily set to five seconds for clients and one minute for emulated clients in landmarks: clients have smaller timeout to avoid excessively long ("stuck") measurements. Due to JavaScript limitations, it is not possible to have more details about every timeout failure (i.e. it is not possible to discriminate between connectivity issues, DNS failures or browser misconfigurations, among other causes). When services are overloaded, clients can reach that five seconds timeout and mark the measurement as failed. While in this case the QoE is clearly degraded, it is worth noting that the affected service *might still work* albeit with extreme latencies. As an example, we monitored the QoE of e-lyco [1], a local school virtual environment. On April 27, 2020, e-lyco reported degraded website performance. While we observed a stark increase of measurement failures from end users at that time, there were also a number of measurements from landmarks showing a PLT of more than 30 seconds with only degraded performance.

Conversely, QoE estimation might appear good when a third-party service is *actually* unavailable. In many cases, such unavailable services show an *error* or *maintenance* page. However, some third-party services still return "200 OK" HTTP status code instead of the expected HTTP error codes that would correctly result in measurement failure. Because DiagSys collects the *number* and *size* of all the resources loaded by a third-party service, it is possible to detect when the number of resources is abnormally low (e.g. indicating an error, redirection or maintenance page). For instance, we monitored the number of loaded resources from the bison-futé service [2] showing live traffic jams and road accidents in France. Because this service shows a dynamic map, it makes about 200 additional requests for small "tile" images. We noted frequent service disruptions, with no loaded resource and a page size of less than 500 bytes (compared to usually more than 6000), clearly showing that something was wrong despite nominal QoE timings.

1. `https://www.e-lyco.fr`
2. `https://www.bison-fute.gouv.fr/maintenant.html`

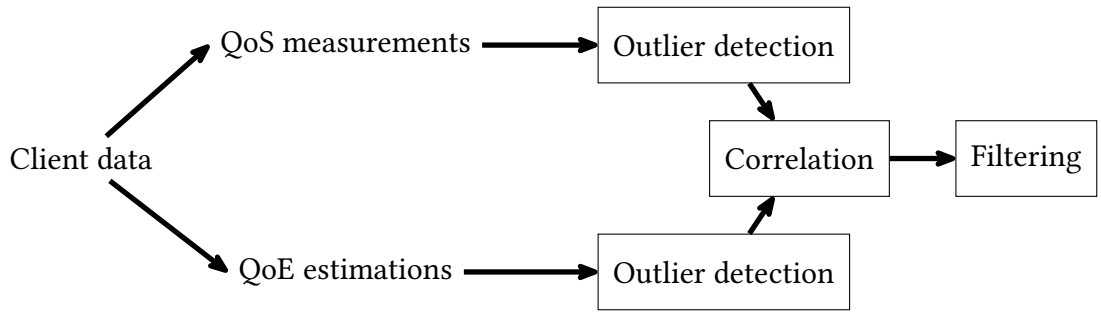


Figure 5.10 – Workflow of our labelling process. For each client, we first pinpoint outliers for QoS and QoE metrics. Then, we *correlate* the outliers in an attempt to find the QoS metrics responsible for QoE degradations. Finally, we filter-out false positives with a causality test.

These examples advocate for a more *accurate* labelling of measurements in our dataset, in order to know which samples represent a nominal and expected QoE, and which ones do represent QoE degradations and failures. Our first approach to perform such labelling was to look for information published from third-party services (e.g. status pages, social network feeds, newsletter). However, most QoE degradations are not reported, especially for degradations affecting a limited number of users over a short period. Service status pages are also criticized for *being unreliable* since updates are usually published manually: for instance, AWS's status page is rarely updated, even during obvious global outages [228]. As another extreme example, the status page for OVH datacenter "SBG2" reported "no server down" during several hours on March 10, 2021, well after the fire that destroyed the building on that same day [229]. Other public sources include specialized forums, troubleshooting websites (like DownDetector [124]), network operators mailing lists and automated detection systems such as IODA [230]. Browsing these feeds remain a manual and tedious process: while large services and network outages are well documented in practice, we were unable to find reliable reports for most failures found in our dataset.

5.2.3 Our proposal: labelling from historical data

With the objective to build practical RCA models using metrics from end users, we now propose to label measurements samples with a relatively simple outlier correlation method. Our workflow is illustrated in Figure 5.10. The first problem to solve is the *detection* of QoE degradations for every client: we use a simple outlier detection technique, leveraging historical data collected over several months. We apply a similar operation to pinpoint anomalies in QoS metrics, and we correlate found outliers to produce *plausible* root causes for the observed

QoE degradations. After an additional filtering of obvious false-positives, we obtain *surrogate ground-truth* labels, that we later use in Section 5.3.

Outliers correlation To overcome the lack of accurate root cause ground truth for the QoE degradations we measured, we take a statistical approach to label our dataset samples, and produce a surrogate ground truth which—although imperfect—we can use to evaluate RCA techniques. We leverage our historical data collected over several months to define *nominal* values for every combination of client and metric (at each measurement point, being a landmark or third-party service). From these nominal measurements, we identify *outlier values*: by marking all values below $\text{Q1} - 1.5\ \text{IQR}$ and above $\text{Q3} + 1.5\ \text{IQR}$ as outliers. We pinpoint outlier samples for both landmark metrics (QoS) and third-party QoE metrics. Then, we label samples by grouping them by client and time slots of 30 minutes and by *correlating* QoE degradations with QoS outliers. Intuitively, this correlation helps in the identification of the dependencies between QoE and QoS metrics. Grouping samples is necessary in practice, since clients probe a limited number of landmarks in every measurement sample: 30 minutes slots allow to collect metrics from ten landmarks per slot on average.

- In a given slot with *degraded QoE*, if outliers come from more than three different landmarks, it is likely that the client's connection to Internet is at fault. Hence we label samples in this slot with the root cause "uplink failure".
- If a QoS metric provides the majority of outliers in a slot, this metric is marked as the *root cause* for QoE degradations in this slot.
- Finally, when multiple potential root causes remain, we pick the one with the largest relative difference from the IQR threshold.

Note that due to the lack of validation examples, we were unable to map jitter and packet loss statistics to plausible root causes accurately. As a result, we only consider the *latency, download* and *upload* QoS measurements in our root cause labelling process. With this approach, the five millions measurements of the DiagSys dataset are processed and labelled in a few hours on a recent laptop. The returned root causes directly map to landmark metrics, in a similar way than DiagNet (Subsection 4.2.1).

Filtering-out false positives However, one major drawback is that many root causes are actually false positives an anomalous QoS might not be the cause of a QoE degradation—correlation does not imply causation. Again using the historical knowledge we collected with

DiagSys, we propose to eliminate most false positives using conditional probability estimations. Let us define the events L_i as "landmark metric i is nominal" and S_i as "service QoE metric i is nominal". Conversely, $\overline{L_i}$ (resp. $\overline{S_i}$) indicates that "landmark (resp. service) metric i is an outlier". We detect false positives by estimating the conditional probability that a service metric i *does not* exhibit a QoE degradation given that a landmark QoS metric j is an outlier, i.e. $P\left(S_i|\overline{L_i}\right)$. The closer this probability is to 1, the less plausible that the said landmark metric can be a root cause for observed the QoE degradation. In the following, we estimate conditional probabilities empirically using Binomial distributions and the associated Wilson score for the confidence interval [231]:

$$\hat{p} = \frac{\text{Number of samples where an event occurs}}{n : \text{Total number of samples}}$$

$$(p_-, p_+) = \left(\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}\,(1-\hat{p})}{n} + \frac{z^2}{4n^2}}\right)\frac{1}{1+\frac{z^2}{n}} \qquad \text{(Wilson score interval)}$$

We set $z = 2.58$ to tolerate an estimation error of 1%. Note that due to the random sampling of landmarks and services, we do not necessarily have measurements for all pairs of metrics in all slots. To overcome this limitation, we define the event $X_{i,j}$ as "in a given slot, metrics i and j have been measured by the same client". This leads to the final approximation equation:

$$P(S_i|\overline{L}_j, X_{i,j}) = \frac{P(S_i, \overline{L}_j|X_{i,j})}{P(\overline{L}_j|X_{i,j})} \geq \frac{p_-(S_i, \overline{L}_j|X_{i,j})}{p_+(\overline{L}_j|X_{i,j})} \text{ with probability } 0.99 \qquad (5.1)$$

We filter-out false positives by eliminating pairs of metrics with an estimated conditional probability above 0.80. This value was empirically chosen to filter most false positives while retaining interesting correlations. Decreasing it further would significantly reduce the knowledge available for RCA training and evaluation, while increasing it would not filter enough false positives. Among the 65 624 potential root causes found with outliers correlation, 28 745 (44%) remain after this filtering.

Validation We confirmed that this method returns plausible results by considering *reference services* deployed in landmarks: it is clear that a QoS outlier on a landmark metric must trigger a QoE degradation for the services co-hosted with this landmark. For instance, we periodically injected artificial latency into one landmark hosted in the Beauharnois OVH region. During these periods, the reference service hosted in the same server showed degraded QoE, with the surrogate root cause being (correctly) labeled as "latency to OVH Beauharnois".

We also verified that services hosted by one provider were correlated with landmarks hosted in the same providers. As an example, we found that many services have Cloudflare CDN as possible root causes; this makes sense since these services use Cloudflare's services. Conversely, most filtered root causes were indeed unreasonable, for example we found several potential root causes localized in south Asia for a reference service hosted in Europe with no external dependency.

5.3 Applying DiagNet to DiagSys's dataset

We presented how we label QoE degradations with potential root causes using a simple outlier detection mechanism in Subsection 5.2.3: using this mechanism, we were able to label our dataset of real Internet measurements with a *surrogate* ground-truth. In this section, we explore how the RCA methods introduced in Chapter 4 perform on this larger and more diverse dataset. Our goal is to assess whether the techniques we have proposed generalize to a less controlled setting, in which the ground truth regarding root causes had to be *reconstructed* using statistical methods: it is important to note that our labelling method uses the *complete dataset* to pinpoint outliers and root causes. By contrast, when training RCA models, we group measurements made by every client in 30-minutes time slots, while not feeding the unique identifiers of clients as features to the RCA models. Only a few measurements are available during a RCA inference: we only exploit the landmark QoS measurements made by a single client during the appropriate 30-minutes slot in which the QoE degradation is observed.

5.3.1 Changes to the RCA approaches

In this experiment, we essentially reuse the RCA implementations from Chapter 4 (namely the Extensible Naive Bayes, Extensible Random Forest and of course DiagNet), yet with a few key differences.

A first difference touches on the metrics (features) available for RCA: in the DiagSys dataset, due to the limitations of browsers (Section 5.1), we did not collect the *current processing load* of clients. We collected jitter and packet loss statistics, but we were unable to map them as plausible root causes accurately. As a result, for this dataset we consider the following four root cause families: *link latency* (49% of faults), *local (uplink) failure* (39%), *download bandwidth* (7%) and *upload bandwidth* (5%). These families are combined with the 26 landmarks we deployed, yielding a grand total of around 80 possible combinations (to be compared to 55 possible root causes in the controlled dataset of Chapter 4). Despite having more possible combinations, the DiagSys dataset holds about the *same number of faulty samples* than the controlled dataset ($\approx 30\,000$). We therefore substantially increased the expressivity of the RCA models (i.e. by increasing the number of random forest estimators and the number of dimensions of hidden layers in DiagNet). We also note that the fault families (and sampled landmarks) are not uniformly distributed in this dataset; we therefore adjust the Random Forest's class weights to be inversely proportional to their frequencies.

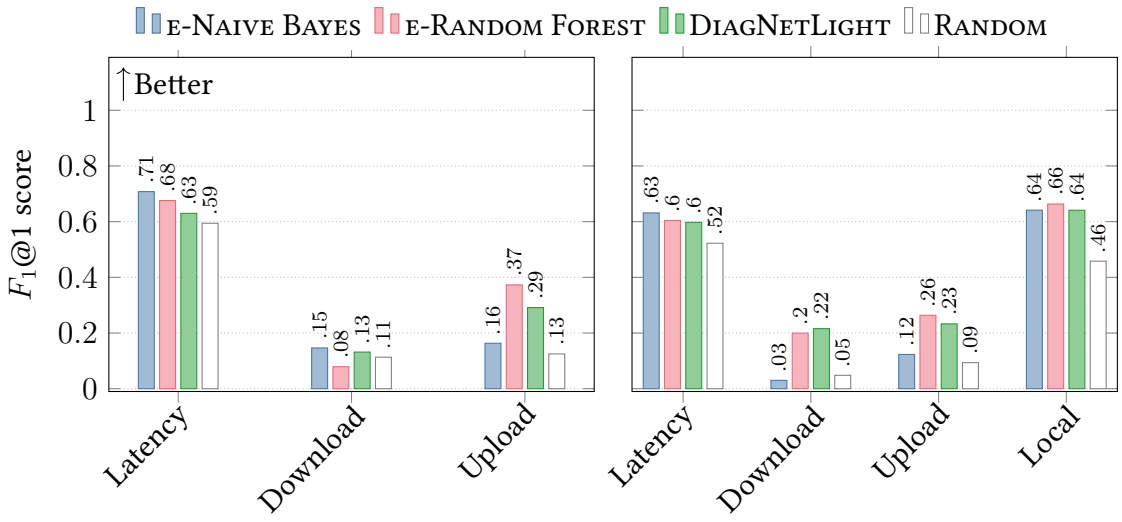
A further complication arises from the fact that the DiagSys dataset is much scarcer than the controlled dataset we have used in Chapter 4. Each DiagSys sample only contains values for *some of the metrics* of the overall metrics set: this is because each client only probes around ten landmark servers per time slot (on average). This is necessary in a realistic deployment: measurements take *time* and *resources*, and we must find an appropriate tradeoff between measurement accuracy and cost. (In our deployment, we try to always include a landmark server *close* to the client according to a database of geographic locations [232].) Moreover, we masked the metrics of three landmarks during model training, in way similar to what we did in Chapter 4: one landmark in North America, one in Europe and one in Asia. The masked metrics are re-enabled during inference (we use 20% of the dataset as evaluation samples). We leveraged the extensible designs of our three algorithms to support these missing metrics. (As a reminder, the naive bayes approach uses "shared" KDE, the random forest sets missing values to zero and DiagNet naturally supports them with convolutions and pooling.)

DiagNetLight: disabling DiagNet's auxiliary model Early results demonstrated that DiagNet was heavily biased towards its *auxiliary model* (Subsection 4.2.5), returning predictions extremely close to the ones of the Random Forest. Therefore, to focus on the evaluation of DiagNet's convolutional network alone, we present in this section the results for DiagNet's predictions *without* ensemble model averaging that we call DiagNetLight.

