THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *16/03/2023* **par :**

Victor PERRIER

LEO/GEO congestion control mechanism based on the contribution of artificial intelligence.

JURY

Toufik AHMED	Université de Bordeaux INP/ENSEIRB-MATMECA	Rapporteur
Cong-Duc PHAM	Université de Pau	Rapporteur
Aline C. VIANA	INRIA	Examinatrice
Jérôme LACAN	ISAE-SUPAERO	Examinateur
Jean-Yves TOURNERET	ENSEEIHT	Co-Directeur de thèse
Emmanuel LOCHIN	ENAC	Directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :
TéSA - ISAE-SUPAERO

Directeur(s) de Thèse :
Patrick GELARD (CNES), *Jean-Yves TOURNERET* et *Emmanuel LOCHIN*

Rapporteurs :
Toufik AHMED et *Cong-Duc PHAM*

Remerciements

Il sera compliqué de remercier tous ceux qui m'ont aidé, de près ou de loin, à l'élaboration de ma thèse, mais je vais essayer de n'oublier personne.

Je voudrais d'abord remercier mes deux directeurs de thèse Emmanuel LOCHIN et Jean-Yves TOURNERET pour leurs conseils, leur patience, leur bonne humeur, le positivisme d'Emmanuel LOCHIN et les corrections orthographiques de Jean-Yves TOURNERET. Je n'aurais peut-être pas persisté si je n'étais pas aussi bien accompagné.

Je tiens aussi à remercier tout le laboratoire TéSA, et principalement sa directrice Corinne MAILHES pour son soutient malgré mon manque de rigueur. Je veux aussi remercier Isabelle VASSEUR pour sa gentillesse et son aide au quotidien.

Je veux remercier mes encadrants de thèse au CNES pour leur aide apportée tout au long de la thèse, à Patrick GELARD, Nicolas KUHN et Emmanuel DUBOIS.

Je remercie également tous les membres de mon Jury de thèse.

Je tiens à remercier mes collègues et amis qui ont su bien m'accueillir à TéSA quand je suis arrivé en stage, et puis lors de ma thèse : Barbara, Lorenzo, Adrien, Raoul, Philippe, Antoine, Selma, Quentin, Oumaima, Patrice... Et puis, un peu moins nombreux, les nouveaux doctorants qui m'ont accompagné lors de la thèse : Corentin, Evelyne, Gaston, Kin, Youssra, Hamish et tous les nouveaux !

Je tiens à remercier ma famille, notamment mes parents Frédérique et Jean-Paul, ma sœur Laetitia et mon frère Camille. Je veux remercier aussi ma copine Julie pour sa patience et ses encouragements.

Et finalement, je remercie tous mes amis, et principalement mes anciens colocataires et voisins qui m'ont donné sans le savoir l'énergie pour continuer, notamment à travers la pandémie.

Contents

List of contribution

Here is the list of contributions produced by Victor Perrier

- Private Continual Release of Real-Valued Data Streams. Perrier, Victor and Asghar, Hassan Jameel and Kaafar, Dali. *26th Annual Network and Distributed System Security Symposium, NDSS 2016.*, 2019.
- Not All Attributes are Created Equal: dX-Private Mechanisms for Linear Queries. Kamalaruban, Parameswaran and Perrier, Victor and Asghar, Hassan Jameel and Kaafar, Mohamed Ali. *Proc. Priv. Enhancing Technol.*, 2020.
- Constrained Deep Reinforcement Learning for Smart Load Balancing. Houidi, Omar and Zeghlache, Djamal and Perrier, Victor and Quang, Pham Tran Anh and Huin, Nicolas and Leguay, Jérémie and Medagliani, Paolo. *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC).* 2022.
- How Attention Deep Learning Can Improve Copa Congestion Control Performance. Perrier, Victor and Lochin, Emmanuel and Tourneret, Jean-Yves and Kuhn, Nicolas and Gelard, Patrick. *The International Wireless Communications and Mobile Computing Conference (IWCMC).* 2022.
- Attention Networks for Time Series Regression and Application to Congestion Control. Perrier, Victor and Lochin, Emmanuel and Tourneret,

Jean-Yves and Gélard, Patrick. *The 4th International Workshop on Network Intelligence in conjunction with IFIP Networking.* 2022.

- Réseaux récurrents d'attention pour la régression de séries temporelles. PERRIER, Victor and LOCHIN, Emmanuel and TOURNERET, Jean-Yves and GÉLARD, Patrick. *XXVIIIème Colloque Francophone de Traitement du Signal et des Images-GRETSI'22.* 2022.

- Priority Switching Scheduler. FINZI, Anais and PERRIER, Victor and LOCHIN, Emmanuel and FRANCES Fabrice. Second review in *International Journal of Satellite Communications and Networking.*

Chapter 1

Résumé français

1.1 Introduction

Cette thèse se concentre sur le problème du calcul d'un taux d'envoi optimal en fonction de l'état du réseau. Historiquement, ce travail a été effectué par le mécanisme de contrôle de congestion TCP. La plupart des variantes de contrôle de congestion sont utilisées pour surveiller le retard et les pertes afin de calculer leur taux d'envoi. Dans cette thèse, nous cherchons à repenser ces métriques et comment elles peuvent être mieux utilisées pour calculer le taux d'envoi optimal. Dans ce contexte, nous proposons d'étudier l'utilisation d'un algorithme de Deep Learning qui semble particulièrement pertinent pour les tâches de contrôle de congestion.

Nous nous sommes également concentrés sur l'amélioration de cet algorithme de Deep Learning afin de faciliter son déploiement et son utilisation dans des scénarios réels.

1.2 Contexte de la thèse

Les algorithmes de contrôle de congestion sont essentiels pour la communication réseau : ils contrôlent essentiellement le rythme auquel les données

sont envoyées dans le réseau et sont donc grandement responsables de la stabilité du réseau. Ils empêchent les applications d'inonder et d'engorger le réseau de paquets.

Les trois principaux objectifs d'un algorithme de contrôle de la congestion sont :

1. utiliser le maximum de la capacité disponible ;
2. maintenir un délai faible et limiter les pertes ;
3. rester équitable avec tous les flux concurrents.

L'importance du contrôle de congestion et ses principales limitations seront développées dans la section 2.2.1. La thèse est principalement motivée par l'application de ces mécanismes de contrôle de congestion aux réseaux satellitaires.

Suite aux constats d'échec développés en introduction des méthodes traditionnelles de contrôle de congestion se basant sur les pertes, de nouveaux mécanismes ont vu le jour.

La plupart de ces nouveaux algorithmes essaient d'atteindre le point de contrôle optimal présenté dans la figure 3.3. Le détail de ces nouveaux mécanismes de contrôle de congestion est présenté dans le chapitre 3.

1.3 Structure de la thèse

La thèse s'articule principalement autour de trois chapitres.

Dans le chapitre 4, nous nous concentrons sur le problème principal de cette thèse : le Machine Learning peut-il aider les algorithmes de contrôle de congestion ? Pour y répondre, nous étudions l'utilisation des "patterns", puis nous introduisons et essayons de prédire de nouvelles métriques qui correspondent à des états cachés du réseau.

Le chapitre 5 se concentre sur le choix de l'algorithme d'apprentissage automatique que nous avons utilisé dans le chapitre 4, et sur les raisons pour lesquelles les alternatives les plus courantes ne fonctionnent pas. Nous comparons les résultats obtenus à l'aide de plusieurs méthodes d'apprentissage automatique.

Dans le chapitre 6, nous nous concentrons principalement sur la contribution présentée au GRETSI [34], et décrivons l'architecture proposée. Ensuite, nous discutons des perspectives et des travaux plus avancés, mais encore non publiés, réalisés sur ces architectures d'attention récurrente.

1.3.1 Chapitre 4: Comment améliorer le contrôle de congestion ?

Dans ce chapitre, nous nous concentrons sur le problème principal soulevé dans cette thèse : comment l'intelligence artificielle peut-elle aider un algorithme de contrôle de congestion ?

Pour répondre à cette question, nous étudions comment l'analyse conjointe des paquets émis et des paquets reçus permettrait à un algorithme d'intelligence artificielle afin d'obtenir des informations sur l'état du réseau.

Ce chapitre présente la notion de "patterns" et explique pourquoi l'utilisation de ces derniers pourrait améliorer les informations de retour et la performance des algorithmes de contrôle de congestion. Nous présentons également de nouvelles métriques qui pourraient aider à estimer l'état interne du réseau, aidant ainsi l'algorithme contrôle de congestion à prendre des décisions plus informées.

Le but de ce chapitre est d'estimer ces métriques internes au réseau avec un algorithme de Deep Learning avec la plus grande précision possible. Les résultats principaux sont détaillés dans la section 4.4.

Nous utilisons un environnement basé sur l'émulation pour valider nos résultats.

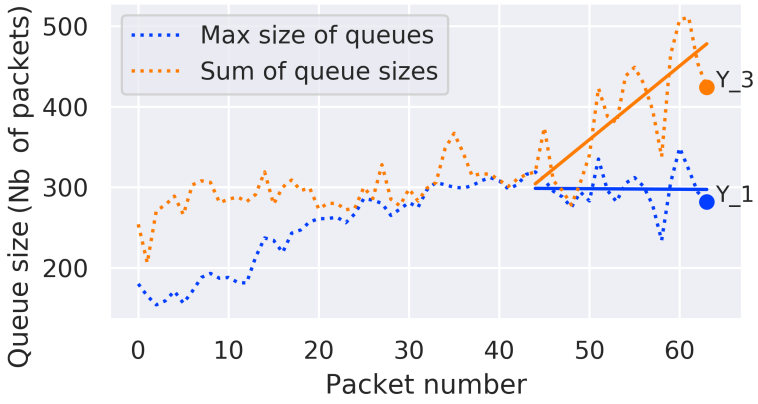


Figure 1.1: Exemple de métriques que l'on cherche à estimer : le taux de remplissage des files d'attentes au goulot d'étranglement (ligne pointillée bleue) et le taux de remplissage tout au long du chemin (ligne pointillée orange). On s'intéresse également à la vitesse d'évolution de ces métriques, représentée par la pente des lignes pleines.

1.3.2 Chapitre 5: Pourquoi les réseaux neuronaux d'Attention sont adaptés ?

Le problème de contrôle de congestion présenté dans la section 3.1.2.3 a été résolu à l'aide d'un estimateur défini par :

$$\min_{i\in[t-L_1,t]} \boldsymbol{x}_i - \min_{i\in[t-L_2,t]} \boldsymbol{x}_i, \tag{1.1}$$

avec $L_1 << L_2$ et où $\boldsymbol{x}_i$ est la série temporelle des RTT (i.e., les temps d'aller-retour des paquets dans le réseau).

L'objectif de ce chapitre est de montrer que les réseaux d'attention sont les réseaux les plus adaptés pour résoudre ce genre de tâches d'estimation faisant appel à des minima et/ou maxima sur des fenêtres glissantes. Les méthodes d'Attention sont comparées à des méthodes plus classiques de Deep Learning telles que les réseaux LSTM.

1.3.3 Chapitre 6: Réduire la complexité de l'Attention

Malgré leurs performances remarquables, les réseaux d'Attention sont difficiles à utiliser à cause de leur taille massive et des entraînements longs. En effet, pour appliquer un modèle d'Attention à une série temporelle, il faut considérer chaque pas temporel. Ainsi, à l'étape $t \in \{1, ..., L\}$, la complexité du calcul est $\mathcal{O}(t^2)$ à cause du produit matriciel. Pour traiter une série temporelle de longueur L, la complexité est donc de l'ordre de $\mathcal{O}(L^3)$. La réduction de ce temps de calcul est la motivation pour trouver une nouvelle architecture aussi performante et plus rapide, ce qui est l'objectif de ce chapitre.

L'architecture proposée afin de réduire la complexité des réseaux d'Attention est la suivante :

- 1) nous concaténons la matrice d'observation $\boldsymbol{X}$ avec la matrice de position $\boldsymbol{P}$.

$$\mathbf{X}_p = [\mathbf{X}, \mathbf{P}]$$

- 2) nous appliquons une couche de réseaux LSTM.

$$\mathbf{H_i} = \mathbf{LSTM}(\mathbf{x}_1, ...\mathbf{x}_i) \in \mathbb{R}^{J \times d}$$

où J est la taille du vecteur produit par le réseau LSTM, à choisir par l'utilisateur.

- 3) nous appliquons le réseau d'Attention qui à partir de la matrice $\boldsymbol{H}_i$ va déterminer quels sont les éléments passés les plus importants.

$$\mathbf{Y}_i = \mathbf{ATTENTION}(\mathbf{W}_q\mathbf{H}_i, \mathbf{W}_k\mathbf{X}_{1;i}, \mathbf{W}_v\mathbf{X}_{1;i})$$

L'idée de cette architecture est de ne pas utiliser l'auto-attention qui est trop coûteuse en temps de calcul, mais de générer grâce à un réseau LSTM le vecteur qui va nous donner les requêtes (J est alors le nombre de requêtes).

- 4) nous appliquons une couche non linéaire (fonction d'activation RELU), comme souvent pour l'apprentissage profond.

$$\mathbf{Z} = \mathbf{FeedForward}(\mathbf{Y})$$

Nous pouvons itérer ce processus en appliquant les étapes 2), 3) et 4) si nous désirons faire un réseau plus profond.

La suite de ce chapitre présente une proposition d'amélioration de l'architecture ci-dessus afin de réduire encore plus la complexité du mécanisme de l'Attention.

Chapter 2

General Introduction

In this introductory chapter, we present the main objective of this thesis, which investigates whether artificial intelligence (AI) can be used to solve congestion control problems over large bandwidth-delay product networks. We first introduce the challenge tackled and then give the context, mostly related to congestion control. The main objective of this chapter is to give the basis to assess why congestion control poorly performs over satellite networks, and further raises several congestion issues over newly deployed megaconstellations such as Starlink [1]

[1] See for instance: "Congested, contested... under-regulated and unplanned", by Stuart Eves, Surrey Satellite Technology Limited, Guildford, UK, Issue #3(29) 2021, ROOM Space Journal of Asgardia, available online `https://room.eu.com/article/congested-contested-under-regulated-and-unplanned`

Contents

2.1 Introduction

This thesis focuses on the problem to compute an optimal sending rate as a function of the network state. The challenge is to assess which kind of sensing is needed to achieve this goal. Historically, this job was done by the TCP congestion control mechanism. However, we do not restrict this thesis to TCP but to any kind of protocol that needs to adapt its sending rate whatever the service is (i.e., reliable with retransmissions or not). Most congestion control variants are used to monitor the delay and the losses to compute their sending rate (for instance, within a congestion window for TCP; a rate control for BBR [7] or certain multimedia applications such as Skype [11]). Both metrics (i.e., delay and loss event) have shown to be under-exploited [46] or too limited when considering only the loss ratio. In this thesis, we seek to rethink these metrics and how they can be better used to compute the optimal sending rate. In this context, we propose to investigate the use of a Deep Learning (DL) algorithm that seems particularly relevant for congestion control tasks. We also focused on improving this DL algorithm to ease its deployment and usage in real-life scenarios.

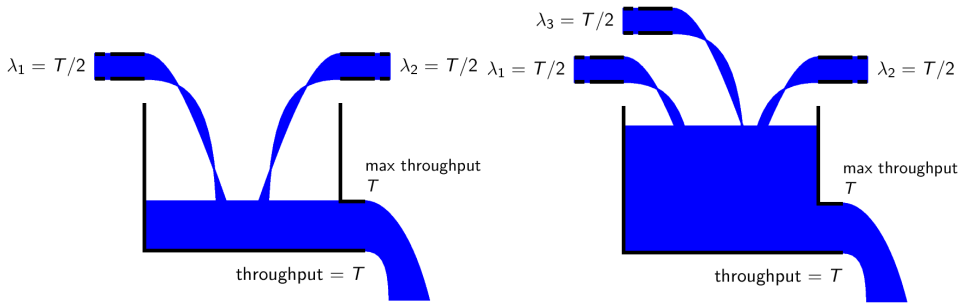
2.2 Context of this thesis

2.2.1 To send or not to send

Congestion Control (CC) algorithms are the backbone of network communication: they essentially control the rate at which data are sent into the network and thus are greatly responsible for the health of the network. They prevent applications from flooding and clogging the network with packets.

2.2.1.1 Why do we need congestion control ?

An easy way to explain how congestion control performs can be illustrated through an analogy with water flowing within pipes, as shown in 2.1.



(a) 2 flows (b) 3 flows

Figure 2.1: Water flows analogy: the pathway of the water within a city

If we consider the entering water flow is not strong enough, the output pipe will not be completely filled, thus limiting a part of the city from accessing a sufficient water flow for instance. On the other hand, if there are too many incoming water flows, the water reservoirs will overflow, thus wasting water.

The difficult part about congestion control is that the water flow needs to be controlled at the source, with only a little information available being: if water had been lost on the way and the time it took to fill the pipes. This is a typical control system with a delayed feedback loop (i.e., the time water needs to go through the network). This delay induces a trade-off between the objectives: if we increase the flow of water, it might cause leakages in the future, but if we do not attempt to reach the maximum capacity, the water network is under-utilized.

Back to IP networks, the water in the city represents the packet flows within the Internet network, pipes are the Internet links between two nodes (they can consist of Ethernet cables, optical cables, mobile networks, or even satellite links) and the water reservoirs are the equivalents of buffers in the networks. Buffers are queues in routers that temporally store packets if the output route is congested, and as reservoirs, if the queue is full the incoming packets will be dropped, thus causing packet losses.

Note that packet losses can have multiple origins (such as link errors, or mobile handovers), but this thesis will only focus on packet losses caused by congestion.

2.2.1.2 The importance of congestion control for quality of experience

Quality of Experience (QoE) is a metric of user satisfaction level that depends on both network Quality of Service (QoS) metrics and user activities. Depending on the context of use, the QoE can be interpreted as follows:

- for a downloading context, the lower the download time, the higher the QoE;
- for a streaming context, the QoE is a factor of multiple variables: the image quality, the time needed to start the video, video stalling, etc.;
- for a phone call, the QoE depends on the audio quality, the jitter, and the latency of the conversation;
- for video games, a good QoE relies on a low latency on the connection.

