KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Analysis of Flow Prolongation Using Graph Neural Network in FIFO Multiplexing System

WEIRAN WANG

Stockholm, Sweden, 2023

Analysis of Flow Prolongation Using Graph Neural Network in FIFO Multiplexing System

WEIRAN WANG

Master's Programme, ICT Innovation, 120 credits
Date: March 14, 2023

Supervisors: Seyed Mohammadhossein Tabatabaee, Prof. Jean-Yves Le Boudec, Computer Communications and Applications Laboratory 2, EPFL
Examiner: Prof. Viktoria Fodor
School of Electrical Engineering and Computer Science
Host organization: École polytechnique fédérale de Lausanne (EPFL)
Swedish title: Analys av Flödesförlängning Med Hjälp av Graph Neural Network i FIFO-Multiplexering System

© 2023 Weiran Wang

Abstract

Network Calculus views a network system as a queuing framework and provides a series of mathematical functions for finding an upper bound of an end-to-end delay. It is crucial for the design of networks and applications with a hard delay guarantee, such as the emerging Time Sensitive Network. Even though several approaches in Network Calculus can be used directly to find bounds on the worst-case delay, these bounds are usually not tight, and making them tight is a hard problem due to the extremely intensive computing requirements. This problem has also been proven as NP-Hard. One newly introduced solution to tighten the delay bound is the so-called Flow Prolongation. It extends the paths of cross flows to new sink servers, which naturally increases the worst-case delay, but might at the same time decrease the delay bound. The most straightforward and the most rigorous solution to find the optimal Flow Prolongation combinations is by doing exhaustive searches. However, this approach is not scalable with the network size. Thus, a machine learning model, Graph Neural Network (GNN), has been introduced for the prediction of the optimal Flow Prolongation combinations, mitigating the scalability issue. However, early research also found out that machine learning models consistently misclassify adversarial examples.

In this thesis, Fast Gradient Sign Method (FGSM) is used to benchmark how adversarial attacks will influence the delay bound achieved by the Flow Prolongation method. It is performed by slightly modifying the input network features based on their gradients. To achieve this, we first learned the usage of NetCal DNC, an Free and Open Source Software, to calculate the Pay Multiplexing Only Once (PMOO), one of the Network Calculus methods for the delay bound calculation. Then we reproduced the GNN model based on PMOO, and achieved an accuracy of 65%. Finally, the FGSM is implemented on a newly created dataset with a large number of servers and flows inside. Our results demonstrate that with at most 14% changes on the network features input, the accuracy of GNN drastically decreases to an average 9.45%, and some prominent examples are found whose delay bounds are largely loosened by the GNN Flow Prolongation prediction after the FGSM attack.

Keywords

Network Calculus, Flow Prolongation, Graph Neural Network, Fast Gradient Sign Method, Delay Bound

Sammanfattning

Nätverkskalkylen behandlar ett nätverkssystem som ett system av köer och tillhandahåller ett antal matematiska funktioner som används för att hitta en övre gräns för end-to-end förseningar. Det är mycket viktigt för designen av nätverk och applikationer med strikta begränsningar för förseningar, så som det framväxande Time Sensitive Network. Även om ett flertal tillvägagångssätt i nätverkskalkylen kan användas direkt för att finna gränsen för förseningar i det värsta fallet så är dessa vanligtvis inte snäva. Att göra gränserna snäva är svårt då det är ett NP-svårt problem som kräver extremt mycket beräkningar. En lösning för att strama åt förseningsgränserna som nyligen introducerats kallas Flow Prolongation. Den utökar vägarna av korsflöden till nya sink servrar, vilket naturligt ökar förseningen i värsta fallet, men kan eventuellt också sänka förseningsgränsen. Den enklaste och mest rigorösa lösningen för att hitta de optimala Flow Prolongation kombinationerna är att göra uttömmande sökningar. Detta tillvägagångssätt är dock inte skalbart för stora nätverk. Därför har en maskininlärningsmodell, ett Graph Neural Network (GNN), introducerats för att förutspå de optimala Flow Prolongation kombinationerna och samtidigt mildra problemen med skalbarhet. Dock så visar de tidiga fynden att maskininlärningsmodeller ofta felaktigt klassificerar motstridiga exempel.

I detta projekt används Fast Gradient Sign Method (FGSM) för att undersöka hur motståndarattacker kan påverka förseningsgränsen som hittas med hjälp av Flow Prolongation metoden. Detta görs genom att modifiera indata-nätverksfunktionerna en aning baserat på dess gradienter. För att uppnå detta lärde vi oss först att använda NetCal DNC, en mjukvara som är gratis och Open Source, för att kunna beräkna Pay Multiplexinng Only Once (PMOO), en metod inom nätverkskalkylen för att beräkna förseningsgränser. Sedan reproducerade GNN modellen baserat på PMOO, och uppnådde en träffsäkerhet på 65%. Slutligen implementerades FGSM på ett nytt dataset med ett stort antal servrar och flöden. Våra resultat visar att förändringar på upp till 14% på indata-nätverksfunktionerna resulterar i att träffsäkerheten hos GNN minskar drastiskt till ett genomsnitt på 9.45%. Vissa exempel identifierades där förseningsgränsen utvidgas kraftfullt i GNN Flow Prolongation förutsägelsen efter FGSM attacken.

Nyckelord

Nätverkskalkyl, Flödesförlängning, Graph Neural Network, Fast Gradient Sign Method, Fördröjningsgräns

Acknowledgments

This master thesis is dedicated to Peipei, my first puppy who has accompanied me for 16 years since the second year at my primary school, but passed away the day when I left Switzerland.

I would like to show my appreciation to Prof. Jean-Yves Le Boudec for this valuable academic research chance in his Computer Communications and Applications Laboratory at EPFL, and also for leading me to the world of Network Calculus. This experience enhances my understanding of the end-to-end delay, a topic which I am keen on. I would also like to thank Hossein Tabatabaee for having supervised me since September 2021 on the whole project. We spent a splendid year, seeking challenging outputs through this project. I feel extremely privileged to have him for so much experienced support.

Specifically for the project implementation, I want to say thank you to Prof. Bondorf Steffen for the of NetCal/DNC usage, Etienne Orliac for the support of the environment configuration on EPFL IZAR cluster, and Hadidane Karim for providing a great base of the implementation of GNN in this project.

In terms of KTH, I appreciate Prof. Viktoria Fodor for being my examiner and teaching me the Queuing Theory and Teletraffic Systems course during my first year of M.Sc. Without this course, I will not be able to understand the key concepts in Network Calculus. Moreover, I also appreciate Prof. Viktoria Fodor's efforts in reviewing my thesis. The process is time-consuming, but I strongly believe my writing skill improves a lot.

Together is my gratitude to KTH and EPFL international offices for organizing this exchange and funding me from September 2021 to August 2022. Studying and researching at EPFL is challenging due to the intensive pressure but fruitful at the same time. This year also provides me with a chance to explore this world. I will never forget my fellow friends in Switzerland for the lovely memory we shared together.

Last but not least, I cannot wait to show my affection to my parents for everything in my life. It is literally hard for me to say this in front of them, but what they've done for me will bury in my mind. Also my affection for the most important person, Kejun Lu for some top secret reasons and hopefully she could be my girlfriend in the coming future :-)

This project and thesis could truly not be finished without so much help and encouragement. Tack så mycket and Merci beaucoup!

Stockholm, March 2023
Weiran Wang

Contents

List of Figures

List of acronyms and abbreviations

EPFL	Swiss Federal Institute of Technology in Lausanne
FFNN	Feed Forward Neural Network
FGSM	Fast Gradient Sign Method
FIFO	First In First Out
foi	flow of interest
FOSS	Free and Open-Source Software
FP	Flow Prolongation
GAN	Generative Adversarial Networks
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
JSMA	Jacobian-based Saliency Map Attack
KTH	KTH Royal Institute of Technology
LUDB-FF	Least Upper Delay Bound Feed Forward
MIMO	Multiple-Input and Multiple-Output
NC	Network Calculus
NFV	Network Functions Virtualization
PMOO	Pay Multiplexing Only Once
QoE	Quality of Experience
QoS	Quality of Service
RTC	Real-Time Calculus
SDN	Software-Defined Networking
SFA	Separated Flow Analysis
TFA	Total Flow Analysis

TMA	Tandem Matching Analysis
TSN	Time-Sensitive Network

Chapter 1

Introduction

In the wake of higher demand for faster and more reliable communication networks over the past decades, many newly-developed systems ranging from daily products to complicated industrial machines are required to provide Quality of Service (QoS) and Quality of Experience (QoE) guarantees. Such examples can be daily life demands represented by reliable online meetings and less-loading-time streaming videos or industrial implementations represented by the design of Time-Sensitive Network (TSN) [1][2] for machine-to-machine communications. Among these guarantees, one important network property, the end-to-end delay, is crucial for the entire application or system. In this thesis, the end-to-end delay will be analyzed by Network Calculus (NC) [3], a mathematical framework which can calculate an upper bound for the end-to-end delay. General background knowledge, the potential problems to be solved, and the corresponding methods will be introduced in this chapter.

1.1 Background

The end-to-end delay is one of the most fundamental and important concepts in the telecommunication area, which has already attracted tremendous research attention in both academic and industrial communities. Given a network topology, it is the time spent by a network packet from its source to its destination. For instance, if the terminal command $ping\ www.kth.se$ is used at the Swiss Federal Institute of Technology in Lausanne (EPFL) campus, it will send network packets from EPFL to KTH Royal Institute of Technology (KTH) shown in Figure 1.1, and the average end-to-end delay, around 96.099 ms, will be achieved as the result of this command.

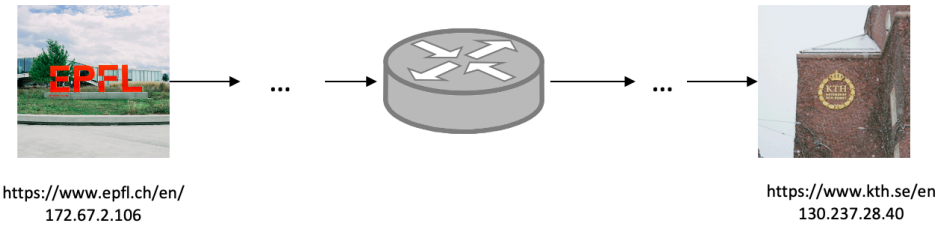


Figure 1.1: A Demonstrative End-to-End Delay from EPFL to KTH

Taking a deeper inspection of this $ping$ command, it is easy to observe that for this network request, there is an end-to-end delay range in a certain interval (mainly depends on when a termination command $ctrl + c$ is conducted), with a minimum value of 86.774 ms and a maximum value of 150.673 ms. This range of end-to-end delays is caused by network fluctuations between EPFL and KTH. In this range of delays, the maximum end-to-end delay, or the so-called worst-case traversal time, is defined as the tightest delay bound [3], matters in the design of network systems, especially for the safety-critical applications or the design of TSN.

To depict the network topology, if the command $traceroute$ is used, the path of this network packet can be visualized, and a series of network packets sending from EPFL to KTH will form the so-called network flow. However, there are definitely other flows traversing through all of or part of the servers in this network topology as can be abstracted into Figure 1.2. Assuming server s_2 and server s_{n+1} represent the EPFL and KTH servers respectively, then flow f_4, the flow in red, will therefore be the $ping/traceroute$ request. The flow waiting to be analyzed is defined as the flow of interest (foi), and is an important concept in this thesis.

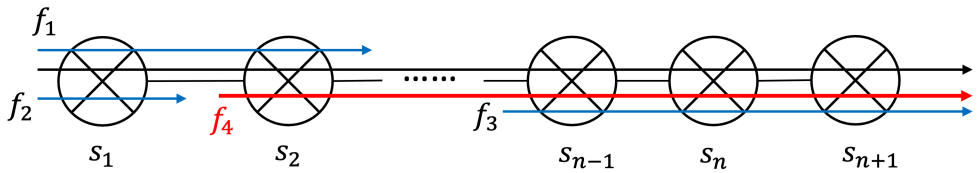


Figure 1.2: An Abstracted Topology Demonstrating the Flow of Interest

In a network setting, computing the exact end-to-end delay is hard, even

if in a First In First Out (FIFO) network. This problem has been defined as NP-hard [4]. In the example above, there is always a range for the end-to-end delay. If we repeat the $ping$ command endlessly, it's still hard to find the worst-case end-to-end delay, owing to a worse end-to-end delay occurring in the next network packet sending with a certain possibility. Therefore, it is difficult to explore an upper bound for the delay, let alone finding the tightest delay bound. The process to find the delay bound can be visualized in Figure 1.3.

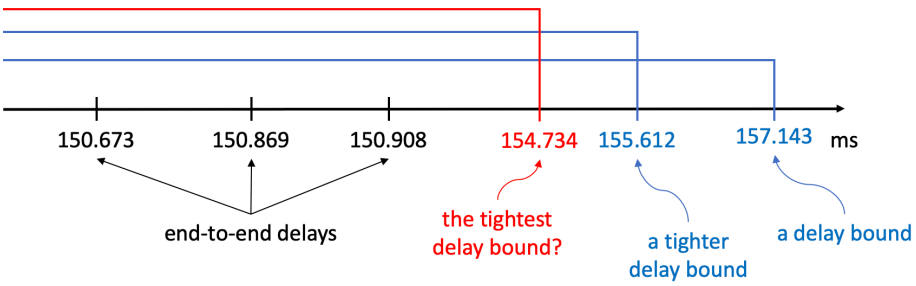


Figure 1.3: The Process to Find a Tighter Delay Bound

Fortunately, NC provides a mathematical tool to compute the delay bound. In terms of the specific calculation means, there are various methods in NC such as Total Flow Analysis (TFA) [5], Pay Multiplexing Only Once (PMOO) [6], Least Upper Delay Bound Feed Forward (LUDB-FF) [7] and etc. All the delay bounds by these calculation methods are valid, while they are distinguished by different invention years, different computational efforts, different usage cases and etc. Taking the TFA as an example, it is recognized as the oldest NC method to compute delay bounds in the FIFO Multiplexing System. Due to its older invention year, it is now known to be inferior to the analysis of the current network systems. However, it is still widely used today for its simplicity and fast execution time [7].

To show the difference regarding tightness, Figure 1.4 compares the delay bounds by four different NC methods on the source-sink tandem network, a type of network where each flow either starts at the first server or ends at the last server, and each flow path is unique. Namely, given a tandem with $\#n$ servers, there will be $\#(2n-1)$ flows, and the foi is the flow whose source is the first server and the sink server is the last server. This also proves the properties of TFA that the delay bounds achieved by this are not tight enough compared to others.

Meanwhile, a comparison experiment among various NC methods

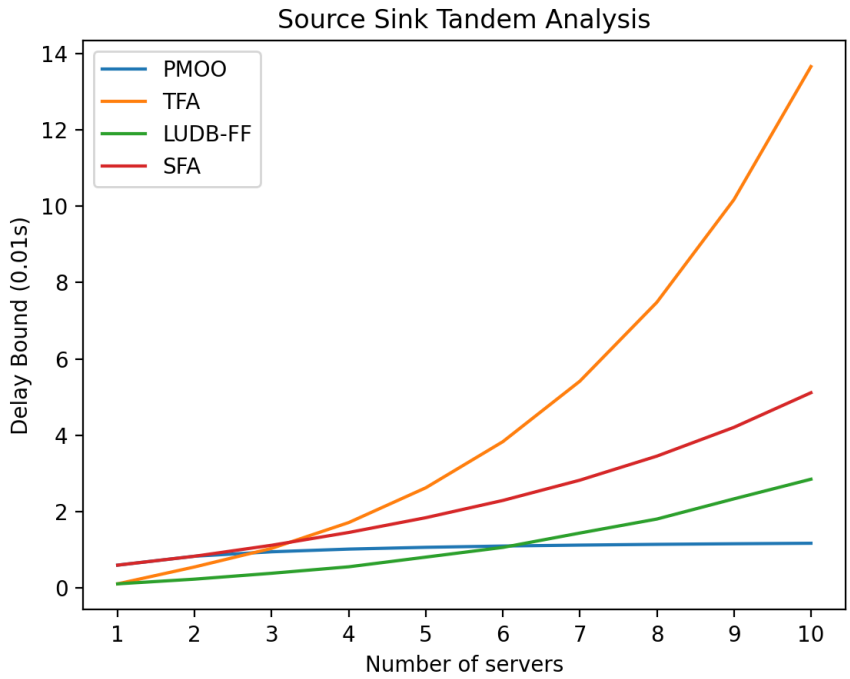
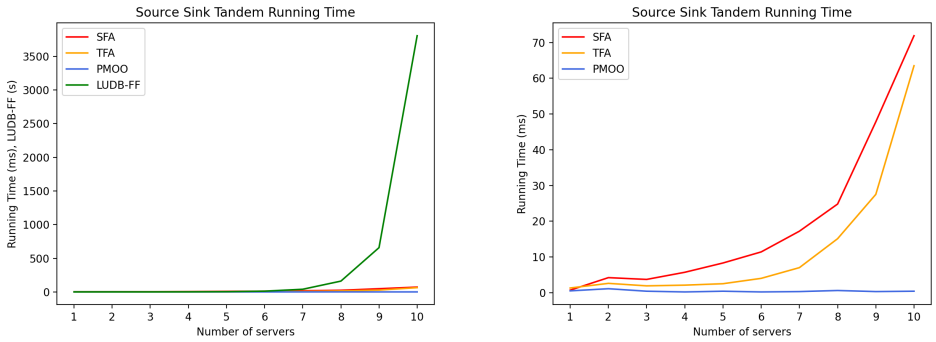


Figure 1.4: Delay Bounds Comparison

execution time is also conducted on an EPFL cluster (The specification will be given in the later chapter). To avoid outliers, each delay bound of a network topology is iterated 10 times to get the average calculation running time depicted in Figure 1.5a. Due to the drastic increase for LUDB-FF, it is hard to observe the growth trends for Separated Flow Analysis (SFA), TFA and PMOO. Thus, Figure 1.5b is shown as well without LUDB-FF for better visualization. Combining the result from Figure 1.4, even though delay bounds obtained by LUDB-FF are tighter, it's super time-consuming from its nearly exponential growth tendency due to its calculation property. Therefore, how to balance the trade-off between the tightness and the execution time is of great importance before the choice of a NC method [8].