5.3.2 Pinpointing fault families

We begin by evaluating the performance of RCA models with respect to the prediction of fault families (what we call the *coarse* prediction). While DiagNetLight already outputs this prediction by design, we tuned the other two baselines (Naive Bayes and Random Forest classifiers) to output such a coarse prediction instead of the finals, fine-grained root causes. We show the results over the evaluation samples in Figure 5.11, distinguishing between "new" and "known" root causes (compared to training samples with some root causes being masked on purpose). We also add the results of a "Random" predictor for comparison: it returns a given class with a probability proportional to the weight of this class in the training dataset.

The Naive Bayes and Random Forest classifiers offer good F_1 scores (Equation 4.6) for the majority classes "Latency" and "Local". However, the minority classes "Download" and "Upload" are clearly disadvantaged in this experiment. For new root causes, the Naive Bayes model have a F_1 score of about 0.15, clearly higher than for known root causes. This is expected, since we noticed the tendency of Naive Bayes to favor unknown metrics (hence unknown causes)



(a) New root causes (b) Known root causes

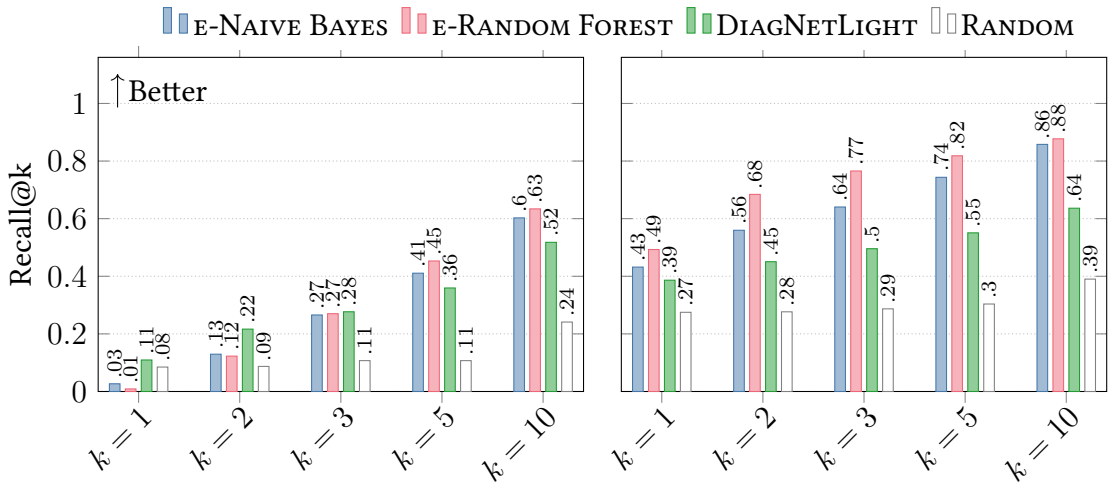
Figure 5.11 – Coarse prediction evaluation over the DiagSys dataset for new and known root causes. DiagNetLight demonstrates slightly superior performance for "Download" and "Upload" classes, while the other baselines show comparable results for the more frequent "Latency" and "Local" classes.

in Section 4.3. The Random Forest classifier gives good predictions for both types of root causes. However, this significantly differs from the observations of Figure 4.5, where Random Forest showed extremely good performance over known causes. We can naturally explain this difference by the larger number of missing metrics, compared to the uniform fault distribution of the controlled dataset.

By contrast, DiagNetLight yields comparable performance for the "Latency" and "Local" classes, while giving relatively good F_1 scores for the "Download" and "Upload" classes. This is true for both new and known root causes, showing a relative tolerance to skewed datasets. DiagNetLight provides better results than the random predictor for all classes, but not by a large margin. Moreover, our results are inferior to the coarse analysis in the controlled dataset (Figure 4.7) that showed F_1 scores over 50% for all classes.

5.3.3 Complete root cause analysis at Internet scale

We now evaluate the prediction of more *fine-grained* root causes, modeled as a combination of a *location* and *fault family*. In Chapter 4, we made our evaluations over precise landmark locations. Since we almost tripled the number of available landmarks, this method is not practical anymore: it would be extremely difficult for a RCA model to distinguish faults



(a) New root causes (b) Known root causes

Figure 5.12 – Recall@k evaluation for new and known root causes, for different values of k. Root causes are grouped by world regions. DiagNetLight shows best results for new root causes up to $k = 3$.

between two landmarks in the same city or region. To illustrate this challenge, we found the Recall@25 to be less than 1% over new root causes for the three considered methods. That is why we chose to *group* root causes in seven *world regions* according to the geographical distribution of our landmarks ("Europe", "Asia", "Australia", "Eastern US", "Central US", "Western US" and a "global" region for CDNs). We also added one *global* region for each CDN landmark. As a result, the predicted root causes could be for instance "latency degradation in Europe", "download bandwidth limitation from Australia" or more simply "local uplink failure". This approach led to a total of 41 possible root causes, a number comparable to our initial DiagNet evaluation.

The results of this evaluation are depicted in Figure 5.12, showing key insights for this dataset. Over new root causes, DiagNetLight offers the best recalls for the first two predictions ($k < 3$). However, *both* Naive Bayes and Random Forest bring in superior results from the fourth prediction onward and for all predictions over known root causes. Surprisingly, the two baselines have comparable results for both types of root causes, showing their robustness against unavailable metrics in general. This again differs from our previous observation, where Random Forest performed better on known root causes and Naive Bayes on new root causes. The random predictor is clearly short of our machine learning models, except for new

root causes and $k = 1$ where it approaches DiagNetLight results. This shows that our statistical methods are making a good contribution for Internet scale RCA, but that important limitations remain.

5.3.4 Discussion and remaining open questions

Summary The performance of the three studied RCA approaches are significantly different over the DiagSys dataset than over the dataset with controlled faults from Chapter 4. Overall, the global performance of models *decrease*: this is not entirely surprising given the challenges to pinpoint uncontrolled failures over Internet. Yet, our models remain superior to a *random* predictor, showing that statistical learning can indeed help in large computer network modeling. All our approaches have close performances on both known and new root causes: the added value brought by DiagNet is no longer apparent. Nevertheless, we must note that the Naive Bayes approach performs surprisingly well, including for known root causes (where it obtained very poor results in Chapter 4). This behavior is certainly due to its capability to handle *unavailable* metrics in its Kernel Density Estimation, something that was not so much needed and therefore not correctly exploited in our previous study.

Additional experiments The observed results could demonstrate the limitations of our statistical approaches, our methodology of labelling with surrogate ground truth or most probably a combination of the two. To tease out the importance of each of these factors, additional sensibility analysis is required. In particular, it would be interesting to evaluate our models with different labelling parameters (e.g. durations of time slots, false-positive probability thresholds) or even different hyperparameters. It would also be interesting to evaluate how the number of plausible root causes and the choice of "world regions" impact the accuracy of each model.

Potential avenues for improvements This evaluation also shows that it is actually very challenging to build reliable RCA models despite a seemingly *simple* labeling approach with outlier detectors (Subsection 5.2.3). This difficulty likely stems from the fact that we *hide* the client identity to the proposed RCA methods: each client has its own view of the global network, with specific nominal QoS values (it is clear that a latency of a few hundreds of milliseconds is acceptable between Paris and Tokyo, but not between Paris and Brussels.) In this Internet-scale dataset, we had an order of magnitude more clients than in the controlled dataset: providing client identifiers as input features would quickly be impractical, especially

since models would need to be retrained for every new client. One alternative could be to input coarse-grained information about clients (such as their ISP, type of device or rough location). However, this might be impractical with the lack of reliable information [141] and recent privacy regulations related to personal data.

Similarly, the Naive Bayes and Random Forest baselines have a significant advantage compared to DiagNet: in their implementation, they can directly distinguish features from two different landmarks (this is clearly necessary for these methods to work at all). By contrast, DiagNet can accept measurements from *any* landmark in any order but without landmark identification, making it more generalizable. One idea to allow DiagNet to distinguish landmarks would be to *embed landmark identifiers* as input features—something we already proposed for neural networks in processing of the scheduling policies in Chapter 3. However, this approach would probably cripple DiagNet's ability to analyze metrics from landmarks added *after* training. One more generalizable idea would be to embed landmarks' geographical or topological *locations* as additional features that would be available even for landmarks unseen during DiagNet's training. This seems to be important for deployments with even more landmarks, as our analysis is based on a relatively limited deployment of 26 landmarks.

Finally, we note that we had to *disable* the auxiliary model to analyze DiagNet's results (resulting in the DiagNetLight variant). From our observations, this was because DiagNet's was overly confident and ignored *new* root causes most of the time—thus exclusively returning the results of the auxiliary model (i.e. a Random Forest). Given the clear benefits of ensemble model averaging we observed in Chapter 4, it appears important to investigate this path. This is especially true since we discovered that the two other envisaged approaches actually give good results on DiagSys dataset overall, both for new and known root causes. While model averaging is probably the simplest ensemble approach, there is no doubt that more sophisticated approaches (such as those described in Subsection 2.4.4) could lead to more accurate RCA models.

5.4 Conclusion

After motivating the need for browser-based measurements for accurate Quality of Service (QoS) and Quality of Experience (QoE) estimations, we presented a set of complementary techniques to implement these measurements in practice. With the security restrictions of modern web browsers, most solutions from the literature no longer work, including solutions based on Java or XPCOM browser plugins. Building upon modern JavaScript and WebExtension capabilities, we have proposed a lightweight measurement script to circumvent these restrictions (including Cross-Origin Resource Sharing (CORS)) while keeping users safe and incurring minimal overhead.

We have deployed our approach (under the DiagSys name) during more than a year and collected a valuable amount of measurement samples from hundreds of volunteer users. The resulting dataset is rich enough to identify noteworthy insights regarding network about web dynamics. For instance, we provided examples of variations of load times during peaks of traffic, service differentiation according to client location, surprising Content Delivery Network (CDN) load balancing configurations and even traffic pattern changes over time. Of course, we have just demonstrated *some examples* of the kind of explorations made possible by our dataset, which illustrate the potential of the collected data *and* of our measurement methods; we are convinced that many interesting results are yet to stem from a deeper exploration of the DiagSys dataset.

In order to overcome the absence of a reliable ground truth about the Internet and service failures we observed in our dataset, we proposed a simple measurement labeling method based on outlier detection and correlation. After some adaptions, we then applied the Root Cause Analysis (RCA) methods we presented in Chapter 4 to this dataset, and found significantly different results compared to the initial evaluation dataset with controlled faults. This additional evaluation highlighted the limitations of our statistical learning proposals for RCA (extensible Naive Bayes, Random Forest and CNN classifiers), hinting that more work is needed for Internet scale RCA with uncontrolled QoE degradations. In a discussion, we provided several avenues to improve DiagNet's performance, such as adding geographical and topological information for clients and landmarks or combining the presented algorithms with more sophisticated ensemble approaches.

Chapter 6

Conclusion

> "[...] on Internet governance, we can therefore say that there are many players, without a clear hierarchy. They sometimes act in concert, and they often oppose each other. And, when they act together, it is not always in the interest of the users."
>
> *Translated from "Cyberstructure" [233]*
> *Stéphane Bortzmeyer*

In this thesis dissertation, we have reviewed how advanced machine learning techniques can help building generalizable and extensible inference models of computer networks. We partitioned the problem space according to the amount of available knowledge over the underlying network: the proposed techniques are indeed different whether one has a *complete* or *limited* knowledge of the network topology and components. In this conclusion chapter, we first review our contributions and highlight immediate future work from the research problems we uncovered (Section 6.1). Then, in an effort to build a more transparent and *measurable* Internet, we provide a wider outlook and advocate in Section 6.2 for a greater collaboration between network operators, service providers, browser developers and end users.

6.1 Summary and future work

In our introduction, we claimed that **statistical learning could be used to model complex computer networks, even with narrowed information**. We now review how our contributions back that claim.

6.1.1 Realistic queuing is needed for data-driven modeling of networks

We started in Chapter 3 by leveraging knowledge of the network topology and of components' characteristics to *predict* the (unknown) expected performances over paths. This problem is known to be difficult due to the many interdependencies between network paths, links

and nodes. Emulation and simulation tools can be used to solve this problem, but they are usually slow and impractical for large networks. Inspired by a proposal from Rusek et al. [11], we use Graph Neural Networks (GNNs) to improve predictions efficiency while preserving their accuracy. Our contribution first exploits the topology and routing knowledge to build *bipartite dependency graphs* between network paths, links and nodes. Using performance information from a network simulator, we then train GNNs models over these dependency graphs. In our evaluation, we showed that the trained models are generalizable to *arbitrary network topologies* not seen during model training.

More specifically, we proposed two GNN approaches. A first one, A1, explicitly maintains representations for *node internal states*. We removed this representation in a second approach (A2), only keeping representations for *paths* and *links*: during initialization, node features are actually embedded in *link representations*. Surprisingly, this (simpler) second approach was actually more accurate than A1 in our experiments. The simulator used to generate experimental data modeled message queues only at *outbound* node interfaces: our evaluation suggests that with this choice, internal states for nodes are not required.

This simplistic queuing model might not, however, fully capture the behavior of *real* routers with *input buffering* or *insufficient internal bandwidth* [182]. As such, realistic queuing models and simulators are needed in order to apply our contributions to real Internet networks. Future work could integrate *real router specifications* in training data simulators to verify if internal states for nodes are beneficial in such realistic scenario (approach A1). However, it could be difficult to *exactly* simulate real routers, due to unknown hardware constraints and the large diversity in router implementations. One potential solution to build generic models that capture all this diversity lies in network emulation: by measuring network performances over *physical* hardware and modifying network characteristics with SDN capabilities, we could collect diverse and realistic training data. Albeit obviously more complex and costly, we believe this method is a relevant next step to improve the accuracy of Knowledge-Defined Networking (KDN) solutions over real SDN networks.

6.1.2 The ground-truth problem for accurate RCA models

Root Cause Analysis (RCA) is an important problem in computer networks: given an observed service anomaly (the symptom), one may want to uncover the reason(s) behind it (the cause) to solve the observed anomaly. However, during the lifecycle of a computer network deployment, it may not be possible to access all its characteristics (due to technical or business reasons). In particular, it is difficult to infer the *dependencies* between services and network

components—a necessary step for RCA in wide-area networks. Chapter 4 presents how extensible machine learning techniques can be used to solve this problem. Metrics are collected by clients actively probing reference "landmark" servers, deployed in multiple network locations: we use these metrics as both features for our machine learning models and as *potential* root causes. The challenge consists in determining which metrics are relevant for which services: because the latency to Tokyo is high does not necessarily mean that the root cause is located in Tokyo, especially if both the service and the client are located in America. However, it may be the case if the troubleshooted service has an internal dependency to a provider in Tokyo.

We proposed two extensible baselines based on Naive Bayes and Random Forest classifiers, along with a novel Convolutional Neural Network (CNN) design for RCA. Compared to the baselines, this latter approach (that we called DiagNet) can be extended easily after training: in an evaluation made over a dataset with controlled faults, we show that DiagNet can pinpoint root causes it has *never* seen during training (using metrics from *untrained* landmarks). Our approach can also be specialized to other services without much re-training. These properties make it a good candidate for practical, Internet-scale RCA.

