Thèse de doctorat

NNT: XXXXXXX

université PARIS-SACLAY

Deterministic scheduling of periodic datagrams for low latency in 5G and beyond

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)

Unités de recherche: (1) Université Paris-Saclay, UVSQ, Données et Algorithmes pour une ville intelligente et durable, 78035, Versailles, France.
(2) Nokia Bell Labs France, 91620 Nozay, France.
Référent: : Université de Versailles-Saint-Quentin-en-Yvelines.

Thèse présentée et soutenue à ....., le .... 2021, par

Maël Guiraud

Composition du jury:

(**Johanne Cohen** Directeur de Recherche , LRI	Président/e)
Prénom Nom Titre, Affiliation	Rapporteur
Prénom Nom Titre, Affiliation	Rapporteur
(**Safia Kedad-Sidhoum** Professeur, CNAM	Examinatrice)
Dominique Barth Professeur, UVSQ	Directeur de thèse
Yann Strozecki Maître de conférence, UVSQ	Coencadrant
Olivier Marcé Ingénieur de recherche, Nokia Bell Labs France	Coencadrant
Brice Leclerc Ingénieur de recherche, Nokia Bell Labs France	Examinateur

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Ordonnancement periodiques de messages pour minimiser la latence dans les réseaux dans un contexte 5G et au delà

Mots clés: Cloud Radio Access Network, Ordonnancement periodique, Heuristiques de recherche locale, Analyse de complexité, Réduction de la latence, Theorie des graphes

Résumé: Cette thèse est le fruit d'une collaboration entre les laboratoires DAVID et Nokia Bell Labs France. L'idée originale est de trouver des solutions algorithmiques pour gérer des flux periodiques de manière déterministe dans les réseaux afin de contrôler et de minimiser le temps de transmission, appelé latence. L'un des objectifs de la 5G (le C-RAN, pour Cloud Radio Access Network) est de centraliser les unités de calculs des antennes radio des réseaux de télécommunications (appelé Radio Access Network) dans un même centre de calcul (le Cloud). Le réseau entre le centre de calcul et les antennes doit être capable de satisfaire les contraintes de latence imposées par les protocoles.

Nous définissions le problème de trouver un ordonnancement periodique pour les messages de façon à ce qu'ils ne se disputent jamais la même ressource, et prouvons que les différentes variantes du problème étudiés sont NP-complets. Nous étudions dans un premier temps le problème pour une topologie particulière dans laquelle tous les flux partagent un même lien. Nous proposons dans un premier temps des algorithmes polynomiaux, de plus en plus evolués, ainsi que des algorithmes FPT peremettant de trouver une solution quand le nombre de route est raisonnable, ce qui est le cas des réseaux C-RAN.

Les algorithmes développés dans cette première partie n'étant pas applicables directement aux topologies plus générales, nous proposons ensuite une forme cannonique au problème qui nous permet de définir une notion de voisinage efficace pour des heuristiques de recherches locales (descente, recherche tabou, recuit simulé). Nous utilisons cette forme cannonique pour définir un algorithme Branch and Bound efficace quand le nombre de route est modéré. Nous proposons aussi une évaluation de performance des solutions proposés par rapport aux solutions courantes de gestion des flux, et montrons que notre modèle est réalisable en pratique grace aux nouveaux equipements en cours de développement.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Deterministic scheduling of periodic datagrams for low latency in 5G and beyond

Keywords: Cloud Radio Access Network, Periodic scheduling, Local search heuristics, complexity analysis, Latency reduction, Graph theory

Abstract: This thesis is the result of a collaboration between DAVID Laboratory and Nokia Bell Labs France. The original idea is to find algorithmic solutions to deterministically manage periodic flows in networks in order to control and minimize the transmission time, called latency. One of the objectives of 5G (C-RAN, for Cloud Radio Access Network) is to centralize the calculation units of the radio antennas of telecommunications networks (called Radio Access Network) in the same computer center (the Cloud). The network between the computing center and the antennas must be able to satisfy the latency constraints imposed by the protocols.

We define the problem of finding a periodic scheduling for messages so that they never compete for the same resource, and prove that the different variants of the problem studied are NP-complete. We first study the problem for a particular topology in which all the streams share the same link. We first propose polynomial algorithms of increased sophistication, and FPT algorithms that allow us to find a solution when the number of routes is reasonable, which is the case for C-RAN networks.

Since the algorithms developed in this first part are not directly adaptable to more general topologies, we then propose a canonical form to the problem which allows us to define an efficient neighborhood notion for local search heuristics (hill climbing, taboo search, simulated annealing). We use this canonical form to define an efficient Branch and Bound algorithm when the number of routes is moderate. We also propose a performance evaluation of the proposed solutions compared to current flow management solutions, and show that our model is feasible in practice thanks to new equipment under development.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Contents

List of Figures

Présentation de la thèse

Les travaux présentés dans cette thèse s'inscrivent dans le contexte du développement de la 5G, et sont plus particulièrement axés sur la réduction de la latence dans les réseaux cœur des opérateurs. L'un des objectifs pour la 5G est de garantir une latence bout en bout la plus faible possible. Réduire la latence dans les réseaux permet non seulement d'améliorer la qualité de service des utilisateurs, et ouvre également la porte au développement d'applications pour lesquelles le temps de réponse est critique (véhicules autonomes, industrie 4.0, ...). Le cas d'application que nous étudions est le Cloud Radio Access Network abrégé en C-RAN. Le but du C-RAN est de centraliser les unités de calcul situées aux pieds de chaque antenne dans un ou plusieurs centres de calcul communs, afin de faciliter la maintenance et de réduire les couts d'exploitation. Les antennes envoient périodiquement des messages aux centres de calcul, qui calculent une réponse et l'envoient aux antennes avec la même périodicité. Le temps écoulé entre l'envoi d'un message par une antenne et la réception de sa réponse doit être inférieur à une durée imposée par le protocole de communication radio. Ces messages envoyés sont très lourds, et utilisent donc beaucoup de bande passante dans les réseaux.

La méthode actuelle de gestion des réseaux, le multiplexage statistique, consiste à dimensionner chaque lien de façon à ce que le flux moyen de messages utilisant un lien puisse emprunter ce lien sans contrainte. Il arrive fréquemment que des flux envoient beaucoup de paquets d'un coup dans le réseau. Quand trop de paquets doivent utiliser un lien en même temps, on parle de contention. Les messages qui ne peuvent pas utiliser le lien directement sont mis dans une file d'attente, que nous appelons buffer de contention. Faire attendre les messages dans ces buffers de contention augmente la latence des paquets. Plus un réseau est chargé, plus il est probable d'avoir de hautes latences dues à la contention. Dans les réseaux C-RAN, les sources envoient periodiquement une grande quantité de messages nécessitant une garantie de faible latence, ils ne peuvent donc pas être gérés grâce au multiplexage statistique. C'est pourquoi nous proposons des solution de gestion deterministe et periodique des flux: le calcul d'un ordonnancement qui définit les dates de passage de chaque paquet dans chaque noeud du réseau. Les sources envoient périodiquement des paquets dans le réseau, toutes selon la même période, fixée par le protocole. Les ordonnancements que nous calculons garantissent l'absence de collision des paquets, quand

l'ordonnancement est répété à l'infini de manière périodique.

Plusieurs groupes de travail proposent aujourd'hui des solutions techniques (présentées dans le chapitre 7) pour aider à contrôler la latence dans les réseaux. Avec ces solutions, les équipements du réseau sont capables d'allouer les ressources de transmission pour certains flux à un instant donné. Des travaux au sein de Nokia Bell Labs visent à aller plus loin pour pouvoir réserver une partie des ressources à un temps donné pour un paquet donné. Il faut toutefois calculer les dates auxquels les paquets doivent arriver dans les nœuds du réseau. Cette thèse se concentre sur le fait d'organiser les paquets, de façon à ce qu'ils n'entrent en collision dans aucun des nœuds, afin de supprimer les buffers de contention. Se passer complètement des buffers de contention n'est pas toujours possible, notamment lorsque les réseaux sont composés de beaucoup de nœuds. Dans ce cas, l'objectif de nos travaux est de minimiser le temps passé par les paquets dans les buffers de contention. Il est important de souligner que dans ce cas-là, le temps d'attente dans les buffers de contentions ne sont plus subis comme pour le multiplexage statistique, mais contrôlés et prévisibles.

Nous modélisons un réseau par un multigraphe orienté acyclique pondéré dont les sommets représentent les points de contention entre messages dans le réseau. Les poids des arcs représentent le temps physique de transmission entre deux points de contention. Deux messages rentrent en conflit s'ils doivent passer par le même point de contention au même moment. Nous considérons que le routage est donné, et nous cherchons à organiser les messages de façon à ce qu'il n'y ai pas de conflit dans le réseau. Nous étudions dans un premier temps le problème sur des réseaux simples et courants, constitués de deux points de contention en série. Nous définissons le problème de décision consistant à choisir la date de passage de chaque message dans chacun de ces deux points de contention de façon à ce qu'aucun message n'ait de conflit avec un autre dans le réseau. Ce problème ressemble à des problèmes classiques d'ordonnancement mais l'envoi périodique de nos flux en fait un problème original et difficile. Nous prouvons dans le chapitre 2 que le problème est NP-complet, même sur des graphes orientés acycliques de faible degré ou de faible profondeur, par réduction de problèmes de coloration d'arcs ou de sommets. Nous proposons donc des heuristiques (algorithme gloutons, métaheuristiques) qui nous permettent de trouver de bonnes solutions en temps polynomial pour tout type d'instance, et des algorithmes FPT (de complexité exponentielle en le nombre de routes mais pas en les autres paramètres du problème) qui trouvent une solution optimale au problème et qui sont suffisemment rapides

sur les instances simples que nous étudions.

Nous étudions dans le chapitre 3 le problème de l'organisation de flux non-synchronisés dans un réseau sans aucun buffer. Même si pour l'instant les protocoles liés au C-RAN ne permettent pas de désynchroniser les antennes (ce qui pourrait être le cas pour de prochaines générations de réseaux mobiles), cette approche est applicable dans d'autres contextes, comme une usine où des robots ne nécessitant pas de synchronisation doivent communiquer rapidement avec un centre de contrôle. Nous cherchons à calculer un temps de départ des messages au début de leur route de façon à ce qu'ils ne soient pas en conflit avec les autres messages, sans que ce temps de départ ne soit considéré comme du temps de contention. Les solutions de ce problème sont toutes optimales en terme de latence: la latence des messages est égale au temps physique de transmission, car aucune latence n'est ajoutée aux messages à cause de la contention. Nous décrivons des algorithmes gloutons de plus en plus évolués visant à optimiser l'impact d'ajouter un message à la solution partielle calculée. Ces algorithmes nous permettent de garantir qu'une solution au problème existe quand la charge du réseau est inférieure à 40% (et même jusqu'a 61% pour des messages de taille 1). Nous proposons aussi un algorithme FPT (quand le problème est paramétré par le nombre de routes) qui nous permet de calculer la solution optimale en un temps raisonnable quand le nombre de routes est inférieur à 20. Nos résultats montrent que le problème ne peut pas être résolu sans buffer de contention quand la charge du réseau est supérieure à 80%. C'est pourquoi nous traitons dans le chapitre 4 le problème d'organiser les flux avec un buffer sur la route, de façon à offrir un plus grand degré de liberté. Nous étudions plus particulièrement le problème de minimisation, c'est-à-dire, trouver une solution qui minimise la latence maximale des routes. Nous proposons une approche en deux parties. Premièrement, nous choisissons les temps d'envoi des messages sur le premier point de contention, et nous résolvons dans un second temps le problème de choisir le temps d'attente de chaque message dans le second point de contention. Pour cela, nous décrivons un algorithme polynomial basé sur le problème d'ordonnancement classique de la littérature, adapté à notre cadre périodique. Nous proposons aussi un algorithme FPT basé sur le même principe, mais qui garantit de trouver la solution optimale. Nous montrons que nous sommes capables de trouver des solutions pour lesquelles la latence est minimale pour 99.9% des instances dans des réseaux très chargés, et que nos méthodes donnent des résultats excellents comparées au multiplexage statistique.

Dans le chapitre 5, nous étudions le problème d'organiser des flux synchronisés sur

tout type de DAG. Dans ce cas, tous les messages sont envoyés en même temps par les sources et nous nous permettons de faire attendre les messages dans des buffers à chaque point de contention du réseau. Nous étudions le problème de minimiser la plus grande latence dans le réseau. Nous commençons par décrire des algorithmes gloutons qui trouvent une solution réalisable pour n'importe quelle charge, qui servent de point de départ aux algorithmes de recherche locale utilisés ensuite. Nous introduisons une forme compacte du problème qui nous permet de définir une notion de voisinage entre les solutions afin d'explorer l'ensemble de ces dernières. Nous étudions les performances des algorithmes de recherche d'optimum local (hill-climbing, recherche tabou, recuit simulé) et nous proposons un algorithme Branch and Bound qui énumère l'ensemble des solutions sous forme compacte, en faisaint suffisemment de coupes pour trouver la solution optimale rapidement. Nous montrons expérimentalement que l'algorithme Branch and Bound est capable de trouver une solution optimale en un temps raisonnable pour 12 routes, tandis que le recuit simulé permet de trouver des solutions bien meilleures que le multiplexage statistique pour n'importe quelle taille d'instance.

Nous étudions ensuite dans le chapitre 6 l'impact de nos algorithmes d'ordonnancement, lorsque les flux C-RAN périodiques et prioritaires partagent le réseau avec des flux Best-Effort, non prioiritaires et dont les arivées suivent un processus stochastique. Nous proposons une méthode d'adaptation de nos algorithmes qui permet de lisser la charge des flux C-RAN tout au long de la période, sans augmenter la latence. Nos expériences montrent que, même si organiser les flux de façon déterministe comme nous le faisons requiert d'utiliser un peu plus de bande passante pour réserver les ressources, la latence moyenne des flux Best-Effort est meilleure qu'avec le multiplexage statistique. Nous montrons aussi le même genre de résultats dans un anneau optique ou l'ordonnancement des flux C-RAN est rendu trivial par les contraintes techniques de la conversion opto-électronique.

Toutes nos approches se basent sur des hypothèses techniques fortes : les flux doivent être parfaitement synchronisés, le réseau doit être intelligent et programmable. Le chapitre 7 fait le point sur les standards récemment développés qui se rapprochent de nos hypothèses. Nous montrons aussi les limites de ces standards, et nous introduisons un équipement en phase de développement qui nous permettrait de réduire la latence dans les réseaux au temps physique de transmission.