Observing Figure 1.4 and Figure 1.5b, PMOO shows outstanding performances in both the delay bound tightness and the execution time. Therefore, it is the main NC method used in this project. LUDB-FF is also used in part of this project, which will be described in detail in Chapter 3.

Even though there are several approaches in NC that can calculate the delay bound, the results are, unfortunately, not usually tight recalling the observation



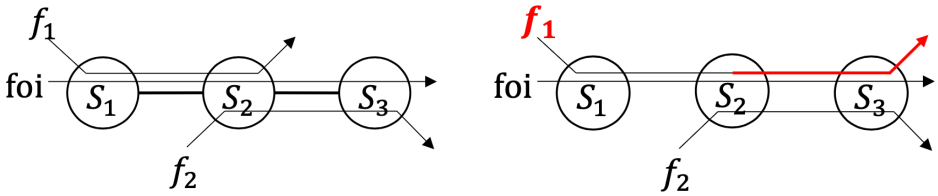
(a) Comparison with LUDB-FF (b) Comparison without LUDB-FF

Figure 1.5: Running Time Comparison

from Figure 1.4. Furthermore, making them tight is a hard problem due to the scalability, because a tighter delay bound usually means more computation-intensive efforts. There are currently two solutions for tightening the delay bound. One is to invent another NC method which is a long-term work for scientists, where fewer than ten methods have been invented recalling the history of NC. Another is to investigate the current NC methods further, where one of the possible directions is to explore whether the network setting will influence the final delay bounds.

Consistent with human perception, a more accurate network model will usually result in a tighter delay bound. However, recent research starting from 2017 [9] has shown quite the opposite, and this introduces a new feature called Flow Prolongation (FP), which has the potential to tighten the delay bound. Regarding the working mechanism, FP assumes that flows take more hops than they actually do, i.e., it prolongs the original sink server to a new one, and actively converts the shape of a network topology. Taking the example shown in Figure 1.6, f_1 is prolonged from its original sink server s_2 to a new sink server s_3.

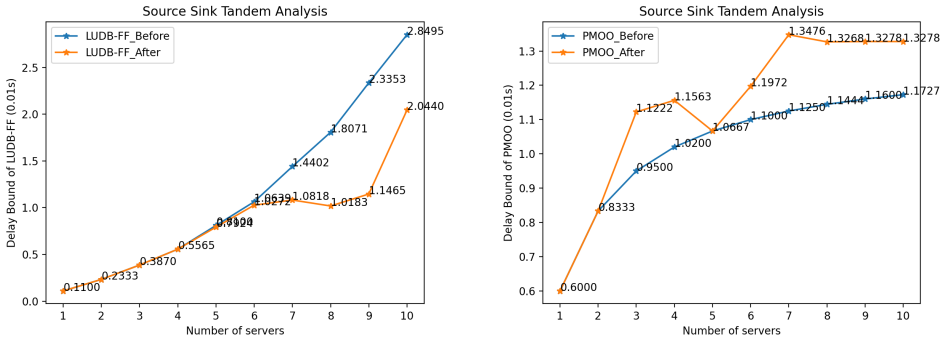
To explain why FP may instead of can tighten the delay bound, a demonstrative experiment is also conducted in the source-sink tandem network, the same network setting used for the comparison experiments of tightness and execution time mentioned above. The results based on LUDB-FF before and after the FP can be observed in Figure 1.7a. However, not all the FPs can tighten the delay bound as compared in Figure 1.7b based on PMOO. Currently, there are no fixed algorithms or definitions to determine which flows should be prolonged while others shouldn't. One straightforward way is the exhaustive search. By using the exhaustive search in FP, all the



(a) A Network Before FP (b) A Network After FP

Figure 1.6: FP Demonstration

possible delay bounds are able to be found, and the tightest delay bound is the one with the minimum value. As a brief summary on FP, it still remains mysterious for academia, even though for those with a deep understanding of NC methods properties, it is still hard to provide a complete explanation of how FP works theoretically. However, some current known reasons will be unboxed in Chapter 2 Section 2.1 Subsection 2.1.5.

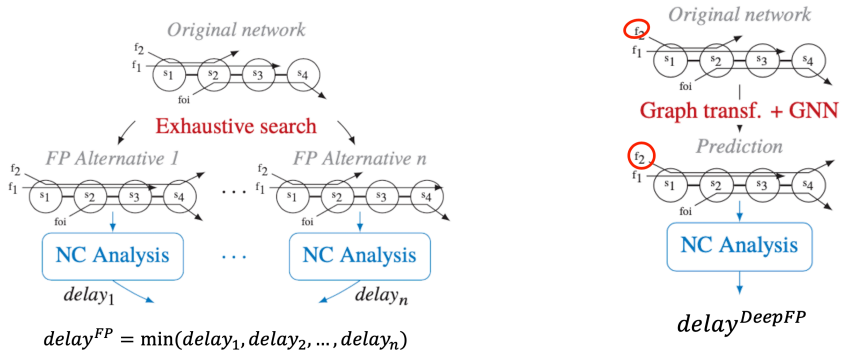


(a) LUDB-FF Delay Bound with FP (b) PMOO Delay Bound with FP

Figure 1.7: FP may Tighten the Delay Bound

Though with the recent five-year research on FP since it was first introduced in 2017, it is far from scalable to find the tightest delay bound via exhaustive search, the process of which can be visualized in Figure 1.8a [10]. Assuming there are n servers with m flows traversing through a tandem topology, there will be $O(n^m)$ magnitude of FP alternatives. For current professional scientists with a deep understanding of various NC methods analysis, it is still hard for them to reduce the number of FP alternatives in the exhaustive search by removing some unnecessary FP combinations. Thus, it is almost impossible to use the exhaustive search when it comes to large network sizes [9] due to the high computational resources and the long execution time.

Academia has been putting numerous efforts to find a feasible way to mitigate the process of finding a tighter delay bound with the help of FP, and Machine Learning has proven to be promising. It has already gained tremendous attention in the past decade, where one of the most popular applications is making predictions, i.e., a Machine Learning model is trained based on a given dataset and then this model will be used to predict brand new cases with similar characteristics of the training dataset. Machine Learning is a general concept where it contains a large number of specific models. In terms of choosing the concrete model, considering that network topologies can be summarized as part of the Graph Theory [11], Graph Neural Network (GNN), a Machine Learning model combining Graph Theory, has proven its potential to find the most suitable FP combinations to tighten the delay bound in a scalable manner depicted in Figure 1.8b [10].



(a) Exhaustive Search in FP

(b) GNN in FP

Figure 1.8: Comparison Between Exhaustive Search and GNN in FP [10]

Previous researches also provide theoretical support for this thought. Fabien Geyer, Alexander Scheffler and Steffen Bondorf have been publishing several papers using Machine Learning models to optimize the process of exploring a tighter delay bound almost every year. Early back in 2019, they derived a framework called DeepTMA combining a graph-based Deep Learning model and Tandem Matching Analysis (TMA), a NC method, together [12]. In 2020, they continued their research and extended TMA with a prediction mechanism by coupling graph-based Neural Network with NC. In 2021, they introduced a new model named after DeepFP, an approach to make FP prediction scalable by using a GNN model [10]. Their recent research in 2022 trains DeepFP again but using Reinforcement Learning instead of GNN, and the results outperform an expert heuristic for FP on small-sized networks

[13].

The process of finding a tighter delay bound in this degree project is mainly inspired by using GNN to mitigate the process of tightening the delay bound with FP [10]. Moreover, we step further to test the robustness of the GNN model on the FP in FIFO Multiplexing System, which will be discussed in the later chapter.

1.2 Problem

Even though GNN has shown its potential to mitigate the process to find a tighter bound via FP, it also raises another problem in how much extent can this model be trusted. This is critical for the design of TSN.

Regarding reliability, the accuracy of the GNN model itself can be tested by the testing dataset as a normal process in Machine Learning. However, recent research on Machine Learning is constantly pushing the models to be more efficient and more secure, especially for adversaries who are likely to fool the pre-trained models. Previous studies have also revealed that several Machine Learning models are found to consistently misclassify adversarial examples. In general, by adding some tiny perturbations into the inputs, the outputs will be mis-recognized by the Machine Learning models. Specifically in this project, one possible scenario could be that modifying the network features slightly may result in that one flow that shouldn't be prolonged in theory but is prolonged in reality by this pre-trained GNN model. This mis-prolongation may further cause a looser delay bound, which goes beyond our expectations. In another word, a delay bound is tightened by GNN before the adversarial attack but is loosened by GNN after the adversarial attack. Such adversaries could happen frequently. For example, after a network is analyzed, where the network settings are changed accidentally, or there is a sudden burst for some flows, or the server capacities are influenced by some external factors such as inadequate power supply or regular server aging. Considering the expression 'attack' is a heated topic both in Machine Learning and Computer Networks, it is worth clarifying that in this thesis, adversarial attacks refer to that Machine Learning models may output different results if the inputs are added by some tiny noises, which is nothing related to the attack in cyber security.

In a nutshell, based on the usage of GNN to tighten the delay bound under the FP, this project will answer the question of whether some tiny changes in the network features will influence the prediction of GNN and consequently lead to a looser delay bound. Moreover, one deeper question is to distinguish

whether this consequence is caused by the adversarial attack or by the accuracy of the pre-trained GNN model.

1.3 Purpose

Considering the significant interest in the end-to-end delay from both academic and industrial communities, it has already been analyzed in multiple network domains and will be continued in the long future due to the increasingly high demands for network evolution. Therefore both sides will benefit if tighter delay bounds can be found by a more scalable and robust means. Considering that both FP itself, and using the GNN to predict the suitable FP are new in research starting from 2017 and 2019 respectively, and this project is the first to test the model robustness by the adversarial attack within our humble knowledge, there are not enough previous research supporting what could happen if a looser delay bound is misused in the time-sensitive network system. Instead, this section will put more effort into demonstrating how NC matters in both academia and industries.

Regarding the benefits of academic research, one example could be the virtualized technology represented by Software-Defined Networking (SDN) and Network Functions Virtualization (NFV). They are considered as the future trend for cloud computing. Among them, function service chains offer a flexible approach to the end-to-end service provision, guaranteeing the QoS. NC makes the evaluation of the service chain end-to-end performance possible in the SDN/NFV architecture [14]. Furthermore, NC is one of the numerous techniques in the field of cybersecurity and is widely used for the calculation of delay bounds [15][16] to the potential cyberattacks prevention. In the physical layer, stochastic NC is applied to derive a closed-form upper bound for the delay target violation probability in the downlink of the Multiple-Input and Multiple-Output (MIMO) [17].

Academia and industry complement each other. Delay-constrained routing optimization framework has been implemented in the NFV-enabled enterprise datacenters to decrease the end-to-end delay and improve the system reliability [18] by using NC methods. It also enables the study of the worst-case transmission delay in the NFV-enabled 5G for ultra-reliable and low-latency communications in industry [19]. NC also provides a foundation for cyber-physical situational awareness to monitor the system faults or malicious activities [20]. In the field of MIMO, NC approaches can assist the formulation of the energy minimization problem so as to achieve high spectral and energy efficiency in future networks [21]. This is what lots of telecommunication

companies are paying attention to recently to promote sustainability. Apart from these, NC can be adopted as a mathematical theory for the performance analysis of the underwater wireless communication networks [22]; model the TSN shapers for industrial automation networks [23]; formulate a QoS constrained decision for electric vehicle charging stations [24]; assess the end-to-end properties of the network slices in Industry 4.0 applications [25] and etc.

Through joint efforts from both academia and industry, energy consumption will be reduced and therefore higher efficiency will be achieved in the networked system design and implementation. If a tighter delay bound could be found in a scalable and robust way, more flows are allowed to use the networked system. Namely, more requests can be satisfied. Simultaneously, safety-critical systems will be enhanced to guarantee the QoS. Considering the critical rule of using NC in various network domains, if a delay bound is loosened by the adversarial attack, the benefits mentioned above will vanish. It's worth mentioning here that this project is research-based and there will be a long way to the industrial applications.

1.4 Goals

The goal of this project is to benchmark whether some small perturbations added in the network features will influence GNN predictions on the FP combinations, and further influence the results of the delay bound. Fortunately, there are two off-the-shelf tools to assist in achieving this goal, NetCal/DNC [26] to calculate the delay bound, and a pre-trained GNN model based on DEBORAH to predict the best prolonged topologies. The following subgoals are achieved by this project:

1. The GNN code is modified and a new model based on PMOO is trained. As is explained in Section 1.1, PMOO outperforms considering both the execution time and the tightness.

2. NetCal/DNC is integrated into GNN so that the delay bound can be calculated for a given network topology when the GNN is used for the prediction of FP. The reason for choosing NetCal/DNC in this project is explained in Section 1.5.

3. Potential attack targets are found based on the GNN prediction results, which will be explained more specifically in Section 3.4.

4. Fast Gradient Sign Method (FGSM), one way of the adversarial attack, is realized based on this project background. Similar to NetCal/DNC, the reason for choosing this method will be explained in Section 1.5. The working mechanism of FGSM in theory will be demonstrated in Section 2.3, and the specific implementation process in this project will be explained in Section 3.5. The attack up to this step is done on the open source dataset[27] used to train the GNN model.

5. After getting the results based on subgoal 4, the results are far from satisfactory, and after analyzing the possible reasons, a decision is made to imitate the Computer Vision since the adversarial attack is usually implemented in the image recognition field. Thus, a larger dataset is created for the adversarial attack purpose where there are larger numbers of servers and flows in one single topology.

6. As the final step, the adversarial attack results are analyzed, i.e., tested whether GNN is fooled to predict the wrong flow prolongation, and thus lossen the delay bound. The numerical analysis is demonstrated in detail in Chapter 4.

1.5 Research Methodology

There are a few choices in the NC calculation tools. Such examples can be NC-Maude [28] written in Maude, CyNC [29] and DIMTOOL [30] developed in MATLAB, as well as WOPANets [31] and DelayLyzer [32] with their own designed GUI. Zhou, Howenstine *et al.* [33] did a survey for various existing NC tools for calculating the delay bounds used in network infrastructures of real-time systems. Among them, NetCal/DNC, also called NCorg DNC [7], a tool in Java is finally chosen. It is developed by Steffen Bondorf and Alexander Scheffler, two NC researchers mentioned in the previous section. Given a network topology consisting of servers and flows crossing these servers with the corresponding network features, NetCal/DNC enables to derive a flow's delay bound as well as a server's backlog bound. Therefore, there is no need to realize the NC mathematical formulas on my own. Instead, the relevant Java codes can be re-written so that the NetCal/DNC can be merged into this degree project. Furthermore, since the Machine Learning part is definitely implemented in Python. Py4j [34], a python package, provides a gateway to connect python and java so that the Java code can be called from my Python part. More specific details will be illustrated in Chapter 3.

In terms of adversarial attacks, there are many categories in this field. In this degree project, FGSM is chosen. It is invented by Goodfellow *et al.* [35], and is one of the first and the most popular attack methods. This attack adjusts the input data by going into the opposite way of the gradient, and thus maximizes the loss function. For unknown domains, FGSM is usually chosen for its classic and simplicity in the adversarial attack field. Furthermore, as its name reveals, it is very fast not only in realization but also in calculation time. More technical details of this attack and how they are combined with this project will be revealed in Sections 2.3 and 3.5.

1.6 Delimitations

As explained in subgoal 5 in Section 1.4, we want to imitate the Computer Vision when creating the new datasets. However, a figure can easily achieve thousands of pixels in a figure. Having such an amount of network features in a network topology is possible, but calculating the delay bound for such a network is almost impossible, even though the whole project is running on a very powerful EPFL server and PMOO outperforms the execution time. Therefore, the first delimitation is that there are still some gaps in terms of the number of features between the dataset created within our best efforts and the pixels. If a dataset from an industry can be obtained, the results will be much more ideal. In this case, the calculation of the delay bound in the dataset creation is not needed, and other NC methods e.g., DEBORAH or LUDB-FF, that perform better under the FP perhaps can be used.

Another delimitation is that only FGSM is implemented in this project due to the time limitation. There are some other methods in adversarial attacks such as Carlini & Wagner [36], Jacobian-based Saliency Map Attack (JSMA) [37], etc, which are worth playing with. Moreover, corresponding to adversarial attacks, the defense process is also interesting and is supposed to be investigated.