In Chapter 5, we attempted to evaluate DiagNet over a dataset of measurements from real end users and third-party web services. To this end, we implemented a measurement framework (DiagSys) running within the *execution environment* of these services, i.e. web browsers. This choice allows easy installation and maintenance of the measurement framework, but raises some challenges due to the security restrictions of modern browsers. We were nevertheless able to collect millions of measurements over more than a year, including network QoS and service QoE, with the help of around 300 unique volunteer end users. We highlighted several case studies showing the relevance of metrics collected *from within the execution environment of web services*.

While we detected obvious *degradations* in QoE metrics indicating failures within Internet, we had no access the *ground-truth root causes* behind these degradations: services and network operators are indeed reluctant to publish such information, especially when failures impact a limited number of end users. For every pair of end user and metric, we computed nominal values over the full measurement period: this allowed us to pinpoint anomalies from outlier measurements. By correlating QoE outliers with QoS ones, we were able to mark each QoE degradation with a corresponding *probable* root cause. Training the same statistical models than previously (i.e. extensible Naive Bayes, extensible Random Forest and DiagNet), we demonstrated that our three models provide good RCA recalls for *trained* root causes. Albeit

close to its competitors DiagNet remains marginally superior when dealing with *untrained*, new root causes.

One potential avenue for improving our statistical methods would be to *embed* slightly more information about the measurement platform and the analyzed network. For instance, when feeding landmark metrics to DiagNet, we do not embed the identity or location of each landmark: adding this information could help DiagNet differentiate between two similar landmarks, leading to better RCA accuracy. Future work could also embed the *known dependencies* between services, and between network components. This could be done in a similar way than what we proposed in Chapter 3: GNN layers could be added to compute advanced representations for potential root causes, later aggregated and processed using DiagNet's extensible CNN.

6.2 Outlook: towards greater collaboration between actors

The labels we computed in the dataset of Chapter 5 remain *rough* and *approximate*: despite our attempts to filter out false-positives, it is clear that our models would benefit from knowing the *real* ground truth behind observed QoE degradations. Indeed, the quality of the final models necessarily depends on the accuracy of the data they are trained upon. In this section, we advocate for a better *collaboration* between Internet stakeholders to improve the wealth of available data for network modeling.

6.2.1 Better transparency is needed from service providers

It is common knowledge that network and service operators do their best to *hide* their internal architecture and dependencies. From a business perspective, this might be necessary to protect the secrets of trade agreements; and from a technical standpoint it might guard against industrial spying and adds some additional security through obscurity. However, this secrecy is in contradiction with the needs of network modeling and RCA as we have shown in this thesis. Without collaborating with these operators, it is difficult to find the root cause of the observed QoE degradation with certainty: public status pages and social media feeds are often too optimistic (see Subsection 5.2.2), inaccurate or may even contain fake information. On April 6, 2021 (first day of a new Covid-19 lockdown for French high-school students), the remote educational tool "Ma classe à la maison" ("My classroom at home") was completely

unavailable for teachers and students. The French ministry of education Jean-Michel Blanquer blamed "foreign cyberattacks" and one hosting provider, OVH [234]. However, other sources (including OVH CEO Michel Paulin [235]) have answered on social media, indicating that the issue came from the undersizing of the educational tool. This critical event for the twelve million French high-school students highlights the needs for *independent* RCA backed by *publicly-available* information.

Documented and—more importantly—automated status pages would greatly help in the understanding of service dependencies. This policy is being implemented by some large actors: for instance, the GitLab deployment tool allows providers to automatically update a public status page according to metrics provided by the monitored service. GitLab itself publishes detailed reports of failures in a dedicated twitter feed, linking to a public, live metric dashboard [1]. Wikimedia (the non-profit foundation hosting the popular Wikipedia encyclopedia) follows a similar path by publishing all the details of its internal architecture and maintaining a similar public dashboard [2]. However, these transparency efforts remain the exception rather than the norm. About network operators, most Internet Exchange Points (IXPs) provide publicly available peering information, a good way to peek into their infrastructure. However, public peering traffic remains anecdotal compared to transit and private (hidden) peering [4]. Worse, some network operators deploy deceptive techniques to make active probes useless (e.g. traceroutes) [236]. As of today, Internet mapping is usually done by independent researchers [8] and may result in inaccuracies due to these efforts. By incentivizing operators to publish details about their network, regulators and lawmakers could improve the situation while empowering Internet users with reliable, independent tools [93].

6.2.2 The way towards standard measurement APIs

There exists several QoS measurement tools based on modern browsers (including DiagSys, our contribution from Chapter 5). However, the results offered by these tools might not be comparable in practice: they may use different types of reference servers or different methodologies. Moreover, such tools are inherently limited by the capabilities of web browsers:

— It is not possible to obtain low-level information about a system and its connectivity (wireless or wired connection type);

1. `https://dashboards.gitlab.com`
2. `https://grafana.wikimedia.org`

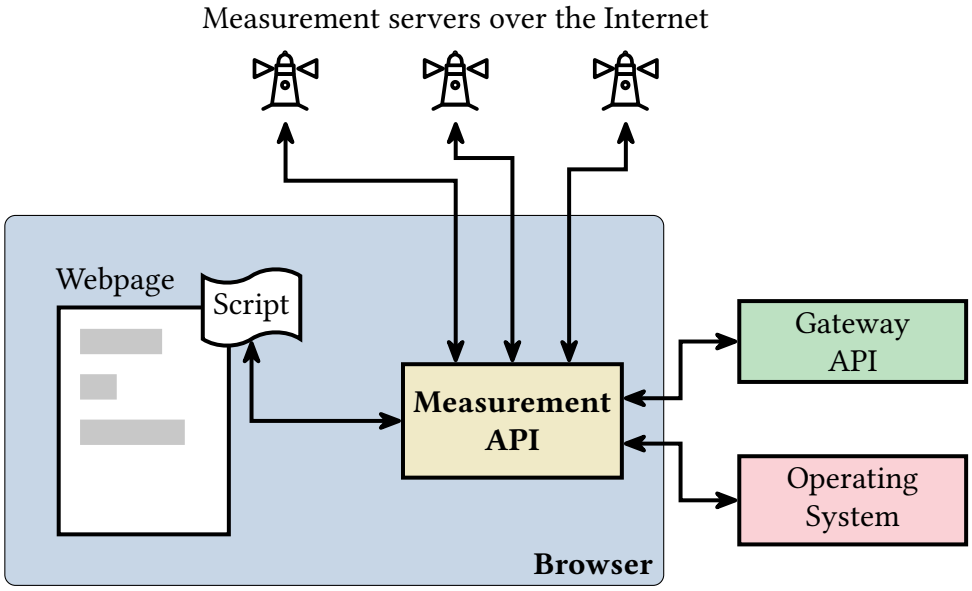


Figure 6.1 – Our proposal for a standard browser measurement API: webpage's scripts could call this (privileged) API to obtain accurate measurements towardslandmark servers. It could provide relevant information from available network gateways and the operating system itself.

— Without webcam authorization, measurements must be done over TCP, a protocol known to be conservative during the *slow-start* phase (Figure 5.3);

— When measuring large bandwidth, a browser might be limited by its actual computing power due to the limitations of JavaScript;

— We showed that estimating the QoE of third-party service requires a WebExtension to *bypass* CORS protections.

In an attempt to solve the first point, the French telecommunication regulator (ARCEP) proposed a measurement API that all ISP must include in their gateway boxes starting from 2022. This API would allow measurement scripts to access relevant information about one customer's connectivity, while ensuring a standardization of measurement scripts [4]. While the remaining three points may be addressed by using a dedicated software installed by users, we believe that browsers *should* provide dedicated APIs for network measurements. For instance, web browsers could provide a APIs (Figure 6.1) for *bandwidth estimation* using UDP or QoE estimation of any website using recognized metrics (e.g. ATF). These special capabilities would run from within browser internals, using optimized and secure system-level features. This new set of APIs could integrate well in the already existing "Web Performance" JavaScript module, and could provide standardized and accurate QoS/QoE measurements for end users,

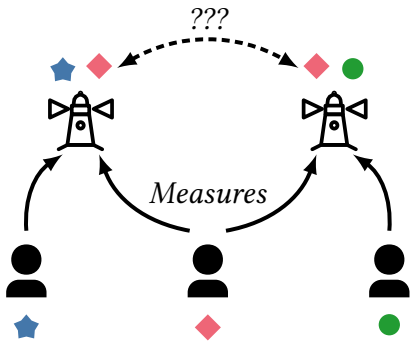


Figure 6.2 – Overview of a federated RCA service. After performing measurements (geometric shapes), end users send these measurements to one or several trusted landmark servers. Through federation, landmarks may now collaborate to build a shared RCA model.

third-party providers, network operators and regulators. For additional security and privacy, this proposal could be enabled in a dedicated browser prompt, for instance similar to the webcam access request: *"Do you authorize this website to measure your Internet connectivity?"*. This should prevent most leaks of sensitive information regarding the network configurations of end users.

6.2.3 Crowdsourcing as a potential solution

While waiting for a more transparent Internet, the *crowdsourcing* of measurements seems to remain a promising path. That is the strategy we used in DiagNet and DiagSys: end users and reference servers *collaborate* by sharing their measurements, in an effort to build a common knowledge about Internet architecture. This strategy is also fostered by public databases maintained by communities (e.g. RIPE Atlas [204] for measurements or PeeringDB for IXP data). In this document, we have described network modeling and RCA services as *centralized* tools, where clients send their metrics and receive *predictions*. With this design, all the relevant metrics are collected in a single point, allowing model training with all available measurements. However, such a design cannot scale realistically to millions of users and would pose privacy problems since a central service could track all metrics from all users without their consent. For instance, it is known that RTT measurements may be sufficient to localize end users over the globe [237, 238]. When combining several measurements, one rogue RCA service could even identify users with high precision [239]. In early experiments, we observed that a basic random forest model could re-identify clients from their QoS metrics with a recall of 95.5% in a pool of 41 possible clients.

As an alternative to centralized services, we have started exploring how RCA services could be *federated* between different providers. End users would be able to select *which* RCA provider(s) they trust—and even host one provider by themselves if they prefer. Figure 6.2 presents a basic overview of a federated DiagNet, where landmarks both provide *measurement endpoints* and a *federated RCA*. In this preliminary architecture, users first collect measurements from several, untrusted landmarks; then they send *all their measurements* to one or several trusted landmarks providing RCA services. The main research challenge is to make landmarks *collaborate* to build RCA models from their collective knowledge. We started evaluating two approaches for DiagNet:

— **Model-sharing approaches** usually work in successive rounds, where landmarks (i) train a local model from their own knowledge, (ii) broadcast this local model to other landmarks and (iii) update their model from the combination of received models from other landmarks. This line of work has been initially presented by Google [240] and has seen a wide development. However, it might suffer from the large network delays between landmarks, non-uniformity of data [241] and is mainly restricted to neural networks.

— By contrast, **data-sharing approaches** propose to share *dataset samples* (i.e. user measurements) instead of whole models. The main challenge is to select *which samples* (or "meta" information about samples) should be shared with other training agents (i.e. landmarks). Common solutions include selecting random elements, building coresets [242] or sending information about the *distribution* of samples. We evaluated several of these techniques, and showed that *random* data sampling was (surprisingly) the solution leading to the most accurate models in our datasets. Data-sharing approaches can be applied to *any family* of machine learning models, since only the training data needs to be shared between landmarks. Moreover, it is significantly faster than model-sharing approaches since a single round is needed. The main downside is that it requires landmarks to *share* the data assigned by end users.

It is worth noting that the federated approaches we have sketched here remain vulnerable to a number of attacks on privacy and model integrity. We believe nonetheless that combined with standard APIs and with a greater transparency from network services and operators, they pave the path towards the future developments of scalable modeling of large computer networks.

Résumé en Français

L'Internet est devenu la principale méthode de communication pour de nombreuses organisations, personnes et machines. Il est alimenté par une fédération de milliers d'opérateurs, qui collaborent pour permettre des communications rapides et résilientes entre n'importe quel appareil sur terre. L'Internet a été à bien des égards un succès retentissant, mais il doit encore relever des défis de taille. En particulier, avec l'augmentation de la demande mondiale d'Internet et de l'offre de services multimédias, il est de plus en plus difficile pour les opérateurs de garantir une bonne qualité d'expérience aux utilisateurs. Cependant, la complexité du réseau est le plus souvent cachée aux utilisateurs finaux : le dépannage des pannes de connectivité reste difficile en pratique. Les récentes avancées dans le domaine de l'apprentissage automatique ont montré que les approches basées sur les données peuvent aider à comprendre des systèmes complexes tels que l'Internet. Dans cette thèse, nous montrons que ces approches peuvent être utilisées pour la modélisation de grands réseaux informatiques, même lorsque les informations disponibles sont limitées.

La complexité croissante des réseaux informatiques

La demande pour un Internet plus rapide a globalement et fortement augmenté au cours des dernières années. Selon les *"Facts and Figures"* de l'ITU [1], plus de 50% de la population mondiale sera connectée en 2020, contre moins de 30% il y a dix ans. En particulier, 69% des personnes âgées de 15 à 24 ans ont accès à l'Internet. Le nombre de ménages disposant d'un accès au haut débit a également doublé, passant de 30% en 2010 à 57% en 2019. Alors que le nombre de personnes connectées augmente, la bande passante dont chaque individu a besoin a également augmenté de manière significative. Cette évolution est naturelle compte tenu des progrès des technologies des médias permettant de capturer, de traiter et d'afficher à moindre coût des contenus haute définition (images, musique, vidéos, jeux, etc.). Par exemple, d'après les estimations de Cisco [2], le nombre de téléviseurs 4K a doublé entre 2018 et 2020. Cette tendance devrait se poursuivre car de plus en plus d'appareils sont également connectés à Internet au fil du temps : les prévisions montrent que plus de 13 milliards d'appareils pourraient être connectés en 2023, y compris des appareils gourmands en bande passante comme les

véhicules autonomes ou les caméras de surveillance nuagiques. La crise du Covid-19 a encore amplifié la demande de trafic à large bande en raison des blocages régionaux et d'un intérêt plus large pour le télétravail. (À titre d'exemple, l'Italie a connu 44% de trafic supplémentaire pendant les confinements de 2020 par rapport à la même période en 2019 [1]).