Introduction

A traduire depuis l'intro fr finie

Chapter 1

Algorithmic and industrial context

1.1 Industrial context

1.1.1 What is 5G ?

Telecoms networks must manage more and more users while continually improving their bandwidth, latency and reliability. Nowadays, 4G is the standard deployed in most of the territory, and 5G is the new technology under deployment. The term 5G defines a set of functional specifications. The organism that proposes these specifications is the ITU (International Telecommunication Union), an agency of the United Nation responsible for information and communications. For several years, the ITU-R (radiocommunication component of the ITU) has been working to determine the functional aspects that 5G must satisfy. Figure 1.1 from [1] illustrates some of those functional aspects, that ITU-R has formally referenced under the name IMT-2020 [2] (the requirements of 4G are referenced as IMT-advanced): a bitrate up to 20Gbps ($\times$20 compared to 4G), low end to end latencies down to 1ms (10 times lower than in 4G, we are focusing on this aspect in this thesis). Also, 5G aims to offer an higher connection density (up to 1 million device/km^2), with an higher traffic capacity (from 0,1$Mbit/s/m^2$ in 4G to 10$Mbit/s/m^2$ in 5G) thanks to a wider use of the spectrum. Other aspects as a better energy efficiency (100 times better in 5G than in 4G) or a better mobility are required.

All these characteristics allow various application cases. Figure 1.2 taken from [3] establishes a non-exhaustive list of them, according to their different technical constraints. One of them is **low latency**, required for applications like motion control that work in real time. Also, 5G aims to develop dynamic programmable networks, for greater flexibility of use. This thesis focuses on developing algorithm to dynamically manage flows in the network in order to provide low latency.

On the other hand, a higher bandwidth is useful for applications like video streaming,

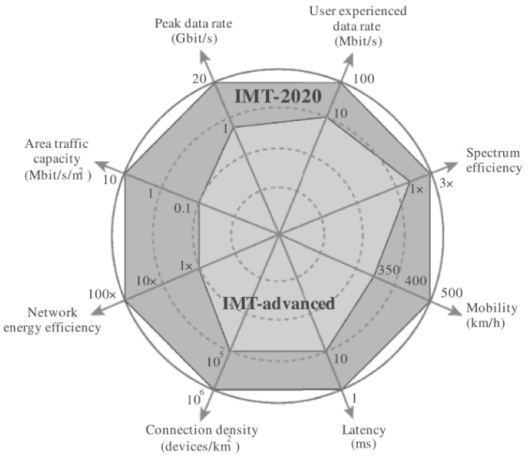

Figure 1.1 – 5G performances required by ITU-R ([1])

augmented reality or ensuring the connectivity of a large number of terminals. By relaxing latency and bandwidth constraints, it is possible to expand further the number of devices (up to 1 million) for applications like sensors networks.

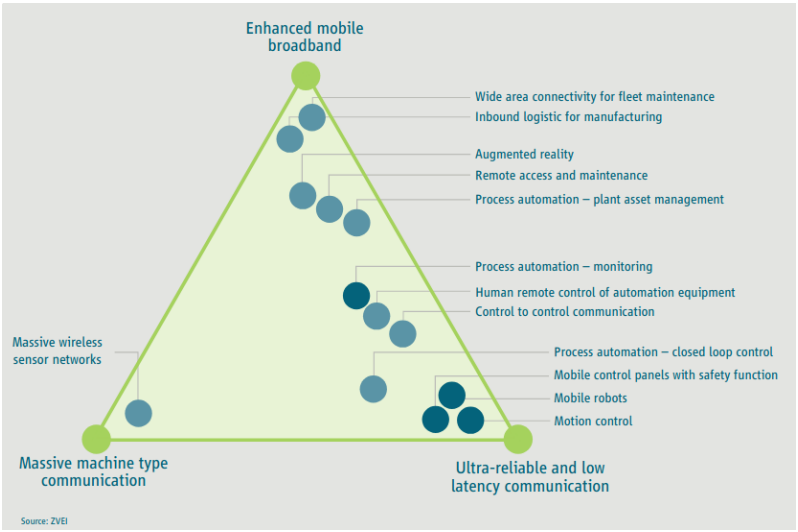

Figure 1.2 – Some examples of use cases for 5G ([3])

1.1.2 Which aspect of 5G do we focus on ?

To meet these 5G functional specifications, the network equipments must follow technical standards. The 3GPP (3rd Generation Partnership Project) is an union between several standard organizations which defines the technical specifications for 5G. 3GPP regularly publishes releases, that regroup new specifications. The first release focused on 5G was

Release 15 (R15), released in 2018 [4].

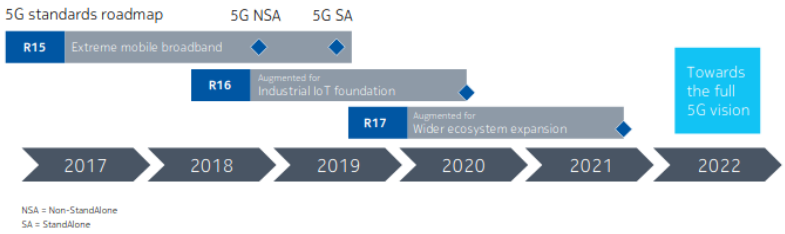

Figure 1.3 – 3GPP releases 15, 16 and 17 calendar ([4])

Release 15 focused on increasing throughput and interworking between 4G and 5G, and introduced the notion of URLLC (Ultra-Reliable Low-Latency Communication). URLLC consists in ensuring low packet loss and low latency communications. Indeed, several use cases (smart factories, control operations, ...) needs highly reliable communications in which the latency must be guaranteed. The objective of this thesis is to compute schemes for low latency communications. To do so, the network equipments must be able to manage the traffic by discriminating different kinds of flows and following computed scheduling to manage them. Even if current network does not yet have such capabilities, releases 16 an 17 have deepened the notion of URLLC and this topic is widely studied.

1.1.3 Current way to manage networks : Statistical Multiplexing

The constraints expressed for low latency architectures and 5G standard are hard to meet in current networks. In IP or even Ethernet networks, the traffic usually suffers of delay due to contention.

As we just mentioned, the current network nodes (routers for IP networks, switches for Ethernet networks) are not able to schedule packets. The only function of the nodes is to forward the packets to the right output port. The objective is to offer a good average quality of service for a minimal price. When a single input flow uses an output port, there is no issue. If several packets coming from several flows require the same output port at the same time, we talk about **contention**. Some packets are then put in a **contention buffer** until the port is available. The additional latency induced by contention buffers is one of the most important causes of delay. In order to avoid this situation, the bitrates of the links is dimensioned according to the use case. It is then calculated according to the average bitrate of the flows on the network. When there is too much packets in the buffer, the oldest packets of the buffer are **lost**.

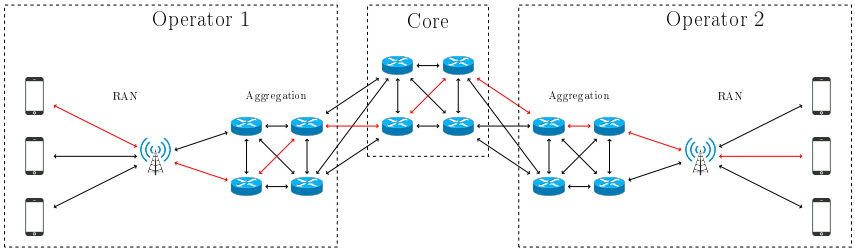

Figure 1.4 – An End to End communication between two mobiles.

Such an approach ensures an easy deployment and management of the networks and most of the time a good quality of service for a minimal cost for network providers, but it may induce packet losses and high latencies. This is called statistical multiplexing [5, 6].

URLLC aims to ensure good end to end latency communications. To achieve such a goal, each component of the communication network must satisfy a low latency: the radio communications, and the core network. This thesis focuses on the core network.

1.1.4 Radio Access Network

To understand the network this thesis focuses on, we now describe how a radio access network works.

Current mobile network (aka cellular network) architecture consists in a distributed radio access networks: the mobile terminals connect to a base station (BTS for Base Transceiver Station as a generic name, eNB for evolved Node B in 3GPP LTE "4G" standard or gNB for 5G) that encompasses all the sub-systems needed to realize mobile communication [7]. It mainly comprises the radio part, that furnishes the connection between the mobile terminal and the BTS, and the network part that provides control and management functions like mobility support (the main functionality being the support of handover from one BTS to another, i.e. the ability to pursue a communication when moving from the range of an antenna to another). The BTS are connected together by the Aggregation Network of the operator, itself connected to the core network, in order to ensure communications with other operators Radio Access Networks(RANs). Figure 1.4 illustrates a communication between two mobiles using a different operator.

1.1.4.1 Cloud RAN

One possible direction for next generation networks is to become centralized radio network architectures (C-RAN, for Cloud Radio Access Network) to reduce consumption costs and

power at the base stations [8]. These C-RAN architectures include simplified base stations on the field. Depending on the architecture choice, it can be restricted to the radio part and the digital to analog conversion only. This can be identified by different names depending on the reference document, including **RU** for Remote Unit or **RRH** for Remote Radio Heads. The latter will be used in the thesis. The other components of the C-RAN are the processing units. One can distinguish two levels of processing units: the **DUs**, for Distributed Units that are able to ensure only a part of the computation tasks, and the **CUs** for Centralized Units that computes the most centralized tasks. In this thesis, we consider only CUs, that are also called **BBUs** for BaseBand Units, and we use this term in the thesis. The BBUs are located in the cloud.

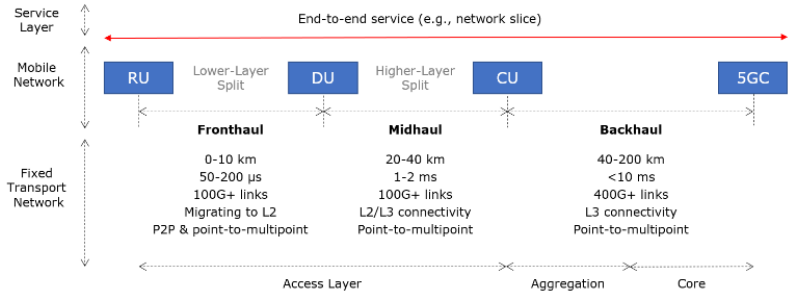

Figure 1.5 – Latency requirements between different part of the network.

The cloud is defined as the execution of a program in a data center (gathering the BBUs) ordered by another machine (RRHs) connected to the data center through a transparent network. The execution may be indifferently performed on virtualized machines, or bare metal, or any combination of the two. The network between RRHs and BBUs is called "Fronthaul Network", or "Fronthaul" for short. Figure 1.6 illustrates an example of fronthaul in which several BBUs are gathered in the same datacenter.

1.1.4.2 Split

As mentioned above, in C-RAN, most of the computation tasks of the BTS must be centralized in the BBU. There are several components that can be centralized, but the more we centralize the ressources, the higher the latency constraints are. Figure 1.7 illustrates two different choices of split for the BTS. The first one, called "Full centralization" leaves only the radio functions to the RRH, while the second one, called "partial centralization", keeps the baseband processing function inside the RRH. The methods presented in this thesis allow to send an high amount of data in the network while minimizing latency. Such

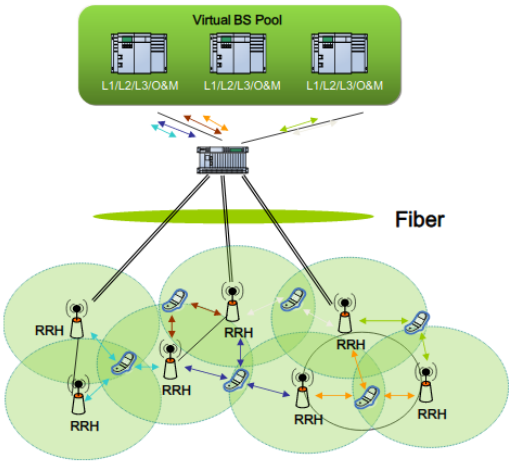

Figure 1.6 – An example of fronthaul network for Cloud RAN

a result matches well with the first split, this is why we talk about BaseBand Units (BBU) here. Even if we choose to work with the most critical split, the amount of data is just a parameter of the problem presented here, and any kind of split could be managed by our algorithms.

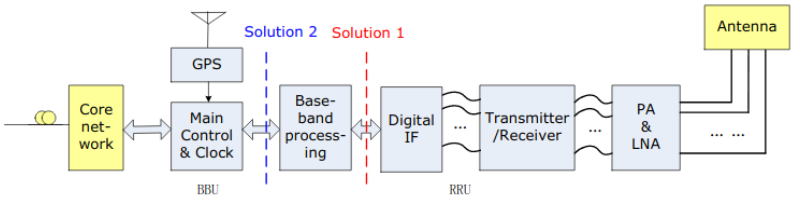

Figure 1.7 – Two different split for Cloud-RAN

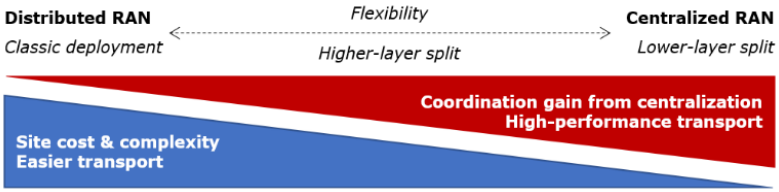

Figure 1.8 – The more centralized is the RAN, the hardest is the transport and the lowest is the cost of coordination for the antennas

This kind of architecture must address the problem of the latency in the transfer process between the RRHs on the field and BBUs in the cloud. Low latency is already critical for the deployment of C-RAN approach in LTE "4G" networks. The standard requires hard time constraints for functions like HARQ (Hybrid Automatic Repeat reQuest) that needs

to be processed in less than $3ms$ [7]. Considering processing time into the BBU, the time budget over the network can be as low as $400\mu s$ for a round trip. One specificity in this C-RAN context is not only the latency constraint, but also the periodicity of the data transfer between RRH and BBU (this HARQ constraint must be enforced for each frame emitted every millisecond). Looking beyond current mobile network generation, one must have in mind that upcoming 5G standards will require to reach end-to-end expected latency as low as $1ms$ (depending on targeted services) [9]. New scheduling and new technologies have to be considered to guarantee delay constrained periodic data transfers.

1.1.5 Technical solution for low latency

Statistical multiplexing is the most common mechanism used to manage packet based networks in the last 40 years. While tools [10] ensure a latency lower than a given value for 95% of the packets, such a guarantee is not sufficient in our context in which all packets must satisfy latency constraints. Indeed, mechanisms like Express Forwarding [11] can be used to prioritize some packets over the others, but they fail to guarantee the delivery of a given packet in a given time delay when several packets compete for the same resource.

The best current solution is to rely on an almost full optical approach, where each end-point (RRH on one side, BBU on the other side) is connected through direct fiber or full optical switches [12, 13]. This architecture is very expensive and hardly scales in the case of a mobile network. As illustrative purpose, a single (one operator) mobile network in France is composed of about 10,000 base stations. This number will increase by a factor of 2 to 20 with the emergence of "small cells" which increases base station density to reach higher throughput [14, 15]. It is thus needed to find a solution to offer low latency over commoditized packet based networks.