Finally, also due to the time limitation, the attack in this project is only done on the network features. Other ways of changing the network also attract our eyes such as modifying the flow paths or removing one of the servers. These scenarios could also happen in the real life. For instance, one server is down due to some reasons such as electricity overloading or regular maintenance. We hope to see whether the modification of the network topology will also affect the GNN predictions and the final delay bounds as well.

1.7 Structure of the thesis

This thesis is organized as followed: Chapter 2 presents relevant background information in theory about NC, GNN, and FGSM. NC is a relatively complicated mathematical framework, so only the background knowledge related to this project will be explained in brief. Chapter 3 presents the experiment settings and methods in practice, i.e., how to use the NetCal DNC tool to calculate delay bounds; how is GNN reproduced; how GNN is used for the prediction of FP; how FGSM is implemented for the adversarial attack to test the robustness of the model, and the creation of a new dataset. Chapter 4 evaluates the results of GNN model performance. Several examples will show how FGSM can fool the model for a looser delay bound. Finally, conclusions and future work will be presented in Chapter 5.

Chapter 2

Background

This chapter provides basic background knowledge about how NC, GNN and FGSM work in theory. How to combine the GNN into FP, and how the FGSM is conducted into the FP, thus influencing the NC delay bound results will be demonstrated in Chapter 3.

2.1 Network Calculus

Early back in the 1970s, NC has been investigated. "Network Calculus" [38] written by Wilton R.Abbott, and J.M. Grange's paper "Effects of the Coupling Between Components of the Reliability of an Integrated Function" [39] are the two earliest papers that can be found on Google Scholar. Then in 1987, Cruz did deep research on NC and applied it to a wide variety of models for network operations in his Ph.D. dissertation [40]. In 2001, Prof. Jean-Yves Le Boudec from EPFL, published a book "Network Calculus: A Theory of Deterministic Queuing Systems for the Internet" [3], providing a theoretical foundation for many subsequent researches in this area. For KTH's information, Prof. James Gross is conducting research on NC, and is examining a Ph.D. level course "FEO3330 Network Calculus" [41].

In general, NC offers a mathematical framework for the performance evaluation of Communication Networks based on interpreting a system as a queuing theory model. It provides a deep insight into flow problems encountered in networking and is a tool to calculate an upper bound for a flow end-to-end delay introduced in Chapter 1. Furthermore, it can also calculate an upper bound for the server buffer sizes, but it is not considered within this thesis. Apart from the basic NC knowledge, a variant of NC focusing on real-time systems is the so-called Real-Time Calculus (RTC) [42], which enables

Ethernet with real-time capabilities for time-sensitive transmission.

2.1.1 Network Calculus Overview

NC relies on flows, more specifically, the so-called foi defined in Chapter 1. A network system can be abstracted as Figure 2.1 with a given input flow $R(t)$ and a corresponding output flow $R^*(t)$ traversing the system, where a network system can be anything ranging from a single server to sophisticated industrial telecommunication architectures.

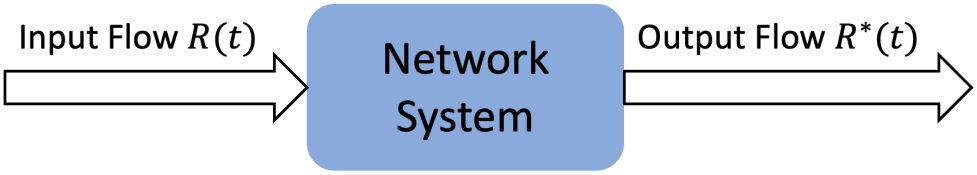
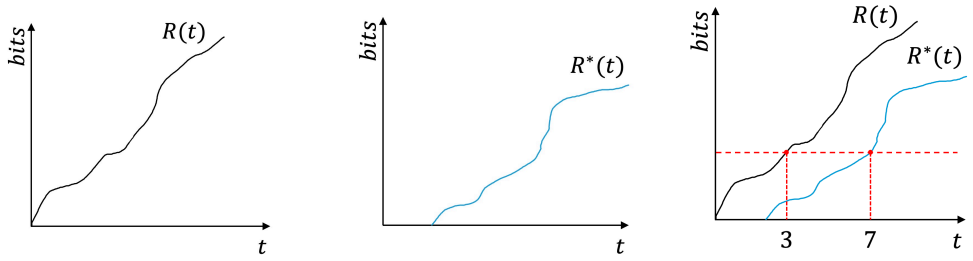


Figure 2.1: How NC Views a Network System

A non-decreasing cumulative function is used in NC, which models the number of bits entering the system since time $t = 0$ demonstrated in Figure 2.2 (These figures are drawn by me instead of observation data support. They are used only for the demonstrative explanation purpose of NC overview). Recalling the scenario of the $ping$ request from EPFL to KTH, the figure 2.2a is what the EPFL server observers and accordingly figure 2.2b is for KTH's perspective. Figure 2.2c demonstrates how the flow cumulative function depicts the end-to-end delay. Assuming that a bit enters the Network System at time $t = 3$ and leaves the system at time $t = 7$, the end-to-end delay will be 4. Obviously, this delay might not be the worst-case delay, nor the upper bound guarantee, but as can be observed from this cumulative function that the horizontal differences between the input flow and the output flow are the end-to-end delays. Consequently, it introduces the following three key concepts: **arrival curves**, **service curves**, and the **delay bound**. Briefly speaking, the service offered to flows is guaranteed by means of a service curve, while an arrival curve constraints the amount of data for the incoming flows into the network system. A delay bound can be obtained by combining the service curve and the arrival curve for the foi. More details will be discussed in Sections 2.1.2, 2.1.3, and 2.1.4.

Before the introduction of the following concepts, it is worth mentioning that the Min-Plus Algebra [43] is often used in NC. For any two non-negative



(a) Input Flow (b) Output Flow (c) End-to-End Delay

Figure 2.2: Flow Cumulative Function

functions, $\forall f_1, f_2 \geq 0$, the Min-Plus Convolution and Deconvolution are defined in equations (2.1) and (2.2) respectively. These will also be used in the following sections.

$$f(t) = f_1 \otimes f_2 = \inf_{t \geq s \geq 0} \{f_1(s) + f_2(t-s)\} \tag{2.1}$$

$$f(t) = f_1 \oslash f_2 = \sup_{s \geq 0} \{f_1(t+s) - f_2(s)\} \tag{2.2}$$

2.1.2 Arrival Curves

To avoid a sudden infinite number of flows entering the system, the traffic sent by sources is usually constrained by the arrival curve α. The arrival curve defines the maximum number of bits that can enter the system regardless of when the observation is started, which can also be expressed by a mathematical function:

$$R(t) - R(s) \leq \alpha(t-s), \forall t \geq s \geq 0 \tag{2.3}$$

where α is a monotonic non-decreasing function $\mathbb{R}^+ \rightarrow [0, +\infty]$. Due to the relationship with leaky buckets [44], the specific arrival curves are usually defined in a piecewise format:

$$\alpha(t) = \gamma_{r,b}(t) = \begin{cases} rt + b & \text{t >0} \\ 0 & \text{t = 0} \end{cases} \tag{2.4}$$

Before the explanation of this piecewise function, a brief introduction of Leaky Bucket working mechanism is needed. In Leaky Bucket, there is a bucket that can hold size b of fluid. The fluid will leak at the rate of r unit/second into the hole of this bucket when the bucket is not empty. So,

taking the disassembly of $\alpha(t) = rt$ and $\alpha(t) = b$ separately, the previous means at any time interval Δt, the number of bits for a flow allowing to traverse through the system is limited to $r \times \Delta t$. This also refers to the rate of a physical link connecting two servers where this flow passes by is r bits/second at maximum. The latter constant function $\alpha(t) = b$ indicates the maximum bits for a flow that can be sent is b bits in the sudden beginning, where b is also defined as burst in the flow. Thus, $\alpha(t) = \gamma_{r,b}(t) = rt + b$ means that the source of this flow is allowed to send b bits at the beginning, but no more than r bits/second for the later run.

Recalling equation (2.3), since time points t and s can be chosen at random numbers as long as the condition $\forall t \geq s \geq 0$ is satisfied, a set of overlapping intervals can be depicted in figure 2.3.

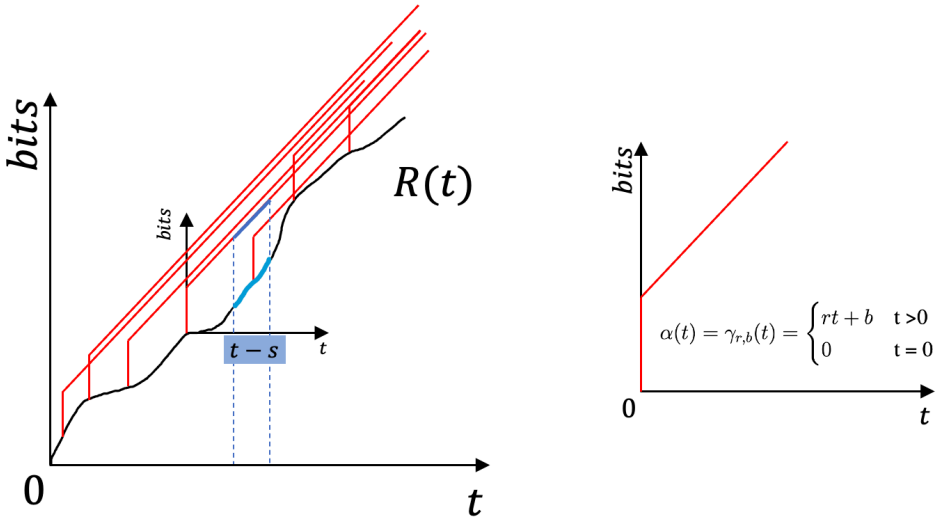


Figure 2.3: Arrival Curve

By using the Min-Plus Convolution equation (2.1), the arrival curve equation (2.3) has an equivalent format:

$$R(t) - R(s) \leq \alpha(t-s) \Leftrightarrow R(t) \leq R(s) + \alpha(t-s) \Leftrightarrow R \leq R \otimes \alpha \quad (2.5)$$

2.1.3 Service Curves

Corresponding to the arrival curves, the network system will in return guarantee the QoS to the flow requests by the concept of the service curves

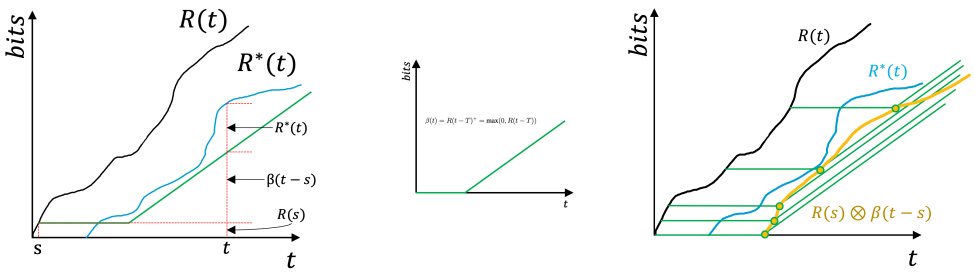
β. The service curves define the minimum service or forwarding capability of the system. Concretely, during a time interval Δt when a flow is traversing through a system, this flow will receive an amount of service with the minimum value of $R \times \Delta t$. The rate R is defined as the service rate. $\forall t \geq 0$, there always exists $\exists s \in [0, t]$ so that following equation (2.6) can be satisfied.

$$R^*(t) \geq R(s) + \beta(t - s) \tag{2.6}$$

Similar to the arrival curve, equation (2.6) is equivalent to equation (2.7) by using the Min-Plus Convolution (2.1). There are several ways for the service curve such as Rate-Latency [45], the Deficit Round Robin [46][47], Packetized Generalized Processor Sharing [48][45], and etc. In this project, Rate-Latency is used to describe the service curve due to its simplicity for both understanding and implementation. Specifically, there is a latency reflecting the time interval T which the network system does not serve any flows, and after this, bits of the flow will be served at a constant rate R. This behavior can be formulated by equation (2.8), and depicted by figure 2.4a.

$$R^* \geq R \otimes \beta \tag{2.7}$$

$$\beta(t) = R(t - T)^+ = \max(0, R(t - T)) \tag{2.8}$$



(a) Service Curve (b) Minimum Output

Figure 2.4: Service Curve and Minimum Output

By using the service curves rate latency (2.8), the cumulative function for the input flow depicted in figure 2.2a and the Min-Plus Convolution equation (2.1), the minimum output of the network system can be visualized from figure 2.4b. This also reclaims the definition of the service curve that the output $R^*(t)$ should be larger than $R(s) \otimes \beta$ as can be referred from the equation (2.7).

Obviously, there is usually more than one flow served on the server recalling the abstracted figure 1.2, and the server capacity is not infinite. Therefore, several flows shall share the service curve on the server. The residual service curve [27] is used to indicate this case. For instance, given flow f_1 with an arrival curve α_1 and flow f_2 with an arrival curve α_2, they are served on a server s with the service curve β. The residual service curve for f_1 means that the service curve f_1 can obtain in the case of coexistence with f_2 on server s. This is mathematically formulated in equation (2.9) where $1_{\{\delta>0\}}$ means the binary parameter will be 1 if δ satisfies the condition, otherwise 0.

$$\beta_{f_1}(t,\delta) = \sup\{\beta(t) - \alpha_2(t-\delta)\} \cdot 1_{\{\delta>0\}} = \beta \ominus_\delta \alpha_2 \tag{2.9}$$

Apart from the residual service curve, the concatenation is also needed for the calculation of the delay bound. Similar to multiple flows on a single server, there are usually multiple servers in one network system depicted in figure 2.5. Recalling that the network system can be viewed as a black box, the equation (2.7) can also be written as $R^* \geq R_{n-1} \otimes \beta_n \geq (R_{n-2} \otimes \beta_{n-1}) \otimes \beta_n \geq \ldots \geq (((R \otimes \beta_1) \otimes \beta_2) \otimes \ldots \otimes) \beta_n = (((R \otimes \beta_1) \otimes \beta_2) \otimes \ldots \otimes) \beta_n$. Consequently, given the setting that a server i with the service curve β_i, the concatenation of the service curve is written in equation (2.10)

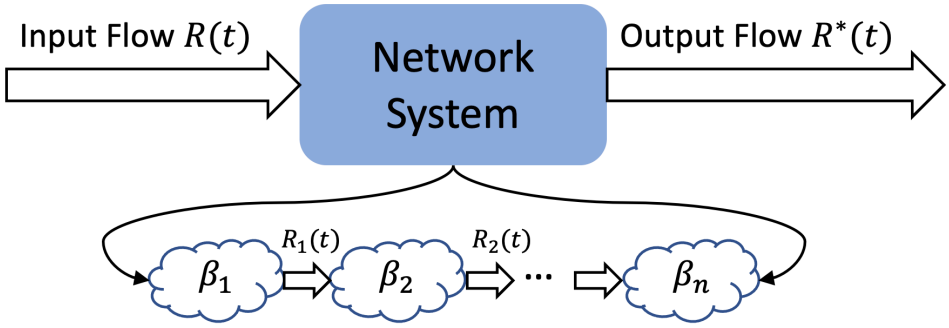


Figure 2.5: Service Concatenation

$$\beta = \beta_1 \otimes \beta_2 \otimes \ldots \otimes \beta_n \tag{2.10}$$

2.1.4 Delay Bound

Recalling the horizontal deviations depicted in Figure 2.2c represent the end-to-end delays, they indicate that the end-to-end delays can be any values in a

certain range, i.e., any values reflected by the horizontal deviations between the input flow $R(t)$ and the output flow $R^*(t)$ in Figure 2.2c. This phenomenon also accords with the *ping* request from EPFL to KTH example mentioned in Chapter 1.

For a network with a single flow and a single server, the arrival curve $\alpha(t)$ and the service curve $\beta(t)$ can be calculated by arrival curves and service curves equations mentioned in Subsections 2.1.2 and 2.1.3. Therefore, Figure 2.2c can be transformed into Figure 2.6 with the delay bound $\frac{b}{R}+T$ and the backlog bound $rT+b$ combining the equations (2.4) and (2.8). The largest vertical deviation is defined as the backlog bound, indicating the largest buffer size of a network system. It is another important research concept in NC, but it is not the focus of this thesis.