QoE is of course first limited by the Internet provider connection quality (both in assured capacity and delay). But a good congestion control algorithm is just as important, as it directly impacts the throughput and the Round Trip Time (RTT) the user is experiencing. In this thesis, all applications are considered, as all proposals developed in the next chapters can be used for any kind of application enabling a rate-control algorithm.

2.2.1.3 The three goals of congestion control

Summed up by these three items, the objectives of a congestion control algorithm are:

1. to use the maximum of the available capacity;

2. to keep delay low and limit losses;

3. to remain TCP-fair between all concurrent flows.

Once again, we remind that we are interested in congestion control, reliable or not, and not specifically in the TCP congestion control that is unbreakable from the reliable service. For instance, TFRC [16], an unreliable rate-based CC algorithm designed for multimedia, falls in the context of this thesis.

As previously introduced with the water analogy, there is a trade-off between these three goals. It seems impossible to maximize these three goals simultaneously with a decentralized system or without involving the network as in XCP [23]. We can only expect to reach the Pareto front [43] and set the trade-off to an arbitrary equilibrium. In the next paragraphs, we explain why these goals are antagonistic.

Use the maximum of the available capacity

In most cases, Internet services want to send data at the highest possible rate because QoE is often positively related to the throughput. As a result, one of the main goals of congestion control is to reach the maximum sending rate possible.

In Fig. 2.1, we represent the output capacity of T. Maximizing the throughput would mean that the sum of the N user's throughput reaches the limit: $\sum_{i=1}^{N} \lambda_i \geqslant T$.

Keeping delay low, and limiting the losses

At a given buffer, if the data packets arrive faster than the output link can handle, congestion occurs as illustrated in 2.1b. This means that the size of the buffer increases until overflow, resulting in packet losses. In our example 2.1b, the input rate is greater than the output rate, so the size of the queue increases.

This is a problem because, for most applications, a congestion event results in a decrease in the QoE. Obviously, while the buffer becomes con-

gested, new incoming packets observe an increase in their transit delay as several packets need to be served before exiting the queue. The RTT is thus increased and when the buffer is full, packets are dropped.

A naive solution would be to increase the size of the router queue to prevent packet loss. However, this increase leads to a phenomenon known as bufferbloat [15]. So queue should be kept small to prevent buffers from bloating [3].

Be fair between all concurrent flows

The last requirement for a good congestion control algorithm is to share the capacity between all flows fairly. A given flow should not preempt the whole available capacity opportunistically. Note that fairness is usually defined between flows, and not between applications of even users.

2.2.1.4 On the failure of congestion control

Unfortunately and for 30 years, no solution that maximizes simultaneously these three previously presented goals have been found. This can be explained as follows:

- the first two goals are antagonistic, if we attempt to maximize the throughput, we mechanically increase the number of packets in queues and thus increase the delay. A trade-off is needed between these two goals, and it is usually chosen following the QoE metric and the link type (for example for a satellite link the delay is already long, so we try to minimize losses);

- this is a decentralized optimization problem: all CC algorithm has to independently determine the optimal user's throughput at the same time. There is no way of knowing the number of flows using the same link;

- the amount of feedback information available for the CC algorithm is

limited. We only know with a delay if a packet arrived, and the time needed to reach the destination;

- the topologies of the network are ever-changing and increasingly complex, new hybrid architectures are more common (mixing satellites, mobile, and optic fiber links);
- each buffer has a different and unknown (from the point of view of the CC algorithm) queue size, thus leading to different behaviors. For example, if the queue size is large, the CC algorithm will overshoot the estimation of the capacity and fill the queue, thus increasing delays. Moreover, a little queue size will often drop packets when CC algorithms are trying to sense the optimal throughput;
- some links (Wi-Fi or mobile networks) are prone to losses, therefore confusing the CC algorithm because losses are detected, but are not caused by congestion;
- there is still a large amount of other varying parameters: throughput, latency, queuing policy, etc.;
- the network condition can quickly change over time, as new flows are created or ended. It is a dynamic environment and the number of concurrent flows can dramatically change, inducing a huge swing in traffic.

2.2.2 Internet over satellite today

During the past 20 years, the need for satellite communication systems has continuously grown. SATCOM systems are now widely used for Internet access. Despite some proposals designed for GEO links but not deployable end-to-end such as TCP Hybla [6], SATCOM links often challenge CC algorithms mostly designed for terrestrial wired networks.

2.2.2.1 GEO

The first generation of satellites used for Internet access is on a geostationary orbit at 35,786 km from earth. At this distance, it takes a radio signal approximately 240ms to 280ms to travel back and forth (it depends, of course, on the antenna position on earth) [27]. For an Internet packet, if the GEO channel is used in both directions, and if we consider other sources of delays from network services, it can add up to an 800ms round trip time. This is much higher than the typical 5–40 ms most Internet user experience with cable and optical fiber networks. This long distance is also responsible for losses caused by the diminishing signal intensity.

Another issue with GEO is that their orbit is around the equator, and they have a hard covering higher latitudes.

2.2.2.2 LEO

Requirements for lower delays, propagation loss, and more considerable earth coverage have sparked the development of LEO satellite systems. They are now widely used for Internet access with satellites constellation such as Iridium (670 km orbits), Globalstar (1,420 km orbits), or even the new Starlink (550 km orbits). The typical RTT of these constellations is around 40-50ms. A user of these systems can experience RTT around 200-250ms if had over causes of delay and use the satellite link in both directions. Unlike geostationary satellites, LEO satellites do not stay in a fixed position over the equator to cover a wider part of the earth, especially at higher latitudes. However, as they are closer to the ground, they cover much smaller areas, so constellations need many more satellites to cover the same place. The tracking and the communication of the type of satellites are also more complex, as they move very fast and need to communicate with many earth stations (a Starlink satellite needs only 90 minutes to make the orbit around the earth).

2.2.2.3 Incompatibilities with TCP

As explained before, most TCP-based congestion control algorithms are not adapted for these networks.

Random Losses

The presence of random losses on satellite channels is not negligible. It depends on the constellation type, visibility, weather conditions, etc. In that case, the basic TCP answer of cutting the congestion window in half is clearly the wrong answer. Even a 1% loss rate may have a significant impact on TCP throughput [4]. This is an issue because it is nearly impossible to disambiguate the origin of the losses (random or caused by congestion).

High Bandwidth-Delay product (BDP) link

Satellite links are part of a more significant category of high BDP links. A network with a large BDP is commonly known as a long-fat network. A link is considered with a high BDP if it is significantly larger than 105 bits (12,500 bytes) [21].

Typically, tuned TCP mechanisms have a time performing well inside a long fat network. As previously mentioned, the optimal congestion window corresponds to the BDP of the link. However, to achieve that large congestion window, the sender needs to send a lot of information on the network and wait for the duration of an RTT to send more. However, if the RTT is high, TCP may take a long time to reach the optimal congestion window. To solve this issue, TCP needs carefully tuned parameters (such as congestion window threshold).

The result is mainly a significant throughput degradation.

2.2.2.4 Performance Enhancing Proxies (PEP)

As previously seen, a large RTT harms the TCP throughput, a solution is to split the end-to-end connection into multiple parts and use different parameters to transfer data across the other legs with Performance-Enhancing

Proxy (PEP) [5]. This system has numerous TCP connections, with smaller RTTs tuned explicitly for each part of the link. It uses standard TCP, and the user is unaware of the split happening.

However, using more secure transport protocols (e.g. QUIC [20]) makes it difficult to split flows, and it is now necessary to design an end-to-end satellite-compatible solution [28].

2.3 Research Problems

This thesis aims to understand and improve congestion control mechanisms in the context of satellite networks. CC is already a pretty complex problem to solve by itself. Still, the behavior of TCP and other CC algorithms is even more complicated if we operate on a topology with satellites because these links inherently have a high BDP. The existing literature already focused on satellite-adapted CC algorithms, so the idea we seek to explore follows a different approach, which is to use diverse Machine Learning and statistical algorithms to model, estimate, and understand the behavior of CC.

2.4 Contributions

During the progression of this thesis, multiple scientific contributions (already published or expected to be submitted) have been produced:

- some interesting results about the use of ”packet flows pattern” to probe the network are discussed in 4. It is shown that the modulation of the instantaneous rate of packets might lead to a better or worse estimation of the network state;
- a paper published at the 2022 IWCMC conference [35] shows that network internal information can be estimated with the help of Deep Learning algorithms. These new metrics are essential for a CC algo-

rithm, as they increase the information extracted from the network, allowing it to get closer to the three CC goals. This publication is explained in 4.

- a paper published at the 2022 IFIP conference [33] improves the previous article by employing a more efficient version of the ML algorithm. This publication is explained in section 6;

- a paper published at GRESTI 22 [34] focuses on the explanation of the machine learning algorithm used by the previous two articles. This improvement over the classic attention algorithm allows the estimator to run faster while still preserving the precision of such networks. This publication is detailed in section 6.

- finally, we discuss the possibility of improving the result in [34] by reducing, even more, the algorithm's complexity. Serious leads are discussed and proposed in 6.

2.5 Structure of the Thesis

- Chapter 3 introduces the background and the state-of-the-art works necessary to understand our work and motivation.

- Chapter 4 focuses on the main problem of this thesis: can ML help CC algorithms? To answer this question, we study the use of "patterns", then introduce and attempt to predict new metrics that correspond to hidden states of the network. We then discuss results presented in [35] and [33].

- Chapter 5 focuses on the choice of machine learning algorithm that we used in 4, and why the most common alternatives are not working. We compare the results using multiple ML methods.

- Chapter 6 is mainly dedicated to the contribution presented in [34], and describes the proposed architecture. The second part of this chapter discusses prospects and more advanced, but still not published work, done on these recurrent Attention architectures.

- Finally, chapter 7 concludes this thesis and introduces future work possibilities.

Chapter 3

Background and State of the art

As presented in the previous chapter, congestion control mixed with satellite networking is a difficult task to solve with standard algorithms. This sparked the creation of numerous new algorithms based on either new metrics, or using new approaches based on machine learning algorithms. This is the core of the thesis, and in this chapter, we present both the state of the art of these new mechanisms, and the Machine Learning tools that we worked with.

In the first part, we focus on congestion control. Before anything else, the concept of the optimal control point is presented. Then, we introduce and explain the current state of the art of congestion control algorithms. We aim at presenting these new CC algorithms because they are based on either the measure of new metrics, or the use of Machine Learning mechanisms, which both inspired our work.

In the second part, we explain the necessary background in Machine Learning algorithms to understand the algorithms used in chapters 4, 5 and 6.

Contents

3.1 Congestion Control

3.1.1 Background

3.1.1.1 TCP presentation

TCP is one of the main protocols used for the transport layer. However, we only focus here on the congestion control algorithm used in the TCP variant CUBIC as it remains the most deployed today [32]. CUBIC relies on packed feedback. It is a communication protocol between a client and server, and for each packet sent in either direction, the receiver sends an acknowledgment message (ACK) assuring that the packet arrived. This ACK provides a limited but essential information: the packet reached its destination, and we can derive from this message the time it took for the data to go through the network. A loss can be detected is either it takes too long to receive the ACK message in comparison to the other data packets, or if the ACK of the 3 following packets already arrived.

All observation made on CUBIC can be made on other variation of TCP that mainly relies on losses.

To control the sending rate of data packets, CUBIC relies on the concept of the congestion window. The congestion window corresponds to the maximum data size of "in-flight" packets at a given moment. An "in-flight" packet is a packet between the moment it has been sent and the moment the ACK packet is received. The bigger the congestion window, the more packets are traveling on the links to the receiver at the same time. It means that a higher congestion window translates to a larger sending rate. If there is no other flow, the optimal congestion window on a specific link corresponds to the Bandwidth-Delay Product (BDP) of that link. Indeed, the BDP is the product of a data link's capacity (in bits per second) and its round-trip delay time (in seconds). If the congestion is too large, the buffers along the link will be filled. If it is too small, the link is underused.

Slow-Start phase

At the beginning of a connection, the sender needs to estimate the optimal sending rate, by increasing the congestion window until it detects congestion losses. This is called the "slow start". At the beginning of this phase, the congestion window is very small but doubled at each RTT. It does so until losses are detected, or it reaches a threshold. In this phase, the congestion window increases exponentially.

Congestion Avoidance phase

When the "slow-start" ends, the Congestion Avoidance phase starts. During this phase, the congestion window also increases linearly until losses are detected. It uses linear increments to prevent overestimating the network capacity and flooding the network. Once the CUBIC algorithm detects losses, it will reduce the congestion window by half and starts the linear increase again. This is what we call Additive-Increase-Multiplicative-Decrease (AIMD). When a loss is detected, the lost packets are re-transmitted during the "fast recovery".

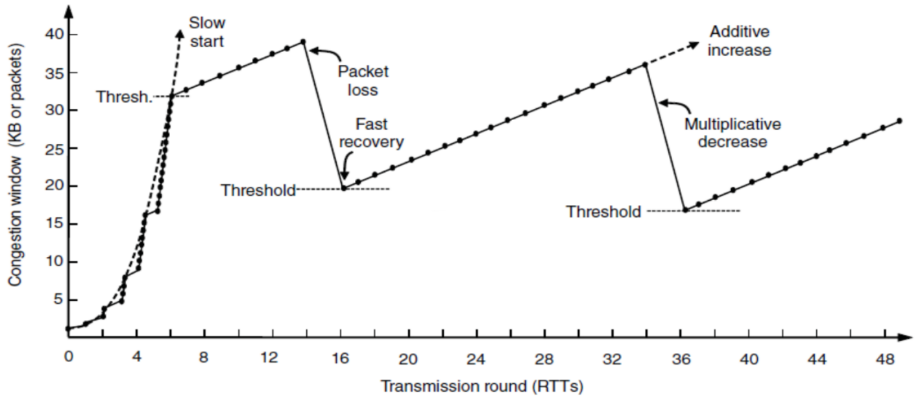


Figure 3.1: Slow-start and Congestion Avoidance phases (from [36]). We can observe the saw-tooth behavior of CUBIC in the congestion avoidance phase.

This AIMD mechanism also allows for each flow sharing the same bottleneck to reach a fair equilibrium, as shown in Fig. 3.2.

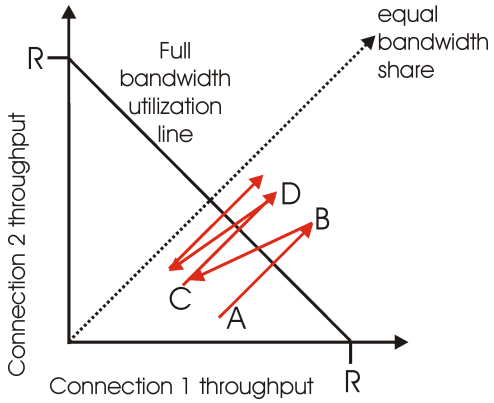


Figure 3.2: Two concurrent CUBIC flows will reach a fair equilibrium because of the AIMD mechanism (from [38]) The red line represents the evolution of the sum of the two flow capacities.

Let us explain Fig. 3.2. The goal should be to stabilize the red line around the intersection between the plain line and the dashed line (the capacity is shared fairly, and the flows use its entirety):

- at the beginning (point A) there are two flows (1 and 2) sharing the capacity, and flow 1 uses more capacity than flow 2;
- as the flows are in the congestion Avoidance phase, the bandwidth increase linearly until losses are detected (point B);
- once losses happen, the bandwidth is divided by two for each flow (point C);

Once this cycle is over, the network capacity is shared more fairly between the connections because as the congestion window of 1 was larger at point B, it was reduced more during the Multiplicative-Decrease process. This cycle then repeats itself until both flows reach the same capacity share.

This behavior, in which the sender get closer and closer to the fairness line, holds, no matter how many senders there are; senders with faster sending rates will always cut their rates by more than senders with slower rates.

3.1.1.2 TCP limitations

The variants of TCP based on losses detection for CC (as CUBIC and NewReno) have a lot of limitations.

All losses are considered congestion losses.

In the congestion avoidance phase, all detected losses have the same effect, it decreases the congestion window by half. This is a good response if the loss is caused by a larger-than-necessary congestion window, and the flow is flooding and clogging the network. However, losses can happen for other reasons: other flows are clogging the bottleneck, the size of the buffer is poorly configured and is too small, or the losses are random and caused by the nature of the link (Wi-Fi, Satellite, or Mobile links).

CUBIC behavior causes losses.

As it was presented before, CUBIC's inherent behavior to guess the optimal congestion window is to reach to clog the bottleneck. It goes directly against the second goal of CC algorithms of avoiding losses and keeping the delay low.

It has the same behavior every time.

As explained before, the diversity of network topologies and the use of radio-based channels prone to random losses make it so that a strictly ruled algorithm such as CUBIC may not have the most efficient answer and, thus, is not optimal.

This specific behavior is also a drawback because of the diversity of the application on the Internet. Indeed, the QoE of each user may depend on different criteria, and a universal behavior proposed by CUBIC cannot fit all needs.

There have been attempts to adapt TCP to some type of network or application, but the use of these algorithms remains very specific.

3.1.1.3 The Optimal Control Point

An important notion used in this thesis is the notion of the Optimal Control point: achieving full capacity while minimizing the queue load. Seeking this point is the root objective of BBR congestion control [7] later presented in this chapter. It is also a perfect general illustration of a CC objective, which is to operate as closely as possible to this point.

To better assess what is this optimal control point, let's consider a simple model where a fixed-sized bottleneck queue with a service μ bit/s, a max queue size q_m, and a current queue size or load q_l, is crossed by a single flow at λ bit/s. This flow might encounter three states:

1) if $\mu > \lambda$, the queue is always empty and packets are passed without delay;

2) if $\mu < \lambda$ and $q_l < q_m$, packets are then stored and the end-to-end delay increases;

3) finally, if $\mu < \lambda$ and $q_l = q_m$, arriving packets are dropped and severe congestion occurs.

The delay and delivery characteristics considering this bottleneck link are illustrated in Fig. 3.3. The problem is thus to operate as close as possible to this optimum operating point.

TCP Vegas was the first attempt to solve this problem based on a delay-based CC that estimates the RTT of a flow by tracking the time of a sent packet and its corresponding received Acknowledgment (ACK) packets. The objective was to interpret a raise of the RTT as a congestion signal, as illustrated by the state 2) above (i.e. $\mu < \lambda$ and $q_l < q_m$). However, TCP Vegas has poor performance in several scenarios [1], which has led to a hybrid approach combining delay and loss-based approaches.