Although 3GPP standards for 5G are not completely frozen yet, the core network is designed to use ethernet technology. Time Sensitive Networking (TSN) [16, 17] is a task group of IEEE that develops some standards for ethernet. We focus on several of those standards which allow a control of the latency. The model and the algorithms of this thesis make the hypothesis that the network components are detailed, able to collect some information and send it to a centralized entity, to differentiate several kinds of flow and to forward it at an exact date, imposed by the controller. TSN standards propose technical solution for those hypothesis, but an approach like TSN is still based on statistical laws and is limited to ensure a perfect control of the latency. TSN technologies ensure an end-to-end

latency bounded for all packets, but not minimal, as we propose. The limits of TSN and the technical solution we propose are detailed in Chapter 7.

In this thesis, we work on deterministic flows. Thus, we propose algorithms to compute deterministic scheduling of the flows in the network, while minimizing the latency due to buffering. Remark that minimizing the buffering latency allows not only to meet latency constraints of applications, but also to leave additional time for others sources of latency (additional computations, longer length of fibers, etc...). When computing a scheduling, we must take into account the **periodicity** which makes the problem difficult to solve. Not only a datagram must not collide with the datagrams sent by the others BBU/RRH in the same period, but also in the other periods.

1.2 Algorithmic related works

We first describe the algorithmic problems studied in this thesis, see Chapter 2 for details. We consider a network in which the routing is fixed. Several flows share the network, sending periodically a message of the same size from a source to a target. This message represents several packets in practice, but in our model we consider them contiguous. The messages can also be buffered in the nodes of the network in order to let another message use a shared ressource. The objective is to compute the buffering time of every message in every node of the network, such that the global latency (i.e. the largest latency of a flow) is minimal. Note that the periodic aspect of C-RAN makes the problem more difficult. Indeed, since the traffic is periodic and the network highly loaded, we must deal with contentions coming from successive periods while computing the scheduling. The problem we address is to compute periodic schedulings. This means we compute the solution on one period and the scheduling remains the same for every period. There are several variations of the problem according to the shape of the network, the synchronization of the messages in a period or the constraints on buffering.

We present in this section several families of problems and approaches, which are close to what is studied in this thesis but which mostly fail to model our problem faithfully or which are too general to derive useful algorithms.

1.2.1 Algorithmic Approaches

Wormhole Because we consider contiguous messages, we first focused on wormhole problems [18, 19]. Wormhole problems consider graphs representing interconnection networks, in which the vertices are the nodes of the network and the edges are the physical links. Long messages are sent in the network, using ressources during a long time. A *deadlock* occurs when several messages are waiting for each others to release a ressource. The main problem consists in avoiding deadlocks. The algorithmic solution proposed consists in multiplexing the physical channels in several virtual channels, and to develop routing algorithms to determine the virtual channel used by each message. The problem is very similar, but we do not have the same degrees of freedom: in our network, the routing is given. Moreover, our messages require the entire capacity of the links and we cannot use multiplexing, except in a variation of our model presented in Chapter 6. Furthermore, deadlocks can not occur in the networks we model, since the considered routings are **coherent**[20]. A routing is coherent if two routes share a single path (i.e. a sequence of contiguous links) in the network.

On a technical aspect, wormhole switches [19] are designed to read only the header of the messages before forwarding it instead of buffering the entire messages as in store-and-forward [21]. This method has a huge impact on the latency, particularly on long messages, and we go further by trying to remove all buffering in the switches.

Graph coloring for ressources allocation One of our approach consisted in representing the shared ressources of the network by a conflict graph. The vertices of the graph represent the routes of the networks and there is an edge between two vertices if their associated routes share the same ressource (i.e. an output port on a switch). The edges of the graph are labeled by the difference of the physical length of the links between the sources of the routes and the conflict point. A proper coloring of such a graph is a scheduling of the network without collisions between the messages. Several graph colorings have been introduced to model similar problems. In [22] the objective is to minimize the number of wavelength in shared links for optical networks. Flow allocations problems are studied in [23], applied to the allocation of the frequencies in radio networks. Unfortunately, they do not take into account the periodicity of the scheduling and the associated problems are already NP-complete.

Circular Coloring The only coloring taking into account periodicity is the circular coloring [24, 25]. As an example, circular coloring can model a road intersection as a graph in which the vertices are the flows and there is and edge between two vertices if the flow must not overlap. The problem is to find a proper vertex coloring of the graph. Such an approach is close to our problem of coloring conflict graphs. However, the models presented do not consider the weight on the arcs, and cannot easily capture general graphs. Also, because the problem is already NP-hard , this approach did not catch our attention. Circular arc coloring [26] also manages jobs that can be related to periodic messages. The authors consider a clock and several arcs around this clock, representing the time needed by a job. The objective is to find the minimal number of men needed to complete the job. If two arcs overlap, the same man cannot be affected to both jobs. Nevertheless, this problem has been shown NP-hard [27] and the model is really far from ours.

1.2.2 Scheduling approaches

Train Scheduling The train timetabling problem [28] and its restriction, the periodic event scheduling problem [29] are generalizations of the problem we study. Indeed, they take the period as input and can express the fact that two trains (like two messages) should not cross. However, they are much more general: the trains can vary in size, speed, the network can be more complex than a single track and there are precedence constraints. Hence, the numerous variants of train scheduling problems are very hard to solve (and always NP-hard). Most of the research done [28] is devising practical algorithms using branch and bound, mixed integer programming, genetic algorithms...

Linear Programming for Latency Constrained Network We define in this thesis a simple network topology called the *star shaped network*. In star shaped networks, all flows go through the same link, and there is only two relevant contention points (one in the way to the BBUs, and one in the way back to the RRHs).

Variation on the problem of scheduling periodic messages for Cloud-RAN have been investigated in [30, 31, 32, 33]. In these papers, authors study the practical problem of scheduling a few number of flows in a star shaped network. Given a network in which the routing is set, the objective is to schedule several periodic flows with a critical latency and several best-effort flows, as we do in Chapter 6. To do so, the authors use new technical standards (TSN, detailed in Chapter 7) that allow to prioritize flows, and linear program-

ming in order to compute a schedule between the critical flows. In the same spirit, the use of an SMT solver rather than linear programming is proposed in [34]. The flows described in these papers are different from ours. While we consider that a single long message is sent by a source every period, the authors propose flows in which several little packets are sent. The scheduling are computed on multiple periods while we compute a scheduling on one period, which can be repeated periodically. Furthermore, the experiments are made on small topologies, because these approaches does not scale neither with the number of routes nor the number of conflict points on each route, while we propose polynomial time algorithms that give satisfying solution for every kind of topology. However, this kind of approach, as explained in [31], can be used as a standardization tool to verify the viability of solutions computed by faster algorithms.

A polynomial algorithm for single processor scheduling The problem we address in this thesis is similar to classical single processor scheduling problems, defined as follows. Given a set of jobs, a release time and a deadline for each job, find a scheduling to minimize the global completion time (i.e. the time at which all jobs have been processed). The approach we adopt in Chapter 4 to solve our problem on a star routed network, is a two stage approach. The second stage, when forgetting about periodicity, is similar to a scheduling problem with a single processor with the goal of minimizing the completion time (makespan). The algorithm described in [35], solves this problem in polynomial time when all tasks have the same processing time. We use it as a building block of several of our algorithms in Chapter 4 and Chapter 5, where we adapt it to the case of periodic jobs. When the jobs have different processing times, the problem is NP-hard [36].

Two flow shop scheduling On the star routed network studied in Chapter 3, the problem of scheduling messages for Cloud-RAN without buffers nor periodicity is similar to a two flow-shop scheduling problem. In two flow-shop scheduling, the objective is to schedule the jobs on two processors, each job must be first processed by the first processor and can then be processed by the second one after a delay which depends on the job. This two-flow shop problem is NP-hard [37], and while it can be reduced to the synchronized version of our problem, presented in 5, the correspondence is less clear with the unsynchronized version of Chapter 3. Moreover, our problem adds the constraint of periodicity, hence no algorithm for two flow shop scheduling can be used as is.

The problem of scheduling tasks on multiple processors periodically does not seem

related to our problem. In [38] the problem is the following: Given a set of tasks, with for each of them a duration and a period, find a periodic scheduling minimizing the number of processors on which the periodic tasks are scheduled. The model is different from ours because we consider messages of same size and period, and we want to schedule all messages on the same ressource.

In general, the problem of cyclic scheduling [39, 40] have been extensively studied. The problem has many variants, considering or not precedence constraint, different message sizes, or different periodicity for each message, etc... Most of these problems have been shown to be NP-hard. In all these problems, the objective remains the same: minimizing the period. The main difference, which makes methods for cyclic scheduling quite different from our approach, is that the period is an input of our problem, which cannot be modified. While cyclic scheduling tries to optimize the throughput by changing the period, we try to optimize the latency for a fixed period.

Conclusion

One of the trend of 5G and networks in general is to be able to ensure end to end communication with a low latency. To do so, it is important to reduce the latency sources in each part of the network. In this thesis, we focus on the packet switched network connecting the radio antennas to the operator's core network. In packet switched networks, the major source of latency is the buffering latency due to contention. Current approach of network management consists in multiplexing the flows on a network dimensioned following statistical laws, and this may induce high contention buffering time. In order to reduce the contention, several technical solutions are currently in development to control critical traffics more precisely than statistical multiplexing. With those solutions, the switches follows all over the time a precise scheduling of each flow it must forward at each date. In this thesis, we study the problem of computing this scheduling. Knowing the topology of the network, and the different flows it supports, our objective is to compute the scheduling of the traffic for every node of the network, while minimizing the latency of the flows. Because we consider applications sending all over the time a datagram periodically, the scheduling must be periodic. Scheduling problems for networks have been extensively studied in wormhole routing, train scheduling or frequency allocation, but none of these studies deal with the periodicity of the flows we want to schedule, which makes the prob-

lem innovative and difficult. Circular scheduling or coloring problems seems related to our problem but the model are too far from ours, and the problem are not the same. Single processor scheduling, with makespan constraint can be adapted for periodicity to build some of our algorithms.

Chapter 2

Model, Problems

The model and the proofs presented in this chapter has been introduced in [41] and extended in a long version of the paper in [42].

2.1 Model

Let $[n]$ denote the interval of n integers $\{0,\ldots,n-1\}$.

2.1.1 Routes and Contention Points

We study a communication network constituted of pairs of vertices between which messages are sent periodically. The routing between each pair of such nodes is given: a **route** is a sequence of vertices $(s, c_1, \ldots, c_l, t)$. A vertex appears only once in a route, that is there is no loop in a route. Each vertex c_i corresponds to a contention point, that is the beginning of a link of the communication network shared by several routes. Hence, all vertices appear in several routes, except s, the first vertex of the route, and t, the last vertex of the route, which are exclusive to the route and represent the source and the target of the message. When modeling a C-RAN network, the first vertex represents the sending of the message by the RRH and the last vertex represents the same RRH that receives the answer the BBU has sent back.

The set of routes is denoted by $\mathcal{R}$. A route is interpreted as a directed path in a directed multigraph constituted of all routes, where the sets of arcs of the routes are disjoint. The routes contain no loop nor cycle, since all vertices of a route are different. Thus, the directed multigraph is acyclic. An arc in the multigraph may represent several physical links or nodes of the modeled network, which do not induce contention points.

Each arc (u,v) of a route r is labeled by an integer weight $\omega(r,u)$. It represents the time elapsed between the sending of the message of the route r in u and its reception in v. The

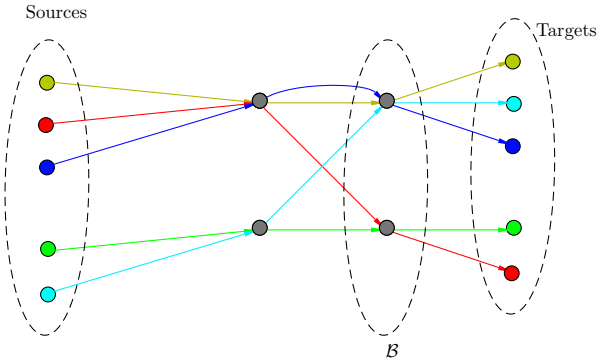

Figure 2.1 – A routed network, each route is represented by a colored path

weight of a vertex u_i in a route $r = (u_0, \ldots, u_l)$ is defined by $\lambda(r,u_i) = \sum_{0 \leq j < i} \omega(r,u_j)$. It is the time needed by a message to go from the first vertex of the route to u_i. The **length** of the route r is defined by $\lambda(r) = \lambda(r,u_l)$.

On each route, we can buffer the message only in the BBU. Since the BBU does not correspond to a contention point, we identify the BBU with the next contention point in the route. The set of these contention points with possible buffering is denoted by $\mathcal{B}$. Each route have thus only one vertex in $\mathcal{B}$. A **routed network**, which models the telecommunication network, is a triple $N = (\mathcal{R}, \mathcal{B}, \omega)$, see Figure 2.1 for an example.

2.1.2 Dynamic of Datagrams Transmissions

In this thesis, we consider a discretized time. The unit of time is called a **tic**. This is the time needed to send an atomic data in a link of the network. We assume that the speed of the links is the same over all the network. We are developing a prototype of this work based on ethernet base-X [43], see Chapter 7, using standard values for the parameters of the network: the size of an atomic data is 64 bits, the speed of the links is 10Gbps, and the length of a tic is thus about 6.4 nanoseconds.

In the process we study, a message, called a **datagram**, is sent on each route from the source node. The **size** of a datagram is an integer, denoted by τ, it is the number of tics needed by a node to emit the full datagram through a link. In this thesis, we assume that τ is the same for all routes. It is justified by our application to C-RAN, where all source nodes are RRHs sending the same type of message. There is no fragmentation: Once a datagram has been emitted, it cannot be fragmented during its travel in the network.