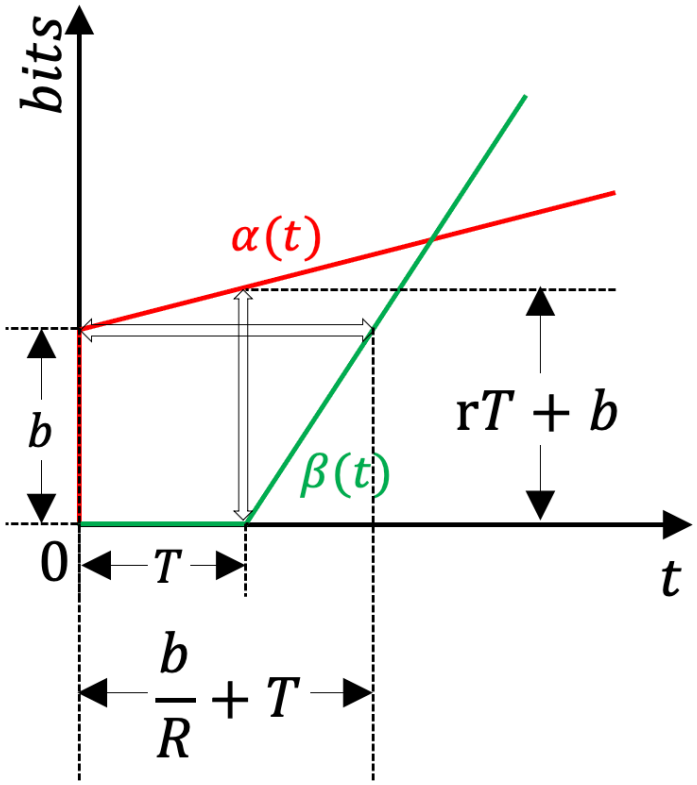


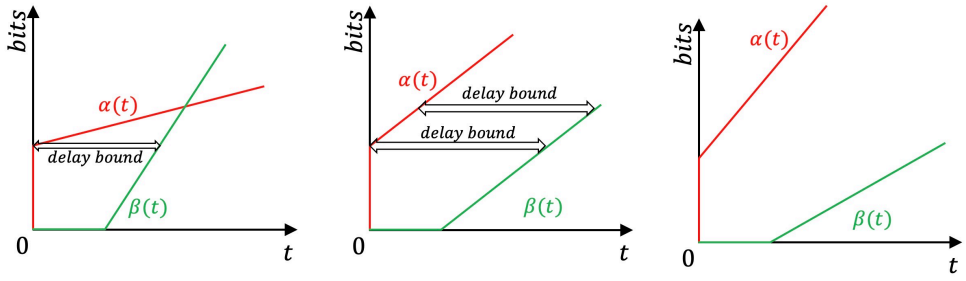
Figure 2.6: Delay Bound and Backlog Bound

Two relationships needed to be discussed here. First, regarding the end-to-end delays $d(t)$ and the delay bounds $h(\alpha, \beta)$, it is defined as followed [3]:

Assuming that a network flow constrained by the arrival curve α traverses a network system offering the service curve β, the end-to-end delay should satisfy the equation (2.11) This also accords with Figure 1.3. Moreover, this also explains why Figure 2.2c can be transformed into Figure 2.6. It is mainly because all flows can not exceed the guarantee of the arrival curve α, so do the servers with the service curve β. The end-to-end delay $d(t)$ visualized from the horizontal deviation between the input flow $R(t)$ and the output flow $R^*(t)$ is constrained by these two curves. Consequently, the largest horizontal deviation between these two curves can be used to depict the delay bound $h(\alpha, \beta)$.

$$d(t) \leq h(\alpha, \beta), \forall t \tag{2.11}$$

Second is the relationship between the arrival curve rate r and the service curve rate R. If $r < R$, a delay bound with a specific value can be calculated for sure as shown in Figure 2.7a. This means that the network system can handle the flow requests, and such kind of network system is defined as robust. For $r = R$, due to the same distances between the two polylines, there will be numerous delay bounds in this case of Figure 2.7b. This refers to the flow requests reaching the limit of the network system capacity. For $r > R$, as can be seen from Figure 2.7c, no largest horizontal deviation can be found. Therefore, there are no delay bounds in this scenario. In another understanding, when the number of bits in this flow exceed the server capacity, the network system is not able to serve this flow request, and consequently, the end-to-end delay will be infinite, which also causes an infinite delay bound.



(a) $r < R$ (b) $r = R$ (c) $r > R$

Figure 2.7: Relationship between r and R

In sum, for a given network topology, as long as the arrival curve and the service curve can be calculated, the delay bound can be consequently obtained.

2.1.5 Flow Prolongation

Recalling the working mechanism of FP in Figure 1.6, it actively prolongs some flows to new sink servers and therefore changes the shape of a network topology. It is recently discovered that the FP can tighten the delay bound [9].

To explain the reason behind, the maximum output curve is needed. Once Figure 2.6 is derived, a new curve will be defined, i.e., the maximum output curve. Different from the minimum output depicted in Figure 2.4b, the maximum output accords with the worst case input when the backlog bound is triggered, shown as $\alpha^*(t)$, the yellow polyline in Figure 2.8. By using the deconvolution equation (2.2), the maximum output can be defined as equation (2.12), which will be used to calculate the delay bounds before and after FP.

$$\alpha^*(t) = (\alpha \oslash \beta)(t) \tag{2.12}$$

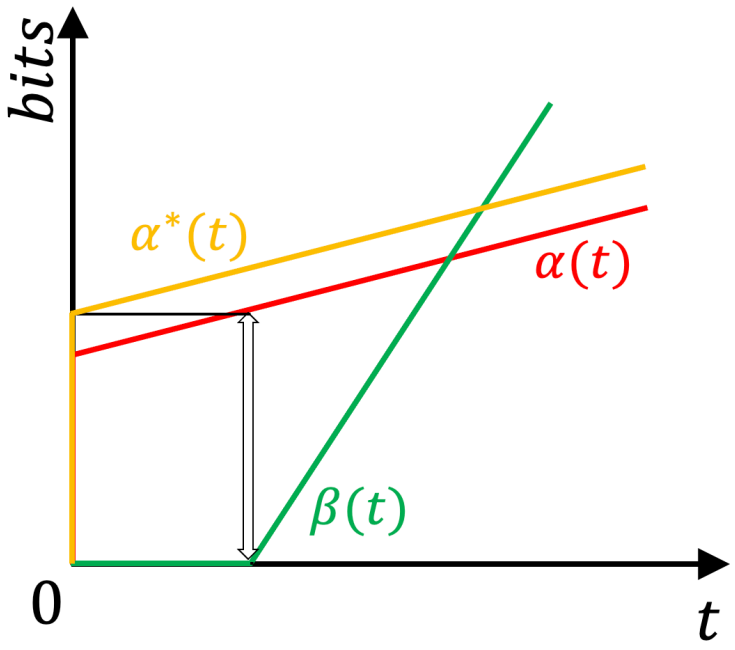


Figure 2.8: Maximum Output

Given the network settings in Figure 1.6, the delay bound before the FP can be calculated by combining equations (2.5), (2.9), (2.10), (2.11) and (2.12).

$$h(\alpha_{foi}, ((\beta_1 \otimes (\beta_2 \ominus_\delta \alpha_2)) \ominus_\delta \alpha_1) \\ \otimes (\beta_3 \ominus_\delta (\alpha_2 \oslash (\beta_2 \otimes ((\alpha_{foi} + \alpha_1) \oslash \beta_1)))))) \tag{2.13}$$

Similarly, the delay bound after FP shown in figure 1.6b can be calculated by the same set of equations and the result is:

$$h(\alpha_{foi} + \alpha_1, \beta_1 \otimes ((\beta_2 \otimes \beta_3) \ominus_\delta \alpha_2)) \tag{2.14}$$

With the absence of deconvolution $\oslash$, this has been proven by previous research [27][9] that the FP has the potential to tighten the delay bound. However, as can be observed from Figure 1.7 that FP may also loose the delay bound, it still remains mysterious which flows should be prolonged. Thus, the most rigorous and accurate way is by the exhaustive search, but it is not scalable when the network size becomes large.

2.2 Graph Neural Network

2.2.1 Graph Neural Network Overview

The graph, as a basic data structure in computer science, is inevitable in life. In graphs, every entity can be linked by tens of or even thousands of connections. By this means, data relationships can be depicted in an easy way. However, in reality, solving problems related to graphs is more difficult due to scalability issues. Therefore, graph theory [49] is also one of the most important and challenging areas in computer science history. GNN is a class of neural network models for processing data that can be represented as graphs [50][51]. In terms of the research on GNN, they have already been widely conducted in the past decades since the first proposal in 2008 [50]. Nowadays, GNN is witnessing a large number of applications in the fields such as image and video processing [52], natural language processing [53], biology [54], social networks [55] and etc.

2.2.2 Graph Neural Network Working Mechanism

Information in the form of scalars/vectors/matrices/tensors are embedded as nodes in the graph where the edge between two nodes indicates the relationship. To visualize this connectivity relationship of n nodes, a square $n \times n$ Adjacency Matrix A is used. The element inside the adjacency matrix can show which two nodes are connected. For instance, element $n_{i,j}$ means

there is a link pointing from node n_i to node n_j. If this adjacency matrix A is symmetric, namely $A^T = A$, it manifests the graph is undirected, otherwise directed. Considering not every two nodes are connected with each other, the adjacency matrix is usually sparse. Another important feature is that if any two lines or columns are switched in order, this behaviour will not influence the connectivity relationship between two nodes even though the adjacency matrices before and after the order changes are different visually. This feature will benefit the creation of the adjacency matrix and can guarantee the permutation invariant for the results of GNN.

GNN can be built via a message passing mechanism [56], a channel for neighbour nodes sharing the information. Specifically, the information of one node will be updated by aggregating the original information of itself and its adjacent neighbours. The step of aggregation is defined as pooling, a common terminology used in neural networks. In the later iterations, the information of a larger-distance neighbour may also be aggregated into this node so that the long-distance message passing can be realized. Not only the node vector itself but also all the relevant adjacent nodes and the connecting edge vectors can be used in the final prediction step, which is also defined as the so-called attention mechanism. Namely, the attention mechanism will consider all the relevant messages of this node for the prediction. This is also the benefit of GNN because node/edge vectors leverage the connectivity property of a graph.

In essence, the process of how GNN works is that given an input graph, this graph will go through the layers of GNN, during which the graph will be transformed with updated information by message passing, but the graph shape will remain the same with the changes of node/edge feature values. This is also known as the permutation invariant property. The transformed graph will be transmitted through other GNN layers e.g., the classification layer, where the prediction values can be generated. Figure 2.9 demonstrates this process.

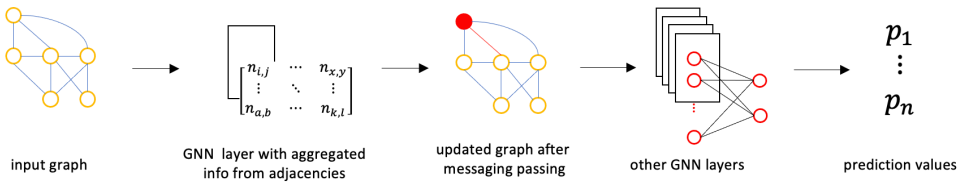


Figure 2.9: Schematic for GNN Working Mechanism

2.3 Fast Gradient Sign Method

2.3.1 Adversarial Attack Overview

Noises, just like graphs, are around human beings. In an ambient sound field, they refer to the noise itself; in images/videos, they are some unwanted tiny points affecting visual perception, and in the telecommunication area, noises are always present in any devices or communication links. Tremendous efforts have been implemented to diminish side effects of noises, but noises cannot be completely eliminated.

In terms of neural network, researches have found that by adding some tiny noises into the inputs manually, the output results may be very different. This noise is defined as the Adversarial Attack. The earliest example can be traced back to 2014 when Goodfellow, Shlens and Szegedy [35] first did an experiment on adding some perturbations into images, and found that the classification results given by the neural network model went to a totally opposite way. This discovery breaks a new ground where people begin to pay attention to the robustness of a model. Later on, a new way of training the model is also proposed based on the adversarial attack by Goodfellow, the so-called Generative Adversarial Networks (GAN) [57], which is also a landmark in Machine Learning.

2.3.2 FGSM in Theory

FGSM is the method used in the first adversarial attack experiment in 2014, aiming to disturb the model so as to cause the misclassification. This method is remarkably powerful and intuitive so that it is still widely used nowadays. Gradient is the core target on which this method focuses. Normally, gradient is calculated in the backpropagation to update weights among neurons so as to minimize the loss function. In the view of the convex loss function, it goes into the direction of the global minimum value. However, FGSM goes into the opposite direction, i.e., the adversarial attack modifies the input data to maximize the loss function value based on the same backpropagation gradients. It can be formulated into equation 2.15.

$$\hat{x} = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{2.15}$$

Specifically, x is the original input data; y is the correct classification target, and θ represents the weight and biases parameters in GNN. In total $J(\theta, x, y)$ means the loss function of applying the GNN with weight and biases

parameters θ on input-output pairs (x, y). Therefore, the gradient on the input x will be $\nabla_x J(\theta, x, y)$. Afterwards, a $sign(a)$ function is imposed, defined in equation 2.16. This guarantees that the gradient goes into the direction to enlarge the loss function. ϵ is defined as the perturbation factor, influencing in which degree of the adversarial attack will be conducted. In the end, $\hat{x}$ is generated, indicating the input after being added the noises, the so-called tiny modification/perturbations.

$$sign(a) = \begin{cases} 1 & \text{a} > 0 \\ 0 & \text{a} = 0 \\ -1 & \text{a} < 1 \end{cases} \tag{2.16}$$

2.3.3 FGSM Examples

There are two typical and representative examples on FGSM. One is the animal recognition shown in figure 2.10 given by the paper "Explaining and Harnessing Adversarial Examples" [35]. It is easy for us humankind to recognize that this image after FGSM is still a panda. However, neural network holds the belief that it is a gibbon with 99.3% of confidence, even though it used to believe that this image is a panda without the adversarial attack.

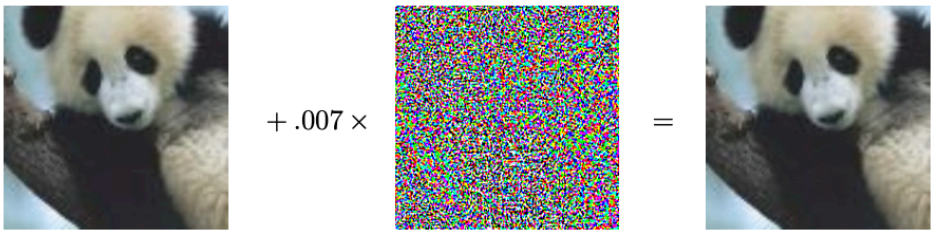


Figure 2.10: FGSM on Animal Recognition [35]

Another example shows how different ϵ values can affect the handwriting recognition in the dataset of MNIST shown in figure 2.11 [58]. Without the adversarial attack, i.e., $\epsilon = 0$, neural networks can recognize every handwriting correctly demonstrated in the first line of the figure 2.11. When ϵ is non-zero, i.e., noises are added into the handwriting figure, neural networks

begin to make mistakes. When ϵ reaches 0.3, as can be seen from the last line of figure 2.11, quite an amount of noises are added and it is somehow difficult for us, human, to recognize the correct number as well. Figure 2.11 is also a good example to explain the relationship between x and $\hat{x}$. x is the original input in the first line when ϵ is zero, and $\hat{x}$ is the attacked input in the rest of the lines when ϵ is not zero. These groups of results also accord with the perception that a larger perturbation factor ϵ will not only lead to vague visual image inputs, but also results in a more obvious misclassification.

Figure 2.11: FGSM on Handwriting Recognition [59]

In this chapter, key background information about NC, GNN, and FGSM is illustrated. How they are implemented in this project will be discussed in detail in the next Chapter. For more information about these three domain knowledge, here are the recommended references. Prof. Le Boudec book's "Network Calculus: A Theory of Deterministic Queuing Systems for the Internet" [3] is reader-friendly to understand both the general and advanced knowledge of NC, where there are a mass of theoretical mathematical equations and demonstrations. Sanchez-Lengeling, Reif, *et al.* established a website playground to demonstrate how GNN works, and there are dozen of parameters, models and examples with which can be played on this website [51]. "Explaining and Harnessing Adversarial Examples" [35] as the first paper in FGSM enjoys a high citation on Google Scholar, and this paper is worth repeatedly reading for the understanding of the adversarial attack. For FGSM, one popular direction nowadays is how to defend the misclassification that happens after the attack, or even before the attack happens so that the accuracy and the robustness of the model can be guaranteed.

Chapter 3

Experiment Method and Setting

Chapter 2 introduces how GNN, NC, and FGSM work separately in theory. In this Chapter 3, more details related to how they are combined and implemented in this project will be unboxed. First, the hardware specifications of where this project runs are introduced briefly. Section 3.2 will demonstrate how to input a network topology into the NetCal DNC so as to calculate the delay bound. Section 3.3 will reproduce the GNN model [27] training process. Section 3.4 will reveal how this GNN model can make a prediction on a brand new network apart from those stored in the training and testing datasets. Finally, section 3.5 will show how the FGSM will fool the GNN model so that some flows are prolonged by mistake, and thus influencing the calculation of the delay bounds.

3.1 Hardware Specifications

The whole project is running on EPFL clusters. Considering machine learning is engaged in this project, the codes can be categorized into two parts: machine learning related and the rest due to the usage of the Graphics Processing Unit (GPU) on the tensor calculation.

For codes where there are no needs for the GPU, they are running on Fidis [60]. Each compute node of this cluster is composed with 2 Intel Broadwell processors running at 2.6 GHz with 14 cores each. 336 out of 408 compute nodes have 128GB of RAM, while the rest have 256 GB of RAM.