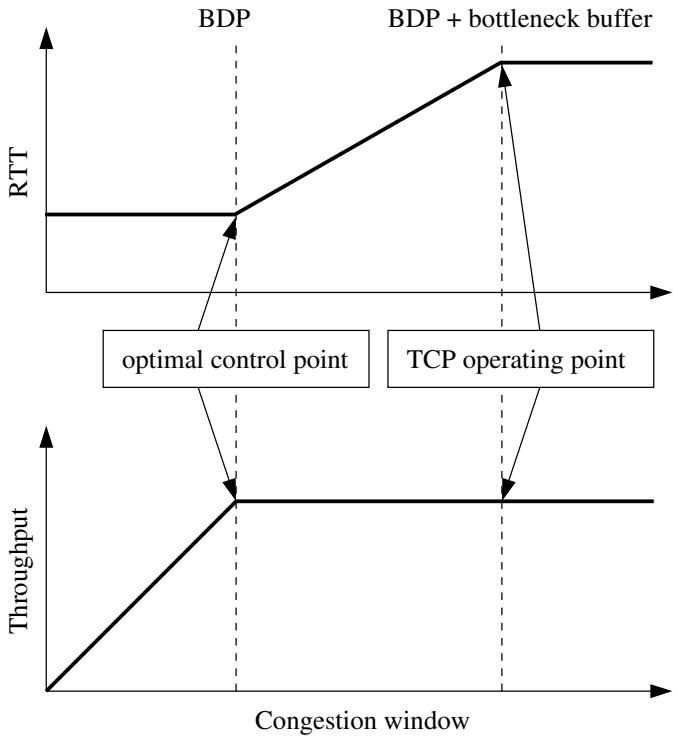


Figure 3.3: Optimal control point as shown in [7]. The aim is to use the maximum throughput while keeping the uses of the buffer as low as possible.

3.1.2 State of the art

As congestion control behavior is very to model, and the topologies and networks are becoming increasingly difficult to model, computer scientists have explored new types of congestion control algorithms that solve the issues in TCP and try to reach that optimal control point.

In this section, we are going to present the state-of-the-art algorithm and those who inspired our work.

3.1.2.1 REMY CC

The algorithm REMY [44], based on machine learning (ML), was a pioneer in this domain. Rather than manually adding rules to react to congestion signals, the algorithm learns on a specific topology the optimal reaction to

have. However, as explained in [46] and [45], the pace of convergence to achieve a consistent CC algorithm can take more than 24 hours of offline learning and remains specific to a given architecture. Indeed, if we try to apply the congestion control to another topology, with different capacities and delays, it might do worse than CUBIC. However, it is the algorithm that sparked the apparition of ML-based congestion control algorithms.

3.1.2.2 RAPID

Although it is not used in the congestion avoidance phase, we introduce here the RAPID [26] algorithm, because it set the stepping stones for using traffic shaping (or "patterns") to gain more information over the available capacity.

Indeed, the aim of the RAPID algorithm is to guess the available capacity faster than CUBIC does (at the slow start phase). It reaches its goal by using traffic shaping and sending packets with a logarithmic pattern, as illustrated in Fig. 3.4.

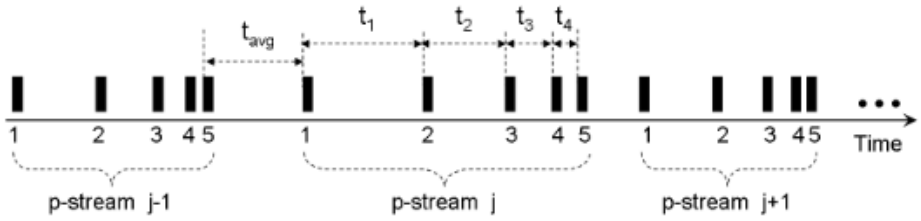


Figure 3.4: Pattern used in Rapid [26], it uses auto-congestion to have more information on the network bandwidth.

So instead of just doubling the congestion windows (and therefore the rate) at each RTT, it modulates the instantaneous packet rate and estimates whether it creates auto-congestion (congestion created by the flow itself). If it detects auto-congestion, it means that the instantaneous rate reached the available capacity (without the average sending rate of the pattern reaching that capacity).

3.1.2.3 Copa

We present Copa [2] as it inspired the use of metrics to guess the state of the network that we present in chapter 4.

Following the same way of thinking, the authors use the modulation of the instantaneous sending rate to guess some internal metric of the network.

The target sending rate is proportional to $\frac{1}{\delta d_q}$ (where δ is a parameter that set the trade-off between the throughput and the delay: a larger δ means that we favor low packet delay. d_q is an estimation of the queuing delay). If there are only Copa flows traversing the link, all the flows will synchronize to the same target sending rate because the metric used is common between all the flows. In that case, they will all follow the rate in Fig. 3.5. Indeed, when the rate increase, it fills the buffer, then d_q increases, causing the rate to decrease. This is why the rate follows a periodic pattern.

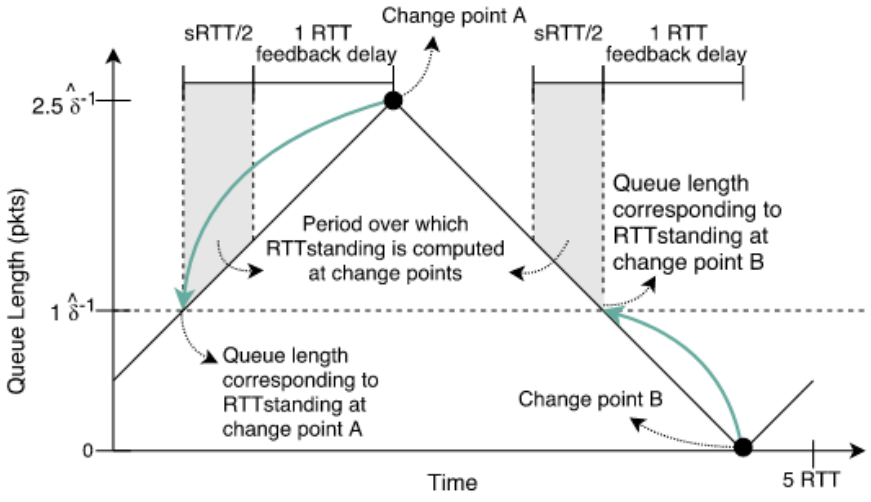


Figure 3.5: Copa mechanism following [2].

The estimation d_q is computed with the following estimator:

$$\min_{i \in [t-L_1, t]} \boldsymbol{x}_i - \min_{i \in [t-L_2, t]} \boldsymbol{x}_i, \tag{3.1}$$

with $L_1 << L_2$ and where $\boldsymbol{x}_i$ is the time series of RTTs, i.e., the round trip times of packets in the network. − Following the Copa article notations, we note $RTT_{\text{standing}} = \min_{i \in [t-L_1,t]} \boldsymbol{x}_i$ and $RTT_{\text{min}} = \min_{i \in [t-L_2,t]} \boldsymbol{x}_i$

The rationale behind the estimator is that flows are following the pattern in Fig. 3.5, meaning that periodically, all the buffers are empty. This allows $\min_{i \in [t-L_2,t]} \boldsymbol{x}_i$ to estimate the delay caused by the link alone without the congestion. $\min_{i \in [t-L_1,t]} \boldsymbol{x}_i$ corresponds to the actual delay over a much shorter period of time.

This estimator demonstrates good performance compared to currently used techniques such as CUBIC. However, it relies on the hypothesis that Copa is the sole CC algorithm being used on that link because the flow will synchronize the buffer will be emptied periodically. However, if another CC is used on the same link, that estimator may not be as accurate, and the target rate is not enough to compete with other CC algorithms.

One of the Copa drawbacks is that it struggles with competing CC algorithms on the same link, and has to decrease δ to remain competitive. Another issue is that if there are multiple Copa flows sharing the same link with different RTTs, their sending rate will not synchronize, and therefore the estimation of d_q will not be accurate.

3.1.2.4 BBR

BBR [7] is one of the most well-known alternatives to TCP. It is mainly used on YouTube. BBR is mainly based on the assumption that all congestion issues on one link are happening on the bottleneck on that link, and measuring the optimal rate for that bottleneck is enough. This paper is presented here because the idea of changing the instantaneous rate is later used in 4.1

During the congestion avoidance phase, instead of using the congestion window as TCP does, BBR directly aims at sending the optimal bandwidth. Let us note that sending rate B. As illustrated in Fig. 3.6 The algorithm

works with the following steps :

- periodically, BBR increases its sending rate to $1.25B$ (the gain in 3.6) during one cycle (of multiples RTTs);
- if around 1 RTT later it detects an increase in the rate of ACKs packets, it means that the available capacity was greater than the sending rate, and it then raises B up to the rate of ACKs packets;
- if the rate of ACKs message remains unchanged, and the RTT increases, it means that there is no more capacity available and congestion is happening on the link because of BBR. To address this issue, BBR does the following step;
- BBR decreases its sending rate to $0.75B$ during one cycle, to empty the buffers.

The upside of BBR is that the algorithm gathers meaningful statistics about the network (i.e. the available capacity at the bottleneck of the link). As for TCP, it is also not disturbed by random packet losses.

A serious drawback is that BBR is not TCP-fair with other CC algorithms and is overtaking the available bandwidth. BBR is also not sensitive to a loss signal, meaning that if shallow buffers are along the link, it won't decrease its sending rate despite huge losses. It can be problematic if we care about loss re-transmission.

3.1.2.5 PCC Vivace

PCC [13, 14]is a CC algorithm developed as a model-free approach. Indeed, it differs from Copa and BBR because the sending rate is computed by an online optimization algorithm, and it is not based on a human-made model of network behavior. It doesn't need to know how the network works under the hood, it just needs to know what metrics to optimize.

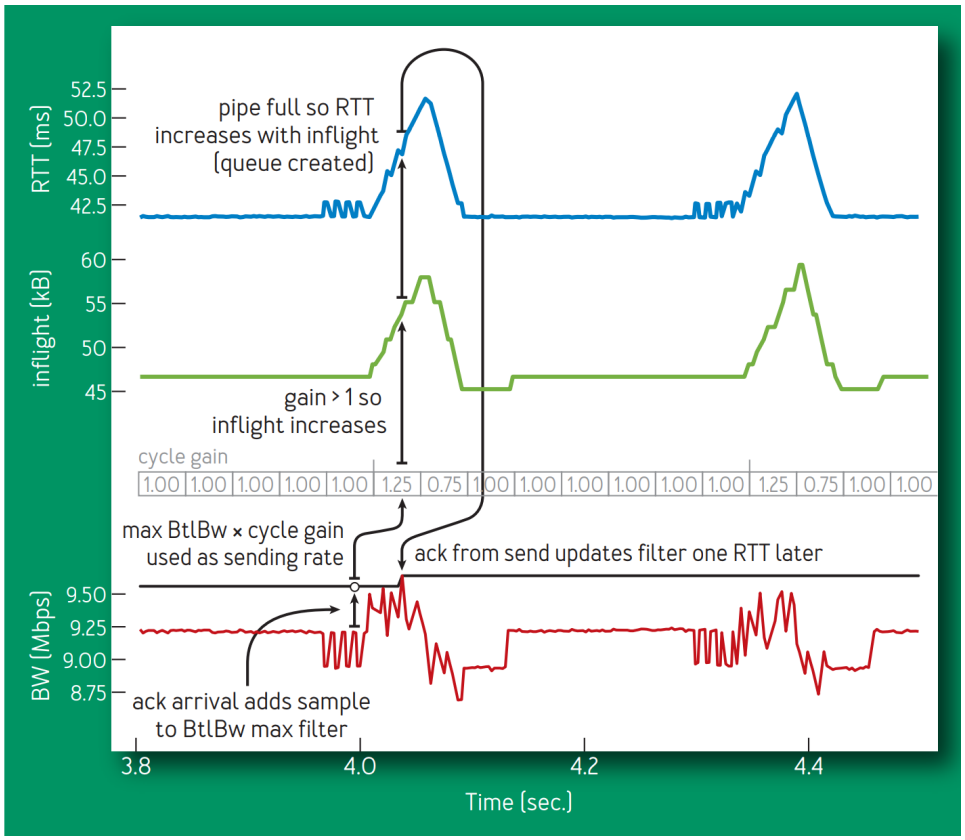


Figure 3.6: BBR mechanism following [7]

This is what we call the utility function. In the case of PCC Vivace (the last extension of PCC) the utility function is as follows:

$$U = x^t - bx\frac{d(RTT)}{dT} - cxL \tag{3.2}$$

where :

- x is the sending rate;
- RTT is the observed RTT is a short time window. thus $\frac{d(RTT)}{dT}$ is the observed RTT gradient;
- L is the loss proportion (between 0 and 1);
- $0 < t < 1$, $b \geqslant 0$ and $c > 0$ and parameters of this utility function.

Therefore, if PCC online learning algorithm tries to increase the utility function step by step, it will try to :

- maximize the sending rate x;
- decrease the gradient of the RTT, thus decreasing the RTT;
- Minimize losses.

Of course, it cannot fully optimize all these three goals, but it can reach a point on the Pareto front, where the trade-off is set with the three variables t, b, and c. The choice of these parameters is also crucial for an equilibrium when multiple PCC flows are competing.

To maximize the utility function, PCC uses online optimization. Time is divided into consecutive MIs (Monitor Intervals). At each MI, PCC tries to change the parameter to optimize (the sending rate). It tries a lower then a high sending rate and decides according to the result if the rate will increase or decrease.

Directly trying to minimize the RTT would lead to wrong results, thus the gradient of the RTT is used. Indeed, let us imagine there is only one flow over a single large buffer, and the sending rate is superior to the link capacity. The buffer is thus being filled and RTTs are increasing. PCC will test the sending rate during two MIs, first a higher and then a lower sending rate. As the buffer was still increasing during the two MIs, the RTT during the second MI is higher, thus telling that a higher sending rate is better. If we use the RTT gradient, this doesn't happen.

The benefits of using PCC are :

- it is very flexible. Indeed, depending on the application, PCC can try to change the parameters t, b, and c, and change the trade-off. For example, if an application is delay-sensitive, it may set the penalty b higher;

- there is no need of any prior assumption on the network, or any model of it.

However, this black-box aspect of the algorithm can be problematic if we need to understand the decision.

3.1.2.6 Summary of congestion control algorithms

Name	Method	Drawbacks
CUBIC	Losses detection	Sensitive to random losses, fill the buffers, handicapped when RTT is high
VEGAS	Losses and Delay	Sensitive to random losses, fill the buffers, handicapped when RTT is high
Remy CC	Offline Machine Learning	Topology specific, black-box
BBR	Estimate the optimal bandwidth	Not-fair, prone to losses with shallow buffers
Copa	Estimate the delay spend in queues	Not competitive enough, issues if different RTTs on the same link
PCC	Utility maximization	Black-box

Table 3.1: Summary of the congestion control algorithm

3.2 Machine Learning

In this section, we describe and introduce the different algorithms we used in the thesis and the necessary background. The task we try to achieve here is a regression task: we try to build an estimator of a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$.

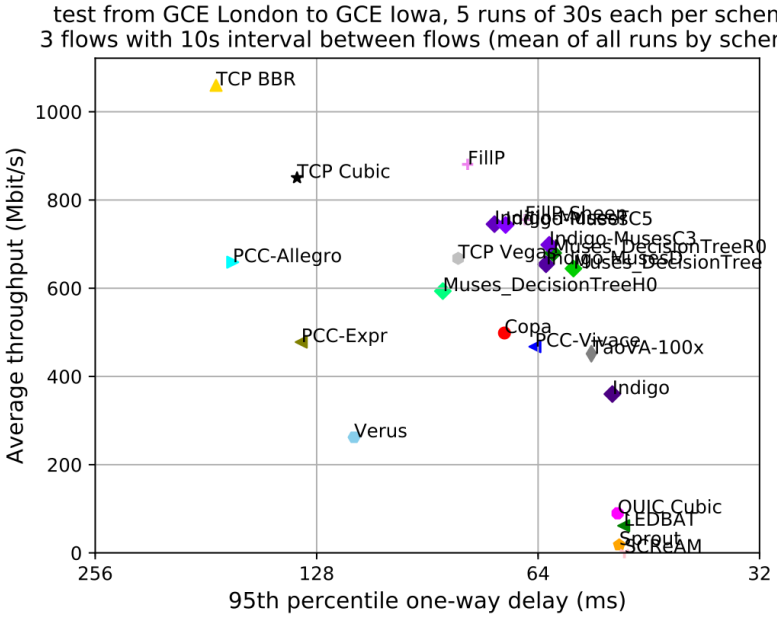


Figure 3.7: Comparison of performance of 17 CC algorithms made on the Pantheon [46] platform.

3.2.1 Background

3.2.1.1 Neural Networks (NN)

Neural networks are a type of model used in Machine Learning (ML) inspired by biological neural networks.

A neural network is based on the connection of nodes called artificial neurons. Each connection in a neural net allows a neuron to transmit a signal to the following set of neurons. An illustration of the architecture of a neural net can be found in Fig. 3.8. As shown in the figure, a neural net can have multiple layers and when it uses hidden layers it is referred to as a deep neural network leading to the DL framework.

A formal neuron is of the form $\boldsymbol{z} = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$, where:

- $\boldsymbol{x}$ is the input (the n-dimensional signal);
- $\boldsymbol{w}$ is a n-dimensional vector of weights that give more or less impor-

tance to the elements of $\boldsymbol{x}$;

- b is a scalar bias;
- σ is the neuron's activation function. We can use a multitude of functions, such as *sigmoid*, *tanh*, or *RELU*;
- $\boldsymbol{z}$ is the output (the signal transmitted to the following sets of neurons).