Let $r = (s, \ldots, t)$ be a route. In order to avoid contention, it is possible to buffer datagrams in the contention points in $\mathcal{B}$. An **assignment** A of a routed network $N =$

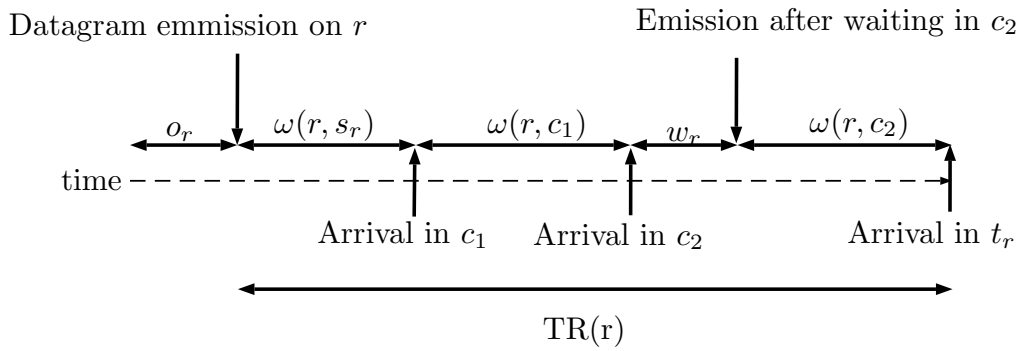

Figure 2.2 – Timeline of a datagram during its travel on a route $r = (s_r,c_1,c_2,t_r)$, with $c_2 \in \mathcal{B}$

$(\mathcal{R}, \mathcal{B}, \omega)$ is a function which associates to each route $r \in \mathcal{R}$, the pair of integers $A(r) = (o_r,w_r)$. The value o_r is the **offset**, the time at which the datagram is available in the first vertex of r. The value w_r is the **waiting time**: the datagram is buffered for w_r tics in $u_j \in \mathcal{B} \cap r$, the vertex representing the BBU. The **arrival time** of a datagram in the vertex u_i of r, is the first time at which the datagram sent on r reaches u_i, and is defined by $t(r,u_i) = \lambda(r,u_i) + o_r$ if $i < j$ and $t(r,u_i) = \lambda(r,u_i) + o_r + w_r$ otherwise.

Let u_l be the last vertex of the route r, the **transmission time** of the datagram on r is denoted by $TR(r,A)$ and is equal to $\lambda(r) + w_r$ or equivalently $t(r,u_l) - o_r$. This is the total time taken by the process we study: the sending of the datagram from the RRH to the BBU and the return of the answer back to the RRH. We can decompose this time into $\lambda(r)$, the *physical latency* of the process and w_r, the *logical latency*. We define the **transmission time** of an assignment A as the worst transmission time of a route: $TR(A) = \max_{r \in \mathcal{R}} TR(r,A)$. Figure 2.2 represents the different events happening during the lifetime of a datagram sent on a route r.

2.1.3 Periodic Emission of Datagrams

In the previous section, we have explained how *one* datagram follows its route. However, the process we model in this thesis is *periodic*: for each period of P tics, a datagram is sent, from each source node in the network, at its offset. The process is assumed to be infinite, since it must work for an arbitrary number of periods. For a given route, we use the same offset and waiting time in all periods, for simplicity of implementation in real networks and to make our problem more tractable from a theoretical perspective. Hence, at the same time of two different periods, all datagrams are at the same position in the network:

the assignments built are themselves periodic of period P. Thus, we only need to consider the behavior of the datagrams on each node of the network during a single period, and to apply the same pattern to every subsequent period. Using a different offset for each route corresponds to sending their datagram at a different time in the period. This matches our hypothesis that the emissions of the RRHs need not to be synchronized but they share a common global time, useful for coordination of their emissions.

Let A be an assignment of a routed network $N = (\mathcal{R}, \mathcal{B}, \omega)$. Let us denote by $[r,u]_{P,\tau}$, the set of tics used by a datagram on the route r at vertex u in a period P, that is $[r,u]_{P,\tau} = \{t(r,u) + i \mod P \mid 0 \leq i < \tau\}$. This set of tics depends on A, but A is omitted in the notation, since it is always clear from the context. Let us consider two routes r_1 and r_2, they have a **collision** at the contention point u if and only if $[r_1,u]_{P,\tau} \cap [r_2,u]_{P,\tau} \neq \emptyset$. The assignment A is said to be **valid** if, for all contention points u and routes r_1 and r_2 containing u, r_1 *and* r_2 *have no collision* at u. The validity of an assignment depends on P the period and τ the size of the datagrams, thus we say that A is a valid (P,τ)-**assignment**. When P and τ are clear from the context, we denote $[r,u]_{P,\tau}$ by $[r,u]$ and say that A is a valid assignment.

Figure 2.3 illustrates two valid periodic assignments for different values of P and τ, but the same network. The three routes are depicted by three different colors. If we let $P = 2$ and $\tau = 1$, then there is a (2,1)-periodic valid assignment with waiting times zero by taking 0 as offset for each route. However, for the same routed network but $P = 5$ and $\tau = 2$, there is no solution to the problem with waiting times zero. If we allow 1 tic of waiting time for one route, we can build the valid assignment ((0,0),(2,1),(0,0)).

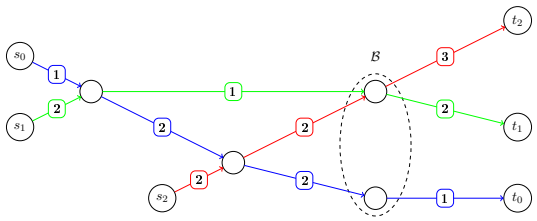

Figure 2.3 – A routed network with ((0,0),(0,0),(0,0)) as a (2,1)-periodic valid assignment and ((0,0),(2,1),(0,0)) as a (5,2)-periodic valid assignment

2.1.4 Periodic Assignment for Low Latency

The period P, as well as the size of a datagram τ are fixed in our C-RAN settings, but not the buffering policy. Hence, the aim of this section is to find a valid assignment which

minimizes the worst latency of the transmissions over the network, that is $TR(A)$. We denote by MINTRA the problem of finding the minimal value of $TR(A)$, for a given period, datagram size and routed network. For simpler hardness proofs and easier reductions, we rather study the decision version of MINTRA, that we call PALL for **P**eriodic **A**ssignment for **L**ow **L**atency. Each route must respect a time limit called a *deadline*. These limits are encoded in a deadline function d, which maps to each route r an integer such that $TR(r,A)$ must be less than $d(r)$. We define the **margin** of a route r in a routed network N with deadline function d as $d(r) - \lambda(r)$. The margin is a bound on the waiting of a route in a valid assignment.

Periodic Assignment for Low Latency

Input: A routed network N, the integers P, τ and a deadline function d.

Question: Does there exist a (P,τ)-periodic assignment of N such that for all $r \in \mathcal{R}$, $TR(r,A) \leq d(r)$?

In the next section, this problem is proved to be NP-hard. In Chapter 4, we propose heuristics solving the search version of PALL (computing a valid assignment), also denoted by PALL for simplicity. In the definition of PALL, we have chosen to bound the transmission time of each route, in particular we can control the worst case latency. It is justified by our C-RAN application with hard constraints on the latency.

We say that an assignment is **bufferless** when the waiting time of all routes are zero. The assignment can then be seen as a function from the routes to the integers (the value of the offset, the waiting time is omitted). We consider a restricted version of PALL, requiring to find a bufferless assignment and studied in Chapter 3. This is equivalent to using the deadline function $d(r) = \lambda(r)$, that is the transmission time must be equal to the size of the route, which implies $w_r = 0$ for all $r \in \mathcal{R}$. This problem is called **P**eriodic **A**ssignment for **Z**ero **L**atency and is denoted by PAZL. Studying PAZL is simpler: in an instance, there is no need to precise $\mathcal{B}$ in the routed network nor the deadline function and a solution is just an offset for each route. Moreover, a solution to PAZL is more efficient when implemented in real telecommunication networks, since we do not need to deal with any contention buffer at all. A switch taking full advantage of the absence of buffer is presented in Chapter 7.

An unusual property of assignments is that given a routed network and a deadline, we may have a (P,τ)-periodic assignment but no (P',τ)-periodic assignment with $P' > P$: the existence of an assignment is not monotone with regard to P.

Proposition 1. *For any odd P, there is a routed network with a (2,1)-periodic bufferless*

assignment but no $(P,1)$*-periodic bufferless assignment.*

Proof. Let us build N, a generalization of the routed network given in Figure 2.3. Let n be an integer, the vertices of the routes are the $v_{i,j}$, v_i^1 and v_i^2, with $0 \leq i < j < n$. There are n routes denoted by r_i, for $i \in [n]$. The route r_i is equal to $(v_i^1, v_{i,1}, \ldots, v_{i,n-1}, v_i^2)$. The weights of the arcs are set so that $\lambda(r_i, v_{i,j}) - \lambda(r_j, v_{i,j}) = P$, where P is an odd number smaller than n. It is always possible by choosing appropriate values for $\omega(r_i, v_{i,j-1})$ and $\omega(r_j, v_{i-1,j})$. In such a graph, there is no (P,τ)-periodic assignment with zero waiting time, since the problem reduces to finding a P-coloring in a complete graph with $n > P$ vertices, the colors being the offsets of the routes.

If we consider a period of 2, for all $i \neq j$, $\lambda(r_i, v_{i,j}) - \lambda(r_j, v_{i,j}) \mod 2 = 1$, hence two datagrams of same offset and size 1 do not have a collision at $v_{i,j}$. Therefore, the bufferless assignment defined by $A(r_i) = 0$ for all $i \in [n]$ is a valid (2,1)-periodic assignment of N. □

Let us introduce a few parameters quantifying the complexity of a routed network. The **contention depth** of a routed network is the number of vertices of the longest route of the network minus two. It is the number of contention points on the route with the most vertices on the network, since the first and the last vertex are private to the route. The **width** of a vertex is the number of routes which contains it. By definition, the first and last vertex of a route are of width 1, while all other vertices are of width at least 2 (otherwise they can be removed). The **contention width** of a routed network is the maximal width of its vertices. Remark that a (P,τ)-periodic assignment of a routed network must satisfy that P/τ is larger or equal to the contention width. Now, let us fix P and τ, for a given vertex of contention width c, we define its **load** as $c\tau/P$. It represents the proportion of the period used by datagrams at this contention point. The load of the routed network is the maximum of the loads of its vertices. A routed network must have a load less or equal to one to admit a valid assignment.

2.1.5 The Star Routed Network

In this section, we define a family of simple routed networks modeling a Multipoint-to-Multipoint fronthaul (see figure 2.4), which has been designed for C-RAN [13]. Let $N = (\mathcal{R}, \mathcal{B}, \omega)$ be a routed network, we say it is a **star routed network** if and only if the routes are $\{r_0, \ldots, r_{n-1}\}$, r_i is (s_i, c_1, c_2, t_i) and $\mathcal{B} = \{c_2\}$ (datagrams can wait in c_2). Star routed networks have contention depth two but a maximal contention width of n. The load on

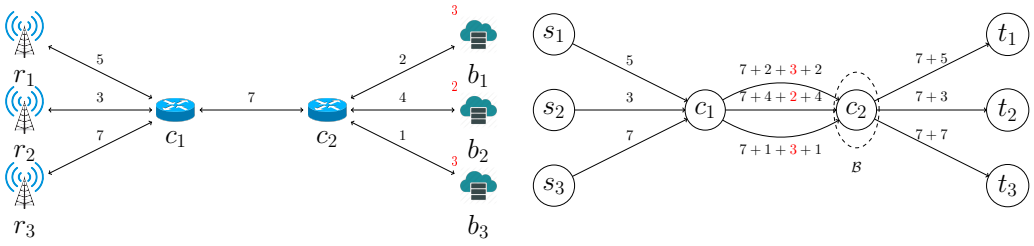

Figure 2.4 – Left, a physical fronthaul network and right, the star routed network modeling a round trip in the fronthaul network. The computation time in the BBU is given in red.

each of the two contention points is thus $n\tau/P$.

The fronthaul network we model with star routed network has a single shared link, which connects all RRHs at one end and all BBUs at the other end. The links are all *full-duplex*, meaning that the datagrams going from RRHs to BBUs do not interact with those going in the other direction. This property does not need to be enforced in our theoretical modeling, but it matches real fronthaul network and we will use such examples for our experiments. The two contention points c_1 and c_2 model the beginning of the shared link (used to go from the RRHs to the BBUs) and the other end of the shared link (used in the other direction). The computation in the BBU of an answer to a datagram on the route r takes some time. In the star routed network, this time is encoded in the weight of the arc between c_1 and c_2 in r. The weight $\omega(r,c_1)$ is the time needed to go through the shared link, then to arrive at the BBU, plus the computation time and the time to return to the shared link, see Figure 2.4.

Star routed network may seem simplistic, but every network in which all routes share an arc and satisfy a coherent routing condition can be modeled by a star routed network. It is common in fronthaul networks, since often all the BBUs are located in the same data-center. In such a situation, we can see the weights of the arcs (c_1,c_2) either as all equals (in that case PAZL is trivial, see Chapter 3) or different due to the structure of the network inside the data-center and the various hardwares used for the BBUs.

When solving PALL or PAZL on a star routed network, a period, a datagram size and a deadline function are also given. When the period is fixed, if we can modify the deadline function, we can do several assumptions on the parameters of the star routed network without loss of generality. We say that a star routed network is **canonical**, for a period P, if the weights of the arcs between c_1 and c_2 are in $[P]$ (because the computations are made modulo P) and the others are equal to zero. Hence, $\lambda(r_i)$, the length of a route is

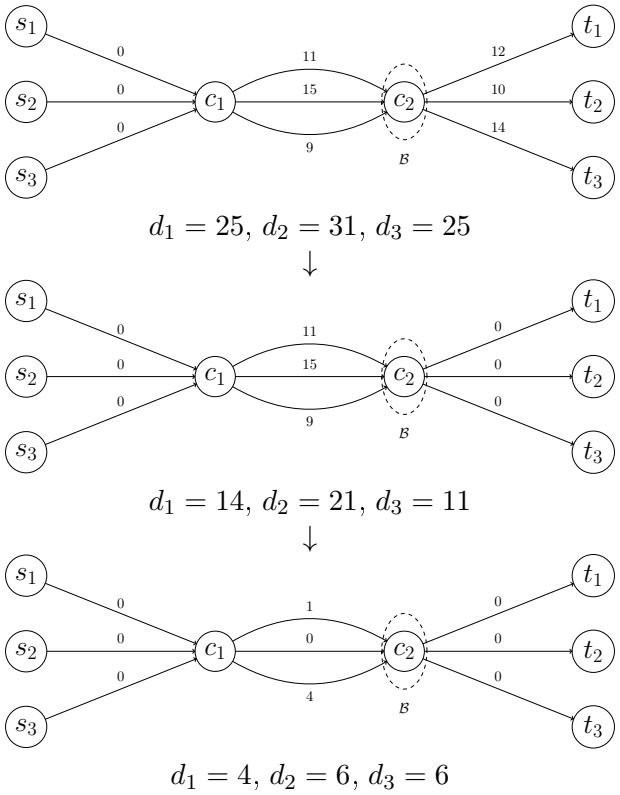

Figure 2.5 – Transformation by the proof of Proposition 2 of the star routed network of Figure 2.4 to its canonical form, initially with $\tau = 1$, $P = 5$, $d_1 = 30$, $d_2 = 34$, $d_3 = 32$.

equal to the length of its arc (c_1,c_2). Moreover, $\lambda(r_0) = 0$. See Figure 2.5 for an example of the canonical star routed network of Figure 2.4.