Pour faire face à cette demande croissante, les opérateurs Internet ont investi pour étendre leur réseau : de 2019 à mi-2020, 200 Tbit/sec de capacité supplémentaire ont été ajoutés au niveau mondial selon l'ITU, portant le total estimé à 700 Tbit/sec. Depuis 2018, la bande passante moyenne mondiale fixe (resp. mobile) des connexions des utilisateurs a grimpé à un taux de 20% (resp. 27%) par an, et atteindra probablement 110 Mbit/sec en 2023 (resp. 44 Mbit/sec) [2]. Si cela signifie que davantage de câbles intercontinentaux, de fibres locales et d'antennes sont déployés, la stratégie utilisée par les opérateurs pour gérer leurs réseaux a également dû évoluer. L'un des changements importants est le passage au SDN, où le plan de contrôle du réseau est découplé du plan d'acheminement (des données) : Ce paradigme permet une plus grande adaptabilité aux demandes dynamiques de leurs clients. Selon [2], 40% des opérateurs interrogés ont indiqué être déjà passés à de tels réseaux, et 55% des autres opérateurs prévoient un déploiement dans les deux ans. Le SDN permet en théorie d'automatiser la plupart des tâches d'ingénierie du trafic, donnant lieu à des configurations de réseau optimisées, des politiques de routage et d'échange de trafic entre opérateurs. Dans les réseaux 5G (dont la conception permet le SDN), les opérateurs sont incités à déployer des fonctions réseau "près de la périphérie" pour permettre de nouveaux cas d'utilisation tels que le *"cloud gaming"* ou l'analyse télémétrique à grande échelle. En pratique, cela signifie déployer d'importantes ressources informatiques dans des centres de données ou même des antennes près des utilisateurs finaux. Les services et applications Internet doivent également adapter leurs infrastructures pour répondre à la demande, en déployant des systèmes distribués sur plusieurs sites dans le monde, par exemple. Là encore, malgré un large éventail d'offres de services nuagiques, ces systèmes distribués ajoutent naturellement de la complexité et de nouvelles dépendances dans l'équation. À titre d'exemple, Netflix a déployé des serveurs de livraison dans plus de 500 points de présence pour faire face à la demande mondiale de streaming vidéo [3], permettant au service de devenir le plus grand fournisseur de contenu en France en 2020 [4], devant Google et son service vidéo Youtube.

Toute cette complexité supplémentaire peut toutefois se retourner contre nous et entraîner de graves pannes. Le 17 juillet 2020, alors qu'elle tentait de résoudre un problème relativement

bénin de congestion du réseau, une équipe d'ingénieurs de Cloudflare a mis à jour la configuration d'un routeur. Cependant, ce changement de configuration contenait une erreur qui redirigeait l'ensemble du trafic de Cloudflare vers un seul emplacement du réseau : cela a conduit à la panne mondiale de la plupart des 12 millions de sites Web desservis par l'entreprise pendant environ 30 minutes [5]. Les systèmes automatisés peuvent également connaître des échecs spectaculaires : le 30 août 2020, CenturyLink/Level3 (l'un des plus grands opérateurs de réseau selon CAIDA [6]) a déclenché une petite modification pour protéger un client contre une cyberattaque. Une chaîne d'événements automatisés a conduit CenturyLink/Level3 à annoncer un grand nombre de changements de routage à d'autres opérateurs de réseau, mettant ainsi hors service une grande partie de l'Internet pendant près de cinq heures [7]. Dans ces exemples, les relations entre la cause racine et les symptômes observés n'étaient pas évidentes, en raison des nombreuses dépendances entre les composants du réseau. Pour les observateurs externes tels que les chercheurs ou les utilisateurs finaux, le dépannage et la compréhension des pannes d'Internet sont encore plus complexes, car la plupart de ces dépendances sont cachées (pour des raisons techniques ou commerciales). Bien que certaines informations publiques soient disponibles par le biais des points d'interconnexion (IXPs), la plupart de la topologie de l'Internet et des relations entre les parties prenantes de l'Internet restent difficiles à déduire [8].

Approches axées sur les données

Les logiciels d'automatisation, d'optimisation et de prise de décision sont largement utilisés par les opérateurs de réseaux pour tenter d'offrir une bonne qualité de service à leurs clients et de réduire les coûts opérationnels. Historiquement basés sur des règles de décision statiques conçues par des experts, ces programmes peuvent avoir du mal à comprendre les dépendances complexes des réseaux informatiques modernes. On constate cependant un intérêt croissant pour les approches *statistiques* et *axées sur les données* de la conception et de l'exploitation des réseaux : les algorithmes *d'apprentissage automatique* tirent parti des grandes quantités de données recueillies par les opérateurs pour produire des modèles pertinents de leurs réseaux. Pourtant, les applications des techniques d'apprentissage automatique aux réseaux informatiques et à l'Internet restent difficiles en pratique en raison de la *grande échelle* des réseaux gérés. Ces réseaux sont en effet difficiles à instrumenter afin d'obtenir des mesures continues et des informations actualisées. De plus, avec l'évolution récente vers le *fog-* et le *edge-computing*, de plus en plus de décisions de gestion sont prises automatiquement par des équipements ayant une connaissance *limitée* du réseau complet.

Au cours de la dernière décennie, les progrès de l'informatique ont permis de développer des solutions révolutionnaires avec des modèles de *réseau neuronal* et l'*apprentissage profond*. Un exemple très populaire est AlphaGo de Google, le premier logiciel capable de battre des joueurs de Go professionnels [9] (2016). D'autres exemples concernent un large éventail de domaines, allant du sujet historique de l'analyse d'images à la synthèse de contenu multimédia, en passant par les modèles de traduction pilotés par les données par exemple (une bonne partie de ce résumé a d'ailleurs été traduite par un modèle de réseau neuronal !). Grâce à leur expressivité, ces solutions se sont avérées plus efficaces pour modéliser des systèmes et des dépendances complexes que la plupart des approches traditionnelles. En particulier, si l'on dispose de suffisamment de données, les modèles d'apprentissage profond sont capables d'extraire ("apprendre") des relations complexes cachées dans les données d'entrée, ce qui conduit à des solutions plus précises et généralisables.

Dans cette thèse, nous soutenons que l'apprentissage statistique peut aider à modéliser de grands réseaux informatiques aux interdépendances complexes, même avec des informations limitées sur les réseaux évalués. Nous démontrons cette affirmation en appliquant les récentes avancées des techniques de traitement d'images et de graphes aux réseaux informatiques, avec des données collectées à partir de réseaux *simulés* et *réels*.

Nos résultats plaident en faveur de l'intégration de techniques plus axées sur les données dans la conception et la gestion des réseaux informatiques. Alors que ces réseaux deviennent de plus en plus complexes et difficiles à gérer, ces techniques offrent la possibilité d'améliorer les performances des réseaux tout en réduisant les risques de pannes et les coûts opérationnels, des paramètres fondamentaux pour les opérateurs de réseaux. D'autre part, nos contributions montrent également que des observateurs externes peuvent utiliser l'apprentissage statistique pour étudier les caractéristiques d'un réseau informatique malgré le peu d'informations préalables dont ils disposent à son sujet. Ceci est très important pour un certain nombre d'acteurs, notamment les utilisateurs désireux d'évaluer les opérateurs de réseaux ou les campagnes de recherche tierces. Bien que nous concentrions cette thèse sur les réseaux informatiques, nous pensons que la plupart des contributions peuvent être appliquées à d'autres domaines de systèmes interdépendants avec des changements limités. Par exemple, nous pensons que nos techniques sont susceptibles de s'appliquer à tout système cyber-physique (par exemple, les réseaux intelligents ou les systèmes de contrôle industriels), aux réseaux sociaux et de transport ou aux processus chimiques et biologiques.

Problématiques de recherche et découvertes

Dans cette thèse, nous étudions différents problèmes liés à l'affirmation que nous avons faite dans la dernière section. Plus précisément, nous différencions principalement les problèmes en fonction de la *quantité d'information disponible* utilisable pour l'apprentissage statistique. En effet, les solutions aux problèmes considérés sont en réalité très différentes lorsque l'on a accès à la topologie complète du réseau et aux caractéristiques des composants internes, par rapport à celles où seules des mesures limitées sont possibles et où le réseau est considéré comme une "boîte noire". Cette différenciation correspond plus ou moins à la disparité des connaissances entre un opérateur de réseau (ayant une vue d'ensemble relativement bonne de son réseau) et les observateurs externes. En conséquence, les défis que nous considérons sont différents en fonction de la quantité de connaissances disponibles :

— Dans un premier problème, nous considérons le cas d'un opérateur de réseau désireux d'estimer les paramètres de performance du réseau, et connaissant la topologie du réseau et les caractéristiques des composants du réseau ; Ces topologies et caractéristiques peuvent être actuellement déployées ou concerner un changement de configuration planifié ;

— Ensuite, nous nous demandons si la ou les causes racines des défaillances observées peuvent être identifiées avec un apprentissage statistique et des informations limitées sur un réseau. Une solution à ce second problème permettrait à toutes les parties prenantes de l'Internet de *comprendre* et de *résoudre* les défaillances, ce qui est très difficile aujourd'hui en raison du manque d'informations disponibles.

Les modèles statistiques sont généralement construits *à partir* d'un environnement spécifique (par exemple, un réseau informatique), *pour* ce même environnement. Cependant, les solutions pratiques aux défis susmentionnés doivent rester *généralisables* à tout réseau informatique, quelle que soit sa taille. En proposant des modèles extensibles et généralisables *par conception*, nous nous assurons que nos contributions répondent à cette exigence.

Modélisation de réseaux pour l'estimation rapide des performances

En raison des interdépendances complexes entre les composants d'un réseau informatique, il est généralement difficile de *prédire* comment une configuration de réseau donnée se comportera en pratique. Il s'agit d'un problème majeur pour les opérateurs de réseau désireux d'optimiser leurs configurations sans introduire de problèmes : de petits changements

dans une partie du réseau pourraient entraîner des changements radicaux de performance dans d'autres parties apparemment sans rapport. Historiquement, les opérateurs ont utilisé des connaissances d'experts et des simulations, mais ces approches peuvent être imprécises et lentes. Dans le contexte des réseaux dynamiques à grande échelle, de nouvelles solutions robustes et efficaces sont aujourd'hui nécessaires.

Le paradigme KDN [10] a été proposé à cet égard. Il propose d'exploiter les capacités combinées des réseaux SDN (collecte de métriques et gestion de réseau facilitées) et de l'apprentissage automatique pour *concevoir* et *contrôler* efficacement de grands réseaux informatiques complexes. Avec le KDN, les réseaux doivent être modélisés à partir de la configuration connue du réseau afin que des décisions automatisées puissent être appliquées. Un des objectifs de cette modélisation est de *prédire la performance du réseau en fonction des changements de configuration* afin de prendre les meilleures décisions. Inspirés par un travail précédent basé sur les réseaux neuronaux en graphe (*Graph Neural Networks*) [11], nous proposons dans cette thèse de modéliser explicitement les dépendances entre les composants du réseau dans de multiples graphes bipartis. Nous appliquons des techniques récentes sur ce modèle des dépendances et obtenons des prédictions avec une très bonne précision pour des réseaux de grande taille en quelques centaines de millisecondes (en comparaison, les outils de simulation nécessitent plusieurs *minutes* pour les mêmes prédictions). En particulier, notre contribution peut supporter des routeurs avec diverses *politiques d'ordonnancement*. Nous montrons qu'elle peut être généralisée à différentes topologies et configurations de réseau et nous explorons son fonctionnement interne avec des visualisations des représentations apprises. Ce travail a été réalisé dans le cadre d'un défi d'apprentissage automatique co-organisé par l'ITU et le Barcelona Networking Center [12].

Analyse des Causes Racines depuis la périphérie

Les utilisateurs de l'Internet fixe et mobile rencontrent régulièrement des dégradations de performances et des problèmes de connectivité lorsqu'ils accèdent à des services sur Internet. Lors de ces incidents, il est difficile pour les utilisateurs finaux d'identifier les *causes racines* derrière les symptômes observés : est-ce dû à une mauvaise couverture sans fil ? Une interruption de service ? Une mauvaise configuration du dispositif ? Les Fournisseurs d'Accès à Internet (FAI) sont naturellement blâmés par les utilisateurs pour de nombreuses dégradations perçues, puisqu'ils sont chargés de fournir une bonne connectivité Internet. Par conséquent, leurs équipes d'assistance clientèle sont souvent sollicitées pour des symptômes qui ne sont pas réellement causés par eux-mêmes. Cela entraîne des coûts d'assistance supplémentaires

et une dégradation de la relation client : les services clients des fournisseurs de télévisions et de télécom ont été classés parmi les pires aux États-Unis en 2018 [14].

Les utilisateurs finaux et les FAIs bénéficieraient tous deux d'une solution automatisée capable de déterminer l'origine la plus probable des pannes. Les clients seraient alors en mesure de localiser rapidement les pannes tout en évitant la plupart des appels téléphoniques coûteux en temps, menant à une plus grande satisfaction (75% des citoyens américains des plus jeunes générations préfèrent les communications écrites dans les relations avec les services client). Inversement, les FAIs ne recevraient que les rapports relatifs aux causes racines susceptibles de relever de leur domaine de responsabilité. Les services Internet tiers impactés bénéficieraient également d'une meilleure expérience client et de rapports automatiques, car même de petites dégradations des performances peuvent entraîner une baisse des revenus [16, 17].

Dans ce contexte, les informations disponibles sont drastiquement réduites par rapport au problème précédent : en faisant l'analyse du point de vue d'un seul utilisateur, il n'est pas possible d'avoir accès à la topologie complète et aux dépendances des services Internet. Pour résoudre ce problème, nous proposons d'utiliser une combinaison de mesures participatives et l'utilisation de serveurs de mesure de référence (*"landmarks"* ou "mires de test") déployés dans de nombreux sites Internet : en collectant des mesures à partir de ces points d'observation, nous montrons qu'il est possible d'identifier de nombreuses causes racines, même sans collaboration avec les FAIs. Nous exploitons un ensemble de données collectées avec des défaillances contrôlées pour construire des modèles de causes racines, et montrons qu'un réseau neuronal convolutif surpasse deux solutions alternatives dérivées d'algorithmes classiques d'apprentissage automatique. En particulier, nos modèles *"Convolutional Neural Network"* sont *extensibles* et peuvent supporter un nombre *dynamique* de mires de tests par conception.

Exploitation des navigateurs web pour la collecte de métriques réseau

La troisième contribution que nous présentons dans cette thèse est liée à la *collecte* de métriques pertinentes pour l'analyse des causes racines. Dans ce dernier travail, nous soulignons le besoin de métriques sur la qualité d'expérience, et nous plaidons pour *l'instrumentation de l'environnement d'exécution des appareils des utilisateurs finaux* à cet égard. Plus précisément, nous soutenons que les navigateurs web sont de bonnes plateformes pour la collecte de métriques qui peuvent fournir des estimations précises à la fois sur la qualité d'expérience et sur la qualité de service réseau. Nous proposons un ensemble de méthodes basées sur les capacités des navigateurs, en tenant compte des restrictions de sécurité des navigateurs web modernes.