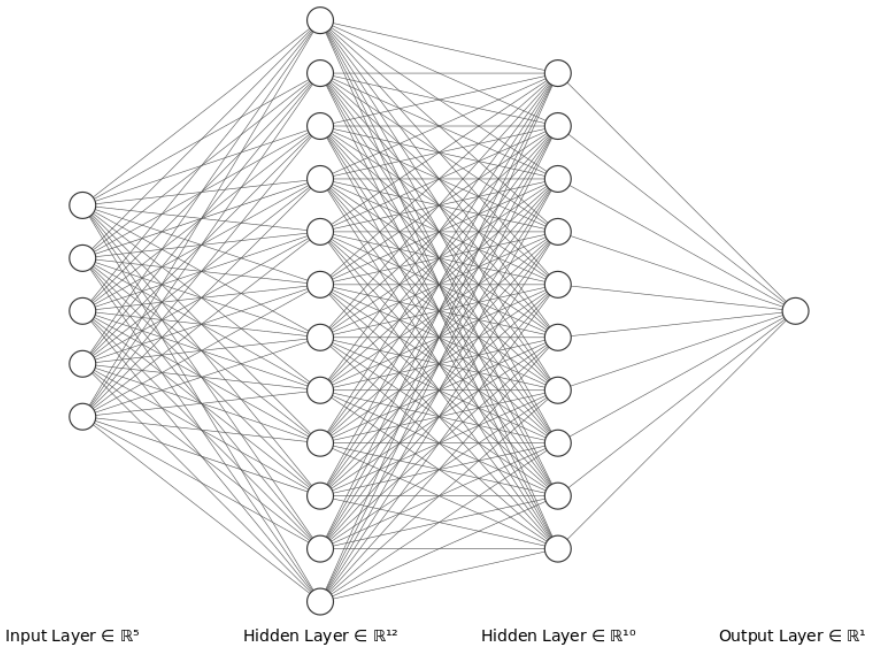


Figure 3.8: Illustration of the architecture of a neural network. The input layer is of dimension 5. Each set of neurons after the input layer (except the final layer) is called a hidden layer because the values of these neurons are not directly seen by the user.

The function created by the neural network can be used as an estimator of some function of interest. In this case, one needs to find the values of the parameters $\boldsymbol{w}$ and b for each neuron such that the neural network applied to a vector $\boldsymbol{x}$ is close enough to the function we try to approximate. It is possible to approximate almost any type of function, thanks to the Universal Approximation Theorem recalled below:

Universal Approximation Theorem.

Let

- $\phi(\cdot)$ be a non-constant, bounded, monotonically-increasing, and continuous function defined on $\mathbb{R}$ and taking its values in $\mathbb{R}^p$;
- $p \in \mathbb{N}^+$;
- $I_p = [0,1]^p$;
- $C(I_p)$ be the space of continuous functions on I_p.

Then, $\forall f \in C(I_p), \forall \epsilon > 0$, $\exists N \in \mathbb{N}, \alpha_i \in \mathbb{R}^p, \beta_i \in \mathbb{R}, \forall i = 1, ..., N$ such that $\forall \boldsymbol{x} \in I_p, |F(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon$ with $F(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_i \phi(\alpha_i^T \boldsymbol{x}), \forall \boldsymbol{x} \in I_p$.
It means that with enough neurons, and the right type of activation function, a neural network can create any estimation of any continuous function with a desired precision. Note that in most applications, it is important to have more than one layer in the neural network. Some definitions associated with neural networks are recalled in Table 1.2.

Definitions	
fully connected	A neural network is *fully connected* when all the neurons of a layer are connected to all of the neurons from the second layer
feed-forward	A *feed-forward* neural network is a NN where nodes do not form any cycle
back-propagation	The algorithm used to compute the gradient of the parameters of a neural network
gradient descent	The mechanism used for the optimization of a neural network

Table 3.2: Vocabulary for neural networks.

3.2.2 Supervised Learning

In all the machine learning tasks described in this thesis, we use a supervised learning regression setup. It is an ML paradigm where the aim is to learn a

function ($f\colon \mathbb{R}^n \longrightarrow \mathbb{R}$) that maps a feature vector $\boldsymbol{x}$ (the input) of dimension n to a real-world function (output). In the learning phase, both the input and output of the neural network have to be known and they are used to minimize an appropriate loss function. Generally, in a regression task, the Mean Square Error (MSE) defined in (3.3) is used as the cost function

$$\frac{1}{L}\sum_{i=1}^{L}\|f(\boldsymbol{x}_i)-\boldsymbol{y}_i\|^2, \tag{3.3}$$

where L is the number of input-output pairs belonging to the training set, $(\boldsymbol{x}_i, \boldsymbol{y}_i)$is the i^{th} pair of this set and f is the model learned by the NN. The optimization of the cost function (3.3) with respect to the model parameters requires the use of an optimization algorithm. In this work, we have used stochastic gradient descent algorithms such as the very popular ADAM algorithm [25].

3.2.3 State of the art

In this thesis, we will mainly use machine learning algorithms having time series as input. Note that it is difficult to use standard neural networks or other methods such as support vector regression in this context because these algorithms require the length of the input to be fixed. As a consequence, we will consider specific networks that are adapted to time series whose size can vary from one example to another. These networks are defined in the following sections.

3.2.3.1 RNN

A simple recurrent neural network [30] layer is very similar to a fully-connected feed-forward neural network layer since it has a set of weights mapping the previous layer to each neuron of the recurrent layer, a bias term for each neuron, and an activation function. However, there is also

a recurrent part of the neural network. The output of the neural network at the time step i is also fed to the neural network at the time step $i+1$. This output is what we call the state or the hidden vector. The main difference between RNN and classic neural networks is that RNN depends on the previous state for the current state computation. The architecture of these RNNs is displayed in Fig. 3.9 for the example of a Long Short-Term Memory (LSTM) network.

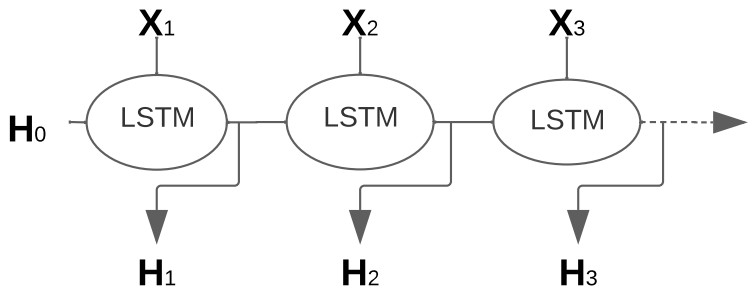


Figure 3.9: Architecture of a specific RNN referred to as LSTM.

To perform the supervised learning, we propose to train different models with the following cost function (very similar to (3.3) but adapted to take into account the time dependence) :

$$\frac{1}{L}\sum_{i=1}^{L}\|f(\boldsymbol{X}_{1:i})-\boldsymbol{y}_i\|^2, \tag{3.4}$$

where $\boldsymbol{y}_i$ is a vector that contains the metrics we want to estimate at time i, $\boldsymbol{X}_{1:i} \in \mathbb{R}^{i\times d}$ is a matrix containing the observations up to time i (in (3.1) this matrix is composed of the RTT, but other complementary observations could be considered as well), L is the length of the time series and f is the model defined by the NN to be trained.

3.2.4 LSTM and GRU networks

RNN networks allow deep learning architectures to be used for regression and classification tasks for time series. They can use the information contained in the past of a time series during the learning phase using back-propagation. However, if this information is far-away from the current time instant, it can take a long time to learn because of the gradient disappearance problem [18].

To overcome this issue, other methods have been introduced to improve the principle of RNNs. These methods are for instance based on more original architectures such as GRUs [8] and LSTM [17]. The architecture of these mechanisms is illustrated in Fig. 3.10. Note that this more complex architecture allows information from the far past to stay in the memory longer bypassing the gradient disappearance issues.

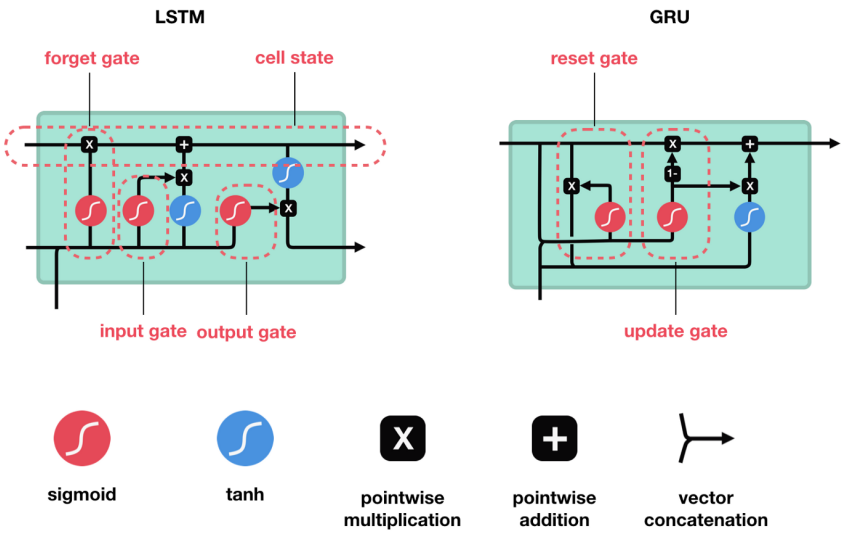


Figure 3.10: Illustration of the GRU and LSTM mechanisms. The top plain line represents the hidden vector information. The line coming from the bottom is the vector $\boldsymbol{x}$ at the current time step. LSTM networks use another hidden vector, called the "cell state". Image from [37]

3.2.4.1 Attention

More recently, a new architecture called Attention has shown interest in several applications [12]. First used in natural language processing tasks, this architecture makes it possible to obtain more precise prediction models, considering that there is sufficient computing power. For example, we can cite the work of [41], which shows that Attention is enough to solve translation or text interpretation tasks. Before explaining how vanilla Attention networks perform, notations are introduced in Fig. 1.10.

Notations	
Symbol	Signification
d	dimension of the Time Series
L	Length of the Time Series
M	Number of layers
$\mathbf{X} \in \mathbb{R}^{L \times d}$	Multivariate Time Series
$\mathbf{W}_k \in \mathbb{R}^{d \times d}$	
$\mathbf{W}_q \in \mathbb{R}^{d \times d}$	Matrices of Attention parameters
$\mathbf{W}_v \in \mathbb{R}^{d \times d}$	
$\mathbf{P} \in \mathbb{R}^{L \times d}$	Position encoding matrix concatenated with $\mathbf{X}$

Table 3.3: Notations for the Attention mechanism.

Attention is a time series processing mechanism initially built to perform language translation tasks. However, it can easily be extended to other domains, such as time series regression. To illustrate how Attention works, we use the running example displayed in Fig. 3.11. This example shows a sentence where the relationships between each word are represented either by plain or dashed lines, depending on whether there is a strong or weak relationship between the two terms. These relationships are defined thanks to a weight matrix. This weight matrix (where the sum of each row is 1) allows for assessing the relationships between elements of the time series.

The Attention mechanism itself is not sensitive to the relative position of the time series elements $\boldsymbol{X}_i \in \mathbb{R}^d, i = 1, ..., L$. To solve this issue, it uses

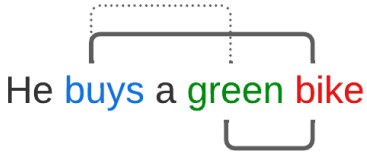


Figure 3.11: Example of the attention mechanism for a sentence. A dashed line corresponds to a weak connection between words, whereas a plain line is used for a strong connection.

a position matrix $\boldsymbol{P} = (p_{i,j})$ with $i, j \in \{1, ..., L\} \times \{1, ..., d\}$ defined as [41]:

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{100^{2j/d}}\right) & \text{if } j = 2n, n \in \mathbb{N} \\ \cos\left(\frac{i}{100^{2j/d}}\right) & \text{if } j = 2n+1, n \in \mathbb{N} \end{cases}$$

The position matrix allows defining the position of a vector in a time series, like a clock can define a time instant with three hands for seconds, minutes, and hours. Note that the use of sine and cosine functions for $p_{i,j}$ ensures that positions $p_{i,j}$ are in $]-1, 1[$. The attention layer can then be defined as follows:

$$\text{ATTENTION}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}, \tag{3.5}$$

with

$$\text{softmax}(\boldsymbol{X})_{i,j} = \frac{e^{\boldsymbol{X}_{i,j}}}{\sum_{k=0}^{d} e^{\boldsymbol{X}_{i,k}}},$$

and

$$\begin{cases} \boldsymbol{K} = \mathbf{W}_k \boldsymbol{X}, \\ \boldsymbol{Q} = \mathbf{W}_q \boldsymbol{X}, \\ \boldsymbol{V} = \boldsymbol{W}_v \boldsymbol{X}, \end{cases}$$

where $\mathbf{W}_k$, $\mathbf{W}_q$ and $\mathbf{W}_v$ are parameter matrices that are determined during the training phase. This example illustrates a particular case of Attention known as self-Attention (where $\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{V}$ are functions of $\boldsymbol{X}$). Note that the result of (3.5) is a linear combination of the elements of the matrix

$\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_t)^T$. The roles of the matrices $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ can be explained by the SQL language since they represent queries (q), keys (k) and values (v). Note also that the duo of matrices $\mathbf{W}_q$ and $\mathbf{W}_k$ allows the model to determine which elements of the past are useful for the regression task.

A multiple-head attention mechanism uses multiple-attention mechanisms in a parallel way on the data. The motivation for using this mechanism is that an attention operation after training is usually specialized in a specific task (e.g., computing a minimum, a mean, etc ...). Adding multiple attention heads allows the model to extend its representative capabilities.

The whole attention architecture has been presented in [41] and summarized in Fig. 3.12. In our case, we only use the encoder part of this process.

3.3 Satellite communications

This section presents the current state of satellite communication and explains why congestion is still a problem today.

3.4 The rise of mega-constellations

Over the last five years, the interest in satellite internet constellations raised due to the dropping cost of launching and increasing demand for internet access. This led to the creation of a few companies aiming at bringing satellite internet access to a broader audience:

- Starlink is a satellite constellation operated by SpaceX. It is composed of approximately 3 000 small LEO (550km) satellites at the moment, and 12 000 more are planned to be deployed;

- the Kuiper constellation, operated by Blue Origin, aims at sending

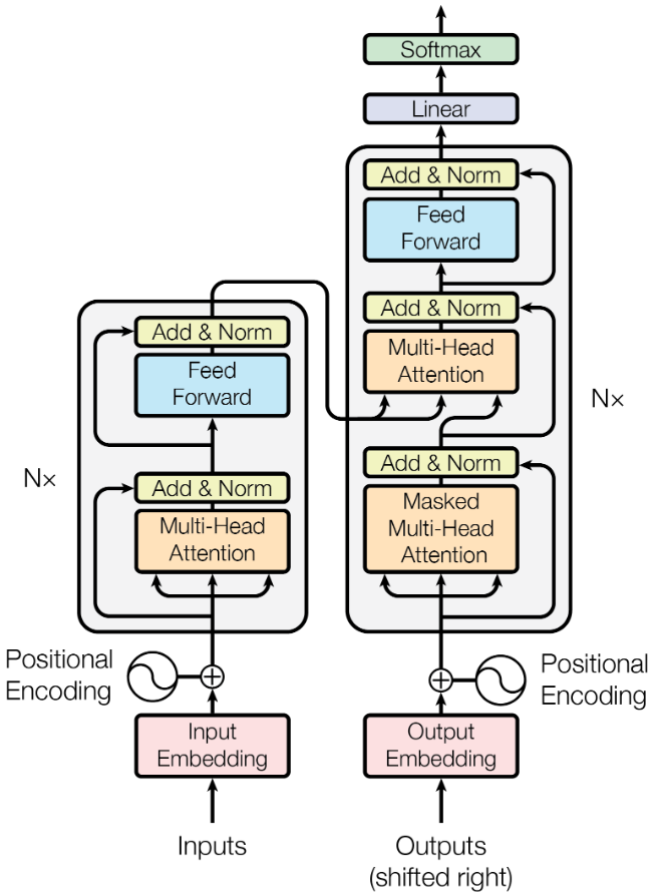


Figure 3.12: Transformer mechanism (the left part of this mechanism is used in this thesis).

3 236 satellites in LEO (600km);

- OneWeb satellite constellation is being deployed with currently 462 of the 648 scheduled LEO satellites.

This new type of constellation with many satellites is called a mega constellation.

This type of internet access is growing a lot these last years, as shown in Fig. 3.13, and it will continue as the coverage and the number of available satellites continues to increase.

The performance of such networks is relatively good, as shown in recent

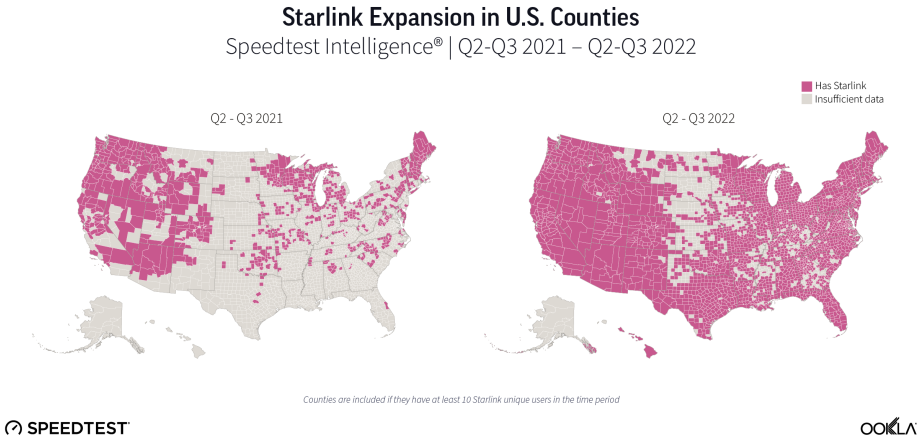


Figure 3.13: Expansion map of Starlink usage in the US between 2021 and 2022 produced by Ookla. We notice a huge increase in the number of users.

studies [31][22]. However, satellite links are still prone to random loss caused by handovers and link errors. Indeed, François Michel et al. [31] show that there is still up to 0.45% of lost packets even in a noncongested environment. Fig. 3.14 shows that CC algorithms are still inadequate for satellite networks, as they each perform worse in this environment.