Regarding codes related to machine learning, GPU is highly needed. They are running on Izar [61]. There are 72 computes node in this cluster in total where 70 of which are equipped with 2 Xeon-Gold processors running at 2.1 GHz, with 20 cores each, and 2 Nvidia V100 PCIe 32 GB GPUs. Half of these 70 compute nodes are with 192 GB of DDR4 RAM, and half of them are with

384 GB of DDR4 RAM. There are also two more powerful compute nodes equipped with 4 Nvidia V100 SXM2 32 GB GPUs and 768 GB of DDR4 RAM.

3.2 The Usage of NetCal DNC

As is discussed in Chapter 1, NetCal DNC [26] is chosen as the Free and Open-Source Software (FOSS) to calculate the delay bound in this project. This section will demonstrate how to transfer a network topology into the input of NetCal DNC, and obtain the delay bound output.

Recalling equations (2.8) and (2.4) as well as figures 2.4a and 2.3, there are four key features for a given network setting, namely, the server rate R, the server latency T, the flow rate r, and the flow burst b. In this project, all these four network features are deterministic, i.e., once a network setting is given, all these network features will be set as immutable rather than dynamic as time goes by. Figure 3.1 shows how a network topology is quantified based on the abstracted network topology Figure 1.2 embedded with four key network features.

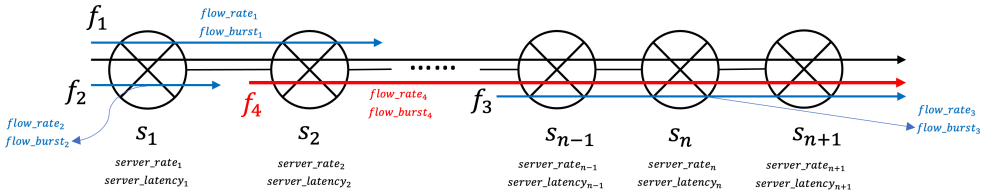


Figure 3.1: Topology with Network Features

To use the NetCal DNC tool, these four features should be defined by user inputs. Two internal functions of creating the service curves by the Rate Latency and the arrival curves by the Token Bucket will be triggered. This completes the step of calculating the two important curves recalling figure 2.6, and these two curves will also be embedded into the servers and flows respectively. Moreover, to embed the arrival curve into the flows, the source and the sink/destination servers of this flow should be defined.

The server order should be claimed as well and the server should be connected with each other as a tandem. Due to the usage of the tandem network topology in this project, the servers are connected one by one, namely, server s_n is connected between servers s_{n-1} and s_{n+1} as is shown in figure 3.1

instead of in a shape of tree or circle. For some networking topologies in the training and testing dataset, servers are connected in a descending order. Taking figure 3.1 as an example, a descending order shows the opposite server id order, meaning that from left to right, server ids are $n+1$, n, $n-1$, ... to 1, while in an ascending order, it is what is shown as figure 3.1. In the attacked dataset, all the servers are connected in an ascending order to refuse confusion. The illustration of these datasets will be discussed later in this chapter.

After the prerequisites of the network setting definition, various NC methods can be called, such as TMA, SFA, PMOO and LUDB-FF. It is worth mentioning that to use LUDB-FF, an additional CPLEX Optimization Studio [62] needs to be installed beforehand.

3.3 GNN Model Reproduction

As is claimed in Chapter 1, Geyer, Scheffler and Bondorf proposed a GNN model [27] for the prediction of FP. However, they don't open-source the GNN model, so the reproduction of GNN model is the priority. To train this GNN model, training and testing datasets are needed. Fortunately, these datasets are open-sourced by Geyer Fabien where the parameters can be referred from Table 3.1. The datasets are designed with the network setting with the four key network features discussed in section 3.2. To show the foi, Deborah and PMOO delay bounds are added under the foi's information block with some explored combinations of FP alternatives. The datasets are stored in a .pbz format [63] developed by Google.

Table 3.1: Dataset Parameters Used to Train the GNN Model [27]

Parameter	**Min**	**Max**	**Mean**
# of servers	4	10	7.8
# of flows	5	35	24.5
# of cross-flows	1	21	4.1
# of prolong. comb. (PMOO-FP$_{foi}$)	2	4024	16.8
# of prolong. comb. (DEBORAH-FP$_{foi}$)	2	131072	247.1
Flow path length	3	9	4.1
Number of nodes in graph	11	128	43.3

For the GNN model training inputs, the training dataset only defines the four key network features, flows' paths and explored FP combinations under

fois. These data cannot be directly recognized by GNN. As the pre-requisite for the GNN, graphs are supposed to be defined, where nodes and edges are two basic elements in the graph data structure. A process of the graph transformation from the original network topologies to a GNN recognized topology can be referred from Algorithm 1 [27]. Specifically, servers and flows in a network topology are represented by nodes in the graph structure, and two server nodes are connected with each other via an edge in the graph if there is a link between them in the real network topology. To indicate the flow path, edges are added between this flow and servers where this flow passes by. For the analysis of foi and its FP alternatives, additional nodes are added to indicate whether other flows are worth prolonging, and additional edges will be added as well to indicate the new sink server of this prolonged flow.

To visualize Algorithm 1, figure 3.1 can be viewed as the input netowrk topology $\mathcal{N}$, and figure 3.2 can be viewed as the output graph $\mathcal{G}$. As can be seen from the blue and red edges connecting flows and servers, these edges represent the flow paths, where red is used for the indication of the foi, and blue for the normal flows. Green edges are added for the purpose of FP with one prolongation node $P_{f,s}$ on each edge. These additional nodes $P_{f,s}$ will be valued by GNN to indicate to which server this flow should be prolonged shown in the statement 'prolong to which flows?'. One node is also added to the foi to judge whether this topology for this foi is worth prolonging or not, shown by the statement 'worth prolonging?'. If this foi judged by GNN does not worth prolonging, then no more FP combinations based on this foi will be explored. In other words, no green edges will be added.

Up till now, a network topology $\mathcal{N}$ has been transformed into an undirected graph $\mathcal{G}$, satisfying the definition of graphs. One more step for the input of the GNN is the transformation of this graph to tensor, which is the format frequently used for neural networks. Figure 3.3 depicts the network features tensors. In terms of columns, the tensor is divided into three parts from the top to the bottom, i.e., server features in grey, flow features in blue and FP features in red. For the convenience of explanation, labels are used to mark different features. 0, 1, 3 are binary numbers, indicating whether this element in the tensor is a server or a flow or a FP node. As is discussed in the previous chapters for several times, server rate, server latency, flow rate and flow burst are also stored in this tensor with the label number 4,5,6,7 separately. Label 2 indicates whether a flow is foi or not. Label 8 reveals to which server is this flow prolonged to. It is worth mentioning here that for one network topology and a given foi, there exist multiple FP options. Therefore, for a single flow which is possible to be prolonged, there are different hops represented by

Algorithm 1 Graph transformation of network $\mathcal{N}$ for flow f_{foi}

Input:
$\mathcal{N}$:= network topology stored in the dataset
$\mathcal{G}$:= empty undirected graph as an input
Output:
$\mathcal{G}$:= corresponding undirected graph of $\mathcal{N}$ after the transformation

```
// add server nodes and the links between the servers
for server s_i in network topology N do
    G.addNode(s_i)
end
for link (s_i, s_j) in network topology N do
    G.addEdges(s_i, s_j)
end
// flows are also added as nodes
// links between this flow and servers are used to indicate their paths
for flow f_i in network topology N do
    G.addNode(f_i)
    for server s_j in f_i.path() do
        G.addEdge(f_i, s_j)
    end
end
// prolong the flows excluding the foi for flow f_i in network N excluding f_foi
do
    for server s_j in f_foi.path() do
        if prolongation P_{f_i}^{s_j} of flow f_i to s_j is valid then
            G.addNode(P_{f_i}^{s_j})
            G.addEdges((f_i, P_{f_i}^{s_j}), (P_{f_i}^{s_j}, s_j))
        end
    end
end
```

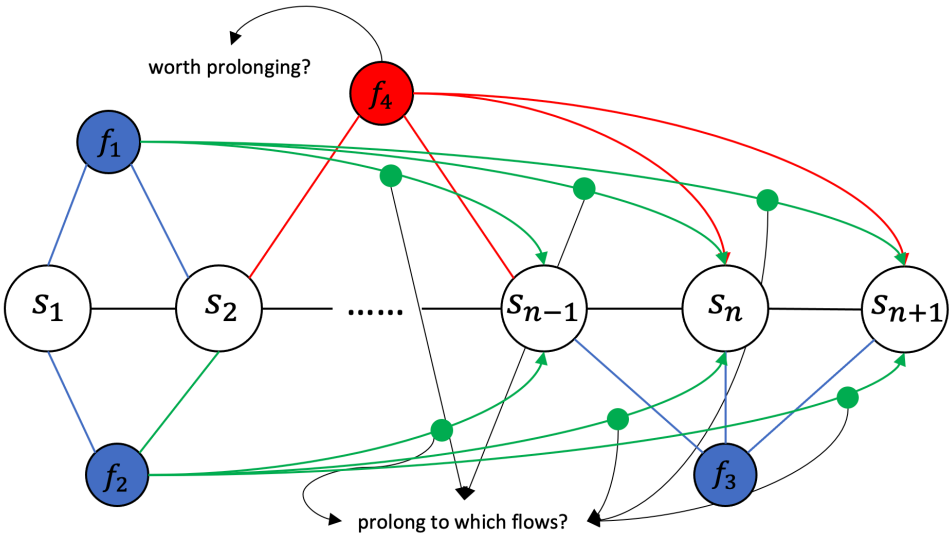


Figure 3.2: Transformed Topology Graph $\mathcal{G}$

figure 3.2. For instance, server s_{n-1}, s_n, and s_{n+1} will all be stored in the tensor for the sink servers of flow f_1 FP possibility.

Recalling the adjacency matrix discussed in section 2.2, it is used to show the connection of servers and flow paths based on transformed graph 3.2. Take the example of figure 3.2 again. The adjacency matrix could be represented by equation (3.1), where 1 means there is a link between these two nodes, otherwise zero. It also accords with the property of sparsity and symmetry for an adjacency matrix of an undirected graph.

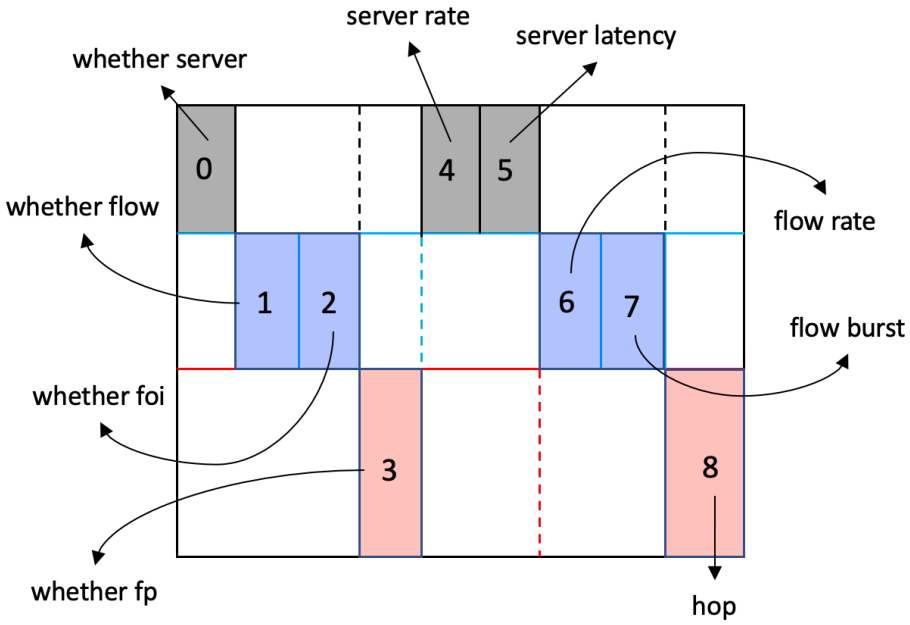


Figure 3.3: Original Network Features Tensor

$$
\begin{array}{c}
 \\
s_1 \\
s_2 \\
\vdots \\
s_{n+1} \\
f_1 \\
f_2 \\
f_3 \\
f_4 \\
FP_{f_1,s_1} \\
\vdots \\
FP_{f_2,s_{n+1}}
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccc}
s_1 & s_2 & \cdots & s_{n+1} & f_1 & f_2 & f_3 & f_4 & FP_{f_1,s_1} & \cdots & FP_{f_2,s_{n+1}}
\end{array} \\
\left(\begin{array}{ccccccccccc}
0 & 1 & \cdots & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 0 \\
1 & 0 & \cdots & 0 & 1 & 0 & 0 & 1 & 0 & \cdots & 1 \\
\vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & 1 & 1 & 0 & \cdots & 1 \\
1 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\
1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 \\
0 & 0 & \cdots & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
0 & 1 & \cdots & 1 & 0 & 1 & 0 & 0 & 0 & \cdots & 0
\end{array}\right)
\end{array}
\tag{3.1}
$$

Till this step, everything is ready for the GNN model training. Gated Recurrent Unit (GRU) is frequently used in GNN to trigger the edge attention mechanism discussed in section 2.2. During the edge attention phase, message passing among neighbouring nodes will be realized. Feed Forward Neural

Network (FFNN) is the common step used for the parameters passing among neurons in the calculation of the binary cross-entropy loss function at each epoch expressed by equation (3.2). $T_{f,s}$ is a binary number where 1 stands for this flow should be prolonged, and 0 otherwise. This is also defined as 'target' in the supervised machine learning. $T_{f,s}$ is obtained directly from the training dataset, guiding the optimal FP combinations. $P_{f,s}$, as discussed above, indicates the hypothetical possibility for flow f to be prolonged to server s. The value of $P_{f,s}$ is calculated by GNN model, and will also be discussed in section 3.4. Later on, backpropagation will be implemented for the update of weights and biases as an essential step for all neural networks training procedure. The purpose of backpropagation is to minimize the loss function so that the prediction value $P_{f,s}$ will be close to the target value $T_{f,s}$. The specific layer information can be referred from figure 3.2 where 9 in the initial layer represents the 9 key network settings shown in Figure 3.3.

$$loss(T,P) = \sum_{f \in Flow, s \in Server} (T_{f,s} log P_{f,s} + (1 - T_{f,s}) log(1 - P_{f,s})) \quad (3.2)$$

Table 3.2: GNN Layers [27]

Layer	**NN Type**	**Size**
init	FFNN	$(9, 96)_w + (96)_b$
Memory unit	GRU cell	$(96, 96)_w + 2 \times (288, 96)_w + (96_b)$
Edge attention	FFNN	$(192, 96)_w + (96)_b + (192, 96)_w + (1)_b$
out hidden layers	FFNN	$2 \times (96, 96)_w + (96)_b$
out final layer 1	FFNN	$(96, 1)_w + (1)_b$
out final layer 2	FFNN	$(96, 1)_w + (1)_b$

1.4×10^{-4} and 30 are chosen for the learning rate and epoch respectively based on several trials. For the accuracy evaluation, the testing dataset is used. The accuracy for the model based on PMOO is 65%, slightly lower than 69.6% achieved by the paper [27]. As a digression, a model based on Deborah is also trained since a delay bounds calculated by Deborah are also stored in the dataset. The reproduction accuracy is 76.4%, much higher than 60.9% in the paper [27]. However, it is also mentioned in Chapter 1 that Deborah failed owing to the lack of maintenance. Therefore, the following numerical analysis mainly depends on the model based on the PMOO.

3.4 GNN Predictions on FP

This section will introduce how this pre-trained GNN model make predictions on a network topology. The key parameter is the prediction values added to the foi and the potential FP alternatives, namely statement 'worth prolonging' and 'prolong to which flows' in Figure 3.2.

The statement 'worth prolonging' belongs to the first prediction value on foi, showing whether other flows are worth prolonging in this network topology based on this foi. The threshold value is 0.5, which means that if this prediction value is larger than 0.5, then flows in this network topology with this given foi are worth prolonging, otherwise, are not.

The second prediction values based on the statement 'prolong to which flows' accords with $P_{f,s}$. This wil decide where to prolong a flow if this flow is necessary to be prolonged to, the criteria of which is the highest value. If the prolongation node value on the original sink server of this prolong, it means that this flow will not be prolonged and remain the same sink server.

Table 3.3 reveals one example of the first topology sotred in the training dataset with the foi 6. The first prediction value is 0.98929, much larger than 0.5, meaning there is high possibility for the flows in this scenario to be prolonged. Focusing on flow f_2 where its original path is from server s_7 to s_3 (This is because the servers in this example are connected in a descending order), the highest prediction value is 0.78166 to sink server s_2, larger than 0.23976 remaining the same sink server s_3 and 0.004808 by prolonging to server s_1. This mean that flow f_2 will be prolonged to server s_2 from its original server s_3.

In this way, given a new network topology, GNN will make a decision on which flows should be prolonged to which servers and generate a network topology after the FP. One more point worth considering here is connected to the adversarial attack. The potential attack targets for GNN to be fooled are the prediction values reaching to the threshold. Specifically speaking, when the first prediction value is close to 0.5, it is highly possible for the GNN to turn over the decision after the network features are changed by a little bit. For example, the first prediction value is 0.6 before the adversarial attack, while it switches to 0.4 after the attack. In this case, the GNN model will prolong the network before the attack while do nothing for the scenario after the attack. It is the same for the second prediction values when some of the highest values are quite close to each other.