Pendant plus d'un an, nous avons déployé et collecté des mesures en utilisant nos contributions et avec l'aide d'utilisateurs volontaires. Nous avons mis en évidence plusieurs résultats intéressants à partir de l'ensemble de données collectées, notamment les variations de qualité d'expérience dans le temps, le repérage d'une différenciation du trafic ou d'importants changements dans le routage Internet. Nous proposons également une approche statistique pour identifier les défauts et leurs causes racines observés dans notre ensemble de données. Cet effort de collecte de données est nécessaire pour toute solution axée sur les données : il a permis d'évaluer les techniques que nous avons proposées avec de *vrais* défauts et dégradations de qualité d'expérience. Bien que les performances globales soient plus faibles que lors de l'évaluation sur un jeu de données avec des fautes contrôlées, notre étude démontre tout de même l'intérêt de l'apprentissage statistique pour le problème d'analyse des causes racines et soulève un certain nombre de questions et de pistes de recherche intéressantes pour les travaux futurs dans ce domaine.

Plan du document

Cette thèse suit le synopsis que nous avons décrit dans la section précédente, chaque chapitre étant écrit pour être autonome.

— Nous commençons dans le Chapitre 2 en introduisant quelques définitions importantes sur les *réseaux informatiques*, suivies d'informations supplémentaires sur la modélisation de ces réseaux et l'analyse des causes racines. Dans la section 2.4, nous exposons également certaines notions de base de l'apprentissage automatique, notamment les développements récents des réseaux neuronaux.

— Dans le Chapitre 3, nous détaillons l'application de réseaux neuronaux en graphe généralisables pour la prédiction de performance sur les chemins du réseau. Ce chapitre comprend les détails de nos résultats lors du défi GNN co-organisé par l'ITU et le Barcelona Neural Networking Center.

— Le Chapitre 4 présente notre deuxième contribution, DiagNet, un modèle de réseau neuronal extensible pour l'analyse de causes racines à partir d'équipements d'utilisateur final. En particulier, nous montrons comment DiagNet se comporte par rapport à deux autres modèles dérivés d'approches d'apprentissage statistique classiques (c'est-à-dire un classifieur extensible naïf Bayesien et un classifieur extensible de forêt aléatoire). Pour évaluer nos modèles, nous exploitons un ensemble de données collectées sur quelques semaines avec des fautes injectées et contrôlées.

— Dans le but d'évaluer comment DiagNet se comporte avec des fautes d'Internet *réelles*, nous avons conçu et déployé une plateforme de mesure complète pendant plus d'un an. Nous présentons cette plateforme, DiagSys, dans le Chapitre 5. Cette dernière contribution montre que la modélisation de réseaux avec des fautes réelles est beaucoup plus difficile qu'avec des fautes contrôlées. Cependant, nous avons réussi à construire des modèles statistiques à partir de cet ensemble de données, et nous présentons nos résultats dans ce chapitre.

— Enfin, nous concluons cette thèse dans le Chapitre 6, où nous détaillons également des perspectives et des idées prometteuses pour la modélisation des réseaux et les services d'analyse des causes racines pour les années à venir.

Publications

Papers included in this dissertation:

— Loïck Bonniot, Christoph Neumann, and François Taïani. « DiagSys: Network and Third-Party Web-Service Monitoring from the Browser's Perspective (Industry Track) ». In: *Proceedings of the 21st International Middleware Conference Industrial Track*. Association for Computing Machinery, 2020, pp. 16–22. ISBN: 978-1-4503-8201-4. DOI: 10.1145/3429357.3430520. URL: https://hal.archives-ouvertes.fr/hal-02967290

— Loïck Bonniot, Christoph Neumann, and François Taïani. « Towards Internet-Scale Convolutional Root-Cause Analysis with DiagNet ». In: *35th IEEE International Parallel & Distributed Processing Symposium*. 2021. DOI: 10.1109/IPDPS49936.2021.00084. URL: https://hal.archives-ouvertes.fr/hal-02534888

Papers not included:

— Loïck Bonniot, Christoph Neumann, and François Taïani. « PnyxDB: a Lightweight Leaderless Democratic Byzantine Fault Tolerant Replicated Datastore ». In: *2020 International Symposium on Reliable Distributed Systems (SRDS)*. 2020, pp. 155–164. DOI: 10.1109/SRDS51746.2020.00023. URL: https://hal.archives-ouvertes.fr/hal-02355778

Papers under submission at the time of writing:

— Under submission at Neural Information Processing Systems (NeurIPS 2021):
Loïck Bonniot, Christoph Neumann, François Schnitzler and François Taïani.
« Computer Network Modeling with Graph Neural Networks ».

— Under submission at ACM SIGCOMM Computer Communication Review (CCR):
José Suárez-Varela, Miquel Ferriol-Galmés, Albert López, Paul Almasan, Guillermo Bernárdez, David Pujol-Perich, Krzysztof Rusek, Loïck Bonniot, Christoph Neumann, François Schnitzler, François Taïani, Martin Happ, Christian Maier, Jia Lei Du, Matthias Herlich, Peter Dorfinger, Nick Vincent Hainke, Stefan Venz, Johannes Wegener, Henrike Wissing, Bo Wu, Shihan Xiao, Pere Barlet-Ros, Albert Cabellos-Aparicio. « The Graph Neural Networking Challenge: A Worldwide Competition for Education in AI/ML for Networks ».

Acronyms

AI Artificial Intelligence
API Application Programming Interface
AS Autonomous System
ATF Above The Fold
AWS Amazon Web Services

BGP Border Gateway Protocol
BLE Bluetooth Low Energy

CDN Content Delivery Network
CNN Convolutional Neural Network
CORS Cross-Origin Resource Sharing
CPU Control Processing Unit

DHT Distributed Hash Table
DNS Domain Name Service
DRR Deficit Round Robin
DSL Digital Subscriber Line

ECMP Equal-cost multi-path

GBN German Backbone Network
GÉANT2 Gigabit European Advanced Network Technology 2
GNN Graph Neural Network
GPU Graphical Processing Unit

ICMP Internet Control Messaging Protocol
IP Internet Protocol
IQR Interquartile Range
ISP Internet Service Provider
ITU International Telecommunication Union
IXP Internet Exchange Point

KDE Kernel Density Estimation
KDN Knowledge-Defined Networking

LSTM Long Short-Term Memory

MAE Mean Absolute Error
MAPE Mean Absolute Percentage Error
MLP Multilayer Perceptron
MSE Mean Square Error

NAT Network Address Translation
NSFNET National Science Foundation Network

PLT Page Load Time
PoP Point of Presence

QoE Quality of Experience
QoS Quality of Service

RCA Root Cause Analysis
ReLU Rectified Linear Unit
RNN Recurrent Neural Network
RTT Round Trip Time

SDN Software-Defined Networking
SVM Support Vector Machine

t-SNE t-distributed Stochastic Neighbor Embedding

TCP Transport Control Protocol

ToS Type of Service

TTL Time To Live

UDP User Datagram Protocol

WFQ Weighted Fair Queuing

XSS Cross-Site Scripting

Bibliography

[1] ITU-D. « Measuring digital development - Facts and figures ». In: *ITU Publications* (2020).

[2] Cisco. *Cisco Annual Internet Report (2018-2023)*. Tech. rep. 2020. DOI: 10.1016/s1361-3723(20)30026-9.

[3] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. « Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix CDN ». In: *SIGCOMM CCR* 48.1 (2018), pp. 28–34. DOI: 10.1145/3211852.3211857.

[4] ARCEP. *L'état d'internet en France*. Tech. rep. 2020.

[5] John Graham-Cumming. *Cloudflare outage on July 17, 2020*. 2020. URL: https://blog.cloudflare.com/cloudflare-outage-on-july-17-2020/ (visited on 03/31/2021).

[6] CAIDA. *ASRank*. 2021. URL: https://asrank.caida.org/ (visited on 03/31/2021).

[7] Angelique Medina. *CenturyLink/Level3 Outage Analysis*. 2020. URL: https://blog.thousandeyes.com/centurylink-level-3-outage-analysis/ (visited on 03/31/2021).

[8] Kevin Vermeulen. « lmproved algorithms for capturing Internet maps ». PhD thesis. 2020.

[9] BBC News. *Google achieves AI 'breakthrough' by beating Go champion*. 2016. URL: https://www.bbc.com/news/technology-35420579 (visited on 03/31/2021).

[10] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcn, Marc Sol, Victor Munts-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, Giovani Estrada, Khaldun Ma 'ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, Albert Cabellos, Fernando Ramos, Eduard Alarcón, Marc Solé, and Victor Muntés-Mulero. « Knowledge-Defined Networking ». In: *SIGCOMM CCR* 47.3 (2017), pp. 2–10.

[11] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. « Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN ». In: *SOSR*. Jan. 2019. ISBN: 978-1-4503-6710-3. DOI: 10.1145/3314148.3314357.

[12] ITU. *ITU AI/ML in 5G Challenge*. 2020. URL: https://www.itu.int/en/ITU-T/AI/challenge/2020/Pages/default.aspx (visited on 02/15/2021).

[14] Bruce Temkin. *2018 Temkin Experience Ratins, U.S.* Tech. rep. 2018.

[15] Hiver. *Customer Support Through The Eyes of Consumers in 2020*. 2020.

[16] Diana Zeaiter Joumblatt, Jaideep Chandrashekar, Branislav Kevton, and Renata Teixeira. « Predicting User Dissatisfaction with Internet Application Performance at End-Hosts ». In: *INFOCOM*. 2013.

[17] Hyunwoo Nam, Kyung Hwa Kim, and Henning Schulzrinne. « QoE matters more than QoS: Why people stop watching cat videos ». In: *INFOCOM*. 2016. ISBN: 978-1-4673-9953-1. DOI: 10.1109/INFOCOM.2016.7524426.

[20] Andrew S Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Computer Networks. Prentice Hall PTR, 2010. ISBN: 978-0-13-066102-9.

[21] Vangie Beal. *WiFi Definition & Meaning*. 2021. URL: https://www.webopedia.com/definitions/wifi/ (visited on 04/02/2021).

[22] John Hawkinson and Tony Bates. *RFC 1930: Guidelines for creation, selection, and registration of an Autonomous System (AS)*. 1996.

[23] Ramon Puigjaner. « Performance Modelling of Computer Networks ». In: *LANC*. 2003. ISBN: 1-58113-789-3.

[24] Gabriel A. Wainer. *Discrete-event modeling and simulation: a practitioner's approach*. 2009. ISBN: 978-1-4200-5336-4.

[25] Francisco J. Suárez, Pelayo Nuño, Juan C. Granda, and Daniel F. García. « Computer networks performance modeling and simulation ». In: *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*. Ed. by Mohammad S. Obaidat Zarai, Petros Nicopolitidis, and Faouzi. Morgan Kaufmann, 2015, pp. 187–223. ISBN: 978-0-12-801158-4. DOI: 10.1016/B978-0-12-800887-4.00007-9.

[26] Robert B Cooper. *Queuing Theory*. 2nd. 1981.

[27] Chee-Hock Ng and Boon-Hee Soong. *Queueing modelling fundamentals: With applications in communication networks*. 2nd. 2008.

[28] B. Filipowicz and J. Kwiecień. « Queueing systems and networks. Models and applications ». In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* 56.4 (2008), pp. 379–390.

[29] Leonard Kleinrock. *Queueing systems, volume 2: Computer applications*. 1976.

[30] Anthony Ephremides. « Limitations of Queueing Models in Communication Networks ». In: *Performance Limits in Communication Theory and Practice*. 1988, pp. 143–153.

[31] Martin Reiser. « A Queueing Network Analysis of Computer Communication Networks with Window Flow Control ». In: *IEEE Transactions on Communications* 27.8 (1979), pp. 1199–1209. DOI: 10.1109/TCOM.1979.1094531.

[32] Anthony Ephremides and Bruce Hajek. « Information theory and communication networks: An unconsummated union ». In: *IEEE Transactions on Information Theory* 44.6 (1998), pp. 2416–2434. DOI: 10.1109/18.720543.

[33] Carolina Osorio and Michel Bierlaire. « An analytic finite capacity queueing network model capturing the propagation of congestion and blocking ». In: *European Journal of Operational Research* 196.3 (2009), pp. 996–1007. DOI: 10.1016/j.ejor.2008.04.035.

[34] Anthony Ebert, Paul Wu, Kerrie Mengersen, and Fabrizio Ruggeri. « Computationally efficient simulation of queues: The r package queuecomputer ». In: *CoRR* (2019). DOI: 10.18637/jss.v095.i05. URL: https://arxiv.org/abs/1703.02151.

[35] Carl Adam Petri and Wolfgang Reisig. « Petri net ». In: *Scholarpedia* 3.4 (2008), p. 6477.

[36] Jacium Wang. *Timed petri nets: Theory and application*. Springer, 1998. ISBN: 978-1-4615-5537-7.

[37] Jonathan Billington and Michel Diaz, eds. *Application of Petri Nets to Communication Networks*. Springer, 1999. ISBN: 978-3-540-65870-2.

[38] F. Bause. « Queueing Petri Nets a formalism for the combined qualitative and quantitative analysis of systems ». In: *PNPM*. 1993. ISBN: 0-8186-4250-5. DOI: 10.1109/PNPM.1993.393439.

[39] E. Rodríguez, J. L. Arqués, R. Rodríguez, M. Nuñez, M. Medina, T. L. Talarico, I. A. Casas, T. C. Chung, W. J. Dobrogosz, L. Axelsson, S. E. Lindgren, W. J. Dobrogosz, Leila Kerkeni, Paula Ruano, Lismet Lazo Delgado, Sergio Picco, Liliana Villegas, Franco Tonelli, Mario Merlo, Javier Rigau, Dario Diaz, and Martin Masuelli. « On the Use of Queueing Petri Nets for Modeling and Performance Analysis of Distributed Systems ». In: *Petri Net, Theory and Applications*. Ed. by Vedran Kordic. Intech Open, 2008, pp. 534–566. ISBN: 978-3-902613-12-7.

[40] Jerry Banks, John Carson II, Barry Nelson, and David Nicol. *Discrete-Event System Simulation*. 5th. Pearson, 2010. ISBN: 978-0-13-606212-7.

[41] University of Washington NS-3 Consortium. *ns-3 Network Simulator*. 2021. URL: https://www.nsnam.org (visited on 01/22/2021).

[42] András Varga and Rudolf Hornig. « An Overview of the OMNeT++ Simulation Environment ». In: *SIMUTools*. 2008. ISBN: 978-963-9799-20-2.

[43] Sally Floyd and Eddie Kohler. « Internet research needs better models ». In: *SIGCOMM CCR* 33.1 (2003), pp. 29–34. DOI: 10.1145/774763.774767.

[44] Bob Lantz, Brandon Heller, and Nick McKeown. « A network in a laptop ». In: *HotNets*. 2010. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466.

[45] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. « Kollaps: Decentralized and Dynamic Topology Emulation ». In: *EuroSys*. 2020.

[46] InterDigital. *AdvantEdge mobile edge emulation platform*. 2021. URL: https://github.com/InterDigitalInc/AdvantEDGE (visited on 01/22/2021).

[47] The Kubernetes Authors. *Kubernetes*. 2021. URL: https://kubernetes.io (visited on 01/22/2021).

[48] OneLab consortium. *FIT Future Internet Testing Facility*. 2021. URL: https://fit-equipex.fr/ (visited on 01/22/2021).