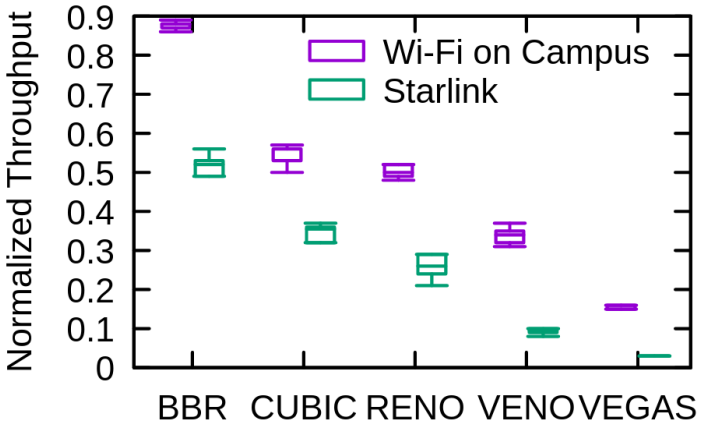


Figure 3.14: Comparison of CC algorithms performance for Starlink networks and standard network. Results are from Mohamed M. Kassem et al. article [22].

3.5 Congestion is still happening

However, as the number of users is growing, the Starlink network is experiencing the effect of congestion as illustrated in Fig. 3.15

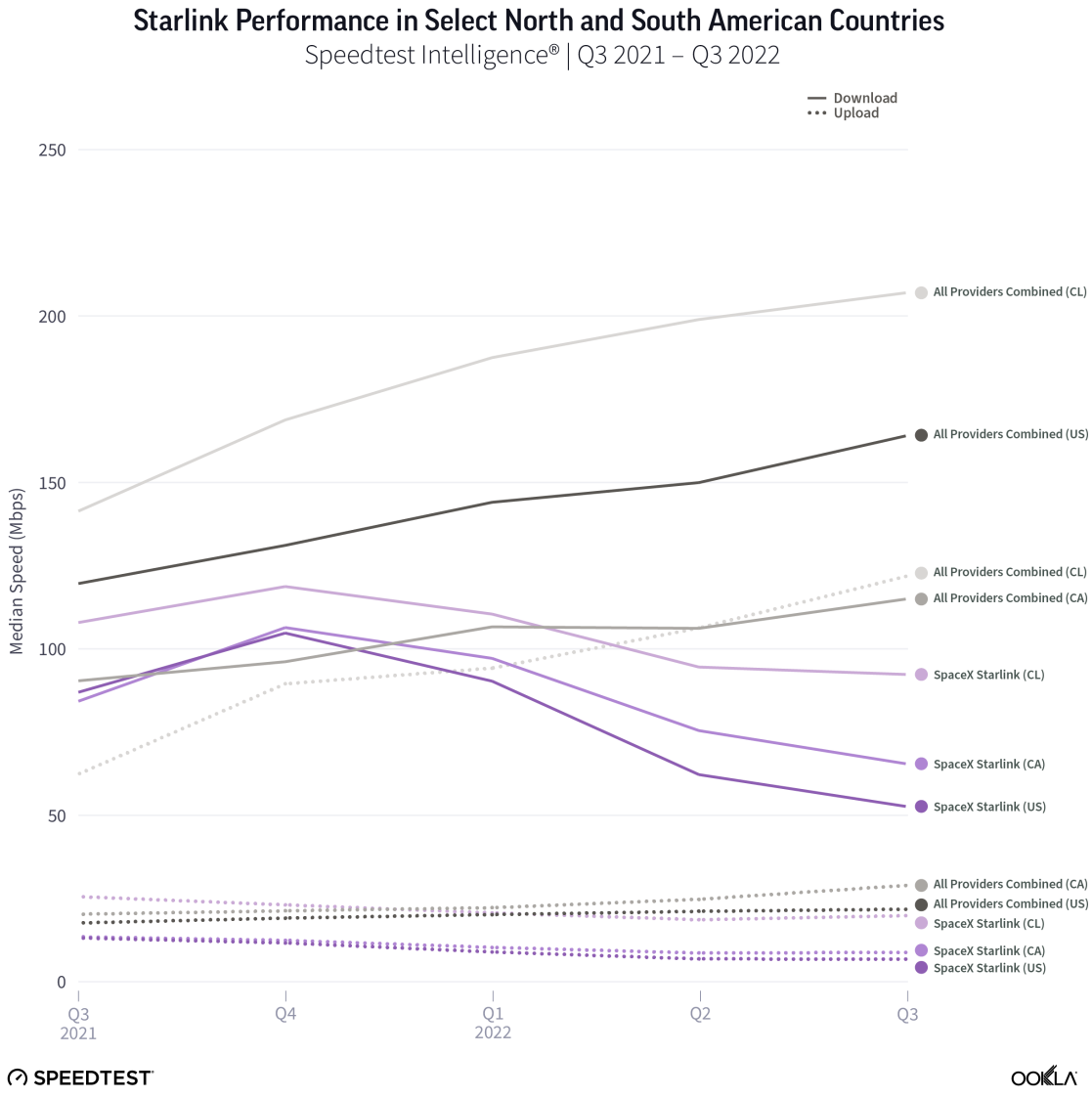


Figure 3.15: Median available throughput for users of Starlink and other internet providers during 2021 and 2022 produced by Ookla.

We notice that from the end of the year 2021 until now, the available

throughput for a Starlink user is decreasing. Indeed the increasing congestion [1] is a concern: it causes Starlink to limit the throughput of user's in congested areas once they reach a threshold during peak hours. This shows that CC algorithms still need to be improved over satellite links, and it is a common use case.

[1] https://mybroadband.co.za/news/broadband/467999-starlink-gets-fair-use-policy-will-start-throttling-in-congested-areas.html

Chapter 4

How to improve congestion Control

In this chapter, we focus on the main problem raised in this thesis: how artificial intelligence can help a congestion control algorithm?

To answer this question, we first investigate how it would be possible to analyze the feedback channel (i.e., acknowledgment packets) conjointly with the sending channel to feed an AI algorithm and obtain information on the network state.

This chapter introduces the notion of "traffic patterns" and why the use of these could enhance the feedback information to increase the performance of CC algorithms. This chapter also discusses the importance of the choice of these patterns, and how it influences the accuracy of the metrics (existing or new ones) we seek to estimate.

At last, the results are presented, and their validity is discussed.

Contents

4.1 Using patterns to sense the network

This idea is inspired by the state of the art with RAPID algorithm [26].

Basically, a pattern is a sequence of packets, sent at a constant bit rate or which follows some bursty characteristics. Each pattern can have different sporadicity. The idea is to inject sensing traffic or to monitor an existing one, conjointly with the feedback channel (i.e. acknowledgment packets). These two features will be then used as entries for an ML algorithm.

We study the impact of the choice of a pattern, and if some pattern shape leads to more useful or precise results.

4.1.1 Definition

First, let us define formally what a pattern of N packets is :

$$(X_i)_{i<N}$$

with

$$\sum X_i = N$$

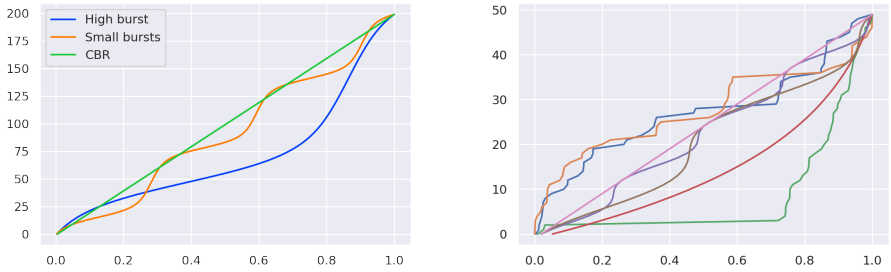
If we send the pattern at a rate $T(packet/s)$, we will then send a packet each $(X_{i(\mathrm{mod}N)}/T)$ seconds.

It means that we shape our instantaneous traffic rate to reach a higher or lower sending speed while keeping the average bandwidth. The time series $X_i, i < N$ corresponds to the inverse of the gain in the BBR algorithm.

4.1.2 What types of patterns are we looking for?

The aim here was to study if the choice of the pattern had any effect on the precision of some estimators. To measure this effect, we had to choose :

- a metric to measure. We decided to focus on the level of congestion on the bottleneck of a network;



(a) Regular and periodic patterns

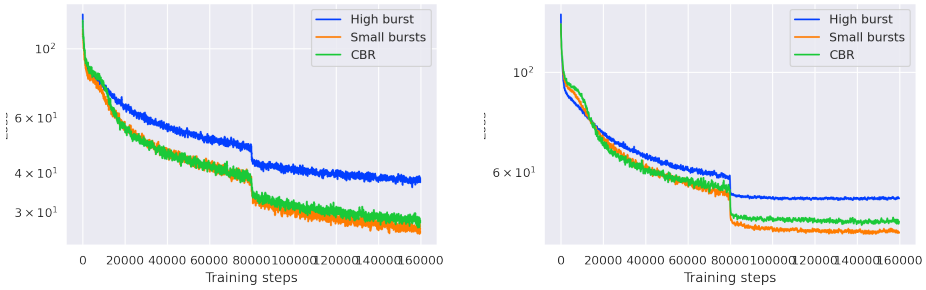
(b) Random patterns

Figure 4.1: Pattern representation. The y axis represents the number of packets being sent. The x axis represents the normalized time (1 is when the pattern ends). With this representation, if the slope is high, the pattern sends a burst of packets.

- an estimator. Our choice focused on SVR and simple networks. The aim was not to use them to get the best prediction possible, but to use them as universal estimators. This means that if the use of a different pattern improves the learning capabilities of the same neural network, the information gained by the use of this pattern is more correlated with the metric we seek to measure. To train the networks, we used the MSE loss (3.3). The features used are the time series of the RTT of each packet sent during the duration of a pattern;

- an experiment. As we were just trying to establish a proof of concept, we did not try to use an emulator yet. A simulator of basic queue and packet behavior was built with Python. More details on this simulator are available in 4.1.3;

- most important, we had to choose what kind of pattern to test. An illustration of this pattern is in Figure 4.1. At first, we did try to measure the effect on completely random patterns, but it was hard to make any conclusion out of the requirements. We then choose regular patterns, as in Figure 4.1a, with fewer parameters (the intensity and frequency of bursts) to see their effect.



(a) Training losses. (b) Test losses.

Figure 4.2: Losses in the training of patterns. The x axis is the training step, and the y axis is the value of the loss. The lower the loss the better the estimation. The step in the middle of the training corresponds to a change in the learning rate.

The conclusion of these experiments was that if the pattern was too "random". Not much information could be found by neural networks, a certain periodicity is needed to gain information. Figure 4.2 shows that with a Constant Bit Rate (CBR) we do have some information. If we change the pattern with a periodic burst, we first notice a decrease in the training and test losses after the end of the training. It means that without a CBR we can gain more information about the current state of the network. Then, if we keep increasing the rate of the burst of packets, we lose information. There is a trade-off between the size, intensity, and frequency of the bursts of packets and the moments where the sending rate is lower.

One way to explain this behavior is that with CBR, the algorithm can only see the current working point of the buffer. If we change the pattern and introduce bursts in the system, the queue will be filled and emptied further, allowing the algorithm to see the behavior of the network at the states near our current working point. However, if the pattern is too bursty and chaotic, the evolution of the RTT will be too erratic to obtain any information on the network state.

4.1.3 Simulation with Python

In the earlier stages of the thesis, we only sought a proof of concept; thus we decided not to implement our experiments on an emulator yet, but to build a Python simulator that mimics the behavior of queues in a network. The simulator works with a Dumbbell (see Fig. 4.3) Topology.

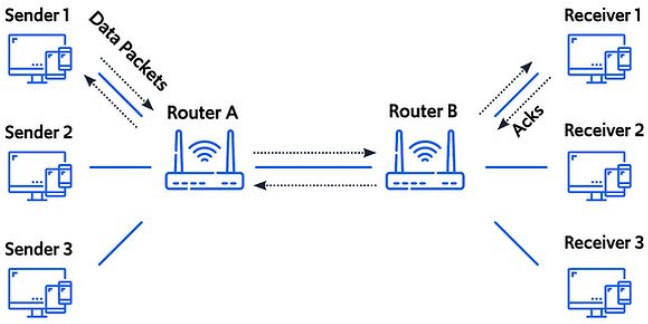


Figure 4.3: Dumbbell topology used for the tests. Image from [39]

And we generated traffic with the following rules :

- it's a simulation, we have perfect information;
- each flow is non-adaptive (the sending rate is fixed);
- each flow follows the same pattern;
- each flow has its own rate;
- the common throughput can be over or under the maximum capacity;
- the number of client/server changes;
- the delay changes.

4.2 Metrics of Interest

While trying to see if we could optimize the choice of a pattern in section 4.1, we were trying to figure out what metric could be important to estimate

for a CC algorithm. We call a metric a measure of the internal state of the network. While we investigated a lot of metrics in 4.1, we will only present the one that proved relevant for CC algorithms and the one we could actually estimate. Indeed, we attempted to see for example directly if it was possible to measure the capacity not used by flows, but without any success. We cannot estimate every parameter of a network.

All CC strategies detailed in 3 basically attempt to estimate the bottleneck queuing size or evolution. The optimal regime targeted by BBR, Copa, and other new algorithms is thus strictly linked to the queuing evolution of the bottleneck. As a matter of fact, we propose to investigate three new metrics of interest used jointly with an existing metric, and to assess their relevance and accuracy. The aim of these metrics is to assess the network congestion level, and we think that having access to them is an improvement for future CC strategies.

These new metrics are introduced below and illustrated in Fig. 4.4:

- Y_1 is the bottleneck buffer size. Note that the bottleneck buffer size (with respect the maximum size) corresponds to the maximum size of all the buffer sizes along the path. The heuristic behind that metric is that there is usually one bottleneck limiting the global performance;

- Y_2 is the slope of the bottleneck buffer size over the last N packets. This feature is computed using a regression over the available data (if the packet is lost, we cannot access the buffer load, because the packet never came through). This feature is important since knowing if a buffer is being filled or emptied is critical for a CC algorithm. This knowledge can help the optimal point to be stabilized, (3.3);

- Y_3 is the average time spent inside buffers over the last N packets. This feature is similar to Y_1, except that all the queues that are on the path are considered. Note that this metric is also used in the Copa

mechanism;

- Y_4 is the slope of the time spent inside buffers over the last N packets. This feature is the equivalent of Y_2 for the variable Y_3.

In an obvious manner, having an accurate estimation of these metrics will undoubtedly improve the performance of a congestion control algorithm.

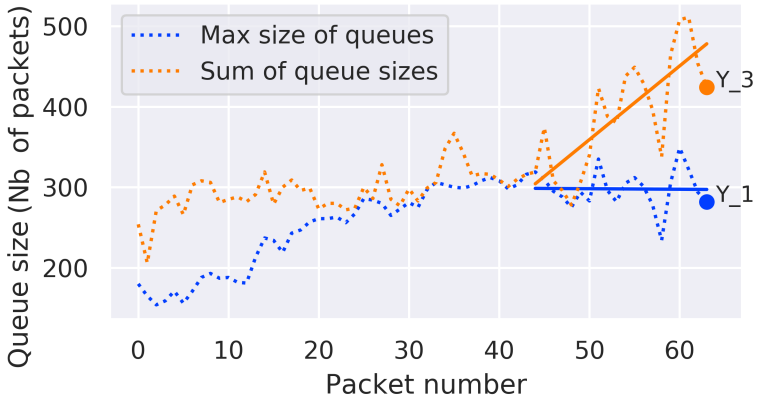


Figure 4.4: Visualization of the metrics. The two plots correspond to the sum of buffer sizes, and to the maximum of the buffer size along the path of a packet. Y_1 is the maximum buffer size for the last packet. Y_2 is the slope of the plain blue line. Y_3 is the sum of all buffer sizes during the last packet transmission. Y_4 is the slope of the plain orange line.

In the scenario presented in Fig. 4.4 we can see the evolution of the bottleneck size with the blue line, and the evolution of the total buffer size along the link with the orange line. The goal is to estimate the evolution of these lines by estimating the last value, and the slope.

4.3 Experimental setup and scenario

As the simulation results were promising, but only worked as a proof of concept (the simulator was internally developed), we decided to use an emulator tool: Mininet. The experimentation was conducted using the parking lot scenario depicted in Fig. 4.5. Each link has a capacity of 50 $Mbps$, and

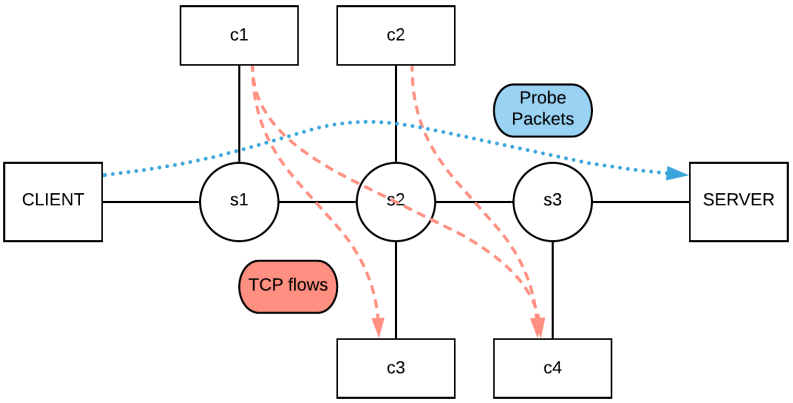


Figure 4.5: Parking lot topology used for the tests.

a nominal delay of 10 ms. Each queue enables FQ-CoDel by default with a size of 500 $pkts$. Packets are sent from the client to the server. TCP flows are sent between c1, c2, c3, and c4 to maintain a constant network load. Several studies show that short-lived flows, mainly generated by Web data transfers caused by user interactions, dominate the Internet traffic [10]. Thus, the TCP traffic is generated in such a way that the length of the TCP flows respects the Pareto principle (80% of short flows, 20% of long flows) as shown in Fig. 4.6 over the long run. The objective of this first experimentation is to probe the network congestion level, i.e., the load of the queues at s1, s2, and s3. The probe flow follows the blue path outlined in Fig. 4.5 and is a TCP flow containing real data. Each TCP flow following the red path has been generated as follows:

- the duration of each TCP connection has a Poisson distribution with parameter $\lambda = 1$;
- the time between two starting flows has an exponential distribution with parameter $\lambda = 10$;
- the server-client pair is randomly selected between the pairs (c1, c3),

(c1, c4), and (c2, c4);

- each TCP flow is generated with the iPerf traffic generator.