Proposition 2. *Let $I = (N, P, \tau, d)$, with $N = (\mathcal{R}, \mathcal{B}, \omega)$ a star routed network, then there is $I' = (N', P, \tau, d')$, with $N' = (\mathcal{R}, \mathcal{B}, \omega')$ a canonical star routed network, such that:*

$$I \in \text{PALL} \Leftrightarrow I' \in \text{PALL} \text{ and } I \in \text{PAZL} \Leftrightarrow I' \in \text{PAZL}$$

Proof. Let us define ω' and d' from ω and d in such a way that there is a bijection between valid assignments of I and I', which proves the proposition. In this bijection, the offsets o_i for an assignment of I will be mapped to o'_i, while the waiting times remain the same.

The routed network N' is equal to N except for the weight function ω'. We set the weights of the arcs (s_i,c_1) to zero in N'. We obtain the bijection between valid assignments of I and I' by setting $o'_i + \omega(r_i,s_i) = o_i$ and $d'(r_i) = d(i) - \omega(r_i,s_i)$. The weights $\omega'(r_i,c_2)$ are also set to 0, it does not change the possible collisions for an assignment but it changes the transmission time, hence we set $d'(r_i) = d'(r_i) - \omega(r_i,c_2)$ to preserve the bijection

between assignments of I and I'.

We let $\omega'(r_i,c_1) = \omega(r_i,c_1) \mod P$. Again, it does not change the collisions since computing a possible collision is done modulo P. However, we must change d' to be $d'(r_i) = d'(r_i) - \omega(r_i,c_1) + \omega'(r_i,c_1)$.

Finally, we assume w.l.o.g. that $\omega'(r_0,c_1)$ is the smallest weight among the weights of the arcs (c_1,c_2). We let $\omega'(r_i,c_1) = \omega'(r_i,c_1) - \omega'(r_0,c_1)$, which implies that $\omega'(r_0,c_1) = 0$. All lengths between c_1 and c_2 change by the same value, hence collisions are not modified. We change $d'(r_i)$ to $d'(r_i) - \omega'(r_0,c_1)$ for all i so that the constraint on the deadline stay the same. □

From now on, we may assume that a star routed network is canonical, using Proposition 2. To give a instance of PALL where the routed network is a canonical star routed network, it is enough to give the weights of the arcs (c_1,c_2) for all routes, the period, the datagram size, and d the deadline function.

Chapter 3 focuses on solving PAZL on star routed network. Chapter 4 will focus on solving PALL on such topologies.

2.2 Hardness of PALL and PAZL

We show in this section that PALL is NP-hard by proving NP-hardness for a restricted version: PAZL with $\tau = 1$. We give two proofs that PAZL is NP-complete. The first proof works even for contention depth two, but not for star routed networks. For contention depth one, the problem is trivial: either the load is less than one and there is a valid assignment with zero waiting time or there is no valid assignment. The second proof works for graphs with contention width 2: the conflicts are locally very simple, but the problem is complex globally nonetheless. Solving PALL is trivial on trees because they can be reduced to one vertex of contention depth one. Thus, it may be interesting to study the complexity of PALL on bounded treewidth (or dagwidth) networks, a common property of real networks [44].

Theorem 1. PAZL *is* NP*-complete on the class of routed networks with contention depth 2.*

Proof. PAZL is in NP since given an offset for each route in an assignment, it is easy to check whether there are collisions, in linear time in the routed network's size.

Let $H = (V,E)$ be an undirected graph and let P be its maximal degree. We consider the problem to determine whether H is arc-colorable with P or $P+1$ colors. The arc coloring problem is NP-hard [45] and we reduce it to PAZL to prove its NP-hardness. To do that, we define from H a routed network $N = (\mathcal{R}, \omega)$ as follows.

Let us choose an arbitrary total order $<$ on V. For each edge $(u,v) \in E$, if $u < v$, there is a route $s_{u,v},u,v,t_{u,v}$ in $\mathcal{R}$. All these arcs are of weight 0. Note that, N is of contention depth 2, as required by the theorem statement.

The existence of a P-coloring of H is equivalent to the existence of a $(P,1)$-periodic bufferless assignment of N. Indeed, a P-coloring of H can be seen as a labeling of its edges by the integers in $[P]$. It induces a bijection between P-colorings of H and offsets of the routes of $\mathcal{R}$, which represent the edges of H. Having no collision on some vertex v implies that all offsets of routes going through v are different, since all arcs are of weight 0. Hence, edges of H incident to v, colored by the offsets of a valid assignment are all of distinct colors. Therefore we have reduced arc coloring to PAZL by a polynomial time transformation which concludes the proof. □

Remark that we have used weights of zero for all arcs in the proof. It is a further restriction to the class of graphs for which PAZL is NP-hard. We could ask the weights to be strictly positive, another possible restriction which makes more sense in our model, since weights represent the delay of physical links. Then, we can prove NP-completeness using the same proof, by setting all weights to the period P.

We now give a hardness proof for routed networks with contention width two but large contention depth. Note that a vertex of contention depth one does not induce a collision and can be removed from the routed network without loss of generality. The presented reduction can be used to prove an inapproximability result. Let MINPAZL be the following problem: given a routed network and τ, find the minimal period P such that there is a (P,τ)-periodic bufferless assignment (a positive instance of PAZL).

Theorem 2. *If* $\mathsf{P} \neq \mathsf{NP}$, *the problem* MINPAZL *on the classe of routed networks of contention width two cannot be approximated in polynomial time within a factor* $n^{1-o(1)}$ *where* n *is the number of routes.*

Proof. We reduce the problem of finding the minimal vertex coloring of a graph to MINPAZL. Let $H = (U,E)$ be a graph, an instance of the problem of finding a minimal vertex coloring. We define the routed network N from H as in Proposition 1.

Let $<$ be an arbitrary total order on U. The vertices of N are in the set $\{v_{u,w} \mid (u,w) \in E\} \cup \{u^1, u^2 \mid u \in U\}$. For each vertex u in H, there is a route r_u in $\mathcal{R}$, whose first and last vertices are u^1 and u^2. In between, the route contains all vertices $v_{u,w}$, following the order $<$ on the w. The weights of all arcs is zero. By construction, a contention vertex corresponds to an edge and belongs to exactly two routes representing the vertices of the edge, thus N is of contention width 2. This reduction is illustrated in Figure 2.6.

The existence of a P-coloring of H is equivalent to the existence of a (P,1)-periodic assignment of N without waiting time: the offset of a route can be identified with the color of the corresponding vertex. Indeed, since all weights are zero, the absence of collision at contention point $v_{u,w}$ is equivalent to the fact that the offsets of r_u and r_w are different and reciprocally.

Therefore, if we can approximate the minimum value of P within some factor such that there is a (P,1)-periodic assignment, we could approximate the minimal number of colors needed to color a graph within the same factor. The proof follows from the hardness of approximability of finding a minimal vertex coloring [46]. □

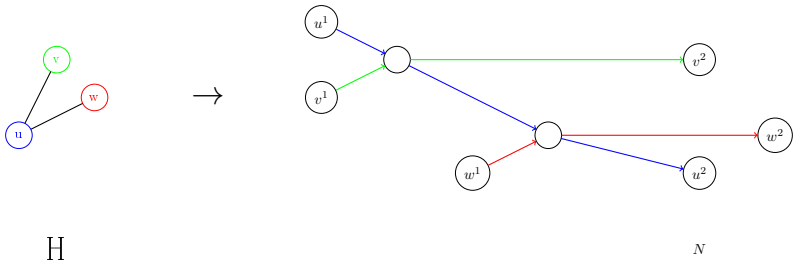

Figure 2.6 – Reduction from k-coloring to MINPAZL

The previous theorem implies that PAZL is NP-complete on the class of routed networks with contention width two. This also underlines the fact that, for general graphs, the best P such that there is a (P,τ)-periodic assignment may correspond to a very small load. We can build on the reduction of the previous theorem to prove that MINTRA, the problem of minimizing $TR(A)$, is hard to approximate too.

Theorem 3. *If* P $\neq$ NP*, the problem* MINTRA*, on graphs of contention width two, cannot be approximated in polynomial time within a factor* $n^{1-o(1)}$ *where* n *is the number of routes.*

Proof. We reduce the problem of finding the minimal vertex coloring of a graph to MINTRA. Let $H = (U,E)$ be a graph, instance of the problem of finding a minimal vertex coloring. We define the routed network N in two steps.

Let the elements of U be $u_0, \ldots, u_{n-1}$. There are n routes in N, denoted by r_i for $i \in [n]$. In their first part, they go from u_i^0 to u_i^1, through some vertices in $\{v_{i,j,k}\}_{i,j,k\in[n]}$ that we later define. Moreover, $u_i^1 \in \mathcal{B}$, that is the waiting time is added at u_i^1. Assume that r_i has offset o_i and r_j has offset o_j and let us fix the datagram size to 1 and the period to n. If r_i and r_i go through some vertex $v_{i,j,k}$, and $\lambda(r_i,v_{i,j,k}) = \lambda(r_j,v_{i,j,k}) + k$, then to avoid a collision, the equation $o_i \neq o_j + k \mod n$ must be satisfied. If r_i and r_j go through $v_{i,j,k}$ satisfying the previous constraints for all $k \neq l$, it implies $o_i = o_j + l \mod n$. It is easy to choose the weights of the two arcs going to $v_{i,j,k}$ to realize the previous condition, whatever the choice of weights of the previous arcs of the routes r_i and r_j.

We ensure, using the vertices $v_{i,j,k}$ for $k \neq i - j$, that $o_i = o_j + i - j \mod n$. It implies that there is some o, such that $o = o_i - i \mod n$ for all $i \in [n]$. Now, for each route r_i, we set the weight of the arc going to v_i^1, from the last vertex of the form $v_{i,j,k}$ in r_i, to be $n - i$. With this construction, we have ensured, that the datagram of r_i arrives at v_i^1 at time o modulo n, for all $i \in [n]$.

The second part of the routes, from v_i^1 to v_i^2 is built exactly as in the proof of Theorem 2. Hence, the waiting time in the vertices v_i^1 plays the exact same role as the offset in the graph of Theorem 2: the valid $(n,1)$-assignments are in bijection with colorings of H, the waiting times corresponding to the colors.

Finally, set the weights of the last arc going to v_i^2, for all $i \in [n]$, such that, for all $i,j \in [n]^2$, $\lambda(r_i) = \lambda(r_j)$. Since all routes are of the same size, $TR(A)$ is equal to the maximal waiting time of A. Hence, the maximum waiting time is equal to the number of different waiting times required to have a valid assignment. A valid $(n,1)$-assignment which minimizes $TR(A)$ is in bijection with a minimal proper coloring of H, which proves the theorem.

□

We would like to prove hardness for even more restricted networks, in particular star routed networks. The problem PAZL on star routed networks is similar to the minimization of makespan in a two flow-shop with delays (see Section 4.1.4), a problem known to be NP-complete [37]. It suggests that PAZL is NP-complete on star routed network, however we have not been able to prove it yet, because the makespan cannot easily be encoded in PAZL. If we relax the definition of routed network by allowing loops, we can model a network with a single half-duplex shared link, that is collisions can happen between datagrams going in both directions. This variant can be shown to be NP-complete by a reduction from the

subset sum problem, as it is done for a similar problem of scheduling pair of tasks [47].

Conclusion

We model C-RAN networks by weighted directed acyclic multigraphs called routed networks. The process we study is Periodic. Datagrams are sent by the sources of the routes at every period. Once a datagram have been sent, it can be buffered in only one contention point of the network. The solutions to this problem are called periodic assignments and are a choice of offset and waiting time (on one contention point) for all datagrams such that there are no collisions between them. An assignment is computed for one period, and repeated all over the time for every period. The value of an assignment A is denoted by $TR(A)$ and corresponds to the largest value on all routes of the time elapsed between the sending of a datagram and the reception of the answer in the RRH. We also introduce the simplest non trivial topology: the star routed networks in which every route shares the same two contention vertices.

We introduce the problem PALL of finding an assignment respecting a given constraint on the latency of each route. We also give its variant PAZL, where the constraint are as strong as possible: the assignment must use no waiting time. We showed that PAZL and PALL are NP-Hard, even for routed network of small contention depth or contention width, using reductions to arc and vertex coloring problems.

Chapter 3

Scheduling Unsynchronized Periodic Datagrams without Buffer

This chapter corresponds to two articles: Section 3 of [41] and all of [48].

In this chapter, we propose algorithm to solve the problem PAZL on *star routed networks*, a problem denoted by PMA for **P**eriodic **M**essage **A**ssignment in [48]. In this context, once a datagram has been emitted, no buffering is allowed in the network for contention. Remark that, if some buffering is necessary for dealing with some technical difficulty, but its duration is deterministic, it can be encoded into the length of the links. The bufferless assignment we design may allow to design completely bufferless networks, using full optical networks or new generation of networks mentionned in Chapter 7, which provide transparent transmission of data without latency inducing opto-electronic conversion or forwarding computations.

In the problem PAZL (and also in the problem PALL studied in Chapter 4), the datagram emissions are not synchronized. This means the sources don't send the datagram at the same time in the period. This is why we can choose the offset on the period without adding latency to the datagram. Such a problem does not perfectly match with C-RAN in actual 5G, but can be used in many other applications in which the sources do not need to be synchronized (industry 4.0, monitoring, sensor network, multicore over a bus, two processors scheduling, ...). Moreover, we hope that future evolutions of the 5G standard will allow to use unsynchronized antennas.

Since PAZL is a more constrained problem than PALL, we simplify the notations. No buffering is allowed, hence the notion of deadline on each route is not relevant anymore because all datagrams have the smallest possible process time, if there is a solution. Thus, an instance of PAZL can be given, for each route i, by the length of the arc (c_1,c_2) on r_i, that is $\omega(r_i,c_1)$, denoted in this chapter by δ_i and called delay. A datagram is emitted an

infinite number of times periodically, hence it is enough to consider any interval of P units of time to completely represent the state of our system by giving the times, in this interval, at which each datagram goes through the two contention points. We call the representation of an interval of P units of time in the first contention point the **first period** and the **second period** for the second contention point.

Recall that an **offset** of a datagram is a choice of time at which it arrives at the first contention point (i.e. in the first period). Let us consider a datagram i of offset o_i, it uses the interval of time $[i]_1 = \{(o_i + t) \mod P \mid 0 \leq t < \tau\}$ in the first period and $[i]_2 = \{(d_i + o_i + t) \mod P \mid 0 \leq t < \tau\}$ in the second period. Two datagrams i and j **collide** if either $[i]_1 \cap [j]_1 \neq \emptyset$ or $[i]_2 \cap [j]_2 \neq \emptyset$. If $t \in [i]_1$ (resp. $t \in [i]_2$), we say that datagram i uses time t in the first period (resp. in the second period). An **assignment** is a function from the datagrams to their offsets, such that there is no collision.