[49] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. « Mahimahi: Accurate record-and-replay for http ». In: *ATC*. 2015. ISBN: 978-1-931971-22-5.

[50] Francis Y Yan, Jestin Ma, Greg D Hill, Riad S Wahby, Philip Levis, Keith Winstein, and Deepti Raghavan. « Pantheon: the training ground for Internet congestion-control research ». In: *ATC*. 2018. ISBN: 978-1-931971-44-7.

[51] Alexander Frömmgen. *Mininet / Netem Emulation Pitfalls A Multipath TCP Scheduling Experience*. Tech. rep. 2017.

[52] Renata Teixeira and Jennifer Rexford. « Managing routing disruptions in Internet service provider networks ». In: *IEEE Communications Magazine* 44 (2006), pp. 160–165. DOI: 10.1109/MCOM.2006.1607880.

[53] Xian Zhang and Chris Phillips. « A Survey on Selective Routing Topology Inference Through Active Probing ». In: *IEEE Communications Surveys and Tutorials* 14.4 (2012), pp. 1129–1141.

[54] Xing Jin, W. P.Ken Yiu, S. H.Gary Chan, and Yajun Wang. « Network topology inference based on end-to-end measurements ». In: *IEEE Journal on Selected Areas in Communications* 24.12 (2006), pp. 2182–2194. DOI: 10.1109/JSAC.2006.884016.

[55] Pietro Marchetta, Antonio Montieri, Valerio Persico, A. Pescape, Italo Cunha, and Ethan Katz-Bassett. « How and how much traceroute confuses our understanding of network paths ». In: *IEEE Workshop on Local and Metropolitan Area Networks* 2016-Augus (2016). DOI: 10.1109/LANMAN.2016.7548847.

[56] Tao Hou, Zhe Qu, Tao Wang, Zhuo Lu, and Yao Liu. « ProTO: Proactive Topology Obfuscation Against Adversarial Network Topology Inference ». In: *INFOCOM*. 2020, pp. 1598–1607. DOI: 10.1109/infocom41043.2020.9155255.

[57] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. « Pinpoint: Problem determination in large, dynamic internet services ». In: *Proceedings of the 2002 International Conference on Dependable Systems and Networks* (2002), pp. 595–604. DOI: 10.1109/DSN.2002.1029005.

[58] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson. « User-level internet path diagnosis ». In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), p. 106. DOI: 10.1145/1165389.945456.

[59] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. « Towards highly reliable enterprise network services via inference of multi-level dependencies ». In: *SIGCOMM*. 2007. ISBN: 978-1-59593-713-1. DOI: 10.1145/1282427.1282383.

[60] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. « X-trace: A pervasive network tracing framework ». In: *NSDI*. 2007. DOI: 10.1.1.108.2220.

[61] Xu Chen, Ming Zhang, Z Morley Mao, and Paramvir Bahl. « Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions ». In: *8th USENIX Symposium on Operating Systems Design and Implementation* (2008), pp. 117–130.

[62] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. « Detailed diagnosis in enterprise networks ». In: *ACM SIGCOMM Computer Communication Review* 39.4 (2009), p. 243. DOI: 10.1145/1594977.1592597.

[63] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, and Albert Greenberg. « WebProphet : Automating Performance Prediction for Web Services ». In: *Context* October (2010), p. 10.

[64] Y. Vardi. « Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data ». In: *Journal of the American Statistical Association* 91.433 (Mar. 1996), pp. 365–377. DOI: 10.1080/01621459.1996.10476697.

[65] Mark Coates, Alfred O. Hero, Robert Nowak, and Bin Yu. « Internet tomography ». In: *IEEE Signal Processing Magazine* 19.3 (May 2002), pp. 47–65. DOI: 10.1109/79.998081.

[66] N. G. Duffield and F. Lo Presti. « Multicast inference of packet delay variance at interior network links ». In: *INFOCOM*. 2000. DOI: 10.1109/infcom.2000.832532.

[67] N.G. Duffield, F. Lo Presti, V. Paxson, and D. Towsley. « Inferring link loss using striped unicast probes ». In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 2. IEEE, 2001, pp. 915–923. ISBN: 0-7803-7016-3. DOI: 10.1109/INFCOM.2001.916283.

[68] Denisa Ghita, Hung Nguyen, Maciej Kurant, Katerina Argyraki, and Patrick Thiran. « Netscope: Practical network loss tomography ». In: *INFOCOM*. 2010. ISBN: 978-1-4244-5836-3. DOI: 10.1109/INFCOM.2010.5461918.

[69] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. « Maximum likelihood network topology identification from edge-based unicast measurements ». In: *SIGMETRICS*. Vol. 30. 1. ACM Press, 2002, p. 11. DOI: 10.1145/511334.511337.

[70] N G Duffield, J Horowitz, F Lo Presti, and D Towsley. « Multicast Topology Inference from Measured ». In: *IEEE Transactions on Information Theory* 48.1 (2002), pp. 1–22.

[71] Ting He. « Distributed Link Anomaly Detection via Partial Network Tomography ». In: *ACM SIGMETRICS Performance Evaluation Review* 45.2 (Mar. 2018), pp. 29–42. DOI: 10.1145/3199524.3199532.

[72] Yao Zhao, Yan Chen, and David Bindel. « Towards Unbiased End-to-End Network Diagnosis ». In: *Transactions on Networking* 17.6 (2009), pp. 1724–1737. DOI: 10.1109/TNET.2009.2022158.

[73] Liang Ma, Ting He, Kin K Leung, Ananthram Swami, and Don Towsley. « Identifiability of link metrics based on end-to-end path measurements ». In: *Proceedings of the 13th ACM Conference on Internet Measurement (IMC 2013)*. 2013, pp. 391–404. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504738.

[74] Wei Ren and Wei Dong. « Robust network tomography: K-Identifiability and monitor assignment ». In: *Proceedings - IEEE INFOCOM*. Vol. 2016-July. 2016. ISBN: 978-1-4673-9953-1. DOI: 10.1109/INFOCOM.2016.7524375.

[75] Hung X Nguyen and Patrick Thiran. « The boolean solution to the congested IP link location problem: Theory and practice ». In: *Proceedings - IEEE INFOCOM*. 2007, pp. 2117–2125. ISBN: 1-4244-1047-9. DOI: 10.1109/INFCOM.2007.245.

[76] Gugan Thoppe, Vivek Borkar, and D. Manjunath. « A stochastic Kaczmarz algorithm for network tomography ». In: *Automatica* 50.3 (Mar. 2014), pp. 910–914. DOI: 10.1016/j.automatica.2013.12.016.

[77] S Tati, S Silvestri, T He, and T La Porta. « Robust network tomography in the presence of failures ». In: *Proceedings - International Conference on Distributed Computing Systems*. 2014, pp. 481–492. ISBN: 978-1-4799-5168-0. DOI: 10.1109/ICDCS.2014.56.

[78] Liang Ma, Ting He, Ananthram Swami, Don Towsley, Kin K. Leung, and Jessica Lowe. « Node Failure Localization via Network Tomography ». In: *Proceedings of the 2014 Conference on Internet Measurement Conference - IMC'14* (2014), pp. 195–208. DOI: 10.1145/2663716.2663723.

[79] Huikang Li, Yi Gao, Wei Dong, and Chun Chen. « Taming both predictable and unpredictable link failures for network tomography ». In: *Proceedings of the ACM Turing 50th Celebration Conference - China on - ACM TUR-C '17*. ACM Press, 2017, pp. 1–10. ISBN: 978-1-4503-4873-7. DOI: 10.1145/3063955.3063990.

[80] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. « Basic Concepts and Taxonomy of Dependable and Secure Computing ». In: *IEEE transactions on dependable and secure computing* (2004), pp. 11–33.

[81] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. « G-RCA: A generic root cause analysis platform for service quality management in large IP networks ». In: *CoNEXT*. 2010. DOI: 10.1109/TNET.2012.2188837.

[82] Jing Chen and Silvio Micali. « Algorand:The Efficient and Democratic Ledger ». In: *CoRR* (2016), pp. 1–75. URL: https://arxiv.org/abs/1607.01341.

[83] Yu Ming Ke, Hsu Chun Hsiao, and Tiffany Hyun Jin Kim. « SDNProbe: Lightweight fault localization in the error-prone environment ». In: *ICDCS*. IEEE, 2018. ISBN: 978-1-5386-6871-9. DOI: 10.1109/ICDCS.2018.00055.

[84] Yang Wu, Ang Chen, Implementation Nsdi, and Ang Chen. « Zeno : Diagnosing Performance Problems with Temporal Provenance ». In: *NSDI*. 2019. ISBN: 978-1-931971-49-2.

[85] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. « Passive Realtime Datacenter Fault Detection and Localization ». In: *NSDI*. 2017. ISBN: 978-1-931971-37-9.

[86] Qiao Zhang, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, Thomas Anderson, and Guo Yu. « Deepview: Virtual Disk Failure Diagnosis and Pattern Detection for Azure ». In: *NSDI*. 2018, pp. 519–532. ISBN: 978-1-931971-43-0.

[87] Chuanxiong Guo, Hua Chen, Zhi-Wei Lin, Varugis Kurien, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, and Bin Pang. « Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis ». In: *SIGCOMM*. ACM Press, 2015. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787496.

[88] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. « NetBouncer : Active Device and Link Failure Localization in Data Center Networks ». In: *NSDI*. 2019. ISBN: 978-1-931971-49-2.

[89] Behnaz Arzani, Selim Ciraci, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, Geoff Outhred, Systems Design, and Implementation Nsdi. « 007: Democratically Finding the Cause of Packet Drops ». In: *NSDI*. 2018. ISBN: 978-1-931971-43-0.

[90] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. « Detecting Peering Infrastructure Outages in the Wild ». In: *SIGCOMM*. 2017. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098855.

[91] Romain Fontugne, Cristel Pelsser, Emile Aben, and Randy Bush. « Pinpointing delay and forwarding anomalies using large-scale traceroute measurements ». In: *IMC*. ACM Press, 2017, pp. 15–28. ISBN: 978-1-4503-5118-8. DOI: 10.1145/3131365.3131384.

[92] Andreas Guillot, Romain Fontugne, Philipp Winter, Pascal Merindol, Alistair King, Alberto Dainotti, and Cristel Pelsser. « Chocolatine: Outage Detection for Internet Background Radiation ». In: *TMA*. 2019. ISBN: 978-3-903176-17-1.

[93] Francois Espinet, Diana Joumblatt, and Dario Rossi. « Framework, models and controlled experiments for network troubleshooting ». In: *Computer Networks* 107.Part 1 (2016), pp. 36–54. DOI: 10.1016/j.comnet.2016.06.001.

[94] Srikanth Kandula, Dina Katabi, and Jean-philippe Vasseur. « Shrink: A tool for failure diagnosis in IP networks ». In: *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. 2005, pp. 173–178. ISBN: 1-59593-026-4. DOI: 10.1145/1080173.1080178.

[95] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. « NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data ». In: *CoNEXT*. ACM Press, 2007, p. 1. ISBN: 978-1-59593-770-4. DOI: 10.1145/1364654.1364677.

[96] R.R. Kompella, J. Yates, A. Greenberg, and A.C. Snoeren. « IP Fault Localization via Risk Modeling ». In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010), pp. 1–14. DOI: 10.1109/TDSC.2009.37.

[97] Radhika Niranjan Mysore, Ratul Mahajan, Amin Vahdat, and George Varghese. « Gestalt: Fast, Unified Fault Localization for Networked Systems ». In: *ATC*. 2014, pp. 255–267. ISBN: 978-1-931971-10-2.

[98] Dan Rubenstein, Jim Kurose, and Don Towsley. « Detecting shared congestion of flows via end-to-end measurement ». In: *Transactions on Networking* 10.3 (2002), pp. 381–395. DOI: 10.1109/TNET.2002.1012369.

[99] Nick Duffield. « Network Tomography of Binary Network Performance Characteristics ». In: *IEEE Transactions on Information Theory* 52.12 (Dec. 2006), pp. 5373–5388. DOI: 10.1109/TIT.2006.885460.

[100] Iain A Stewart. « A general algorithm for detecting faults under the comparison diagnosis model ». In: *IPDPS*. 2010.

[101] Nagao Ogino, Takeshi Kitahara, Shin'ichi Arakawa, Go Hasegawa, and Masayuki Murata. « Decentralized boolean network tomography based on network partitioning ». In: *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Apr. 2016, pp. 162–170. ISBN: 978-1-5090-0223-8. DOI: 10.1109/NOMS.2016.7502809.

[102] Yanghua Peng, Ji Yang, Chuan Wu, Chuanxiong Guo, Chengchen Hu, and Zongpeng Li. « deTector: a Topology-aware Monitoring System for Data Center Networks ». In: *USENIX Annual Technical Conference (ATC)* (2017).

[103] Alexey Ignatiev, Antonio Morgado, Georg Weissenbacher, and Joao Marques-Silva. « Model-Based Diagnosis with Multiple Observations ». In: *IJCAI*. 2019, pp. 1108–1115. DOI: 10.24963/ijcai.2019/155.

[104] Caitlin Gray, Clemens Mosig, Randy Bush, Cristel Pelsser, Matthew Roughan, Thomas C. Schmidt, and Matthias Wahlisch. « BGP Beacons, Network Tomography, and Bayesian Computation to Locate Route Flap Damping ». In: *IMC*. 2020. ISBN: 978-1-4503-8138-3. DOI: 10.1145/3419394.3423624.

[105] Yiyi Huang, Nick Feamster, and Renata Teixeira. « Practical issues with using network tomography for fault diagnosis ». In: *SIGCOMM CCR* 38.5 (2008), pp. 53–58. DOI: 10.1145/1452335.1452343.

[106] Denisa Ghita, Can Karakus, Katerina Argyraki, and Patrick Thiran. « Shifting network tomography toward a practical goal ». In: *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11* (2011), pp. 1–12. DOI: 10.1145/2079296.2079320.

[107] M Steinder and A.S. Sethi. « End-to-end service failure diagnosis using belief networks ». In: *NOMS*. 2002, pp. 375–390. ISBN: 0-7803-7382-0. DOI: 10.1109/NOMS.2002.1015595.

[108] Irina Rish, Mark Brodie, Natalia Odintsova, and Genady Grabarnik. « Real-time problem determination in distributed systems using active probing ». In: *NOMS*. 2004, pp. 133–146. ISBN: 0-7803-8230-7. DOI: 10.1109/NOMS.2004.1317650.

[109] Emre Kiciman, David Maltz, and John C Platt. « Fast Variational Inference for Large-scale Internet Diagnosis ». In: *NIPS*. 2008.

[110] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. « Detection and localization of network black holes ». In: *INFOCOM*. 2007, pp. 2180–2188. ISBN: 1-4244-1047-9. DOI: 10.1109/INFCOM.2007.252.

[111] Ookla. *Speedtest Servers*. 2020. URL: https://www.speedtest.net/speedtest-servers (visited on 04/15/2020).