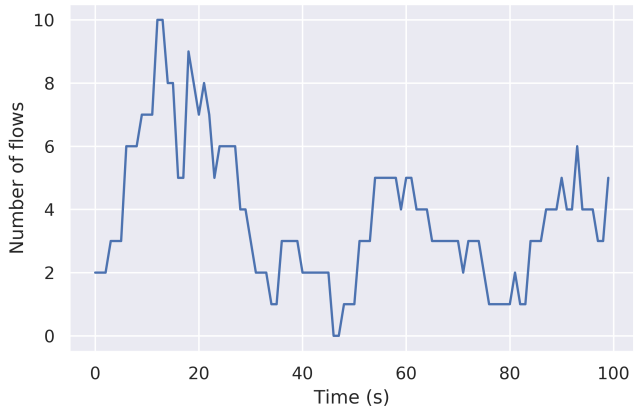
This setup attempts to reproduce realistic and variable network conditions:

- the bottleneck is not stable and moves from s1 to s2 randomly. We can see the evolution of the queue length in Fig. 4.7: both queues can act as the bottleneck depending on the network load and flows;
- the number of TCP flows changes to mimic a network load as shown in Fig. 4.6;
- TCP constantly switches between the slow-start phase and the congestion avoidance phase. This allows us to obtain a very diverse distribution of the variables, representing various kinds of behavior (few flows, congestion, slow-start/cruise-control phase, unbalance between buffer load...).

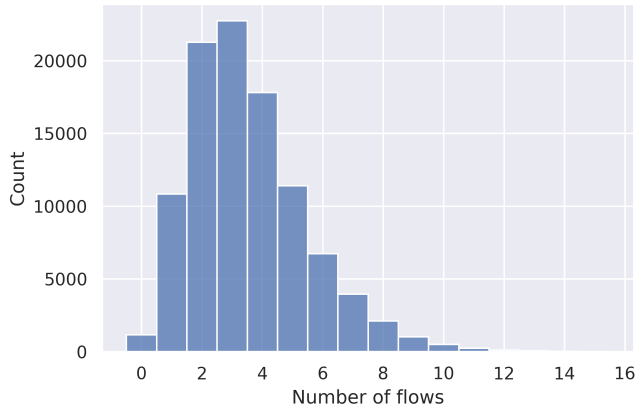
4.3.1 Features and ML algorithm

As mentioned before, we seek to use ML algorithms to predict the network congestion level. As we are using an emulated environment, the internal congestion level is known at each time instant, which allows supervised ML algorithms to be considered. We use then the data collected by the probe flow to estimate the metrics. These collected data are:

- the average rate at which the packets are sent;
- a binary variable indicating which packets have been lost;
- the RTT of each packet, as shown in Fig. 4.4. Note that we have used the complete time series in this work;
- a timestamp indicating when each packet was sent.



(a) Evolution of the number of flows in the network for 100 seconds. The alternation between high-load phases and phases with almost no TCP flows can be observed.



(b) Histogram of the number of concurrent flows in the network with a 1-second resolution. There is a majority of short flows.

Figure 4.6: Visualization of some flow characteristics.

As a consequence, the vector of features that will be used at the input of the ML algorithm is of size $3N + 1$.

In this section, the presented results have been obtained with the help of Attention mechanisms. Chapter 5 explains why this algorithm was used, and how it was trained. The current setting of our training is that we are still working in a supervised setting because we have all the data we need

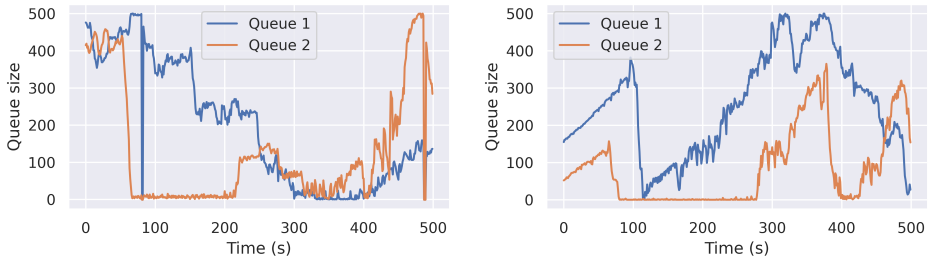


Figure 4.7: Examples of the evolution of the two bottleneck queue lengths at $s1$ and $s2$ for two different time periods. This shows that the bottleneck of the network is changing, and sometimes the first buffer is the limiting factor, and sometimes the second one is.

thanks to the emulator. The features are part of a multidimensional time series, and the size of the time series is not fixed. We are using the MSE loss presented in (3.4).

4.4 Results

Neural networks were trained in a supervised way to predict the internal congestion level from the features provided by the CC algorithm. The predictions of our algorithm are then evaluated in two different ways:

error plots - the x axis represents the real value of the variable to be predicted whereas the y axis is the average error resulting from that prediction. The line represents the average of the estimation error, and the area around the line shows the standard deviation, indicating how the error is spread around its average value. Note that the y axis has been normalized by the maximum range of the x axis, to allow comparison within the total range. We did choose not to normalize by the corresponding real value, indeed it would have been problematic around zero because the normalized error would grow too much. We think that the same error absolute error when the queue is full or empty has the same impact, so the normalization should be the same;

multivariate distributions - the actual value of the variable is represented in the x-axis whereas its prediction is in the y-axis. This representation shows how the predictions are distributed around their means. Note that between two consecutive blue lines, there are 10% of the points of the scatter plot. The top panel of each figure shows the distribution of the variable we try to predict (Y_1, Y_2, Y_3, or Y_4). Finally, the right panel shows the distribution of the predicted variable.

Fig. 4.8a shows that the bottleneck load can be predicted with good accuracy with the proposed ML algorithm. The important point is to know if the bottleneck is full or in an empty state, in order to work close to the optimal point corresponding to the queue almost empty, but used at 100% of its capacities. The prediction of Y_2 (Fig. 4.1) is correct and provides important information, namely if the queue is being filled or emptied. However, the prediction around the edge is slightly worse: the evolution speed is always underestimated. Table 4.1 shows performance measures for the prediction of the Boolean variable ($Y_2 > 0$). We can observe that the prediction is correct in 84% of cases. We think that this information is also very useful for a CC algorithm, and is complementary to the knowledge of Y_1 and Y_3 for finer control. For example, if we are already working at the optimum control point, and ($Y_2 > 0$), then we can slightly reduce the sending window to stabilize Y_2 around zero, with a low value of Y_1.

	Results
True positives	36 %
False negatives	8%
True negatives	47%
Precision	0.84
Recall	0.16

Table 4.1: Scores for Y_2, the slope of the bottleneck buffer size over the last 15 sent packets.

The proposed ML algorithm is also predicting Y_3 (Fig. 4.8c) with high

accuracy, when compared to the simple Copa predictor ($RTT_{\text{standing}} - RTT_{\text{min}}$): both methods underestimate the total load on all the buffers (Y_3), but our approximation error is approximately half of that obtained with Copa. Note again that our aim is not a comparison with Copa or BBR, but rather to improve the performance of existing CC algorithms. Predictions of Y_1 and Y_3 show that the proposed algorithm tends to overestimate the load when the queue is almost empty and to underestimate the load when it is not empty. The estimation provided by Copa is sensitive to the same phenomenon.

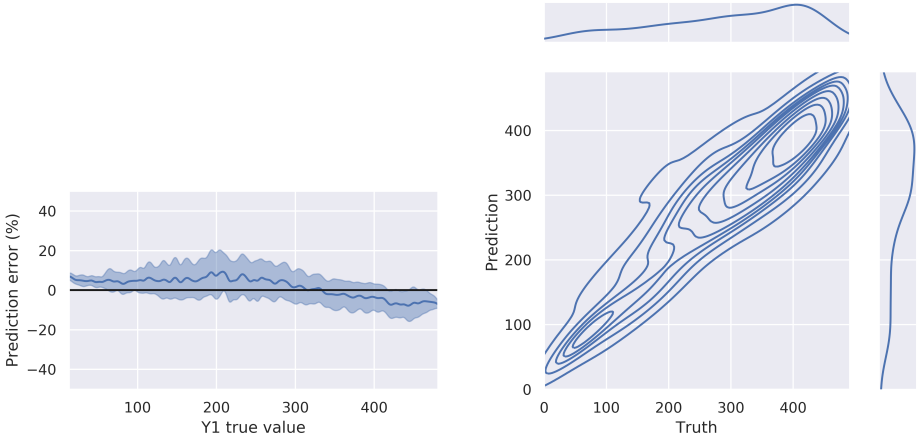
This can be explained by the following reasons:

- when the buffers are full, Copa estimates the load proportionally to $RTT_{\text{standing}} - RTT_{\text{min}}$. This estimation works if, during the aforementioned time window, the buffers are empty. This way RTT_{min} should correspond to the RTT caused by the link, and the estimation should be correct. However, in practice, especially when we work with large buffers, the queues are not emptied fully, and RTT_{standing} is just an overestimation of the RTT of the link. That leads to an underestimation of the buffer load. Copa uses this estimation with the hypothesis that Copa is the only CC being used, but this scenario is not realistic: in a lot of cases, there are different CC algorithms in competition;

- although ML algorithms based on neural networks or Support Vector Regression (SVR) are black boxes we still have some ideas to explain the difference between the predictions and the real value of the metrics. A similar underestimation is observed for these algorithms and for Copa. We think that this can be explained as follows: if we are working on a big buffer, we will rarely see the empty state of that buffer, since the algorithm cannot differentiate between the RTT caused by the state of the link and the RTT caused by the minimal load in the buffer;

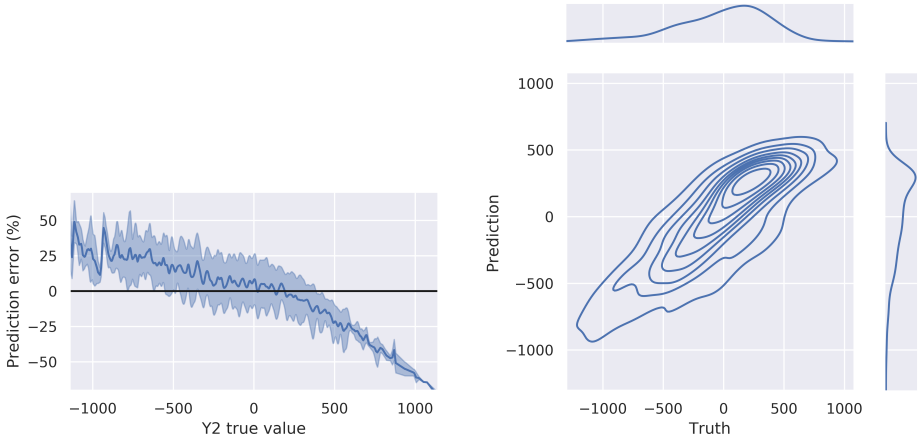
- there is also a small overestimation of the load of the buffer when the queues are almost empty.

We also notice that the error in Fig. 4.8b1 is increasing if the real value of Y_2 is over 500. It is easily explained by the lack of training examples. Indeed in Fig. 4.8b1, the distribution plot on top shows us that there is almost no training data when the filling speed of the bottleneck is higher than 500. This is not really an issue because this training set represents the behavior of the network in the different topologies where we collected our data, thus the prediction error is happening when a rare event is occurring.



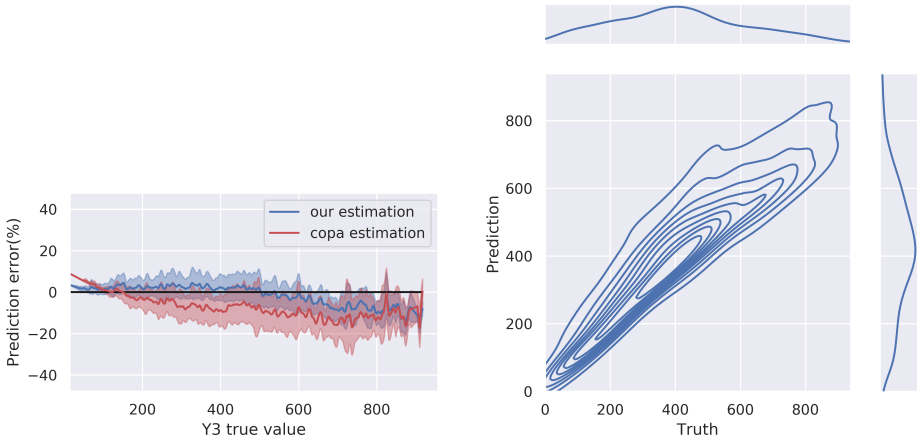
(a1) Errors (a2) Distribution

(a) Y_1: the average size in packets of the bottleneck, for the last 15 sent packets.



(b1) Errors (b2) Distribution

(b) Y_2: the slope of the average size in packets of the bottleneck, for the last 15 sent packets.



(c1) Errors (c2) Distribution

(c) Y_3: the average size in packets of all the buffers along the path of the packets, for the last 15 sent packets.

Figure 4.8: Error and prediction of the three variables using Attention (the error is normalized with the maximum range of the variable).

Chapter 5

How attention can help CC algorithms

This chapter focuses on the choice of machine learning algorithm that we used in chapter 4.

We first formalize the regression task, and precise which features are used for prediction.

Secondly, we show how classic Machine Learning or Deep learning algorithms were not effective enough. We then explain why attention-based algorithms were more successful on congestion control-related tasks.

Then we present our training methodology and how the networks were implemented.

The last part focuses on presenting the architecture of the chosen network, and how it was trained.

Contents

5.1 Finding a minimum

As discussed in the previous chapter, the aim of the ML algorithm used in this thesis is to estimate the metric presented in 4.2. Many estimators were investigated without providing interesting results. One of the metrics we propose to consider is a metric used by Copa used for its CC algorithm, namely Y_3 that is estimated by minimizing (3.1). This estimator has shown good performance and our goal was to propose a supervised ML method that can improve (3.1). The objective of estimating a maximum/minimum led us toward the use of new deep learning methods.

5.2 Choosing an algorithm

In this section, we introduce the different ML algorithms used in this study and justify both their choice and the validity of the results obtained with these algorithms. We have an input of dimension $3N + 1$ containing both discrete (Boolean) and continuous variables described in 4.3.1. The output that we seek to predict is one of the continuous metrics, which restricts the type of algorithm that can be considered. We also need a predictor adapted to regression.

Our first guesses were to use feed-forward neural networks, support vector regression (SVR), and traditional time series prediction methods (ARIMA for example). However, these kinds of algorithms are not adapted to our problem for the following reasons :

- the input size is fixed for feed-forward neural networks and SVR. If these algorithms are trained with a pattern of size 64, it is impossible to extend the results to samples having a different size. We would like to have an algorithm that can adapt to the length of the time series. Indeed, the available amount of data depends on the capacity of the link (the faster the link, the more data), and of course, including all

the data might improve the accuracy of the results,

- after consulting the state-of-the-art, we noticed that Copa for example uses a simple estimator in order to compute Y_3, based on $RTT_{\text{standing}} - RTT_{\text{min}}$. RTT_{standing} is the minimum of the RTT for a short time window whereas RTT_{min} is the minimum for a longer time period. Copa works quite well, so our idea is that being able to compute a minimum from a vector is certainly important if we seek to get a good estimation of some metrics. However, it is very difficult for the mentioned ML algorithms to estimate a maximum function (it is theoretically possible with neural networks with large complexity but is difficult in practice).

In order to fill these gaps, we have studied existing ML algorithms that are adapted to time series. These algorithms include the family of long short-term memory (LSTM) networks [19] and GRU [9]. While these algorithms allow the first concern to be bypassed, they still do not allow a minimum/maximum to be computed. We also investigated Attention [40]. Attention has become the state-of-the-art solution for natural language processing and is adapted to time series as well.

5.2.1 LSTMs

This section describes LSTM networks as they are commonly used for time series, and introduces the rationale of this study. We noticed that some regression tasks, such as the estimator presented in (3.1), cannot be accurately predicted with LSTM networks. Indeed, the predictions obtained with these networks can correctly estimate the maxima of a time series, but they fail to memorize this information for the next time steps as in Fig. 5.1. Note that due to their nature, the same observation holds for all RNN and CNN networks.

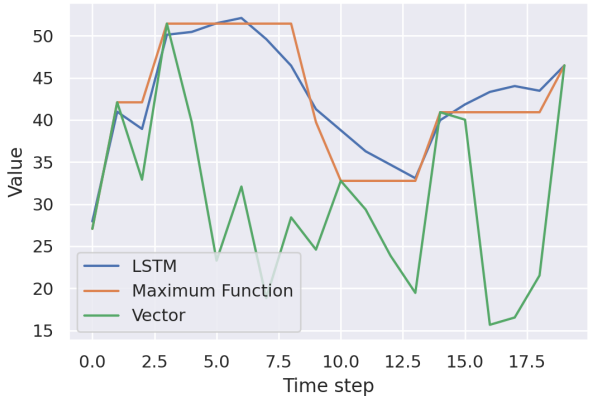


Figure 5.1: The green line is a random vector. The orange line is the maximum of the last 5 values. The blue line corresponds to the output of an LSTM network trained to estimate the function represented by the orange line.

5.2.2 Attention

We found that Attention is able to estimate the maximum/minimum of a vector with better accuracy than other networks. Indeed, the Attention mechanism involves the following computation: $Y = \text{softmax}\left(\frac{QK^T}{\sqrt{n}}\right)$ with Q K and V linear transformations of the input vector X, and n the length of the vector. To illustrate that, if we choose the simplest linear transformation $X = V = Q = K$, the resulting vector is an estimation of the maximum of X, where X could be a vector of RTT expressed in *ms*. As an example,

$$X = [9.3246, 10.4722, 11.5280, 12.6615, 10.6212]$$

leads to

$$Y = [12.6147, 12.6317, 12.6417, 12.6486, 12.6334].$$

Of course, Attention is more complicated than this simple example, and knowing the maximum/minimum is not enough to make a correct prediction. However, this example illustrates the intuition which leads us to consider

that Attention should be adapted to our task. It was indeed confirmed by the results: for the moment, Attention-based algorithms achieve the best performance when applied to our datasets.