The complexity of PAZL on star routed networks is yet unknown. We prove in this chapter that, when parameterized by n the number of datagrams, the problem is FPT. On a slight generalization of the star routed network, with more contention points, but each datagram only going through two of them, PAZL is NP-hard, see Theorem 1. When the shared link is not full-duplex, that is, there is a single contention point and each datagram goes through it twice, we can encode the same non periodic problem, which is NP-hard [47]. Hence, we conjecture that PAZL is NP-hard.

To overcome the supposed hardness of PAZL, we study it when the load of the system is small enough, which is defined here as the number of units of time used in a period by all datagrams divided by the period that is $n\tau/P$. There cannot be an assignment when the load is larger than one; **we prove in this chapter that, for moderate loads, there is *always* an assignment and that it can be found by a polynomial time algorithm**. When an algorithm finds an assignment to the problem PAZL, for any input in some set, we say it solves PAZL **positively** on the set of inputs. This kind of result is very helpful when solving the following optimization version of PAZL: given a set of datagrams, find the largest subset which admits an assignment. A weighted version, where the datagrams have different size can also be considered. An optimal solution to the optimization problem is a set of datagrams corresponding to a load of at most 1. Assume we have an algorithm that always finds an assignment for an instance of load λ. Then, such an algorithm finds an assignment for any subset of load λ and is an approximation algorithm for the optimization problem with approximation ratio λ.

The chapter is composed of two parts. In Section 3.1, we study the case in which the size of the datagrams is unconstrained. This covers the use cases previously mentioned. Section 3.2 focuses on datagrams of size $\tau = 1$ and we also provide methods to convert the general problem to datagrams of size 1, at the cost of additional latency or load. Several algorithms solving the case $\tau = 1$ are proposed and analyzed.

3.1 Greedy Algorithms for Large Datagrams

In this section, we study the case of arbitrary values for τ. When modeling real problems, it is relevant to have $\tau > 1$ when the transmission time of a single datagram is large with regard to its delay, which is the case in C-RAN networks.

A **partial assignment** A is a function defined from a subset S of $[n]$ to $[P]$. The cardinal of S is the **size** of partial assignment A. A datagram in S is **scheduled** (by A), and a datagram not in S is **unscheduled**. We only consider partial assignments such that no pair of datagrams of S collide. If A has domain S, and $i \notin S$, we define the extension of A to the datagram i by the offset o, denoted by $A[i \to o]$, as A on S and $A[i \to o](i) = o$.

We give several simple heuristics and an exact fixed parameter tractable algorithm, in time exponential in the number of datagrams only. All presented algorithms except `Exhaustive Search of Compact Assignments` build an assignment incrementally, by growing the size of a partial assignment. Moreover, algorithms of this section are *greedy*: once an offset is chosen for a datagram, it is never changed.

In the rest of the chapter, we sometimes compare the relative position of datagrams, but one should remember that the time is periodic and these are relative positions on a circle of size P. Moreover, when it is unimportant and can hinder comprehension, we may omit to write *mod* P in some definitions and computations.

We show in the experiments of Section 3.1.6, that PAZL can be very often solved positively, in particular for short routes and when the load is moderate.

3.1.1 Shortest-Longest policy

We first present a simple policy, which works when the period is large with regard to the lengths of the routes. More generally, it works when the length of the routes modulo the period are close. The algorithm is called `ShortestLongest`: it sends datagrams on the shared link from the route with the smallest delay (i.e the shortest arc (c_1,c_2)) to the

largest (i.e. the longest one). There is no idle time in the first period, i.e. a datagram goes through c_1 right after the previous one has left c_1.

Proposition 3. *Assuming the datagrams are ordered by increasing delay and* $n\tau + \delta_{n-1} - \delta_0 \leq P$*, then then* `ShortestLongest` *solves* PAZL *positively in time* $O(n\log(n))$.

Proof. Let us define the assignment $A(i) = i\tau$. Since $n\tau + \delta_{n-1} - \delta_0 \leq P$ and $\delta_{n-1} \geq \delta_0$, we have $n\tau \leq P$. Hence, there is no collision in the first period.

The delays are sorted so that for all i, $\delta_i \leq \delta_{i+1}$. We can also assume, without loss of generality that $\delta_0 = 0$. The interval of time used by i in the second period is $[i]_2 = \{(\delta_i + A(i) + t) \mod P \mid 0 \leq t < \tau\}$. By hypothesis, and because the delays are in increasing order, we have for all $i \leq n$, $n\tau + \delta_i \leq P$. Hence, $[i]_2 = [\delta_i + i\tau, \delta_i + (i+1)\tau[$. Since the delays are in increasing order, $\delta_i + (i+1)\tau \leq \delta_{i+1} + (i+1)\tau$, then $[i]_2 \cap [i+1]_2 = \emptyset$ and we have proved that the assignment A does not induce any collision. The complexity of the algorithm is dominated by the sorting of the delays in $O(n\log(n))$.

□

If the period is slightly smaller that the bound of Proposition 3, there is a collision of datagram $n-1$ with datagram 0 in the first period. Hence, this policy is not useful as a heuristic for longer routes, as confirmed by the experimental results of Section 3.1.6.

3.1.2 First Fit

Consider some partial assignment A, the datagram i uses all times from $A(i)$ to $A(i)+\tau-1$ in the first period. If a datagram j is scheduled by A, with $A(j) < A(i)$, then the last time it uses in the first period is $A(j)+\tau-1$ and it should be less than $A(i)$, which implies that $A(j) \leq A(i) - \tau$. Symmetrically, if $A(j) > A(i)$, to avoid collision between datagrams j and i, we have $A(j) \geq A(i) + \tau$. Hence, datagram i forbids the interval $]A(i) - \tau, A(i) + \tau[$ as offsets for datagrams still not scheduled because of its use of time in the first period. The same reasoning shows that $2\tau - 1$ offsets are also forbidden because of the times used in the second period. Hence, if $|S|$ datagrams are already scheduled, then $|S|(4\tau - 2)$ offsets are forbidden for any unscheduled datagram. It is an upper bound on the number of forbidden offsets, since the same offset can be forbidden twice because of a datagram on the first and on the second period.

Let $Fo(A)$ be the maximum number of forbidden offsets when extending A. Formally, assume A is defined over S and $i \notin S$, $Fo(A)$ is the maximum over all possible values of

$_i$ of $|\{o \in [P] \mid A[i \to o] \text{ has no collision}\}|$. The previous paragraph shows that $Fo(A)$ is always bounded by $(4\tau - 2)|S|$.

Let `First Fit` be the following algorithm: for each unscheduled datagram (in the order they are given), it tests all offsets from 0 to $P - 1$ until one does not create a collision with the current assignment and use it to extend the assignment. If $Fo(A) < P$, then whatever the delay of the route we want to extend A with, there is an offset to do so. Since $Fo(A) \leq (4\tau - 2)|S|$ and $|S| < n$, `First Fit` (or any greedy algorithm) will always succeed when $(4\tau - 2)n \leq P$, that is when the load $n\tau/P$ is less than $1/4$. It turns out that `First Fit` always creates compact assignments (as defined in Proposition 4), that is a datagram is always next to another one in one of the two periods. Hence, we can prove a better bound on $Fo(A)$, when A is built by `First Fit`, as stated in the following theorem.

Theorem 4. *`First Fit` solves* PAZL *positively on instances of load less than* $1/3$.

Proof. We show by induction on the size of S, that $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$. For $S = 1$, it is clear since a single datagram forbid at most $(3\tau - 1) + \tau - 1 = 4\tau - 2$ offsets, as explained before. Now, assume $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$ and consider a route $i \notin S$ such that `First Fit` builds $A[i \to o]$ from A. By definition of `First Fit`, if choosing $o - 1$ as offset creates a collision (W.l.o.g. say this is a collision in the first period), it means that there is a scheduled datagram between $o - \tau$ and $o - 1$, hence all these offsets are forbidden by A. The same offsets are also forbidden by the choice of o as offset for i, then only $3\tau - 1$ new offsets are forbidden, that is $Fo(A[i \to o]) \leq Fo(A) + (3\tau - 1)$, which proves the induction and the theorem. □

3.1.3 Meta-Offset

We propose an alternative greedy algorithm to build a bufferless assignment, which introduce the notion of meta-offset used for next algorithms and always finds an assignment when the load is less than $1/3$. The idea is to restrict the possible offsets which can be chosen for the datagrams. It seems counter-intuitive, since it decreases artificially the number of available offsets to schedule new datagrams. However, it allows reducing the number of forbidden offsets for unscheduled datagrams. A **meta-offset** is an offset of value $i\tau$, with i an integer from 0 to P/τ. We call `MetaOffset` the greedy algorithm which works as follows: for each datagram, in the order they are given, it tries all meta-offsets from 0 to P/τ as offset for the assignment until one does not create a collision with the current partial

assignment. Note that `MetaOffset` works as `First Fit`, but consider only meta-offsets when scheduling datagrams. It is also equivalent to `ShortestLongest` when the delays are sorted in increasing order and that the conditions of Proposition 3 are satisfied.

Let $Fmo(A)$ be the maximal number of meta-offsets forbidden by A when trying to schedule any new datagram.

Theorem 5. *`MetaOffset` solves* PAZL *positively on star routed network and load less than* $1/3$. *The assignment is found in time* $O(n^2)$.

Proof. Let us prove that `MetaOffset` always schedules the n datagrams when the load is less than $1/3$. Let us consider A defined on the datagrams 0 to $i-1$. Let us evaluate $Fmo(A)$, that is the number of values j, such that $A[i \to j\tau]$ is a correct partial assignment (using only meta-offsets). Since i datagrams are scheduled by A, there are i meta-offsets which cannot be chosen to avoid collision in the first period. In the second period, the set of τ consecutive tics used by a datagram forbid at most two meta offsets, since the datagrams are all of size τ, see Figure 3.1. Hence, there are at most $2i$ meta-offsets forbidden by collisions in the second period. We have proved that $Fmo(A) \leq 3i$. There is always a way to extend A into $A[i \to j\tau]$ for some j when there is more meta-offsets than forbidden meta-offsets, that is $Fmo(A) < P/\tau$. Hence, `MetaOffset` terminates and provides a valid bufferless assignment as soon as $P/\tau > 3(n-1)$, which can be rewritten $(n-1)\tau/P > 1/3$: the load is larger than $1/3$.

This algorithm works in time $O(n^2)$, since for the i-th datagram we schedule, we have to try at most $3i$ meta-offsets before finding a correct one. We can test whether these $3i$ offsets cause a collision in the second period in time $O(i)$ by maintaining an ordered list of intervals of tics in the second period used by already scheduled datagram. □

This algorithm, contrarily to the previous one, may work well, even for loads higher than $1/3$. In fact, experimental data in Section 3.1.6 suggest that the algorithm finds a solution when the load is less than $1/2$.

A naive implementation of `MetaOffset` is in $O(nP/\tau)$, while `First Fit` is in $O(nP)$. However, it is not useful to consider every possible (meta-)offset at each step. By maintaining a list of positions of scheduled datagrams in first and second period, both algorithms can be implemented in $O(n^2)$ (n is the number of routes).

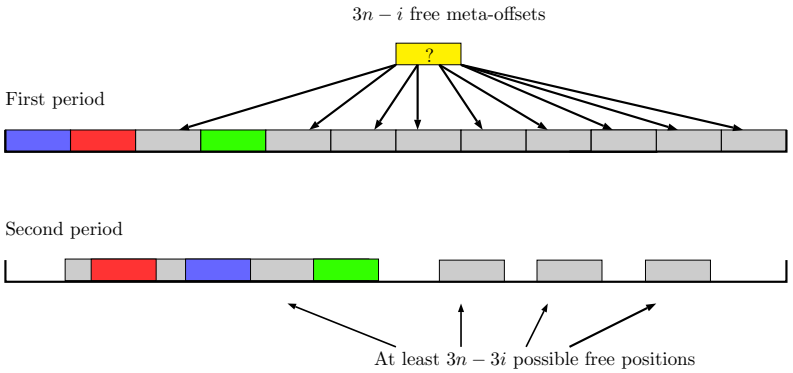

Figure 3.1 – Meta offsets used in the first and second period, when scheduling the $i + 1^{th}$ datagram in `MetaOffset`. There is already 3 datagram in S (in red, blue and green), and the meta offsets available for the yellow datagram are represented in grey.

3.1.4 Compact Tuples

We present in this section a family of greedy algorithms which solve PAZL positively for larger loads. We try to combine the good properties of the two previous algorithms: the compactness of the assignments produced by `First Fit` and the absence of collision in the first period of `MetaOffset`. The idea is to schedule several datagrams at once, using meta-offsets, to maximize the compactness of the obtained solution. We first describe the algorithm which schedules pairs of datagrams and then explain quickly how to extend it to any tuples of datagrams.

We now introduce Lemma 6 to assume $P = m\tau$ and we use it until the end of the section. This hypothesis makes the analysis of algorithms based on meta-offsets simpler and tighter. The load increases from $\lambda = n\tau/P$ to at most $\lambda(1+1/m)$: the difference is less than $1/m < 1/n$, thus very small for most instances. The transformation of Lemma 6 does not give a bijection between assignments of both instances but only an injection, which is enough for our purpose.