[112] Speedtest by Ookla. *Global Fixed and Mobile Network Performance Maps. Based on analysis by Ookla of Speedtest Intelligence data for 2020*. 2020. URL: https://registry.opendata.aws/speedtest-global-performance/ (visited on 02/01/2021).

[113] Venkata N. Padmanabhan, Sriram Ramabhadran, and Jitendra Padhye. « NetProfiler: Profiling wide-area networks using peer cooperation ». In: *IPTPS*. 2005. ISBN: 3-540-29068-0. DOI: 10.1007/11558989_8.

[114] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. « Netalyzr: Illuminating The Edge Network ». In: *IMC*. 2010. ISBN: 978-1-4503-0483-2.

[115] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P Gummadi, Ratul Mahajan, and Stefan Saroiu. « Glasnost : Enabling End Users to Detect Traffic Differentiation ». In: *NSDI*. 2010.

[116] Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson. « Fathom: a browser-based network measurement platform ». In: *IMC*. 2012. DOI: 10.1145/2398776.2398786.

[117] Chathuranga Widanapathirana, Jonathan Li, Y. Ahmet Şekercioğlu, Milosh Ivanovich, and Paul Fitzpatrick. « Intelligent automated diagnosis of client device bottlenecks in private clouds ». In: *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*. 2011, pp. 261–266. ISBN: 978-0-7695-4592-9. DOI: 10.1109/UCC.2011.42.

[118] Bhavish Aggarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N Padmanabhan, and Geoffrey M Voelker. « NetPrints: Diagnosing home network misconfigurations using shared knowledge ». In: *Proceedings of the 6th USENIX symposium on Networked systems design and implementation* Di.July (2009), pp. 349–364.

[119] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. « Taking the Blame Game out of Data Centers Operations with NetPoirot ». In: *SIGCOMM*. 2016, pp. 440–453. ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2934884.

[120] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-ros, Konstantina Papagiannaki, and Peter Steenkiste. « Identifying the Root Cause of Video Streaming Issues on Mobile Devices ». In: *CoNEXT*. 2015. ISBN: 978-1-4503-3412-9. DOI: 10.1145/2716281.2836109.

[121] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. « Home network or access link? Locating last-mile downstream throughput bottlenecks ». In: *PAM*. 2016. ISBN: 978-3-319-30504-2. DOI: 10.1007/978-3-319-30505-9_9.

[122] Heng Cui and Ernst Biersack. « Troubleshooting slow webpage downloads ». In: *Proceedings - IEEE INFOCOM* (2013), pp. 3285–3290. DOI: 10.1109/INFCOM.2013.6567152.

[123] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. « A Knowledge Plane for the Internet ». In: *SIGCOMM*. 2003. ISBN: 1-58113-735-4.

[124] Ookla. *Downdetector*. 2020. URL: https://downdetector.com (visited on 04/15/2020).

[125] Helen J Wang, John C Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. « Automatic Misconfiguration Troubleshooting with PeerPressure ». In: *OSDI*. 2004.

[126] Helen J Wang, Yih-Chun Hu, Chun Yuan, Zheng Zhang, and Yi-Min Wang. « Friends Troubleshooting Network: Towards Privacy-Preserving, Automatic Troubleshooting. » In: *Iptps* (2004), pp. 184–194.

[127] Y Zhang, Zm Mao, and M Zhang. « Effective Diagnosis of Routing Disruptions from End Systems ». In: *Nsdi* (2008), pp. 219–232.

[128] David R. Choffnes, Fabián E. Bustamante, and Zihui Ge. « Crowdsourcing service-level network event monitoring ». In: *SIGCOMM*. 2010. ISBN: 978-1-4503-0201-2. DOI: 10.1145/1851275.1851228.

[129] Kyung Hwa Kim, Hyunwoo Nam, Vishal Singh, Daniel Song, and Henning Schulzrinne. « DYSWIS: Crowdsourcing a home network diagnosis ». In: *ICCCN*. 2014. ISBN: 978-1-4799-3572-7. DOI: 10.1109/ICCCN.2014.6911758.

[130] Kyung Hwa Kim, Hyunwoo Nam, Jin Hyung Park, and Henning Schulzrinne. « MoT: A collaborative network troubleshooting platform for the Internet of Things ». In: *IEEE Wireless Communications and Networking Conference, WCNC* (2014), pp. 3438–3443. DOI: 10.1109/WCNC.2014.6953136.

[131] Maximilian Bachl, Renata Teixeira, Timur Friedman, Claudio Sacchi, and Anna-Kaisa Pietilainen. « Collaborative Home Network Troubleshooting ». In: *CoRR* (2016). URL: https://hal.inria.fr/hal-01415767.

[132] Emmanuelle Anceaume, Erwan Le Merrer, Romaric Ludinard, Bruno Sericola, and Gilles Straub. « FixMe: A self-organizing isolated anomaly detection architecture for large scale distributed systems ». In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7702 LNCS (2012), pp. 1–15. DOI: 10.1007/978-3-642-35476-2_1.

[133] Petar Maymounkov and David Mazières. « Kademlia: A Peer-to-Peer Information System Based on the XOR Metric ». In: *IPTPS*. 2002, pp. 53–65. ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5.

[134] Yu Jin, Nick Duffield, Alexandre Gerber, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. « NEVERMIND, the problem is already fixed: proactively detecting and troubleshooting customer DSL problems ». In: *CoNEXT*. 2010. ISBN: 978-1-4503-0448-1. DOI: 10.1145/1921168.1921178.

[135] Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. « FChain: Toward black-box online fault localization for cloud systems ». In: *Proceedings - International Conference on Distributed Computing Systems* (2013), pp. 21–30. DOI: 10.1109/ICDCS.2013.26.

[136] Christian Callegari, Marco Milanesio, and Pietro Michiardi. « Troubleshooting web sessions with CUSUM ». In: *IWCMC 2015 - 11th International Wireless Communications and Mobile Computing Conference* (2015), pp. 385–390. DOI: 10.1109/IWCMC.2015.7289114.

[137] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. « Localizing Faults in Cloud Systems ». In: *Software Testing, Verification and Validation (ICST)*. 2018, pp. 262–273. ISBN: 978-1-5386-5012-7. DOI: 10.1109/ICST.2018.00034.

[138] Alexander G. Tartakovsky, Boris L. Rozovskii, Rudolf B. Blažek, and Hongjoong Kim. « Detection of intrusions in information systems by sequential change-point methods ». In: *Statistical Methodology* 3.3 (2006), pp. 252–293. DOI: 10.1016/j.stamet.2005.05.003.

[139] C. W. J. Granger. « Investigating Causal Relations by Econometric Models and Cross-spectral Methods ». In: *Econometrica* 37.3 (1969), p. 424. DOI: 10.2307/1912791.

[140] Zachary S. Bischof, John S. Otto, Mario A. Sánchez, John P. Rula, David R. Choffnes, and Fabián E. Bustamante. « Crowdsourcing ISP characterization to the network edge ». In: *Proceedings of the first ACM SIGCOMM workshop on Measurements up the stack - W-MUST '11* (2011), p. 61. DOI: 10.1145/2018602.2018617.

[141] Rodérick Fanou, Bradley Huffaker, Ricky Mok, and K. C. Claffy. « Unintended Consequences: Effects of Submarine Cable Deployment on Internet Routing ». In: *PAM*. 2020. ISBN: 978-3-030-44080-0. DOI: 10.1007/978-3-030-44081-7_13.

[142] Ethem Alpaydin. *Introduction to Machine Learning*. 4th. MIT, 2020. ISBN: 978-0-262-04379-3.

[143] David J. Hand and Keming Yu. « Idiot's Bayes - Not so stupid after all? » In: *International Statistical Review* 69.3 (2001), pp. 385–398. DOI: 10.1111/j.1751-5823.2001.tb00465.x.

[144] Murray Rosenblatt. « Remarks on Some Nonparametric Estimates of a Density Function ». In: *The Annals of Mathematical Statistics* 27 (1956), pp. 832–837. DOI: 10.1214/aoms/1177728190.

[145] George H. John and Pat Langley. « Estimating Continuous Distributions in Bayesian Classifiers George ». In: *Conference on Uncertainty in artificial intelligence*. 1995, pp. 338–345.

[146] Ryan Prescott Adams and David J. C. MacKay. « Bayesian Online Changepoint Detection ». In: *CoRR* (2007). URL: https://arxiv.org/abs/0710.3742.

[147] U C Santa Barbara, Udit Paul, Alex Ermakov, Michael Nekrasov, Vivek Adarsh, and Elizabeth Belding. « #Outage : Detecting Power and Communication Outages from Social Networks ». In: *WWW*. 2020. ISBN: 978-1-4503-7023-3.

[148] Georgios Patounas, Andres Gonzalez, and Ahmed Elmokashfi. « On the utility of coarse-grained mobile network telemetry in detecting performance degradation ». In: *TMA*. 2020. ISBN: 978-3-903176-27-0.

[149] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. « Top 10 algorithms in data mining ». In: *Knowledge and information systems* 14.1 (2008), pp. 1–37.

[150] J Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1992. ISBN: 1-55860-238-0.

[151] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification And Regression Trees*. Chapman and Hall/CRC, 1984. ISBN: 0-412-04841-8.

[152] Mike Chen, A.X. Zheng, Jim Lloyd, M.I. Jordan, and Eric Brewer. « Failure diagnosis using decision trees ». In: *International Conference on Autonomic Computing, 2004. Proceedings.* (2004), pp. 36–43. DOI: 10.1109/ICAC.2004.1301345.

[153] Sishuai Gong, Usama Naseer, and Theophilus A. Beson. « Inspector Gadget : A Framework for Inferring TCP Congestion Control Algorithms and Protocol Configurations. » In: *TMA*. 2020, pp. 1–9. ISBN: 978-3-903176-27-0.

[154] Leo Breiman. « Bagging predictors ». In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/bf00058655.

[155] Robert E. Schapire. « The Strength of Weak Learnability ». In: *Machine Learning* 5.2 (1990), pp. 197–227. DOI: 10.1023/A:1022648800760.

[156] Jerome H. Friedman. « Greedy function approximation: A gradient boosting machine ». In: *Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

[157] David H. Wolpert. « Stacked generalization ». In: *Neural Networks* 5.2 (1992), pp. 241–259. DOI: 10.1016/S0893-6080(05)80023-1.

[158] Casper Solheim Bojer and Jens Peder Meldgaard. « Learnings from Kaggle's Forecasting Competitions ». In: *CoRR* (2020). URL: https://arxiv.org/abs/2009.07701.

[159] Hendrik Jacob van Veen, Le Nguyen The Dat, and Armando Segnini. *Kaggle Ensembling Guide*. 2015. URL: https://mlwave.com/kaggle-ensembling-guide/ (visited on 02/08/2021).

[160] Batiste Le Bars and Argyris Kalogeratos. « A Probabilistic Framework to Node-level Anomaly Detection in Communication Networks ». In: *INFOCOM*. 2019.

[161] Yao Wang, Zhaowei Wang, Zejun Xie, Nengwen Zhao, Junjie Chen, Wenchi Zhang, Kaixin Sui, and Dan Pei. « Practical and White-Box Anomaly Detection through Unsupervised and Active Learning ». In: *ICCCN*. 2020, pp. 1–9. ISBN: 978-1-72816-607-0. DOI: 10.1109/icccn49398.2020.9209704.

[162] Jiyao Hu, Zhenyu Zhou, Xiaowei Yang, Jacob Malone, Jonathan W Williams, Chapel Hill, Implementation Nsdi, and Jacob Malone. « CableMon : Improving the Reliability of Cable Broadband Networks via Proactive Network Maintenance ». In: *NSDI*. 2020. ISBN: 978-1-939133-13-7.

[163] F. Rosenblatt. « The perceptron: A probabilistic model for information storage and organization in the brain ». In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[164] Paul John. Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. John Wiley & Sons, 1994. ISBN: 978-0-471-59897-8.

[165] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. « Gradient-Based Learning Applied to Document Recognition ». In: *Proceedings of the IEEE* 11.86 (1998), pp. 2278–2324.

[166] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. « Deep residual learning for image recognition ». In: *CVPR*. 2016, pp. 770–778. ISBN: 978-1-4673-8850-4. DOI: 10.1109/CVPR.2016.90.

[167] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. « Time-series anomaly detection service at Microsoft ». In: *SIGKDD*. 2019, pp. 3009–3017. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330680.

[168] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. « Learning representations by back-propagating errors ». In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[169] Sepp Hochreiter and Jürgen Schmidhuber. « Long Short-term Memory ». In: *Neural Computation* 9.8 (1997), p. 17351780.

[170] Xingjian Shi, Zhourong Chen, Hao Wang, Dit Yan Yeung, Wai Kin Wong, and Wang Chun Woo. « Convolutional LSTM network: A machine learning approach for precipitation nowcasting ». In: *NIPS*. 2015, pp. 802–810.

[171] William L. Hamilton. « Graph Representation Learning ». In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (2020), pp. 1–159. DOI: 10.2200/S01045ED1V01Y202009AIM046.

[172] Fabien Geyer and Georg Carle. « Learning and generating distributed routing protocols using graph-based deep learning ». In: *Big-DAMA 2018*. 2018, pp. 40–45. ISBN: 978-1-4503-5904-7. DOI: 10.1145/3229607.3229610.

[173] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. « AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN ». In: *IJCAI*. 2019, pp. 4419–4425. DOI: 10.24963/ijcai.2019/614.

[174] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. « Neural message passing for quantum chemistry ». In: *CoRR* (2017).

[175] Barcelona Neural Networking Center. *Graph Neural Networking Challenge 2020*. 2020. URL: https://bnn.upc.edu/challenge2020/ (visited on 02/15/2021).

[176] Kan Guo, Yongli Hu, Zhen Qian, Hao Liu, Ke Zhang, Yanfeng Sun, Junbin Gao, and Baocai Yin. « Optimized Graph Convolution Recurrent Neural Network for Traffic Prediction ». In: *IEEE Transactions on Intelligent Transportation Systems* 22.2 (2020), pp. 1138–1149. DOI: 10.1109/tits.2019.2963722.

[177] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. « PyTorch: An Imperative Style, High-Performance Deep Learning Library ». In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[178] PyTorch geometric. *PyTorch geometric - MessagePassing*. 2020. URL: https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html (visited on 12/04/2020).

[179] Diederik P. Kingma and Jimmy Lei Ba. « Adam: A method for stochastic optimization ». In: *ICLR*. 2015.

[180] Leslie N. Smith. « Cyclical learning rates for training neural networks ». In: *WACV*. April. 2017, pp. 464–472. ISBN: 978-1-5090-4822-9. DOI: 10.1109/WACV.2017.58.

[181] Sewell Wright. « Correlation and causation ». In: *Journal of Agricultural Research* 20 (1921), pp. 557–585.

[182] James F. Kurose and Keith W. Ross. *Computer Networking - A top-down approach*. 6th ed. Pearson, 2012. ISBN: 978-0-13-285620-1. DOI: 10.5555/2584507.