If self-Attention is applied to a matrix $\boldsymbol{X}$ of univariate RTT (i.e., $d = 1$) with $\mathbf{W}_q = \mathbf{W}_k = 1$, the result of the softmax operation provides a matrix of $\mathbb{R}^{L \times L}$ with rows close to 0, except the row corresponding to the index of the maximum of $\boldsymbol{X}$ whose elements are close to 1. Thus, by choosing $\mathbf{W}_v = 1$, the result of the Attention is a matrix whose elements are approximations of the maximum of $\boldsymbol{X}$. If we are interested in the minimum of $\boldsymbol{X}$ (instead of the maximum), one can choose $\mathbf{W}_k = -1$. It is also possible to define more complicated queries, such as finding when the maximum of RTTs occurred (considering $\boldsymbol{X}$ also contains the time information with $d = 2$). In that case, we need to choose $\mathbf{W}_q = \mathbf{W}_k$ as the projection of $\boldsymbol{X}$ on the RTTs axis, and $\mathbf{W}_v$ as the projection of $\boldsymbol{X}$ on the time axis.

This ability of the Attention algorithm to estimate maxima or minima is not discussed in the literature, and it is a feature that we have found to be very interesting, which shows that Attention can be quite powerful for the estimation of CC-related metrics.

5.3 Training methodology and implementation

The aim of this section is to give details on the way we implemented, trained, and used the deep-learning algorithm we presented.

DL framework. During this thesis, all deep-learning experimentations have been performed with the PyTorch framework.

Initialization. The initialization of all trainable parameters is random and follows a normal distribution with a mean of 0 and a variance of $\frac{1}{\sqrt{N}}$ where N is the input size of the matrix the parameter is from.

Optimization Algorithm. ADAM is the selected optimization algorithm, it has been empirically chosen against standard Stochastic Gradient

Descent (SGD) and RMSprop algorithms.

Choosing the learning rate. The learning rate has been empirically selected. For standard DL architecture with only feed-forward neural networks, a learning rate of 0.01 is often good enough. However, Attention networks are harder and longer to train, because a learning rate of 0.01 can cause chaotic training behavior or a plateau in the learning behavior as shown in Fig 5.2. In experiments done in chapter 6 we even choose a lower learning rate for stabilization purposes.

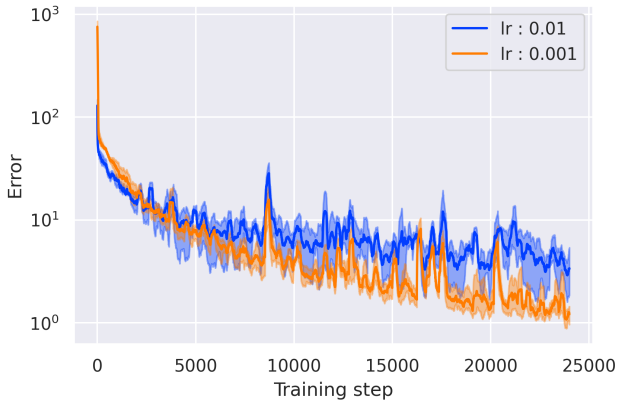


Figure 5.2: Training loss of the attention mechanism with a learning rate of 0.01 and 0.001.

When to stop the algorithm. Generally, we used in this thesis two datasets, one for training and one for testing. The training was stopped when the testing loss stopped decreasing. This early stop helps to prevent overfitting behavior.

Normalization. We added a normalization step before doing the attention step, which improved a lot the convergence process.

5.4 Robustness of the algorithm

This section discusses how to validate the prediction results obtained using neural networks. When an ML algorithm is used to estimate a function of

interest, we need to know and measure the scope of validity of the trained estimator. Indeed, if we train our estimator to work with a specific topology, e.g., the parking lot topology with 2 nodes between the sender and the receiver of the probing flow, it might not work on a network with three nodes. This lack of generalization can be due to over-fitting: the network can learn very well in a restricted environment but cannot generalize its behavior to a larger environment.

In this context, the issue with ML algorithms is that they will not generalize if learning is conducted in a restricted learning space (i.e., our emulated environment), which is different from the real world. In order to study the generalization capacity of the proposed algorithm, we trained our algorithm in a restricted environment, and then tested it with different parameters. This way, it was possible to determine whether the proposed predictor is robust to changes in some of the network parameters. If some parameter changes have an impact on the prediction, the learning dataset needs to be improved and include a higher diversity to improve generalization. More precisely, the parameters that were changed to test the robustness of the proposed estimator include:

- the topology of the network: the number of nodes between the client and the server;
- the capacity of the links;
- the delay of the links;
- the queuing/scheduling mechanism in the buffers.

Some experiments are presented in the next sections showing that the proposed network was not over-training.

5.4.1 Cross Validation - Number of flows

To see whether the number of flows in the training dataset was important, the model considered in this thesis was trained in an environment with 10 concurrent flows at each moment. In a second step, the model was then tested in environments with either 2 or 25 flows. A confusion matrix summarizing the results is shown in Table 2.1. We notice that it is important to have a variable number of flows during the training process because the network is not able to generalize the result when the number of flows is higher. The generalization to a smaller number of flows seems to be less problematic. These remarks explain why we changed the number of flows during the training scenarios 4.3.

	Training 10 flows	Test 2 flows	Test 25 flows
True Positive	30	19	39
False Positive	9	7	9
False Negative	10	6	25
True Negative	51	69	26
Precision	81	88	65
Recall	19	12	35

Table 5.1: Confusion matrix for different scenarios.

5.4.2 Cross Validation - FQ-Codel

In a similar way as the previous section, we decided to test if the model could generalize the network behavior if the type of queuing management changed between the training and the testing sets. The network is trained with a queue management set as FQ-Codel and then tested in an environment without FQ-Codel (with the same number of flows and only one parameter changing). Here a minor loss in precision can be observed, around 5%.

	Training FQ-Codel 2 flows	Test - 2 flows	Training FQ-Codel 15 flows	Test - 15 flows
True Positive	18	20	37	33
False Positive	8	6	10	15
False Negative	4	9	16	14
True Negative	70	64	37	38
Precision	88	84	74	71
Recall	12	16	26	29

Table 5.2: Precision for different scenarios.

Chapter 6

Improving the Attention mechanism

In this chapter, we will focus mainly on the contribution presented on [34], and describe the proposed architecture.

In the second part, we discuss unfinished work done on these recurrent Attention Architectures.

Contents

6.1 Reducing the complexity of Attention networks

Despite their remarkable performance, Attention networks are difficult to use because of their massive size (the GPT-3 model created by OpenAI for text interpretation has 175 billion parameters), and long training times. Indeed, to apply an Attention model to a time series, it should be applied at each time step. Thus, at the t^{th} step (with $t \in \{1, ..., L\}$), the computation complexity is linked to the matrix product. To treat a time series of length L, the complexity is therefore in the order of $\mathcal{O}(L^3)$ (the computation done at step $t-1$ cannot be used to ease the task because of the presence of the non-linear layers). In contrast, methods such as LSTMs have a complexity in the order of $\mathcal{O}(L)$. This computational time is a motivation to find a new NN architecture that is as efficient as Attention but faster, which is the goal of this chapter.

This goal of trying to reduce the complexity of the attention mechanism has been a focus of research in previous years [42] [24] because the main limitation of these huge models is their portability.

6.1.1 Proposed architecture

To overcome both shortcomings of LSTM and Attention networks, we propose a new hybrid architecture defined as follows:

1. The observation matrix $\boldsymbol{X}$ is concatenated with the position matrix $\boldsymbol{P}$ yielding
$$\mathbf{X}_p = [\mathbf{X}, \mathbf{P}].$$

2. An LSTM layer is constructed as follows:
$$\mathbf{H_i} = \mathbf{LSTM}(\mathbf{x}_1, ...\mathbf{x}_i) \in \mathbb{R}^{J \times d},$$
where J is the size of the vector produced by the LSTM network (with

one layer and a unidirectional network), to be chosen by the user.

3. An Attention network is constructed as follows:

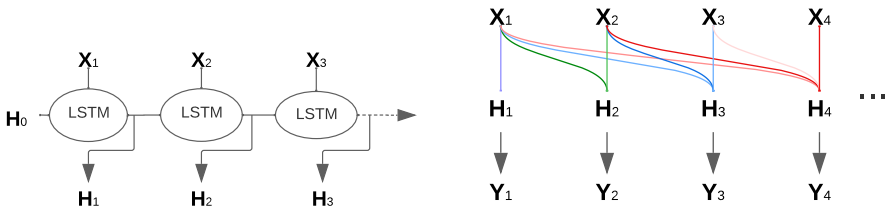
$$\mathbf{Y}_i = \mathbf{ATTENTION}(\mathbf{W}_q\mathbf{H}_i, \mathbf{W}_k\mathbf{X}_{1;i}, \mathbf{W}_v\mathbf{X}_{1;i}),$$

Where the matrix $\boldsymbol{H}_i$ is used to determine the most important elements from the past. The idea of this architecture is not to use self-Attention directly (since it is too computationally intensive) but to generate, thanks to an LSTM network, a vector generating the requests (J is thus the number of requests).

4. A non-linear layer (RELU activation function) is finally introduced as in many DL architectures:

$$\mathbf{Z} = \mathbf{FeedForward}(\mathbf{Y}).$$

The previous steps, 2), 3), and 4), can be repeated for each of the M layers of the network. The interest in this architecture, when compared to LSTM and Attention, will be shown in the next section.



(a) Creation of the vectors $\mathbf{H}_i$ (first step).

(b) Attention used to know which $\mathbf{X}_i$ is used to get $\mathbf{Y}_i$.

Figure 6.1: Two steps of the proposed NN architecture.

The idea behind this new architecture is that instead of making a correlation matrix for the whole time series, we only look at the correlation matrix

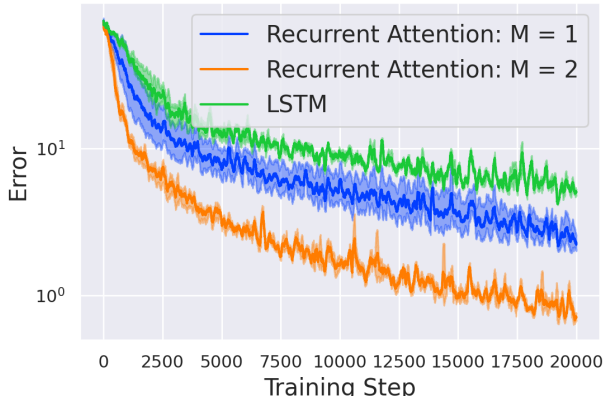
between the time series and a vector of fixed length created by an LSTM network. The performance of this architecture depends of course on the task for which it is used. In our application, we are still interested in features such as the maximum or minimum of a time series subset. It is reasonable to think that not each time series element is essential for the estimation. Indeed, to compute a maximum, we only need one element (chosen by the LSTM network, for example) to look at all the other elements and pick the minimum/maximum. In other more complicated cases, like natural language processing, it may be different since each word has to be connected with the other words to ensure that the sentence has some sense. Note that the number of elements produced by the LSTM network J is a hyper parameter that needs to be chosen by the user.

6.1.2 Results

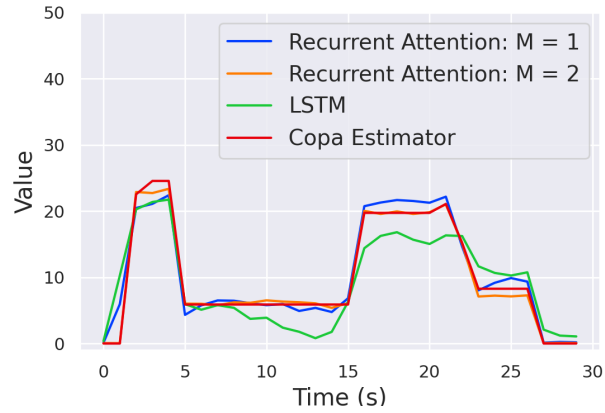
This section evaluates the performance and ability of the proposed NN architecture to seek information from the past of the time series within two use cases: the first experiment considers synthetic data with available ground truth, whereas the second experiment is conducted using real data from the evolution of an IP router queue load.

6.1.2.1 Finding a minimum

As explained in Section 3.2.4, a simple estimator of the queue load is of the form (3.1). This section studies the capacities of the proposed NN architecture to approximate this estimator. The parameters of the NN were chosen by cross-validation leading to $L_1 = 5$ and $L_2 = 30$. The NN was trained using a learning rate of 0.001 and the optimizer ADAM. The RTT time series were randomly generated at each training step according to independent samples from a normal distribution ($\mathcal{N}(\mu, \sigma^2)$ with $\mu, \sigma \sim \mathcal{U}(0, 10)$ fixed for each time series) to prevent over-fitting.



(a) Training loss. The colored envelopes represent the maximal and minimal errors for 10 trained models.



(b) Estimations provided by LSTM and Attention networks for (3.1).

Figure 6.2: Results for the synthetic task of estimating (3.1).

Fig. 6.2a shows that the new NN architecture, not only learns faster how to estimate the function f defined in (3.1), but can also estimate the difference between two minima with more accuracy than an LSTM network, which reaches a learning plateau. Of course, this remains an artificial task and the proposed network was created to solve that kind of task. This first experiment also shows that the deeper the network (i.e., the larger M), the faster the elements from the past can be learned. The poor performance of the LSTM network can be easily explained by the form of f, which is a simple

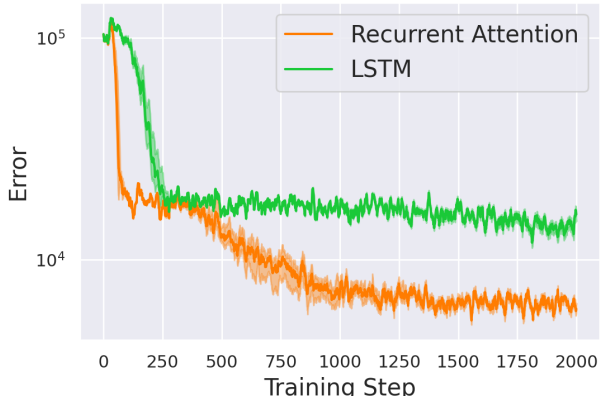
relationship between elements from the past of the time series. Note that LSTM has problems learning how to store elements in its hidden vector and to memorize all values (for example, if the value of RTT is increasing, each value will be at some point a minimum of the sliding window). Conversely, the proposed Attention network directly refers to elements from the past, which can be accessed in one step.

Fig. 6.2b shows that LSTM networks cannot store the relevant information for a correct amount of time. LSTM regression seems to approximate the time series by a piece-wise linear function to minimize the mean prediction error. The LSTM network struggles to use specific past information in the time window of interest. Even if this information were available, the hidden vector used by the LSTM network would have difficulty storing all the relevant information contained in the time window of interest. Conversely, the Attention mechanism successfully uses the values from the past of the time series, yielding better estimates.

6.1.2.2 Application to real networks

This part considers a real application, defined as the prediction of the queue level at a bottleneck for a given network path. For this purpose, we propose to use the time series of RTTs as well as the time of transmission of these packets. The training is done with a learning step of 0.0005 and the ADAM optimizer.

Fig. 6.3a shows that a plateau in the training phase is reached by the LSTM network, while Attention allows the relationship between the data and the network load to be learned quickly and with greater accuracy. As (3.1) is a good approximation of the load in the queues, we can expect that the approximation provided by the network has a strong connection to that equation and so may need to use elements that are located far in the past. These remarks explain the fast learning of the proposed Attention network.



(a) Training loss. The colored envelopes represent the maximal and minimal errors for 10 trained models.

(b) Estimation of the bottleneck load (with emulation) using LSTM and Attention architectures.

Figure 6.3: Results for a concrete task of estimating the current load at the bottleneck in a network path.

Note that the excellent performance of the new NN algorithm is similar to that obtained with a global Attention network. However, the proposed architecture allows faster training. Finally, it is interesting to note that it is not helpful to increase the number M of network layers for this example to obtain a better estimation accuracy.

Note that the parameters of the NN architecture implemented for this case were determined by cross-validation leading to $M = 1$, $J = 3$, $d = 6$,

and 3 heads for the Attention network. The hidden dimension of the LSTM layers was set to 18, and the dimension of the hidden layer in the Feedforward network was 36. Finally, the learning rate used during training was 0.0005.

6.2 Linearizing the attention mechanism

The architecture presented in 6.1 has a complexity of $\mathcal{O}(L^2)$. Indeed, at a given time step t of the time series, the complexity of each step is: $\mathcal{O}(1)$ for the LSTM part, $\mathcal{O}(t)$ for the attention part, $\mathcal{O}(1)$ for the feed-forward part. Over L time steps, the complexity is then $\sum_{t=1}^{L} \mathcal{O}(t) = \mathcal{O}(L^2)$. It is, of course, better than $\mathcal{O}(L^3)$, but the computational cost of this algorithm is still too high to be implemented in real-life scenarios. This section aims to explain and present some ideas we had to try to reduce the complexity of the architecture further.

The first idea developed here is that not all time series elements have the same utility. In our recurrent example of searching the minimum of a moving time window, only the local minima and its timestamp (i.e., the moment it was achieved) are relevant information. If the attention mechanism forgets about the other values, we can expect to observe no change in the quality of the estimations. To verify that this idea is valid, we have considered the network trained in 6.1.2.2 and have studied the result of $\mathbf{ATTENTION}(\mathbf{W}_q\mathbf{H}_i, \mathbf{W}_k\mathbf{X}_{1;i}, \mathbf{W}_v\mathbf{X}_{1;i})$, especially the product of the softmax operation $\text{softmax}(\frac{(\mathbf{W}_q\mathbf{H}_i)(\mathbf{W}_k\mathbf{X}_{1;i})^T}{\sqrt{d}}$. Indeed the softmax scores indicate which elements of the matrix $\mathbf{X}$ are used when computing the buffer usage at each time step.