Table 3.3: Pred Values of Topology 0 in the Training Dataset with foi 6

foi	source server	sink server	PRED1
6	3	1	0.98929
flow id	**source server**	**sink server**	**PRED2**
1	2	2	1.0
1	2	2	8.28881E-09
2	7	3	0.23976
2	7	2	0.78166
2	7	1	0.004808
4	2	2	1.0
4	2	1	7.38276E-09
7	2	2	1.0
7	2	1	8.52992E-09
12	4	2	0.98657
12	4	1	0.00841

3.5 FGSM on Network Features

In this section, more attention will be paid on figure 3.3 because all network features are stored in this matrix. Recalling the equation (2.15), the input of the GNN will be permutated by increasing the value of the loss function.

Accordingly, Figure 3.3 is the input x in equation (2.15). After using the GNN model, the loss function $J(\theta, x, y)$ can be calculated. By backpropagation of the calculation on the gradient of neural network parameters weight and bias, $\nabla_x J(\theta, x, y)$ will be obtained. Function $sign$ will be imposed to guarantee the gradient goes beyond the convex optimization direction. As is discussed, ϵ is the perturbed factor, influencing the extent of this adversarial attack.

In the practical code stage, $torch.clamp$ [64] is also used for the attacked $\hat{x}$. In other words, after calculating $\hat{x} = x + \epsilon \cdots sign(\nabla_x(J(\theta, x, y)))$, $\hat{x}$ should be upated by $\hat{x} = torch.clamp(\hat{x})$. This can clamp all $\hat{x}$ into a certain range. Combined with this project, for example, the server rate cannot be a negative number considering its practical meaning. It is the same for other network features. In this project, the range will be the minimum and the maximum value of its parameter. Take the example of the server rate again. Assuming that there are n servers in one network topology, and therefore there are n server rate r values ranging from r_1 to r_n. A manual constraint is added by

$torch.clamp$ to guarantee the server rates after the attack satisfy the real-life meaning by $min(r) = min(r_1, ..., r_n)$ and $max(r) = max(r_1, ..., r_n)$. It is also the same for other parameters.

During the numerical analysis, it is found that the flow rate R is extremely sensitive to the adversarial attack, i.e., a tiny ϵ value will cause a tremendous flow rate change. Therefore, $\epsilon/100$ is implemented to the flow rate. Figure 3.4 visualizes the process of FGSM on network features.

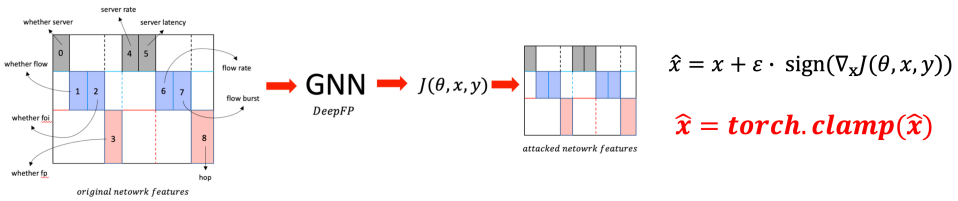


Figure 3.4: FGSM on Network Features

3.6 Adversarial Attack Dataset Creation

The FGSM was first implemented on the open-source training and testing datasets in this project. However, the results are far from satisfactory. Observing the dataset parameters used to train the GNN model from Table 3.1, the number of changeable network features is quite small, ranging from the minimum 18 ($4 \cdot 2 + 5 \cdot 2$) to the maximum 90 ($10 \cdot 2 + 35 \cdot 2$) with the mean value 64.6 ($7.8 \cdot 2 + 24.5 \cdot 2$). It cannot be compared with the image pixel of 784 stored in the MNIST dataset [58]. Considering that the adversarial attack is first and largely used in the field of image recognition, a dataset with larger network settings is needed as an imitation of Computer Vision. However, unlike a pixel in a picture which has only one constraint, i.e., the value of a pixel should be in the range of 0 (black) and 255 (white), there are a series of constraints in the network setting for the creation of the new dataset used for the adversarial attack purpose:

- Recalling the Figure 2.7 showing the relationship between the service rate r and the flow rate R, the aggregated flow rates on one server should not exceed this server's serving capacity, i.e., the server rate of this server r should be larger than the aggregated flow rates passing through this server to guarantee that an endless delay bound won't happen.

Hypothesizing there are N flows whose flow rate is R_{flow_id} traversing through a server, the equation $r >= (R_1 + R_2 + ... + R_n)$ needs to be satisfied.

- Each flow path is advocated to be unique without repetition. If two flows have the same paths, they can be aggregated, so the same flow paths will cause redundancy. In theory, given a network with #n servers, there are #$n \cdot (n+1)/2$ flows at maximum without redundancy.

- Due to the property of delay bound calculation, a flow whose path is completely out of the foi's path will have a tiny influence on the final delay bound result. Therefore, more flows that can be engaged in this network are also advocated. In this case, the source server of the foi is set as one of the first three servers in the network topology, and the sink server of the foi is set to the last server of this network topology.

- The exhaustive search is used to explore tighter delay bounds. For time-saving reasons, once three tighter delay bounds are found by the exhaustive search, the generation of this network topology will terminate and switch to the next network topology. Consequently, the tightest delay bound will be the one among these three values and will be used as the target in the following FGSM step. It is indeed not very rigorous. However considering that computing one delay bound in such a large network is time-consuming, and using the exhaustive search to find a tighter delay bound is also time intensive, it is currently the optimal solution that can be thought to guarantee tighter delay bounds can be explored given a network setting.

- Even though the EPFL servers are powerful, the bottleneck is still met during the new dataset generation. As far as several tests, a network with 30 servers and 232 ($30 \cdot 31/4$) flows meets the EPFL servers' computational limits when the gradients are calculated in the FGSM step. Therefore, even though a larger network size is desired, it is impossible owing to the computational resource restriction. Table 3.4 reveals the network sizes in the newly created dataset for the adversarial attack purpose, though there is still a gap to the number of pixels of an image in Computer Vision.

The specific dataset used for the adversarial attack purpose can be viewed from Table 3.5. Different color blocks indicate different categories. Light yellow (topo id: 0-1999) reveals how the number of servers will influence the

Table 3.4: New Dataset for FGSM

Parameter	Min	Max	Mean
# of servers	20	30	24.9
# of flows	46	232	115.6
Flow path length	1	30	9.3

experimental results; light gray (topo id: 2000-4999) reflects the sparsity of the number of flows; light red (topo id: 5000-5999), light blue (topo id: 6000-6999), gray (topo id: 7000-7999) and light green (topo id: 8000-8999) show different network features respectively, i.e., server rate, server latency, flow rate and the flow burst. Specifically for these four big blocks, only one network feature is changed while others remain the same. For instance the red block, only the server rate is changed while others are immutable. By changing only one feature, we hope to observe how this network feature will influence the adversarial attack results. Finally, the light brown part (topo id: 9000-9999) only does the adversarial attack on the flow features which corresponds to the scenario that given a network setting with good conditions of servers, if the flows are affected by some network perturbations, how this case will influence the calculation of the delay bound.

Table 3.5: Adversarial Attack Dataset Parameter

topo id	**# server** n	**# flow**	**server rate**	**server latency**	**flow rate**	**flow burst**
0 - 999	20 - 30	$n \cdot (n+1)/5$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
1000 - 1499	30 - 40	$n \cdot (n+1)/5$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
1500 - 1999	40 - 50	$n \cdot (n+1)/5$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
2000 - 2499	20 - 30	$n \cdot (n+1)/4$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
2500 - 2999	20 - 30	$n \cdot (n+1)/3$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
3000 - 3499	20 - 30	$n \cdot (n+1)/6$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
3500 - 3999	20 - 30	$n \cdot (n+1)/7$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
4000 - 4499	20 - 30	$n \cdot (n+1)/8$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
4500 - 4999	20 - 30	$n \cdot (n+1)/9$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1
5000 - 5099	25	$25 \cdot (25+1)/9 = 130$	0.01	0.05	0.0005	0.05
5100 - 5199	25	$25 \cdot (25+1)/9 = 130$	0.02	0.05	0.0005	0.05
5200 - 5299	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.05
5300 - 5399	25	$25 \cdot (25+1)/9 = 130$	0.04	0.05	0.0005	0.05
5400 - 5499	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0005	0.05
5500 - 5599	25	$25 \cdot (25+1)/9 = 130$	0.06	0.05	0.0005	0.05
5600 - 5699	25	$25 \cdot (25+1)/9 = 130$	0.07	0.05	0.0005	0.05
5700 - 5799	25	$25 \cdot (25+1)/9 = 130$	0.08	0.05	0.0005	0.05
5800 - 5899	25	$25 \cdot (25+1)/9 = 130$	0.09	0.05	0.0005	0.05
5900 - 5999	25	$25 \cdot (25+1)/9 = 130$	0.1	0.05	0.0005	0.05
6000 - 6099	25	$25 \cdot (25+1)/9 = 130$	0.03	0.02	0.0005	0.05

6100 - 6199	25	$25 \cdot (25+1)/9 = 130$	0.03	0.04	0.0005	0.05
6200 - 6299	25	$25 \cdot (25+1)/9 = 130$	0.03	0.06	0.0005	0.05
6300 - 6399	25	$25 \cdot (25+1)/9 = 130$	0.03	0.08	0.0005	0.05
6400 - 6499	25	$25 \cdot (25+1)/9 = 130$	0.03	0.1	0.0005	0.05
6500 - 6599	25	$25 \cdot (25+1)/9 = 130$	0.03	0.12	0.0005	0.05
6600 - 6699	25	$25 \cdot (25+1)/9 = 130$	0.03	0.14	0.0005	0.05
6700 - 6799	25	$25 \cdot (25+1)/9 = 130$	0.03	0.16	0.0005	0.05
6800 - 6899	25	$25 \cdot (25+1)/9 = 130$	0.03	0.18	0.0005	0.05
6900 - 6999	25	$25 \cdot (25+1)/9 = 130$	0.03	0.2	0.0005	0.05
7000 - 7099	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0001	0.05
7100 - 7199	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0002	0.05
7200 - 7299	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0003	0.05
7300 - 7399	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0004	0.05
7400 - 7499	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0005	0.05
7500 - 7599	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0006	0.05
7600 - 7699	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0007	0.05
7700 - 7799	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0008	0.05
7800 - 7899	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.0009	0.05
7900 - 7999	25	$25 \cdot (25+1)/9 = 130$	0.05	0.05	0.001	0.05
8000 - 8099	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.01
8100 - 8199	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.02
8200 - 8299	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.03

8300 - 8399	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.04
8400 - 8499	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.05
8500 - 8599	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.06
8600 - 8699	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.07
8700 - 8799	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.08
8800 - 8899	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.09
8900 - 8999	25	$25 \cdot (25+1)/9 = 130$	0.03	0.05	0.0005	0.1
9000 - 9999	20 - 30	$n \cdot (n+1)/5$	0.05 - 1	0.01 - 1	0.0001 - 0.0005	0.01 - 1

3.7 Project Implementation

Figure 3.5 reveals the steps to train the GNN model. Same with what has been discussed before, $prepare\_dataset\_pmoo$ code will do the job of changing the dataset stored in the .pbz format to a .pickle format where network topologies are changed to network feature tensors. These tensors will be used for the training process of the model. Once the trained-GNN model is obtained, it can be used for the prediction of the FP on a brand-new topology.

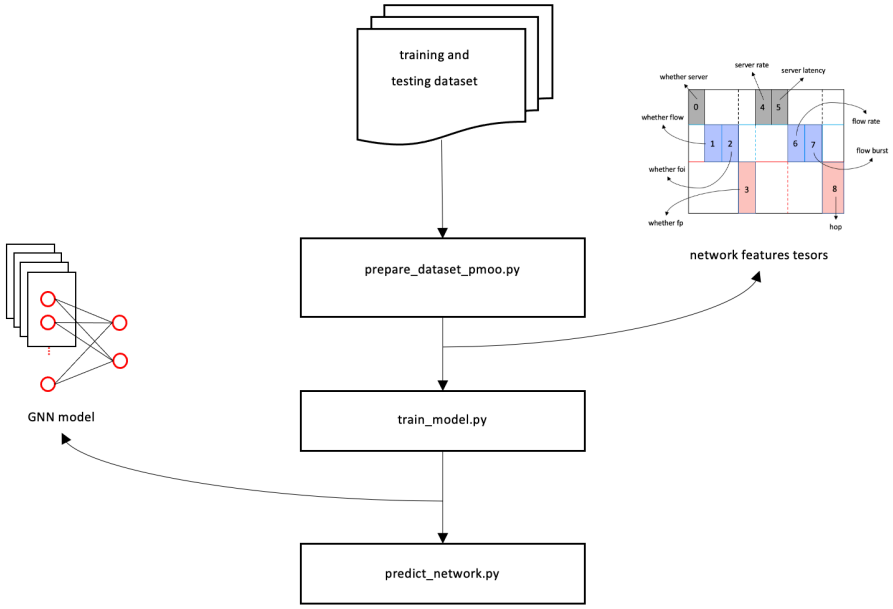


Figure 3.5: GNN Training Steps

To unbox the adversarial attack process, Figure 3.6 can be referred to from the perspective of codes. After the new dataset shown in Table 3.5 is generated for the adversarial attack purpose, the first prediction can be done based on these original network topologies, namely the network features before the attack. Later on, the potential targets can be found as discussed in section 3.4 based on two different criteria on two different prediction values. The attack dataset will be given to the FGSM code to generate a new set of network topologies with the modification of network features. Afterward, these network topologies will be predicted.

As can also be seen from Figure 3.6, four important categories of networks are introduced in four different colors. To clarify the name and its usage,

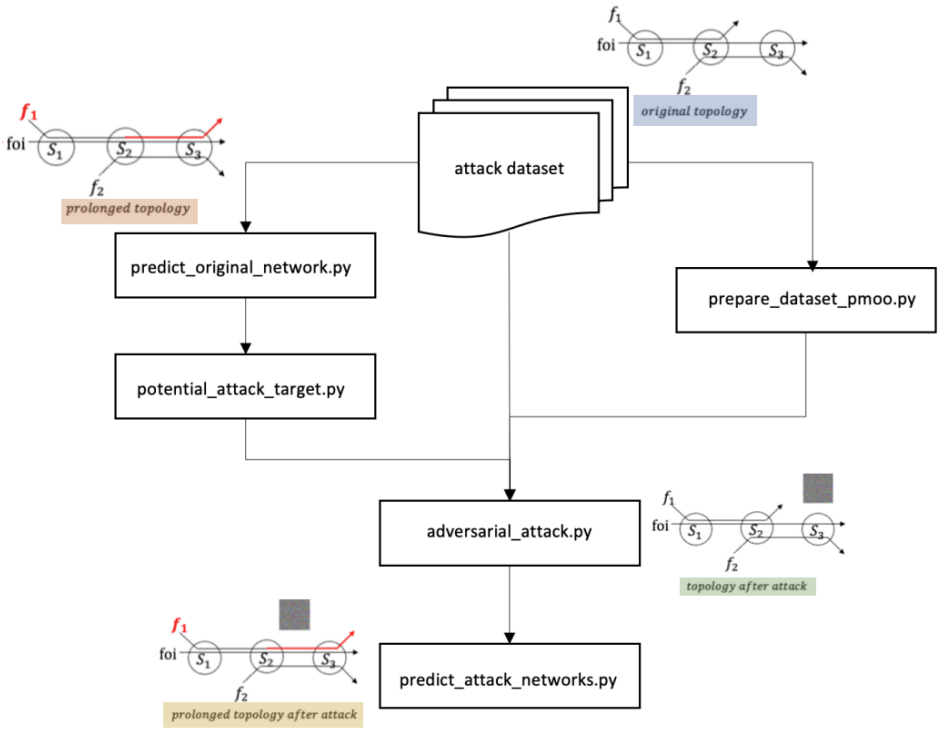


Figure 3.6: Adversarial Attack Steps

Table 3.6 is created with the corresponding color to different networks in Figure 3.6. Acronyms are used for the clarification of different types of networks. The first letter o/a represents whether it is an original network topology or a network topology after the adversarial attack. The second letter b/a indicates whether this is a network topology before or after the FP together with the last two letters fp. For instance, $obfp$ stands for the original network topologies before FP and before FGSM attack. These four network categories are of great importance in the coming Chapter 4.

Table 3.6: Network Categories

category	acronym	FP?	FGSM?
category 1	obfp	×	×
category 2	oafp	✓	×
category 3	abfp	×	✓
category 4	aafp	✓	✓

As a supplementation to Figure 3.6 and Table 3.6, Figure 3.7 is created for a better understanding of the whole adversarial attack process. Specifically speaking, given an original topology without FP and FGSM, how the following topolgoies after GNN and/or FGSM will be generated.