[183] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. « Human-level control through deep reinforcement learning ». In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[184] Ranjan Anantharaman, Yingbo Ma, Shashi Gowda, Chris Laughman, Viral Shah, Alan Edelman, and Chris Rackauckas. « Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks ». In: *CoRR* (2020). URL: https://arxiv.org/abs/2010.04004.

[185] Miquel Ferriol, José Suárez-Varela, Pere Barlet-Ros, and Albert Cabellos-Aparicio. « Applying Graph-based Deep Learning To Realistic Network Scenarios ». In: *CoRR* (2020). URL: https://arxiv.org/abs/2010.06686.

[186] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. « Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition ». In: *TPAMI* 37.9 (2015), pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.

[187] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. « Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps ». In: *CoRR* (2014). DOI: 10.1080/00994480.2000.10748487. URL: https://arxiv.org/abs/1312.6034.

[188] Praveen Tammana, Chandra Nagarajan, Pavan Mamillapalli, Ramana Kompella, and Myungjin Lee. « Fault localization in large-scale network policy deployment ». In: *ICDCS*. 2018. ISBN: 978-1-5386-6871-9. DOI: 10.1109/ICDCS.2018.00016.

[189] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. « Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-fold Metrics ». In: *PAM*. 2018, pp. 1–13. ISBN: 978-3-319-76480-1. DOI: 10.1007/978-3-319-76481-8_3.

[190] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. « "Why Should I Trust You?": Explaining the Predictions of Any Classifier ». In: *SIGKDD*. 2016. ISBN: 978-1-4503-2138-9. DOI: 10.18653/v1/N16-3020.

[191] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. « Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization ». In: *ICCV*. 2017, pp. 618–626. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.74.

[192] Rasmus Eskola and Jukka K. Nurminen. « Performance Evaluation of WebRTC Data Channels ». In: *Symposium on Computers and Communication (ISCC)*. 2015, pp. 676–680. DOI: 10.1109/ISCC.2015.7884873.

[193] David Madigan, Adrian E. Raftery, Chris T. Volinsky, and Jennifer Hoeting. « Bayesian Model Averaging ». In: *AAAI*. 1996. DOI: 10.1007/978-3-030-31150-6_12.

[194] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[195] Srikanth Sundaresan, Yan Grunenberger, Nick Feamster, Dina Papagiannaki, Dave Levin, and Renata Teixeira. « WTF? Locating Performance Problems in Home Networks ». In: *CoRR* (2013). URL: http://hdl.handle.net/1853/46991.

[196] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. « Measuring the Performance of User Traffic in Home Wireless Networks ». In: *PAM*. 2015.

[197] Measurement Lab. *NDT (Network Diagnostic Tool)*. 2020.

[198] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. « Characterizing Residential Broadband Networks ». In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07* (2007), pp. 1–14. DOI: 10.1145/1298306.1298313.

[199] Romain Fontugne, Anant Shah, and Kenjiro Cho. « Persistent Last-mile Congestion : Not so Uncommon ». In: *IMC*. 2020. ISBN: 978-1-4503-8138-3.

[200] Kenneth L Calvert, W Keith Edwards, Nick Feamster, Rebecca E Grinter, Ye Deng, and Xuzi Zhou. « Instrumenting Home Networks ». In: *ACM SIGCOMM Computer Communication Review* (2011). DOI: 10.1145/1851307.1851321.

[201] Vaibhav Bajpai and Jurgen Schonwalder. « A survey on internet performance measurement platforms and related standardization efforts ». In: *IEEE Communications Surveys and Tutorials* 17.3 (2015), pp. 1313–1341. DOI: 10.1109/COMST.2015.2418435.

[202] Srikanth Sundaresan, Renata Teixeira, Georgia Tech, Nick Feamster, Antonio Pescapè, and Sam Crawford. « Broadband Internet Performance : A View From the Gateway ». In: *SIGCOMM*. 2011. ISBN: 978-1-4503-0797-0.

[203] Srikanth Sundaresan, Sam Burnett, Nick Feamster, and Walter De Donato. « Bismark: A testbed for deploying measurements and applications in broadband access networks ». In: *ATC*. 2014. ISBN: 978-1-931971-10-2.

[204] RIPE NCC Staff. « Ripe atlas: A global internet measurement network ». In: *Internet Protocol Journal* 18.3 (2015).

[205] Raimund Schatz, Tobias Hoßfeld, and Pedro Casas. « Passive YouTube QoE monitoring for ISPs ». In: *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012* (2012), pp. 358–364. DOI: 10.1109/IMIS.2012.12.

[206] Sulema Yevgeniya, Amram Noam, Oleksii Aleshchenko, and Olena Sivak. « Quality of Experience Estimation for WebRTC-based Video Streaming ». In: *European Wireless* (2018), pp. 94–99.

[207] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. « Measuring and mitigating web performance bottlenecks in broadband access networks ». In: *IMC*. 2013. ISBN: 978-1-4503-1953-9.

[208] Lucas Dicioccio, Renata Teixeira, Catherine Rosenberg, Lucas Dicioccio, Renata Teixeira, Catherine Rosenberg, and Measuring Home. « Measuring Home Networks with HomeNet Profiler ». In: (2013).

[209] Walter De Donato, Alessio Botta, and Antonio Pescapé. « HoBBIT: A platform for monitoring broadband performance from the user network ». In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8406 LNCS (2014), pp. 65–77. DOI: 10.1007/978-3-642-54999-1_6.

[210] Diana Zeaiter Joumblatt, Renata Teixeira, Jaideep Chandrashekar, Nina Taft, Diana Zeaiter Joumblatt, Renata Teixeira, Jaideep Chandrashekar, Nina Taft, Hostview Annotating, Diana Joumblatt, Renata Teixeira, Jaideep Chandrashekar, and Nina Taft. « HostView : Annotating end-host performance measurements with user feedback ». In: *ACM SIGMETRICS Performance Evaluation Review* 38.3 (2011).

[211] Yanyan Zhuang, Eleni Gessiou, Steven Portzer, Fraida Fund, Monzur Muhammad, Ivan Beschastnikh, and Justin Cappos. « NetCheck: Network Diagnoses from Blackbox Traces ». In: *Nsdi* (2014), pp. 115–128.

[212] Waiting WT Fok, Xiapu Luo, Ricky Mok, Weichao Li, Yujing Liu, Edmond WW Chan, and Rocky KC Chang. « MonoScope: Automating network faults diagnosis based on active measurements ». In: *IM*. 2013.

[213] Pavle Vuletić, Bartosz Bosak, Marinos Dimolianis, Pascal Mérindol, David Schmitz, and Henrik Wessing. « Localization of network service performance degradation in multi-tenant networks ». In: *Computer Networks* 168 (2020), p. 107050. DOI: 10.1016/j.comnet.2019.107050.

[214] Fangfan Li, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. « A large-scale analysis of deployed traffic differentiation practices ». In: *SIGCOMM*. 2019, pp. 130–144. ISBN: 978-1-4503-5956-6. DOI: 10.1145/3341302.3342092.

[215] Othmane Belmoukadam, Thierry Spetebroot, and Chadi Barakat. « ACQUA: A user friendly platform for lightweight network monitoring and QoE forecasting ». In: *Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2019, pp. 88–93. ISBN: 978-1-5386-8336-1. DOI: 10.1109/ICIN.2019.8685878.

[216] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. « Demystifying page load performance with WProf ». In: *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (2013), pp. 473–485.

[217] W3C. *Browser Extensions (Draft)*. 2020. URL: https://browserext.github.io/browserext/ (visited on 05/28/2020).

[218] Simone Basso, Michela Meo, Antonio Servetti, and Juan Carlos De Martin. « Estimating packet loss rate in the access through application-level measurements ». In: *Proceedings of the 2012 ACM SIGCOMM workshop on Measurements up the stack - W-MUST '12* (2012), p. 7. DOI: 10.1145/2342541.2342545.

[219] Simone Basso, Michela Meo, and Juan Carlos De Martin. « Strengthening Measurements from the Edges : Application-Level Packet Loss Rate Estimation ». In: *ACM SIGCOMM Computer Communication Review* 43.3 (2013), pp. 45–51. DOI: 10.1145/2500098.2500104.

[220] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. « Avoiding traceroute anomalies with Paris traceroute ». In: *IMC*. 2006. ISBN: 1-59593-561-4.

[221] W3C. *Resource Timing Level 2*. 2020. URL: https://www.w3.org/TR/resource-timing-2/ (visited on 05/28/2020).

[222] IETF. *The WebSocket Protocol*. 2011. URL: https://tools.ietf.org/html/rfc6455 (visited on 05/28/2020).

[223] Weichao Li, Ricky K.P. Mok, Rocky K.C. Chang, and Waiting W.T. Fok. « Appraising the delay accuracy in browser-based network measurement ». In: *IMC*. 2013. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504760.

[224] W3Techs. *Usage statistics of reverse proxy services for websites*. 2020. URL: https://w3techs.com/technologies/overview/proxy (visited on 05/28/2020).

[225] Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, and Martin Lopatka. « Don't count me out: On the relevance of IP addresses in the tracking ecosystem ». In: *The Web Conference*. 2020. ISBN: 978-1-4503-7023-3.

[226] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. « When to use and when not to use BBR: An empirical analysis and evaluation study ». In: *IMC*. 2019. ISBN: 978-1-4503-6948-0. DOI: 10.1145/3355369.3355579.

[227] RIPE NCC. *BGPlay*. 2020. URL: https://stat.ripe.net/widget/bgplay (visited on 05/28/2020).

[228] Catalin Cimpanu. *AWS outage impacts thousands of online services*. 2020. URL: https://www.zdnet.com/article/aws-outage-impacts-thousands-of-online-services/ (visited on 03/19/2021).

[229] OVH. *OVH Visual Monitoring System - SBG-2 (archived)*. 2021. URL: https://web.archive.org/web/20210310214153/http://status.ovh.com/vms/index_sbg2.html (visited on 03/10/2021).

[230] CAIDA. *IODA - Internet Outage Detection and Analysis*. 2021. URL: https://ioda.caida.org/ (visited on 03/19/2021).

[231] E.B. Wilson. « Probable inference, the law of succession, and statistical inference ». In: *Journal of the American Statistical Association* 22.158 (1927), pp. 209–212.

[232] MaxMind. *GeoIP Products*. 2021. URL: https://dev.maxmind.com/geoip/ (visited on 04/08/2021).

[233] Stéphane Bortzmeyer. *Cyberstructure - Internet, un espace politque*. Ed. by C & F Editions. 2018. ISBN: 978-2-915825-87-9.

[234] Franceinfo avec AFP. *Covid-19 : l'enseignement à distance perturbé par des bugs, Jean-Michel Blanquer évoque "des attaques informatiques"*. 2021. URL: https://www.francetvinfo.fr/sante/maladie/coronavirus/covid-19-l-ecole-a-la-maison-reprend-les-bugs-informatiques-aussi_4361445.html (visited on 04/06/2021).

[235] Michel Paulin. *Tweet about OVHcloud relation with CNED services*. 2021. URL: https://twitter.com/michel_paulin/status/1379426031463895042 (visited on 04/06/2021).

[236] Yves Vanaubel, Jean-Romain Luttringer, Pascal Merindol, Jean-Jacques Pansiot, and Benoit Donnet. « TNT, Watch me Explode: A Light in the Dark for Revealing MPLS Tunnels ». In: *TMA*. 2019. ISBN: 978-3-903176-17-1. DOI: 10.23919/tma.2019.8784525.

[237] Ondrej Krajsa and Lucie Fojtova. « RTT measurement and its dependence on the real geographical distance ». In: *TSP*. IEEE, 2011, pp. 231–234. ISBN: 978-1-4577-1411-5. DOI: 10.1109/TSP.2011.6043737.

[238] Raul Landa, Richard G. Clegg, João Taveira Araújo, Eleni Mykoniati, David Griffin, and Miguel Rio. « Measuring the relationships between internet geography and RTT ». In: *ICCCN*. 2013. ISBN: 978-1-4673-5774-6. DOI: 10.1109/ICCCN.2013.6614151.

[239] Yves Alexandre De Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. « Unique in the Crowd: The privacy bounds of human mobility ». In: *Scientific Reports* 3 (2013), pp. 1–5. DOI: 10.1038/srep01376.

[240] Brendan H. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. « Communication-efficient learning of deep networks from decentralized data ». In: *AISTATS*. 2017.

[241] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. « On the Convergence of FedAvg on Non-IID Data ». In: *CoRR* (2019). URL: https://arxiv.org/abs/1907.02189.

[242] Olivier Bachem, Mario Lucic, and Andreas Krause. « Practical Coreset Constructions for Machine Learning ». In: *CoRR* (2017). URL: https://arxiv.org/abs/1703.06476.

Design information

This thesis has been typeset using LaTeX and Neovim. It uses the Linux Libertine and Libertine Mono fonts at 12 pt. The figures and plots have been designed with `pgf/tikz` and `pgfplot`. The `biblatex` package was used to generate the bibliography. The full source code is available at `https://lesterpig.com/phd`

UNIVERSITÉ DE RENNES 1

Titre : Techniques d'Apprentissage Statistique pour la Modélisation et pour l'Analyse de Causes Racines de Réseaux Informatiques

Mot clés : Internet, Qualité d'Experience, Réseaux de neurones, Production participative

Résumé : Avec la demande mondiale croissante d'Internet, les opérateurs de réseaux et les fournisseurs de services doivent gérer des systèmes de plus en plus complexes et interdépendants. Dans cette thèse, nous explorons comment les techniques d'apprentissage statistique peuvent être utilisées pour aider à la modélisation et à la compréhension des grands réseaux informatiques. Dans une première contribution, nous proposons et évaluons un algorithme de réseau neuronal en graphe pour la prédiction des performances à partir de caractéristiques connues du réseau. Notre deuxième contribution porte sur l'analyse des causes racines à l'échelle de l'Internet : en tenant compte des connaissances limitées sur le réseau, nous évaluons trois techniques d'apprentissage statistique pour ce problème important, les classifieurs naïf Bayésien, forêt aléatoire et réseau neuronal convolutif. Nous montrons les résultats de ces techniques sur un ensemble de données de mesures Internet couvrant une année entière, et collectées avec un ensemble de méthodes basées sur les navigateurs web.

Title: Computer Network Modeling and Root Cause Analysis with Statistical Learning

Keywords: Internet, Quality of Experience, Neural Networks, Crowdsourcing

Abstract: With the global rising Internet demand, network operators and service providers need to manage increasingly complex and interdependent systems. In this thesis, we explore how statistical learning techniques can be used to help modeling and understanding large computer networks. In a first contribution, we propose and evaluate a Graph Neural Network algorithm for path performance prediction given known network characteristics.

Our second contribution focuses on Internet-scale Root Cause Analysis: given limited knowledge about the network, we evaluate three statistical learning techniques for this important problem, including Naive Bayes, Random Forest and Convolutional Neural Network classifiers. We show the results of these techniques over a year-long dataset of Internet measurements, collected with a set of methods based on web browsers.