Fig. 6.4 shows that some elements of the time series have relative importance for future predictions, and others are completely useless. For example, the element around $y = 12$ is used for most of the predictions of the future. One important point is that the number of relatively useful elements is very low in this use case. Our idea was then to only keep the elements that might

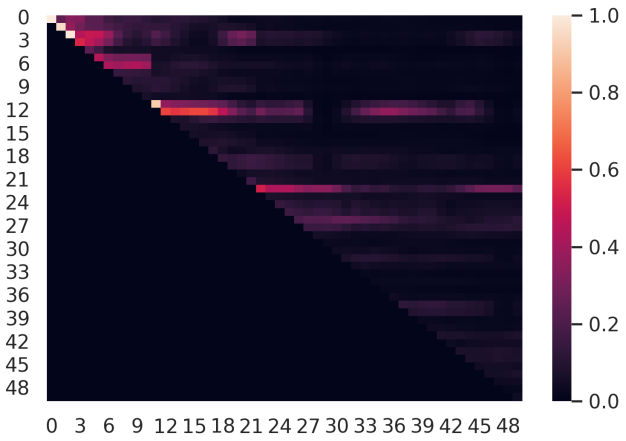


Figure 6.4: Matrix of scores $\boldsymbol{S}$ between the elements $\mathbf{H_i}$ and $\mathbf{x}_j$ with $j \leq i$. j is represented on the y axis, i on the x axis.

be useful for future prediction. Indeed, assume that only the top 20 elements are kept, then Attention is only focused on these 20 elements, and the overall complexity is then reduced to only $\mathcal{O}(L)$. The following scores have been considered to choose which elements need to be kept in the analysis (with $\forall 1 \leq j \leq i \leq L$) :

- $\boldsymbol{S}_{i,j}$ is the attention score between $\mathbf{H_j}$ and $\mathbf{x}_i$,
- $\boldsymbol{R}_{i,j} = \frac{\sum_{k=j}^{L} \boldsymbol{S}_{i,k}}{L-j+1}$ is the mean attention score of $\mathbf{x}_i$ for all $\mathbf{H_k}$,
- $\boldsymbol{U}_{i,j} = \frac{\boldsymbol{R}_{i,j}}{\sum_{k=1}^{j} \boldsymbol{R}_{k,j}}$ is the normalized score.

The score $\boldsymbol{U}_{i,j}$ can be used to select at the time step i the elements to discard and the elements to be preserved for future prediction; for example, we could say that at each time step, we keep the 20 elements that have the highest $\boldsymbol{U}$ value (the ones that are the most useful for future computation). The issue is that the score $\boldsymbol{U}_{i,j}$ depends on future attention prediction. It can only be computed if the future of the time series is known. One solution is to try to predict those scores from the current knowledge of the time series. For

example, when a minimum has to be computed, there is no need for future elements to know that we only need to keep local minima.

To achieve that with a machine learning algorithm, we tried to train a DL algorithm to predict $\boldsymbol{U}_{i,j}, \forall 1 \leq j \leq i$ at each time step i. This architecture would need to produce a positive vector of length i at the time step i, with the sum of its values equal to 1. To achieve this, we modified the Attention algorithm presented in Section 6.1 and stopped the attention step at the softmax computation. This model would satisfy all the constraints explained in this paragraph.

Then at each time step i, we computed the scores $\boldsymbol{S}(i,j)$, $\boldsymbol{R}(i,j)$, and $\boldsymbol{U}(i,j)$ for all $1 \leq j \leq i$. As we want the algorithm to predict the scores $\boldsymbol{U}$, we introduced two new losses related to a probability distribution (MSE would still work, but it not adapted to probability distribution spaces) :

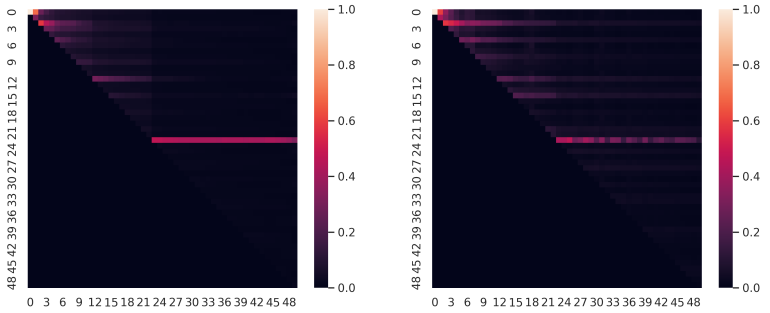
- the Binary Cross Entropy (BCE) loss. Let $\boldsymbol{p}, \boldsymbol{q}$ two discrete probability distribution in $[|1, n|]$. $\text{BCEloss}(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{i=1}^{n}(\boldsymbol{p}_i \ln \boldsymbol{q}_i + (1 - \boldsymbol{p}_i) \ln 1 - \boldsymbol{q}_i)$;
- the χ^2 loss. Let $\boldsymbol{p}, \boldsymbol{q}$ two discrete probability distribution in $[|1, n|]$. $\chi^2\text{loss}(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{n} \frac{(\boldsymbol{p}_i - \boldsymbol{q}_i)^2}{\boldsymbol{q}_i}$.

We trained both Attention algorithms simultaneously, with a learning rate of 0.0005. The choice of the learning rate for the estimation of $\boldsymbol{U}$ has been determined empirically. It is however not always the same as the first learning rate because the loss of the second network depends on the first network, and it may lead to unstable results common in the reinforcement learning field[29].

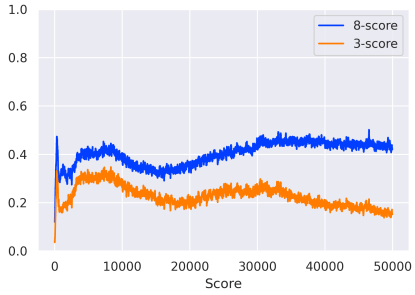
To find whether the training is efficient, we introduce a " n-score". Let us suppose that at each time step i, we keep the best ten elements from the past. It means that we keep the ten elements j from the past that have the highest $\boldsymbol{U}_{j,i}$. However, we have to make this choice based on the estimation

of $\boldsymbol{U}$ noted $\boldsymbol{U}^{\text{est}}$. The 10-score is the proportion of elements in the top 10 values of $(\boldsymbol{U}^{\text{est}}_{(}j,i))_{j<i}$ also in the top 10 values of $(U_{j,i})_{j<i}$. If the 10-score is close to one, one could say that choosing the elements with $\boldsymbol{U}^{\text{est}}$ instead of $\boldsymbol{U}$ is sufficient.

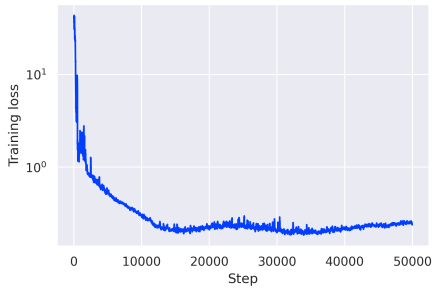
The training with the two presented losses led to the results in Fig. 6.5 and 6.6. The main task is still the synthetic task of finding (3.1).



(a) Comparison of the two Matrices $\boldsymbol{U}$ on the left and $\boldsymbol{U}^{\text{est}}$ on the right.



(b) Evolution of the n-scores, during the training with the BCE loss.

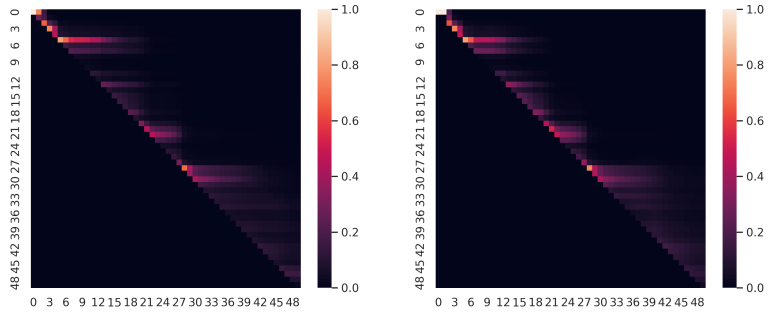


(c) Evolution of the BCE loss during the training.

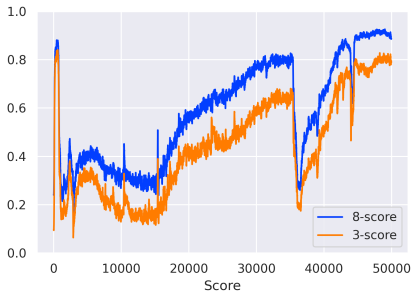
Figure 6.5: Results after training with the BCE loss.

We can observe that using the BCE loss is inefficient, as the n-score is relatively low. However, if we train the networks with the χ^2 loss, we can see that the n-score reaches values close to 0.9, meaning that 90% of the interesting elements from the past are indeed selected.

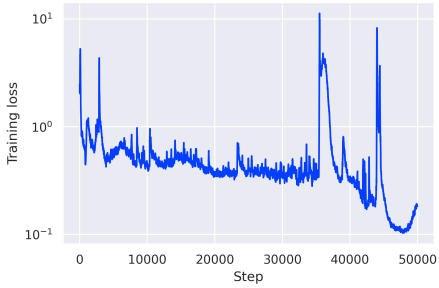
To conclude, we would like to mention that this section is still prospective and that the following problems would deserve to be solved:



(a) Comparison of the two Matrices $\boldsymbol{U}$ on the left and $\boldsymbol{U}^{\text{est}}$ on the right.



(b) Evolution of the n-scores, during the training with the χ^2 loss.



(c) Evolution of the χ^2 loss during the training.

Figure 6.6: Results after training with the χ^2 loss.

- obtaining a better approximation of the scores $\boldsymbol{U}_{i,j}$;
- defining an algorithm that selects which elements to keep;
- comparing the accuracy of the precision with and without element selection. Indeed, we showed for the moment that the selection of the best elements from the past is accurate, but we have no idea how to quantify the global accuracy loss if we forget all the elements not selected;
- to have a formal proof of the interest of the algorithm, we can bound the result of $\textbf{ATTENTION}(\mathbf{W}_q\mathbf{H}_i, \mathbf{W}_k\mathbf{X}_{1;i}, \mathbf{W}_v\mathbf{X}_{1;i})$ if we remove the element that has the smallest value of $\boldsymbol{U}_{i,j}$.

Chapter 7

Conclusion and perspectives

This chapter essentially summarizes the entire work done in this thesis and presents the eventual path for future works.

Contents

7.1 Conclusion

This thesis contributed to both the networking and machine learning fields. In this section, we discuss them and explain how significant they are.

7.1.1 Congestion control

Congestion control and rate control are at the backbone of the Internet. As the topology of networks becomes more complex, the need for new congestion control methods is essential. Indeed we currently assist in the rise of mobile networks (4G, 5G, 6G) and satellite networks (Starlink, OneWeb, and Kuiper) which are used by more users every day but have different behavior than earth-based networks. These new networks have, for example, more random losses and higher latency, making old CC algorithms such as CUBIC based on loss metrics less efficient and relevant. This is the common motivation for new CC algorithms, which use new metrics to have more information about the state of the network. In this context, the metric we introduced in 4.2 might be helpful in the development of future algorithms. Indeed, the metrics we provided help us understand the dynamics of the buffer along the path of the packets more accurately. Moreover, we proposed a deep learning algorithm that accurately estimates these metrics. This work was more of a proof of concept, which showed the possibilities of network state estimation than a brand-new CC algorithm.

7.1.2 Deep Learning

In another context, this thesis proved that Attention-based mechanisms applied for time series tasks were particularly relevant when the estimation relies on the computation of some metric such as the maximum/minimum of a time series. This work also improved the complexity of some deep-learning mechanisms based on Attention by reducing the complexity of the

algorithm from $\mathcal{O}(L^3)$ to $\mathcal{O}(L^2)$. This is particularly useful, even outside the network world, because a low-complexity algorithm is easier to deploy: not every hardware has access to a massive amount of computational power and GPUs.

7.2 Future work

Many subjects have been worked on during this thesis, leaving a lot of promising future work to be dealt with.

Further study the pattern effects on the network. We could further study the impact of patterns on the accuracy of the network state estimations. Indeed the shape of the pattern could be adapted to different situations: if the queue is nearly empty, some patterns might produce more accurate results, but in another scenario, they could harm the performance. This is the idea we had at the beginning of the thesis but didn't have the time to experiment. One lead could be to use a Reinforcement Learning Framework to find and use the most helpful pattern at the right moment.

Build a new CC algorithm. One could also try to build a CC algorithm from the metrics we proposed. Indeed the more we know about the internal state of the network, the better the CC algorithm can act.

Further improve the complexity of attention mechanism. Section 6.2 shows promising leads to improve the complexity of the attention-based mechanism. It is currently a trend in the DL field to reduce the complexity of already successful algorithms to allow them to be deployed to a larger audience. Further work in this field would need to bound the error of the attention result if we remove the least useful element, and show that experimentally it is still efficient. One could try to adapt the number of elements kept in the memory dynamically and automatically.

Acronyms

ACK Acknowledgment. 35

AIMD Additive-Increase-Multiplicative-Decrease. 32

BBR Bottleneck Bandwidth and Round-trip propagation time. 4, 30, 39

BCE Binary Cross Entropy. 93

BDP Bandwidth-Delay product. 24

CBR Constant Bit Rate. 61

CC Congestion Control. 17

DL Deep Learning. 17, 44

GEO Geostationary Earth Orbits. 3, 16, 23

GRU Gated Recurrent Unit. 5, 30, 49

LEO Low Earth Orbit. 4, 16, 23

LSTM Long Short-Term Memory. 5, 30, 49

ML Machine Learning. 44

MSE Mean Square Error. 47

NN Neural Network. 4, 30, 44

PCC Performance-oriented Congestion Control. 4, 30, 40

PEP Performance-Enhancing Proxy. 24

QoE Quality of Experience. 19

QoS Quality of Service. 19

RNN Recurrent Neural Network. 4, 30, 47

RTT Round Trip Time. 19

SGD Stochastic Gradient Descent. 78

SVR Support Vector Regression. 70

TCP Transmission Control Protocol. 4, 30, 31

Bibliography

[1] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, and Limin Yan. Evaluation of tcp vegas: Emulation and experiment. In *ACM SIGCOMM*, page 185–195, New York, NY, USA, 1995.

[2] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *USENIX NSDI*, pages 329–342, Renton, WA, April 2018.

[3] F. Baker and G. Fairhurst. IETF Recommendations Regarding Active Queue Management. RFC 7567, IETF, July 2015.

[4] Alberto Blanc, Konstantin Avrachenkov, Denis Collange, and Giovanni Neglia. Compound tcp with random losses. In *Networking*, 2009.

[5] J. et al. Border. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, RFC Editor, June 2001.

[6] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, 2004.

[7] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *Commun. ACM*, 60(2):58–66, January 2017.

[8] J. Chung et al. Gated feedback recurrent neural networks. In *Proc. Int. Conf. on Machine Learning*, pages 2067–2075, 2015.

[9] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2067–2075, Lille, France, July 2015. PMLR.

[10] D. Ciullo, M. Mellia, and M. Meo. Two schemes to reduce latency in short lived TCP flows. *IEEE Communications Letters*, 13(10), October 2009.

[11] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Skype video congestion control: An experimental investigation. *Computer Networks*, 55(3):558–571, 2011.

[12] A. de Santana Correia and E. L. Colombini. Attention, please! a survey of neural attention models in deep learning. *Artificial Intelligence Review*, pages 1–88, 2022.

[13] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *Proc. 12th USENIX NSDI Conf.*, pages 395–408, 2015.

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[15] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15(3):96–96, 2011.

[16] Mark J. Handley, Sally Floyd, Jitendra Padhye, and Joerg Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. IEEE Press, 2001.

[19] Sepp Hochreiter and Jürgen Schmidhuber. LSTM can solve hard long time lag problems. *Advances in neural information processing systems*, pages 473–479, 1997.

[20] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF, May 2021.

[21] V. Jacobson and R.T. Braden. TCP extensions for long-delay paths. RFC 1072, IETF, October 1988.

[22] Mohamed M Kassem, Aravindh Raman, Diego Perino, and Nishanth Sastry. A browser-side view of starlink connectivity. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 151–158, 2022.

[23] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):89–102, aug 2002.

[24] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[26] V. Konda and J. Kaur. Rapid: Shrinking the congestion-control timescale. In *IEEE INFOCOM*, pages 1–9, Rio de Janeiro, Brazil, 2009.

[27] Hans Kruse. Data communications protocol performance on geostationary satellite links-lessons learned using acts. In *16th International Communications Satellite Systems Conference*, page 1072, 1995.

[28] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, Emmanuel Lochin, Francklin Simo, and David Pradas. Quic: Opportunities and threats in satcom. *International Journal of Satellite Communications and Networking*, 40(6):379–391, 2022.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[30] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[31] François Michel, Martino Trevisan, Danilo Giordano, and Olivier Bonaventure. A first look at starlink performance. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 130–136, 2022.

[32] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The great internet tcp congestion control census. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), dec 2019.

[33] Victor Perrier, Emmanuel Lochin, Jean-Yves Tourneret, and Patrick Gélard. Attention networks for time series regression and application

to congestion control. In *The 4th International Workshop on Network Intelligence in conjunction with IFIP Networking*, 2022.

[34] Victor Perrier, Emmanuel Lochin, Jean-yves Tourneret, and Patrick Gélard. Réseaux récurrents d'attention pour la régression de séries temporelles. In *XXVIIIème Colloque Francophone de Traitement du Signal et des Images-GRETSI'22*, 2022.

[35] Victor Perrier, Emmanuel Lochin, Jean-Yves Tourneret, Nicolas Kuhn, and Patrick Gelard. How attention deep learning can improve copa congestion control performance. In *The International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2022.

[36] Adrian Perrig. Eth zurich course. `https://spcl.inf.ethz.ch/Teaching/2016-osnet/lectures/net_8_6s.pdf`.

[37] Michael Phi. Illustrated guide to lstm and gru. `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21`.

[38] Keith W. Ross and James F. Kurose. Tcp congestion control. `http://www2.ic.uff.br/~michael/kr1999/3-transport/3_07-congestion.html`.

[39] Michael Schapira. Ultimate guide to congestion control. `https://www.compiralabs.com/ultimate-guide-congestion-control`.

[40] Ashish Vaswani et al. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Conf. Advances in neural information processing systems*, volume 30, 2017.

[42] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems*, 33:21665–21674, 2020.

[43] Susan Wei and Marc Niethammer. The fairness-accuracy pareto front, 2020.

[44] Keith Winstein and Hari Balakrishnan. TCP ex machina: computer-generated congestion control. In *ACM SIGCOMM Conference*, pages 123–134, Hong Kong, August 2013. ACM.

[45] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *SIGCOMM Comput. Commun. Rev.*, 43(4):123–134, August 2013.

[46] Francis Y. Yan et al. Pantheon: the training ground for internet congestion-control research. In *USENIX ATC*, pages 731–743, Boston, MA, July 2018.