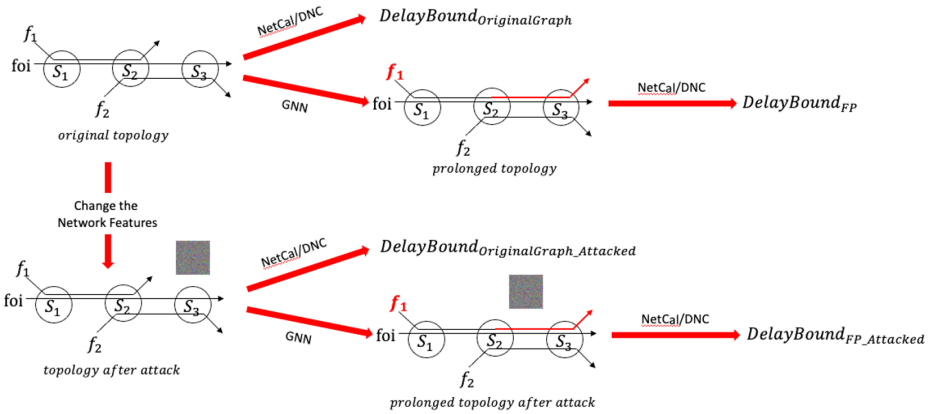


Figure 3.7: Adversarial Attack Visualization

Chapter 4

Results and Analysis

4.1 Analyzed Network Sizes

Based on the dataset features in Table 3.5 and the network categories in Table 3.6, an additional Table 4.1 is demonstrated to show the number of analyzed networks in this project. As can be observed, 10000 original network topologies with different network features are generated. However, only 8500 of them can be prolonged, due to the limitation by the EPFL cluster capabilities recalling the fourth bullet point in Section 3.6. For instance, a network topology with more than 30 servers will trigger the bottleneck of CUDA Out of Memory for the EPFL cluster. Afterward, the potential attack targets are explored. 20 different ϵ values are set for the attack purpose. Finally, 80180 attacked network topologies are generated before and after FP for each. In sum, 178,860 network topologies are engaged in this project.

Table 4.1: Network Number

category	acronym	FP?	FGSM?	# Networks
category 1	obfp	×	×	10,000
category 2	oafp	✓	×	8,500
category 3	abfp	×	✓	80,180
category 4	aafp	✓	✓	80,180

4.2 Two Representative Examples

As can be observed in Section 2.3.3, by adding some tiny noises in the inputs, the outputs of neural network models are largely different from what they are expected to be. In this project, what is expected is that network features are changed slightly by the FGSM, but there are huge differences in the delay bounds predicted by the GNN model. Equations (4.1) and (4.2) expatiate on this expectation.

$$\frac{|DelayBound_{abfp} - DelayBound_{obfp}|}{DelayBound_{obfp}} \rightarrow small \tag{4.1}$$

$$\frac{|DelayBound_{aafp} - DelayBound_{abfp}|}{DelayBound_{abfp}} \rightarrow large \tag{4.2}$$

This expectation also explains why the large difference in delay bound is caused by the FGSM attack instead of the GNN model accuracy. For all the 8,500 original network topologies (the first and second category in Table 4.1) that can be prolonged, there are no such large differences between the delay bounds before and after the GNN prediction. However, after the FGSM, several large differences for delay bound before and after the GNN prediction occur, which means that these large delay bounds difference only happen after the FGSM attack. In other words, this phenomenon is caused by the FGSM attack.

Some good results are achieved among the 178,860 network topologies. Two representative examples are demonstrated below in Figure 4.1.

shown in the medium number

topo id	eps	delay bound obfp	delay bound oafp	delay bound abfp	delay bound aafp	server rate changes %	server latency chagnes %	flow rate changes %	flow burst changes %
6549	0.004	4659.031387	4651.304693	3840.871363	11106.02527	13.33332707	3.333336115	4.74975E-06	7.999999821
6369	0.002	7061.983539	7021.565914	8627.424152	18339.27296	6.666659315	2.499993518	2.666672319	3.999999166

Figure 4.1: Two Representative Examples

Based on the expectations of Equations (4.1) and (4.2), the network features are changed just by a bit after the FGSM. Considering that there are multiple servers and flows in one network topology, the network feature differences are shown in the medium number. The delay bounds with the attack before the GNN prediction are also changed by a bit with 17% and 22% respectively by using the Equations (4.1) (The calculation process is shown in Equations (4.3) and (4.4)). However, the delay bounds influenced

by the GNN prediction due to the attack have 189% and 112% differences, the calculation process of which is shown in Equations (4.5) and (4.6) based on Equation (4.2). These representative results are noteworthy because the network features are only modified a little, but the GNN is fully fooled by the adversarial attack, similar to the panda and digit recognition examples in the field of computer vision mentioned in Subsection 2.3.3. This is catastrophic in the design of the network system, especially in the TSN.

$$\frac{|DelayBound_{abfp} - DelayBound_{obfp}|}{DelayBound_{obfp}} = \frac{|3840.871363 - 4659.031387|}{4659.031387} = 17.56\% \tag{4.3}$$

$$\frac{|DelayBound_{abfp} - DelayBound_{obfp}|}{DelayBound_{obfp}} = \frac{|8627.424152 - 7061.983539|}{7061.983539} = 22.17\% \tag{4.4}$$

$$\frac{|DelayBound_{aafp} - DelayBound_{abfp}|}{DelayBound_{abfp}} = \frac{|11106.02527 - 3840.871363|}{3840.871363} = 189.15\% \tag{4.5}$$

$$\frac{|DelayBound_{aafp} - DelayBound_{abfp}|}{DelayBound_{abfp}} = \frac{|18339.27296 - 8627.424152|}{8627.424152} = 112.57\% \tag{4.6}$$

4.3 GNN Model Accuracy and Tightened Networks Ratio

GNN accuracy can directly reflects how the GNN model performs regards to the FGSM process. Demonstrated in Figure 4.2, it is interesting to find that when ϵ equals 0, meaning that no FGSM attacks happen, the model accuracy drops to a very low level of 9.45% compared to 65% reproduction accuracy and 69.6% accuracy in the paper [27].

Based on our shallow understanding, it is mainly caused by two reasons. Firstly, the GNN model is trained on a smaller size of networks. According to the recent research [13] of Geyer, Scheffler, and Bondor in 2022, the same three researchers of the proposal of this GNN model, the model is only suitable for small sizes of networks. Even though the model mentioned in [13] is trained based on reinforcement learning, the size of the dataset is quite similar to the one they train the GNN model. Therefore, the current GNN model may not

work well when the size of the network becomes large.

Secondly, the accuracy is evaluated based on the network topology shape during the testing process. Specifically, if a network after FP looks exactly the same as the target stored in the dataset, then we match it as a correct prediction. Then the accuracy is the total number of correct predictions divided by the total number of topologies. It is easy to predict in the smaller size of networks but turns difficult in the larger size of networks due to larger numbers of cross flows inside. Moreover, the targets in our newly-created larger dataset are not accurate either. Recalling the reason to save time, once three tighter delay bounds are explored instead of using the exhaustive search, this network generation is marked as finished. Thus, it also exists the possibility that the GNN predict the real targets instead of the ones stored in the dataset. It also leads to the wrong count on the correct predictions.

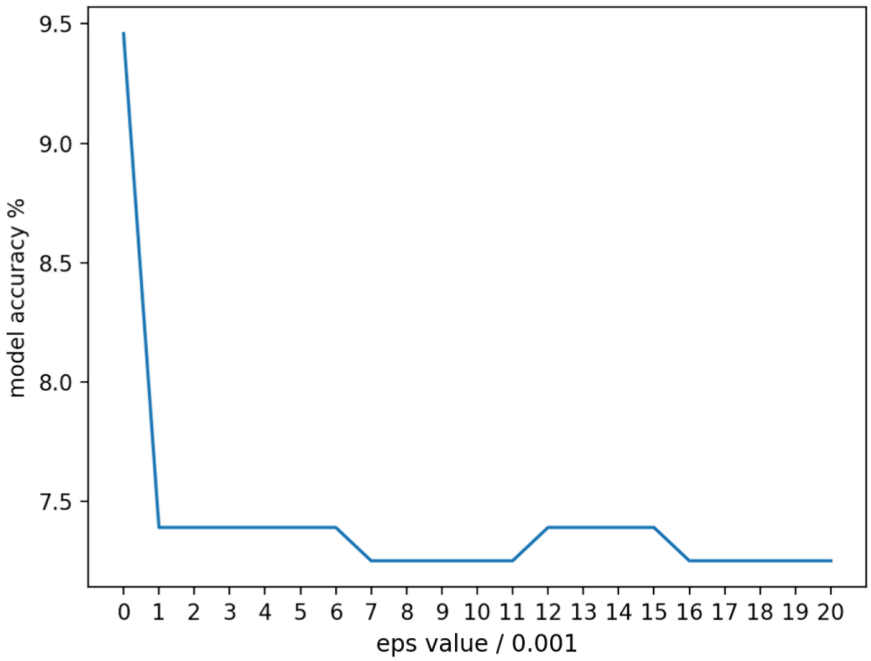


Figure 4.2: GNN Model Accuracy

According to our shallow understanding of why the GNN accuracy is low, the percentage of tightened delay bounds is more interesting and can reflect how FGSM influences the results plotted in Figure 4.3. It can be easily observed that with larger ϵ values, the ratio of tightened delay bounds decreases. Though the gap between 42% and 39% is small, the tendency is

what is expected.

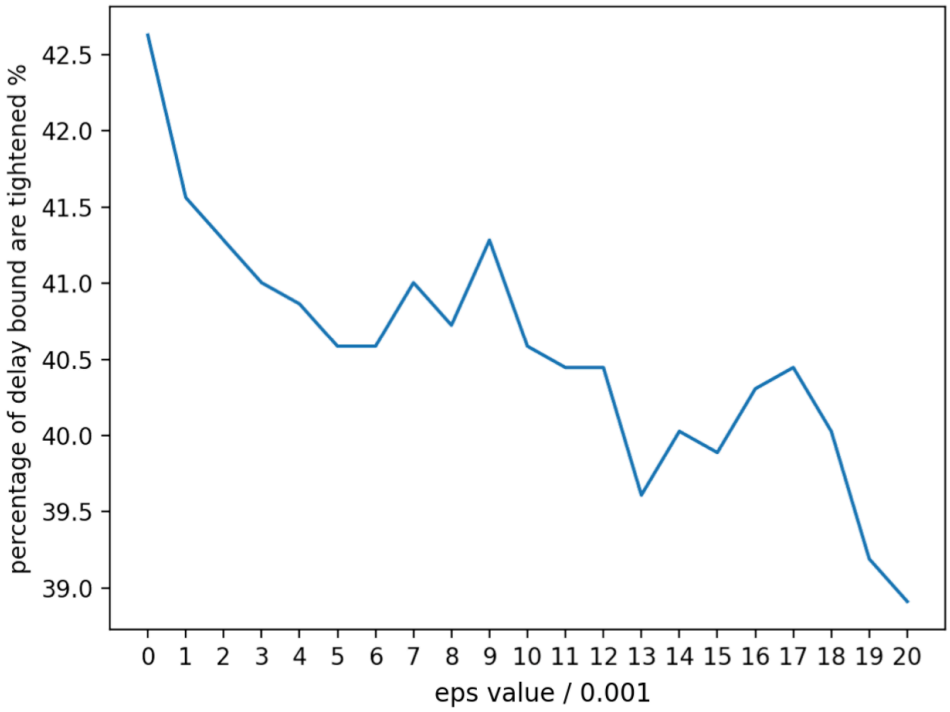
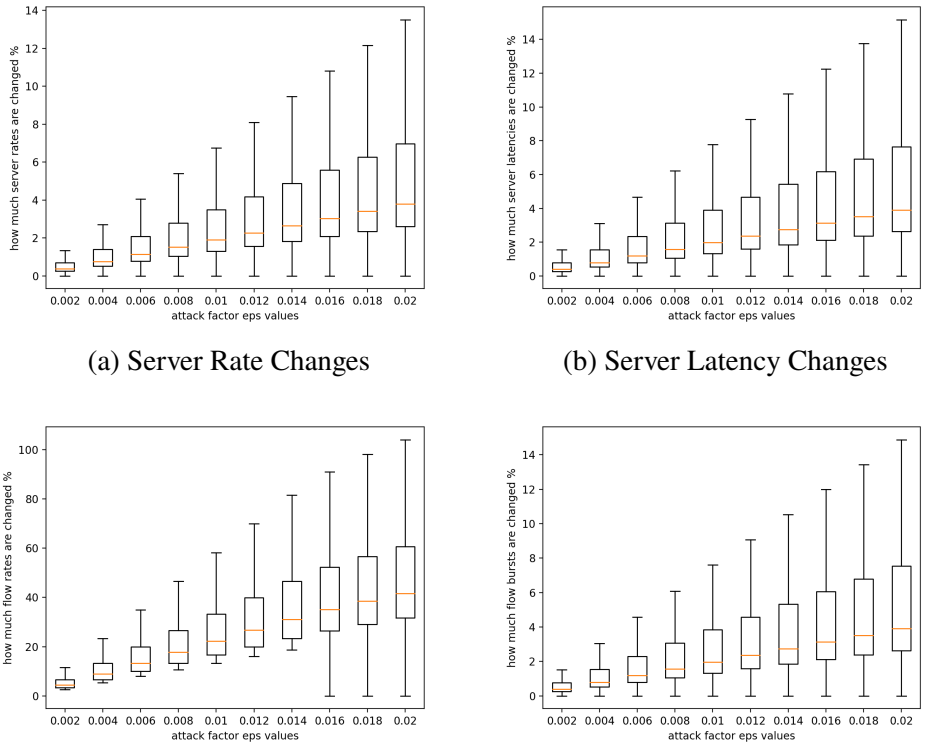


Figure 4.3: Tightened Network Ratio

4.4 Network Feature Differences

Figures 4.4 plots the four key network features changing differences after the FGSM attack. The results are calculated based on Equation (4.7). Also within the expectation that a larger ϵ value will lead to a larger difference. It is also interesting that the flow rate is extremely sensitive to the attack with more obvious changing differences, even though the modification of flow rate is set as $\epsilon/150$ during the FGSM. Unlike the value number in a pixel, there are more realistic meanings in the network topology. For example, the changes in the flow features can represent that in a certain period of time, some flows require more or fewer network resources.

$$\frac{|NetworkFeaturesAfterFGSM - NetworkFeaturesBeforeFGSM|}{NetworkFeaturesBeforeFGSM} \tag{4.7}$$



(a) Server Rate Changes

(b) Server Latency Changes

(c) Flow Rate Changes

(d) Flow Burst Changes

Figure 4.4: Network Features Changes

4.5 Successful Attack Definition

Finally, we are also interested in the attack outcomes under some assumptions given by Equations (4.8), (4.9), and (4.10). It is not difficult to understand Equations (4.8) and (4.9) that for the same network topology of four different categories, the delay bound is tightened by the GNN predictions before the attack, while is loosened by the GNN predictions after the attack. The third Equation (4.10) sets a tolerance rate of 25% to indicate the delay bounds are affected 25% by the FGSM before the GNN FP predictions. This 25% error

tolerance is acceptable in real life. To indicate how FGSM influences the GNN predictions, Equation (4.11) is defined as 'attack influence'. Apparently, with a larger number, it means that the result is affected more by the FGSM attack.

$$DelayBound_{oafp} < DelayBound_{obfp} \tag{4.8}$$

$$DelayBound_{aafp} > DelayBound_{abfp} \tag{4.9}$$

$$\frac{|DelayBound_{abfp} - DelayBound_{obfp}|}{DelayBound_{obfp}} < 25\% \tag{4.10}$$

$$\frac{|DelayBound_{aafp} - DelayBound_{abfp}|}{DelayBound_{abfp}} \tag{4.11}$$

Based on these filters, the result is achieved in Figure 4.5. Although the two representative examples mentioned in Section 4.2 are quite impressive, they are not shown in this Figure 4.5. This is because the number of such kind of representative examples is not large enough compared to the total number satisfying Equations (4.8), (4.9) and (4.10). Thus, for better visualization purposes, they are excluded in Figure 4.5. As can be observed from Figure 4.5, 'attack influence' of the majority number of network topologies locates between $(0\%, 10\%]$.

Recalling Table 3.5 in Section 3.6, a new dataset with multiple groups of network features is created for the FGSM purpose. Figure 4.6 demonstrates how the network sparsity will affect the 'attack' influence. Again, it is discussed in the second bullet point of the dataset creation constraints in Section 3.6 that given a network with $\#n$ servers, there are at most $\#n \cdot (n+1)/2$ flows without redundancy. According to this, n is chosen between 20 to 30, and the denominator 2 of $\#n \cdot (n+1)/2$ is changed from 4 to 9. This explains why the number of flows is shown in a range on the x-axis. Moreover, three representative ϵ values are picked in this figure shown in red, green and blue. Consistent with our conjectures, a denser network will lead to a larger 'attack influence'. It is also worth mentioning here that experiments on other network features are also conducted, but the outcomes are not good enough due to the smaller number of network topologies with specific network features, and the time limitation as well. Therefore, the results are not demonstrated as figures in this section, and they are left as the future work to be discussed in Chapter 5.