Lemma 6. *Let I be an instance of* PAZL *with n datagrams of size τ, period P and $m = P/\tau$. There is an instance I' with n datagrams of size τ' and period $P' = m\tau'$ such that any assignment of I' can be transformed into an assignment of I in polynomial time.*

Proof. Fig. 3.2 illustrates the reductions we define in this proof on a small instance. Let $P = m\tau + r$ with $r \leq \tau$. We define the instance I' as follows: $P' = mP$, $\delta'_i = m\delta_i$ and $\tau' = m\tau + r$. With this choice, we have $P' = m(m\tau + r) = m\tau'$. Consider an assignment A' of the instance I'. If we let $\tau'' = m\tau$, then A' is also an assignment for

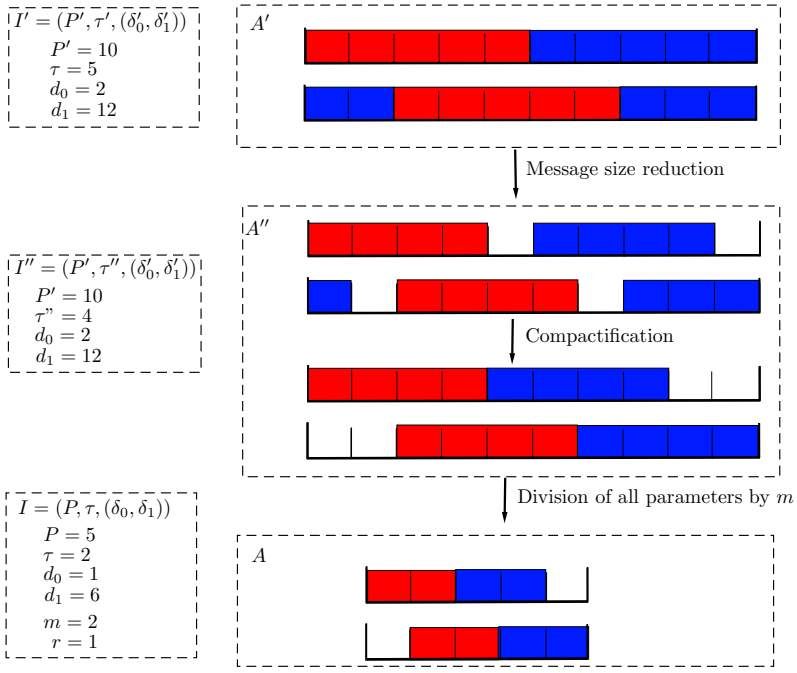

Figure 3.2 – Transformation from A'' to A

$I'' = (P', \tau'', (\delta'_0, \ldots, \delta'_{n-1}))$. Indeed, the size of each datagram, thus the intervals of time used in the first and second period begin at the same position but are shorter, which cannot create collisions. We then use a compactification procedure on A' seen as an assignment of I'', with size of datagrams multiple of m (see Proposition 4 for a similar compactification). W.l.o.g., the first datagram is positioned at offset zero. The first time it uses in the second period is a multiple of m since its delay is by construction a multiple of m. Then, all other datagrams are translated to the left by removing increasing values to their offsets, until there is a collision. It guarantees that some datagram j is in contact with the first one on the first or second period. It implies that either $A'(j)$ or $A'(j) + \delta_j \mod P'$ is a multiple of m and since δ_j is a multiple of m, then both $A'(j)$ and $A'(j) + \delta_j \mod P'$ are multiples of m. This procedure can be repeated until we get an assignment A'' to I'', such that all positions of datagrams in the first and second period are multiples of m. Finally, we define A as $A(i) = A''(i)/m$ and we obtain an assignment of I. □

We are interested in the remainder modulo τ of the delays, let $\delta_i = \delta'_i \tau + r_i$ be the Euclidean division of δ_i by τ. We assume, from now on, that *datagrams are sorted by increasing* r_i. A **Compact pair**, as shown in Fig. 3.3 is a pair of datagrams (i,j) with $i < j$ that can be scheduled using meta-offsets such that $A(i) + (\delta'_i + 1)\tau = A(j) + \delta'_j \tau$, i.e.

j is positioned less than τ unit of times after i in the second period. The **gap** between i and j is defined as $g = \delta'_i + 1 - \delta'_j \mod m$, it is the distance in meta offsets between i and j in the first period. By definition, we can make a compact pair out of i and j, if and only if their gap is not zero.

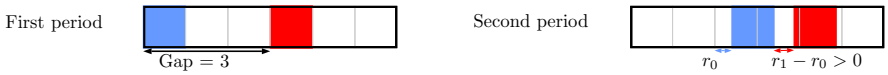

Figure 3.3 – A compact pair scheduled using meta-offsets, with $d'_0 = 2$ and $d'_0 = 0$

Lemma 7. *Given three datagrams of delay δ_1, δ_2 and δ_3, two of them form a compact pair.*

Proof. If the first two datagrams or the first and the third datagram form a compact pair, then we are done. If not, then by definition $\delta'_1 = 1 + \delta'_2 = 1 + \delta'_3$. Hence, datagrams 2 and 3 have the same delay and form a compact pair of gap 1. □

Let `Compact Pairs` be the following greedy algorithm: From the datagrams in order of increasing r_i, a sequence of at least $n/3$ compact pairs is built using Lemma 7. Pairs are scheduled in the order they have been built using meta-offsets. If at some point all compact pairs are scheduled or the current one cannot be scheduled, the remaining datagrams are scheduled as in `MetaOffset`. The analysis of `Compact Pairs` relies on the evaluation of the number of forbidden meta-offsets. In the first phase of `Compact Pairs`, one should evaluate the number of forbidden offsets when scheduling a compact pair, that we denote by $Fmo_2(A)$. In the second phase, we need to evaluate $Fmo(A)$. When scheduling a datagram in the second phase, a scheduled compact pair only forbids *three* meta-offsets in the second period. If datagrams in a pair are scheduled independently, they forbid *four* meta-offsets, which explains the improvement from `Compact Pairs`. We first state a simple lemma, whose proof can be read from Fig. 3.4, which allows bounding $Fmo_2(A)$.

Lemma 8. *A compact pair already scheduled by `Compact Pairs` forbids at most four meta-offsets in the second period to another compact pair when scheduled by `Compact Pairs`.*

Theorem 9. *`Compact Pairs` solves* PAZL *positively on instances of load less than* $3/8$.

Proof. Let n_2 be the number of compact pairs scheduled in the first phase. When scheduling a new pair, the position of the $2n_2$ datagrams on the first period forbid $4n_2$ offsets for

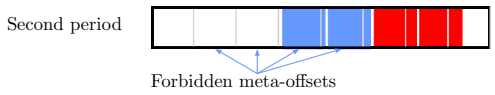

Figure 3.4 – Meta offsets forbidden by a scheduled compact pair (in blue) when scheduling another compact pair (in red)

a compact pair. Indeed, each scheduled datagram can collide with each of the two datagrams which form a compact pair. On the second period, we can use Lemma 8 to bound the number of forbidden offsets by $4n_2$. Hence, we have established that during the first phase, the partial solution A satisfies $Fmo_2(A) \leq 8n_2$. This first phase continues while there are available offsets for compact pairs, which is guaranteed when $Fmo_2(A) \leq m$, that is while $n_2 \leq m/8$. Hence, we assume that $n_2 = m/8$.

In the second phase, a compact pair forbids 3 meta offsets in the second period and 2 in the first. Hence, if we let n_1 be the number of datagrams scheduled in the second phase to build partial assignment A, we have $Fmo(A) \leq n_2 * 5 + n_1 * 3$. `Compact Pairs` can always schedule datagrams when $Fmo(A)$ is less than m, which is implied by $n_2 * 5 + n_1 * 3 \leq m$. Solving this equation, we obtain that $n_1 \geq \frac{m}{8}$ thus the number of datagrams scheduled is at least $2n_2 + n_1 \geq \frac{3}{8}m$. Assuming there are exactly $\frac{3}{8}m$ datagrams to schedule, then $\frac{2m}{8}$ datagrams are scheduled as compact pairs. It is two third of the $\frac{3}{8}m$ datagrams, hence Lemma 7 guarantees the existence of enough compact pairs. Therefore, an assignment is always produced when the load is less or equal to $\frac{3}{8}$. □

`Compact Pairs` can be improved by forming compact tuples instead of compact pairs. A compact k-tuple is a sequence of datagrams $i_1 < \cdots < i_k$ (with $r_{i_1}, \ldots, r_{i_k}$ increasing), for which meta-offsets can be chosen so that, there is no collision, the datagrams in the second period are in order $i_1, \ldots, i_k$ and for all l, $A(i_l) + (\delta'_{i_l} + 1)\tau = A(i_{l+1}) + \delta'_{i_{l+1}}\tau$. The algorithm `Compact k-tuples` works by scheduling compact k-tuples using meta offsets while possible, then scheduling compact $k-1$-tuples and so on until $k = 1$.

Lemma 10. *Given $k + k(k-1)(2k-1)/6$ datagrams, k of them always form a compact k-tuple and we can find them in polynomial time.*

Proof. We prove the property by induction on k. We have already proved it for $k = 2$ in Lemma 7. Now assume that we have found C a compact $(k-1)$-tuple in the first $(k-1)^3/3$ datagrams. Consider the next $(k-1)^2 + 1$ datagrams. If k of them have the same delay modulo τ, then they form a compact k-tuple and we are done. Otherwise,

there are at least k different values modulo τ in those $(k-1)^2+1$ datagrams. Each element of the compact $(k-1)$-tuple forbids one value for the delay modulo τ of a new kth element in the tuple. By pigeonhole principle, one of the k datagrams with distinct delays modulo τ can be used to extend C. We have built a compact k-tuple from at most $(k-1)+(k-1)(k-2)(2k-3)/6+(k-1)^2+1$ datagrams. It is equal to $k+k(k-1)(2k-1)/6$ which proves the induction. □

Theorem 11. *`Compact 8-tuples` always solves* PAZL *positively on instances of load less than* $4/10$*, for instances with* $n \geq 220$*.*

Proof. We need the following fact, which generalizes Lemma 8: A k-tuples forbids $k+j+1$ offsets in the second period when scheduling a j-tuple. It enables us to compute a lower bound on the number of scheduled i-tuples for i equal k down to 1 by bounding $Fmo_i(A)$, the number of forbidden meta-offsets when placing i-tuple in the algorithm. If we denote by n_i the number of compact i-tuples scheduled by the algorithm, we have the following equation:

$$Fmo_i(A) \leq \sum_{j=i}^{k} n_j(j*i+j+i+1).$$

The equation for n_1 is slightly better:

$$Fmo(A) \leq \sum_{j=1}^{k} n_j(2j+1).$$

A bound on n_i can be computed, using the fact that A can be extended while $Fmo_i(A) < m$. Lemma 10 ensures there are enough compact k-tuples, when $n - \sum_{j \leq i \leq 8} n_j$ is larger than $i + i(i-1)(2i-1)/6$. A numerical computation of the n_i's shows that `Compact 8-tuples` always finds an assignment when the load is less than $4/10$ and for $n \geq 220$. □

Th. 11 is obtained for $k = 8$. Taking arbitrary large k and using refined bounds on $Fmo_i(A)$ is not enough to get an algorithm working for a load of $41/100$ (and it only works from larger n).

The code computing the n_i can be found on [49]. To make `Compact 8-tuples` work, there must be at least 220 datagrams to produce enough compact 8-tuples in the first phase. It is not a strong restriction for two reasons. First, the bound of Lemma 10 can be improved, using a smarter polynomial time algorithm to find compact tuples, which better takes into account repetitions of values and compute the compact tuples in both

directions. Second, on random instances, the probability that k datagrams do not form a compact k-tuples is low, and we can just build the tuples greedily. Therefore, for most instances, forming compact k-uples is not a problem and in practice `Compact 8-tuples` works even for small n.

We describe here a last algorithm called `Compact Fit`, which is a simpler variant of the previous one. The idea is, as for `Compact Pairs`, to combine the absence of collision on the first period of `MetaOffset` and the compactness of assignments given by `First Fit`. The datagrams are ordered by increasing remainder of delay modulo τ, and each datagram is scheduled so that it extends an already scheduled compact tuples. In other words, it is scheduled using meta offsets, so that using one less as a meta-offset for some datagram creates a collision in *the second period*. If it is not possible to schedule the datagram in that way, the first possible meta-offset is chosen. This algorithm is designed to work well on random instances. Indeed, it is easy to evaluate the average size of the created compact tuples, and from that, to prove that `Compact Fit` works with high probability when the load is strictly less than 1/2. Figure 3.5 shows how `Compact Fit` builds an assignment from

Message (i)	0	1	2	3
Delay (δ_i)	2	0	3	1
$d_i \bmod \tau$	0	0	1	1

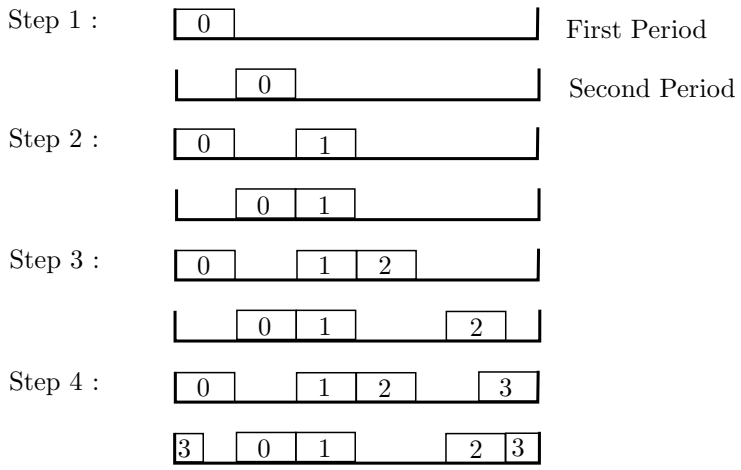

Figure 3.5 – Execution of `Compact Fit` creating two compact pairs with $P = 12$ and $\tau = 2$

a given instance. The datagrams are ordered by increasing remainder of delay modulo τ. A compact pair is built with datagrams 0 and 1. Datagram 2 cannot increase the size of the compact pair, it so creates a new tuple, completed by datagram 3

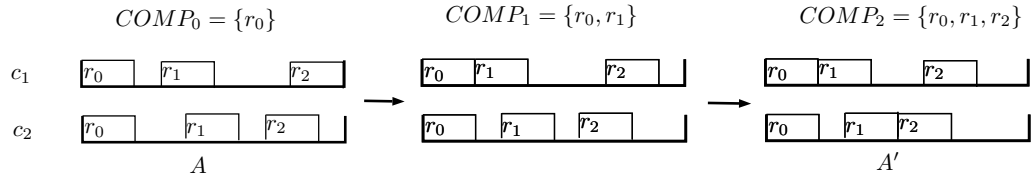

Figure 3.6 – Transformation of a bufferless assignment A into a compact assignment A', following the process of Proposition 4

3.1.5 Compact Assignment

In this section, we show how every bufferless assignment can be put into a canonical form. We use that form to design an algorithm solving PAZL in fixed parameter tractable time (FPT), with parameter n the number of routes (for more on parametrized complexity see [50]). This is justified since n is small in practice, from 10 to 20 in our settings, and the other parameters such as P, τ or the weights are large.

Let $(\mathcal{R},\omega)$ be a star routed network and let A be a bufferless (P,τ)-periodic assignment. We say that that A is **compact** if there is a route $r_0 \in \mathcal{R}$ such that the following holds: for all subsets $S \subset \mathcal{R}$ with $r_0 \notin S$, the bufferless assignment A', defined by $A'(r) = A(r) - 1 \mod P$ if $r \in S$ and $A(r)$ otherwise, is not valid. In other words, an assignment is compact if for all routes r but one, $A(r)$ cannot be reduced by one, that is either in the first or the second period, there is a route r' using the tics just before $A(r)$. See Figure 3.6 for an example of a compact assignment, obtained by the procedure of the next proposition.