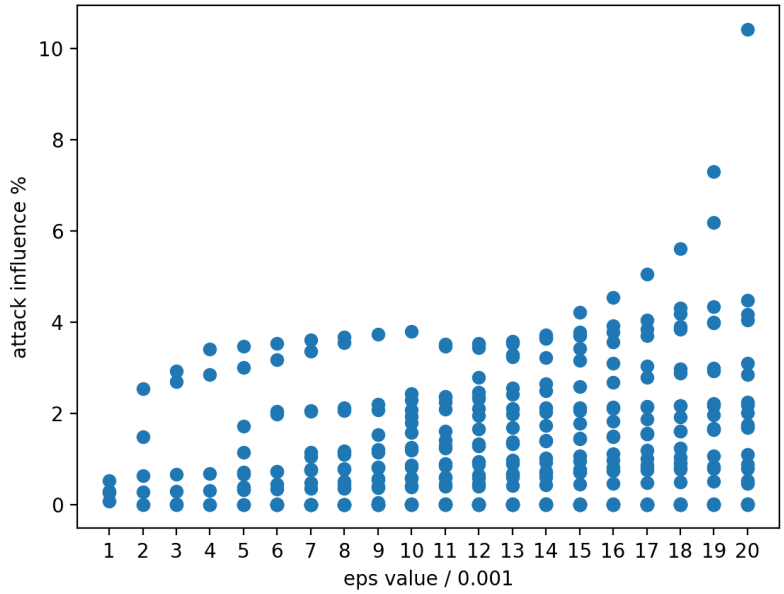


Figure 4.5: 'Attack Influence' Range

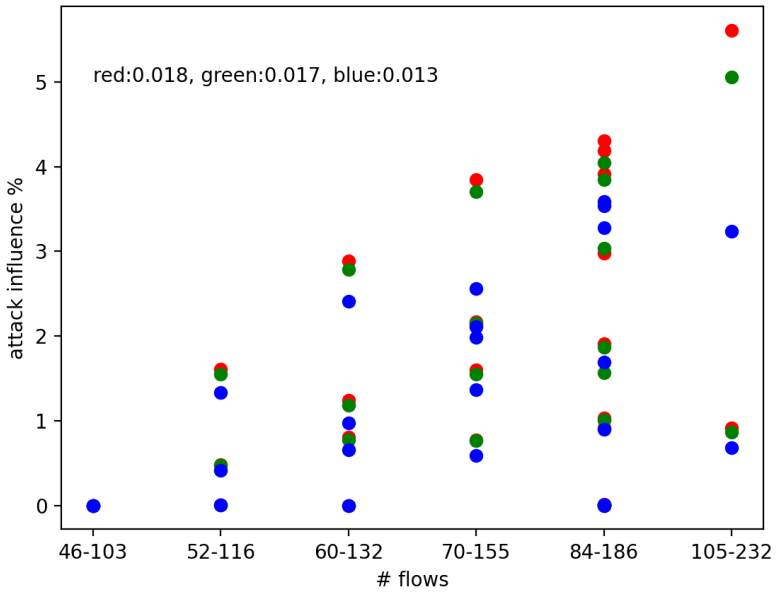


Figure 4.6: How the Network Sparsity Affects 'Attack Influence'

Chapter 5

Conclusions and Future Works

5.1 Conclusions

5.1.1 Tasks Done in This Project

First, PMOO is chosen as the NC method to calculate the delay bound in this project thanks to its remarkable performance in the trade of execution time and delay bound tightness. Thus, a GNN model based on this is reproduced, and an accuracy of 65% is achieved compared to 69.6% in the paper [27]. Second, an FOSS NetCal DNC is integrated into the network topology so that once a new network topology is given, the delay bound can be calculated automatically. Finally, a new dataset with larger numnbers of servers and flows is created for the adversarial attack purpose, and the consequent FGSM attack is implemented.

5.1.2 Attack Results Summary

FP is a newly-investigated method to tighten the delay bounds starting from 2017, and using GNN or more generally speaking, machine learning models to predict FP is still in an initial stage. The current machine learning models' accuracies are not high enough to guarantee an accurate FP prediction. Moreover, current machine learning models are more useful for the smaller size of networks with fewer servers and cross flows. This has been proved by recent researches, and in the project, when the network sizes are increased, the GNN accuracy drastically decreases from 65% to 9.45%.

As a consequence of the FGSM attack in this project, more than 17,000 network topologies have been analyzed where the network, except for the

flow rates, are modified 14% at most after the attack. As far as to our observations, the flow rates are very sensitive to the perturbation, even with a tiny ϵ value. Some promising attacked cases are discovered where with tiny modifications on the network features, the delay bounds are largely loosened due to the FGSM attack, which is critical to the TSN designs. After defining the successful attack, except for some evident observations, most delay bounds after the attack and after the GNN FP are affected up to 10% compared to the original ones. In terms of the sparsity of a network, the denser a network is, the larger 'attack influence' value will be observed.

5.2 Future Works

It is a kind of pity not to generate the dataset with even larger network sizes owing to the hardware restrictions on the EPFL clusters, though they are already powerful enough. Thus, if the industry could provide a dataset for the larger size of networks, a new machine learning model based on this can be trained, and a new benchmark can be conducted for the FGSM attack. We believe that the results in this way will be more rigorous.

Considering that using machine learning models worth more investigation in FP and the FGSM implemented in this project, one more further step could be exploring, i.e., a new model trained by GAN is worth investigating. This machine learning model works based on the adversarial attack, and what interesting is that Ian Goodfellow, the proposer of this model, is also the proposer of FGSM attack.

Opposite to the adversarial attack, a defense procedure could be investigated to prevent the attack, and increase the robustness of the model. This also reaches heated discussion in machine learning nowadays.

Continuing to Section 4.5, it is also worth investigating which network features will mainly influence the attack performance so that both the attack and the potential defense procedure will be more focused.

Finally, GNN is a powerful tool can be used in other fields of the computer network, e.g., source allocation and scheduling problems. It is worth expecting more fruitful results of the important role of machine learning models in the computer network.

References

[1] F. Geyer and G. Carle, "Network engineering for real-time networks: comparison of automotive and aeronautic industries approaches," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 106–112, 2016. doi: 10.1109/MCOM.2016.7402269 [Page 1.]

[2] R. W. L. Coutinho and A. Boukerche, "Guidelines for the design of vehicular cloud infrastructures for connected autonomous vehicles," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 6–11, 2019. doi: 10.1109/MWC.2019.1800539 [Page 1.]

[3] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001. [Pages 1, 2, 15, 21, and 29.]

[4] A. Bouillard, L. Jouhet, and E. Thierry, "Tight performance bounds in the worst-case analysis of feed-forward networks," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–9. [Page 3.]

[5] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch..." in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008. doi: 10.1109/INFOCOM.2008.228 pp. 1669–1677. [Page 3.]

[6] A. Bouillard, B. Gaujal, S. Lagrange, and É. Thierry, "Optimal routing for end-to-end guarantees using network calculus," *Performance Evaluation*, vol. 65, no. 11-12, pp. 883–906, 2008. [Page 3.]

[7] A. Scheffler and S. Bondorf, "Network calculus for bounding delays in feedforward networks of fifo queueing systems," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2021, pp. 149–167. [Pages 3 and 11.]

[8] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus: Design and evaluation of an accurate and fast analysis," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–34, 2017. [Page 4.]

[9] S. Bondorf, "Better bounds by worse assumptions — improving network calculus accuracy by adding pessimism to the network model," in *2017 IEEE International Conference on Communications (ICC)*, 2017. doi: 10.1109/ICC.2017.7996996 pp. 1–7. [Pages 5, 6, 23, and 24.]

[10] F. Geyer, A. Scheffler, and S. Bondorf, "Tightening network calculus delay bounds by predicting flow prolongations in the fifo analysis," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021. doi: 10.1109/RTAS52030.2021.00021 pp. 157–170. [Pages ix, 6, 7, and 8.]

[11] W.-K. Chen, *Applied graph theory*. Elsevier, 2012, vol. 13. [Page 7.]

[12] F. Geyer and S. Bondorf, "Deeptma: Predicting effective contention models for network calculus using graph neural networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019. doi: 10.1109/INFOCOM.2019.8737496 pp. 1009–1017. [Page 7.]

[13] F. Geyer, A. Scheffler, and S. Bondorf, "Network calculus with flow prolongation–a feedforward fifo analysis enabled by ml," *arXiv preprint arXiv:2202.03004*, 2022. [Pages 8 and 53.]

[14] Q. Duan, "Modeling and performance analysis for service function chaining in the sdn/nfv architecture," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018. doi: 10.1109/NETSOFT.2018.8460068 pp. 476–481. [Page 9.]

[15] A. Baybulatov, V. Promyslov, and E. P. Jharko, "On ics cybersecurity assessment with the help of delay calculation," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 971–975, 2021. [Page 9.]

[16] K. Wang, J. Li, J. Wu, and G. Li, "Qos-predicted energy efficient routing for information-centric smart grid: A network calculus approach," *IEEE Access*, vol. 6, pp. 52 867–52 876, 2018. doi: 10.1109/ACCESS.2018.2870929 [Page 9.]

[17] C. Xiao, J. Zeng, W. Ni, X. Su, R. P. Liu, T. Lv, and J. Wang, "Downlink mimo-noma for ultra-reliable low-latency communications,"

IEEE Journal on Selected Areas in Communications, vol. 37, no. 4, pp. 780–794, 2019. doi: 10.1109/JSAC.2019.2898785 [Page 9.]

[18] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 554–568, 2017. doi: 10.1109/TNSM.2017.2723090 [Page 9.]

[19] H. Huang, W. Miao, G. Min, J. Tian, and A. Alamri, "Nfv and blockchain enabled 5g for ultra-reliable and low-latency communications in industry: Architecture and performance evaluation," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5595–5604, 2021. doi: 10.1109/TII.2020.3036867 [Page 9.]

[20] N. Jacobs, S. Hossain-McKenzie, and A. Summers, "Modeling data flows with network calculus in cyber-physical systems: enabling feature analysis for anomaly detection applications," *Information*, vol. 12, no. 6, p. 255, 2021. [Page 9.]

[21] Z. Lin and W. Chen, "Energy-efficient pushing with content consumption constraints: A network calculus approach," *IEEE Transactions on Green Communications and Networking*, vol. 4, no. 1, pp. 301–314, 2020. doi: 10.1109/TGCN.2019.2942848 [Page 9.]

[22] T. Manikandan, R. Sukumaran, M. Christhuraj, and M. Saravanan, "Adopting stochastic network calculus as mathematical theory for performance analysis of underwater wireless communication networks," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020. doi: 10.1109/ICCMC48092.2020.ICCMC-000082 pp. 436–441. [Page 10.]

[23] X. Liu, C. Xu, and H. Yu, "Network calculus-based modeling of time sensitive networking shapers for industrial automation networks," in *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, 2019. doi: 10.1109/WCSP.2019.8927901 pp. 1–7. [Page 10.]

[24] Y. Li, Z. Ni, T. Zhao, T. Zhong, Y. Liu, L. Wu, and Y. Zhao, "Supply function game based energy management between electric vehicle charging stations and electricity distribution system considering

quality of service," *IEEE Transactions on Industry Applications*, vol. 56, no. 5, pp. 5932–5943, 2020. doi: 10.1109/TIA.2020.2988196 [Page 10.]

[25] A. E. Kalør, R. Guillaume, J. J. Nielsen, A. Mueller, and P. Popovski, "Network slicing in industry 4.0 applications: Abstraction methods and end-to-end analysis," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 12, pp. 5419–5427, 2018. doi: 10.1109/TII.2018.2839721 [Page 10.]

[26] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – a comprehensive tool for deterministic network calculus," in *Proc. of the International Conference on Performance Evaluation Methodologies and Tools*, ser. ValueTools '14, December 2014, pp. 44–49. [Online]. Available: https://dl.acm.org/citation.cfm?id=2747659 [Pages 10 and 32.]

[27] F. Geyer, A. Scheffler, and S. Bondorf, "Tightening network calculus delay bounds by predicting flow prolongations in the FIFO analysis," in *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021)*, May 2021. doi: 10.1109/RTAS52030.2021.00021 [Pages 11, 20, 24, 31, 33, 34, 38, 53, and 59.]

[28] M. Boyer, "Nc-maude: a rewriting tool to play with network calculus," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010, pp. 137–151. [Page 11.]

[29] H. Schioler, H. P. Schwefel, and M. B. Hansen, "Cync: a matlab/simulink toolbox for network calculus," in *2nd International ICST Conference on Performance Evaluation Methodologies and Tools*, 2010. [Page 11.]

[30] E. Heidinger, S. Burger, S. Schneele, A. Klein, and G. Carle, "Dimtool: A platform for determining worst case latencies in switched queuing networks," in *6th International ICST Conference on Performance Evaluation Methodologies and Tools*. IEEE, 2012, pp. 45–53. [Page 11.]

[31] A. Mifdaoui and H. Ayed, "Wopanets: a tool for worst case performance analysis of embedded networks," in *2010 15th IEEE International Workshop on Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD)*. IEEE, 2010, pp. 91–95. [Page 11.]

[32] M. Schmidt, S. Veith, M. Menth, and S. Kehrer, "Delaylyzer: A tool for analyzing delay bounds in industrial ethernet networks," in *International Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Springer, 2014, pp. 260–263. [Page 11.]

[33] B. Zhou, I. Howenstine, S. Limprapaipong, and L. Cheng, "A survey on network calculus tools for network infrastructure in real-time systems," *IEEE Access*, vol. 8, pp. 223 588–223 605, 2020. [Page 11.]

[34] B. Dagenais, "Py4j - a bridge between python and java." [Online]. Available: https://www.py4j.org/index.html [Page 11.]

[35] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014. [Pages ix, 12, 26, 27, and 29.]

[36] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. Ieee, 2017, pp. 39–57. [Page 12.]

[37] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017, pp. 15–26. [Page 12.]

[38] W. R. Abbott, "Network calculus," *IEEE Transactions on Reliability*, vol. 19, no. 4, pp. 196–198, 1970. [Page 15.]

[39] J. Grange, "Effects of the coupling between components of the reliability of an integrated function," *IEEE Transactions on Reliability*, vol. 19, no. 4, pp. 195–196, 1970. [Page 15.]

[40] R. L. Cruz, *A calculus for network delay and a note on topologies of interconnection networks*. University of Illinois at Urbana-Champaign, 1987. [Page 15.]

[41] J. Gross, "Feo3330 network calculus 10.0 credits." [Online]. Available: https://www.kth.se/student/kurser/kurs/FEO3330?l=en [Page 15.]

[42] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International*

Symposium on Circuits and Systems (ISCAS), vol. 4. IEEE, 2000, pp. 101–104. [Page 15.]

[43] J. Liebeherr *et al.*, "Duality of the max-plus and min-plus network calculus," *Foundations and Trends® in Networking*, vol. 11, no. 3-4, pp. 139–282, 2017. [Page 16.]

[44] J. S. Turner, "The challenge of multipoint communication," 1987. [Page 17.]

[45] M. Boyer, E. Le Corronc, and A. Bouillard, *Deterministic network calculus: From theory to practical implementation*. John Wiley & Sons, 2018. [Page 19.]

[46] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1995, pp. 231–242. [Page 19.]

[47] M. Boyer, G. Stea, and W. M. Sofack, "Deficit round robin with network calculus," in *6th International ICST Conference on Performance Evaluation Methodologies and Tools*. IEEE, 2012, pp. 138–147. [Page 19.]

[48] J. A. R. De Azua and M. Boyer, "Complete modelling of avb in network calculus framework," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, 2014, pp. 55–64. [Page 19.]

[49] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2. [Page 24.]

[50] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008. [Page 24.]

[51] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, vol. 6, no. 9, p. e33, 2021. [Pages 24 and 29.]

[52] P. Pradhyumna, G. Shreya *et al.*, "Graph neural network (gnn) in image and video understanding using deep learning for computer vision applications," in *2021 Second International Conference on Electronics*

and Sustainable Communication Systems (ICESC). IEEE, 2021, pp. 1183–1189. [Page 24.]

[53] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, "Every document owns its structure: Inductive text classification via graph neural networks," *arXiv preprint arXiv:2004.13826*, 2020. [Page 24.]

[54] K. Huang, C. Xiao, L. M. Glass, M. Zitnik, and J. Sun, "Skipgnn: predicting molecular interactions with skip-graph networks," *Scientific reports*, vol. 10, no. 1, pp. 1–16, 2020. [Page 24.]

[55] X. Tang, Y. Liu, N. Shah, X. Shi, P. Mitra, and S. Wang, "Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 2269–2279. [Page 24.]

[56] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Message passing neural networks," in *Machine learning meets quantum physics*. Springer, 2020, pp. 199–214. [Page 25.]

[57] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020. [Page 26.]

[58] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998. [Pages 27 and 41.]

[59] N. Inkawhich, "Adversarial example generation." [Online]. Available: https://pytorch.org/tutorials/beginner/fgsm_tutorial.html [Pages ix and 28.]

[60] EPFL, "Fidis." [Online]. Available: https://www.epfl.ch/research/facilities/scitas/hardware/fidis/ [Page 31.]

[61] E. SCITAS, "Izar," https://www.epfl.ch/research/facilities/scitas/hardware/izar/. [Page 31.]

[62] IBM, "Installing cplex optimization studio." [Online]. Available: https://www.ibm.com/docs/en/icos/20.1.0?topic=2010-installing-cplex-optimization-studio [Page 33.]

[63] G. Developers, “Protocol buffers.” [Online]. Available: https://developers.google.com/protocol-buffers [Page 33.]

[64] P. Contributors, “Torch.clamp.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.clamp.html [Page 40.]

TRITA-EECS-EX- 2022:00