Proposition 4. *Let $N = (\mathcal{R}, \omega)$ be a star routed network. If there is a (P,τ)-periodic bufferless assignment of N, then there is a compact (P,τ)-periodic assignment of N.*

Proof. Consider A a (P,τ)-periodic bufferless assignment of N. We describe an algorithm which builds a sequence $(r_0,\ldots,r_{n-1})$ and a sequence A_i of valid bufferless assignments. We denote by $COMP_i = \{r_j \mid j < i\}$

Let r_0 be an arbitrary route of $\mathcal{R}$ and $A_0 = A$. For $i = 1$ to n, we choose r_i as follows. Let $A_i = A_{i-1}$. While there is no collision, for all routes $r \in \mathcal{R} \setminus COMP_i$, let $A_i(r) = A_i(r) - 1$. Then choose any route r in $\mathcal{R} \setminus COMP$ such that setting $A_i(r) = A_i(r-1)$ creates a collision and let $r_i = r$. By construction A_i is a valid bufferless assignment, since it is modified only when no collision is created.

We prove by induction on i, that A_i is compact when restricted to $COMP_{i+1}$. For $i = 0$, $|COMP_1| = 1$ and the property is trivially satisfied. Let us consider A_i, by

induction hypothesis, since the offsets of routes in $COMP_i$ are not modified at step i of the algorithm, A is compact when restricted to $COMP_i$.

Consider $S \subseteq COMP_i$ which does not contain r_0. If S contains an element of $COMP_i$, then $S \backslash r_i$ is not empty and by compactness we cannot decrement all offsets of $S \backslash r_i$ without creating a collision. The same property is true for S. If $S = \{r_i\}$, then by construction of r_i by the algorithm, removing one from $A_i(r_i)$ creates a collision. Hence, A_i is compact restricted to $COMP_{i+1}$, which proves the induction and the proposition. □

We now present an algorithm to find a (P,τ)-periodic assignment by trying all compact assignments.

Theorem 12. PAZL $\in$ FPT *over star routed networks when parametrized by the number of routes.*

Proof. Let $N = (\mathcal{R},\omega)$ be a canonical star routed network and let P be the period and τ the size of a datagram. First, remark that for a given assignment and a route r with offset o_r, by removing o_r to all offsets, we can always assume that $o_r = 0$. By this remark and Proposition 4, we need only to consider all *compact assignments* with an *offset* 0 for the route r_0. We now evaluate the number of compact assignments and prove that it only depends on n the number of routes to prove the theorem.

We describe a way to build any compact assignment A by determining its offsets one after the other, which gives a bound on their number and an algorithm to generate them all. We fix an arbitrary total order on $\mathcal{R}$. Let r_0 be the smallest route of $\mathcal{R}$, its offset is set to 0 and we let $S = \{r_0\}$, $S_1 = \{r_0\}$ and $S_2 = \{r_0\}$. S represent the routes whose offsets are fixed, offsets of unscheduled routes are chosen so that they follow a route of S_1 in the first period or a route of S_2 in the second period.

At each step, we add an element to S: let r be the smallest element of S_1, if it is non empty. Then, select any route $r' \in \mathcal{R} \setminus S$ such that $o_{r'} = o_r + \tau$ does not create collision (by construction $o_{r'} = o_r + \tau - 1$ does create a collision in the first period). Then, we update the sets as follows: $S = S \cup \{r'\}$, $S_1 = S_1 \setminus \{r\} \cup \{r'\}$ and $S_2 = S_2 \cup \{r'\}$. If S_1 is empty, r is smallest element of S_2, and we set $o_{r'} = o_r + \tau + \omega(r,c_2) - \omega(r',c_2)$. We can also remove r from S_1 (or from S_2 if S_1 is empty) without adding any element to S. Remark that the value of the offset of the route added to S is entirely determined by the values of the offsets of the routes in S.

Now, remark that any compact assignment can be built by this procedure, if the proper

choice of element to add is made at each step. Hence, this process generates all compact assignments. We now bound the number of compact assignments it can produce. Remark that, when $|S| = i$, we can add any of the $n-i$ routes in $\mathcal{R} \setminus S$ to S. Hence, the number of sequences of choices of routes to add is $n!$ (but some of these sequences can fail to produce a valid assignment). We have not yet taken into account the steps at which an element is removed from either S_1 or S_2, without adding something to S. At each step of the algorithm, we can remove an element or not, there are at most $2n$ steps in the algorithm, hence there are at most 4^n sequences of such choices during the algorithm. As a conclusion, there are at most $4^n n!$ compact assignments.

The algorithm to solve PAZL builds every possible compact assignment in the incremental manner described here, and tests at each step whether, in the built partial assignment, there is a collision, which can be done in time linear in the size of N. Therefore PAZL $\in$ FPT. □

We call the algorithm described in Theorem 12 **E**xhaustive **S**earch of **C**ompact **A**ssignments or `ESCA`. The complexity of `ESCA` is in $O(4^n n!)$. While a better analysis of the number of compact assignments could improve this bound, the simple star routed networks with all arcs of weights 0 has $(n-1)!$ compact assignments. Hence, to improve significantly on `ESCA`, one should find an even more restricted notion of bufferless assignment than compact assignment.

To make `ESCA` more efficient in practice, we make cuts in the search tree used to explore all compact assignments. Consider a set S of k routes whose offsets have been fixed at some point in the search tree. We consider the times used by these routes in the first period. It divides the period into $[(a_0,b_0), \ldots, (a_{k-1},b_{k-1})]$ where the intervals (a_i,b_i) are the times not used yet in the first period. Therefore at most $\sum_{i=0}^{k-1} \lfloor (b_i - a_i)/\tau \rfloor$ routes can still send a datagram through the first period. If this value is less than $n-k$, it is not possible to create a compact assignment by extending the current one on S and we backtrack in the search tree. The same cut is also used for the second period. These cuts rely on the fact that the partial assignment is wasting bandwidth by creating intervals which are not multiples of τ. It helps with instances of large load, which are also the hardest to solve.

3.1.6 Experimental Results

3.1.6.1 Cloud-Ran Parameters

In this section, the performance on random instances of the algorithms presented in Sec. 3.1 is experimentally characterized.

The defaults parameters of experiments of this section are derived from the C-RAN context: a tic correspond to the sending time of 64 Bytes of data on links of bandwidth 10 Gbps. The datagrams are of size 1 Mbit, which corresponds to 2,500 tics.In the Cloud-RAN problem, the period fixed by the protocol HARQ correspond to 21,000 tics. This means that a network with 8 routes is loaded at 95%, which seems unrealistic and cannot be solved for PAZL, as we experimentally show it in this section. Nevertheless, we choose these parameters as a realistic reference and we modify them in order to evaluate our algorithm performances.

All experiments are done on synthetic data generated randomly. We generate the physical fronthaul network represented in Figure 2.4 of Chapter 2. We consider routes which are shorter than τ: a datagram cannot be contained completely in a single arc which is common in our applications. We generate random star routed networks, by drawing uniformly at random the weights of the arcs in [700]. This corresponds to links of the networks of less than 5km between a BBU and an RRH. Then, the corresponding canonical star routed network is built from the generated fronthaul and the algorithms tested on it. This process is mostly equivalent to drawing the delays randomly in [1400].

We consider the following algorithms:

- `ShortestLongest`
- `First Fit`
- `MetaOffset`
- `Compact Pairs`
- `Compact Fit`
- `Greedy Uniform`, the algorithm introduced and analyzed in Section 3.2, used for arbitrary τ
- `Exhaustive Search of Compact Assignments`

In the following experiments, we illustrate how well the algorithms work with regards to the load. To change the load, both parameters τ and n are fixed and we modify the period P, which allows for a smooth control of the load and does not impact the execution time of the algorithms. We generate 10,000 random instances of PAZL of load from 0.5 to 1. We represent, in Figure 3.7, the percentage of success of each algorithm as a function of the load. We make three experiments with 8, 12 and 16 routes to understand the effect of the number of routes on the quality of our algorithms. A bound on the maximal success rate is given by `ESCA` (exhaustive search) which always finds a solution if there is one.

The code in C is available on [49] under a copyleft license. The code has been run on a standard 2016 laptop with a 2.2 Ghz Intel Core i5 and the sources are compiled with gcc version 7.5.0. All experiments on 8 routes end in at most a few dozen seconds.

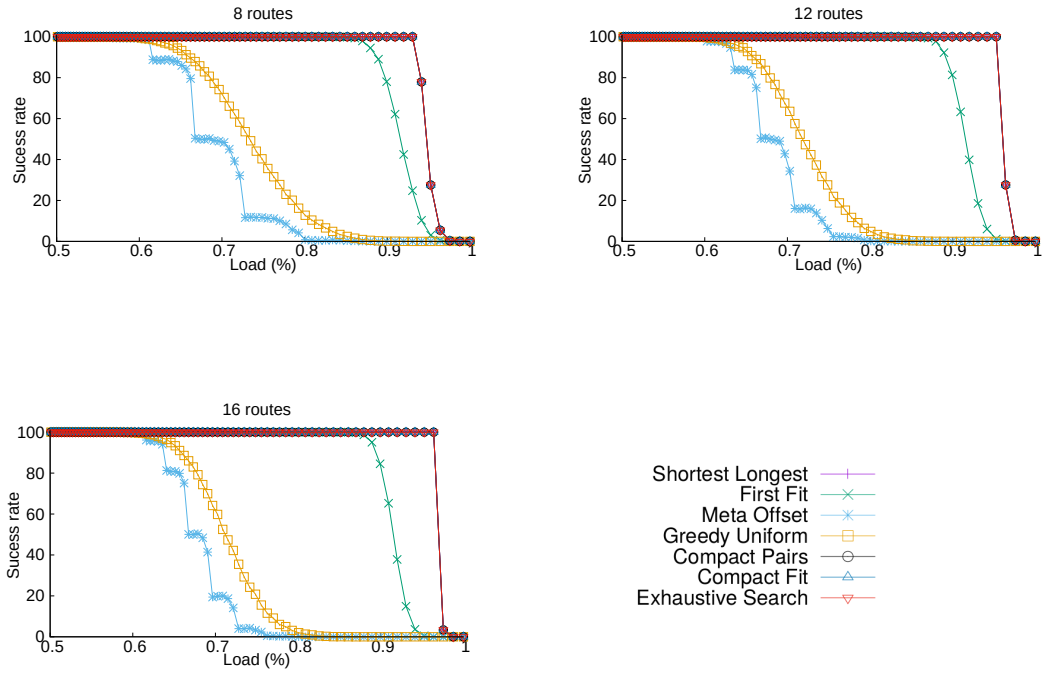

Figure 3.7 – Success rate of algorithms solving PAZL, for short routes and 8 routes, 12 routes and 16 routes

First, we remark that `ESCA` finds a solution even when the load is high. It justifies the idea to look for a bufferless assignment in this short routes regime. It seems that increasing the number of routes increases the success rate of `ESCA`, meaning that the more the routes, the more instances have a bufferless assignment. Second, remark that `ShortestLongest` is

as good as the exhaustive search. While it was expected to be good with short routes (see Proposition 3), it turns out to be optimal for all the random star routed networks we have tried. Therefore, we should use it in practical applications with short routes, instead of the exhaustive search which is much more computationally expensive. Also, `Compact Pairs` and `Compact Fit` have the same performances as `ESCA` and `ShortestLongest`. This is not surprising when the routes are drawn on a lower range than τ. Indeed, since we sort the routes by remainder modulo τ in `Compact Pairs` and `Compact Fit` (which are just there values then), these two algorithms build the same assignment as `ShortestLongest`.

`First Fit` has also excellent performances (100% of success under loads lower to 0.8). However, since we do not have strong theoretical results for this algorithms and it does not performs as well as `ShortestLongest` in this regime, it is not interesting to rely on it.

`MetaOffset` and `Greedy Uniform` both seem to always work when the load is less than 1/2 and have a good probability to work up to a load of 2/3, which is twice better than the theoretical bound. `MetaOffset` presents discontinuities in the probability of success at several loads, which seems to smooth out when the number of routes increases. It can be explained by the fact that `MetaOffset` becomes better when decreasing the load makes the number of available meta-offsets larger, which happens each time τ is added to the period (to decrease the load, the period is increased) and is more frequent when there are more routes.

`Greedy Uniform` and `MetaOffset` performances depend on the number of routes. The more routes there is, the lower the success rate. This is even clearer in experiment of Figure 3.11 in the next section in which there is 100 routes. Remember that `Greedy Uniform` is an algorithm presented and analyzed in Section 3.2. This algorithm is designed for $\tau = 1$ and random delays and has no theoretical guarantee for arbitrary τ and small delay.

3.1.6.2 Larger Route Number

After studying the C-RAN parameters, we want to experiment the performance of our algorithm with a larger number of routes and/or delays drawn in a larger interval. We experiment with several periods and datagram sizes. For each set of parameters, we try every possible load by changing the number of datagrams and give the success rate of each algorithm. Notice that all algorithm except `ESCA` are in polynomial time but are not always able to find a solution, depending on the load or the size of the routes. On the other hand,

`ESCA` finds a solution if it exists, but works in exponential time in n. The success rate is measured on 10000 instances of PAZL generated by drawing uniformly and independently the delays of each datagram in $[P]$ or $[\tau]$ for Figure 3.11.

On a regular laptop, all algorithms terminates in less than a second when solving 10000 instances with 100 datagrams except `ESCA`, whose complexity is exponential in the number of routes (but polynomial in the rest of the parameters). Hence, the optimal value of the success rate given by `ESCA` is only available in the experiment with at most 10 routes (the algorithm cannot compute a solution in less than an hour for twenty datagrams and high load). Note that while `First Fit`, `Compact Pairs`, `MetaOffset`, `Compact Fit` all run in almost the same time, `Greedy Uniform` seems to be three times longer than the other algorithms to run on instances with 100 datagrams. It is expected since, at each step, it must find all available offsets to draw one uniformly at random instead of just choosing one.

In the following experiments, since Lemma 6 explains how to transform any instance into one with $P = m\tau$, we chose for simplicity that $P = m\tau$.

In Figure 3.8 and Figure 3.9 the performances of `ShortestLongest` are abysmal (falls to 10% of success rate when the load is greater than 0.2, and 0% at 0.3 of load) and are thus not represented in this figures, in order to focus on other algorithms. This can be explained by the fact it depends on the difference of size between the longest and the smallest route, which is large here since the delays are drawn in $[P]$. This observation is reinforced by the results of Figure 3.10. Indeed, since the number of route is lower than in the two previous experiment, the probability of drawing a set of delays with a large difference is lower, and thus the success rate of `ShortestLongest` is better.

For all sets of parameters, the other greedy algorithms have the same relative performances. `MetaOffset` and `Greedy Uniform` perform the worst and have almost equal success rate. Remark that they have a 100% success rate for load less than $1/2$, while it is easy to build an instance of PAZL of load $1/3 + \varepsilon$ which makes them fail. The difference between the worst case analysis and the average case analysis is explained for `Greedy Uniform`, when $\tau = 1$ in Section 3.2.

`First Fit` performs better than `MetaOffset` while they have the same worst case. `Compact Pairs`, which is the best theoretically also performs well in the experiments, always finding assignments for load of 0.6. `Compact Fit`, which is similar in spirit to

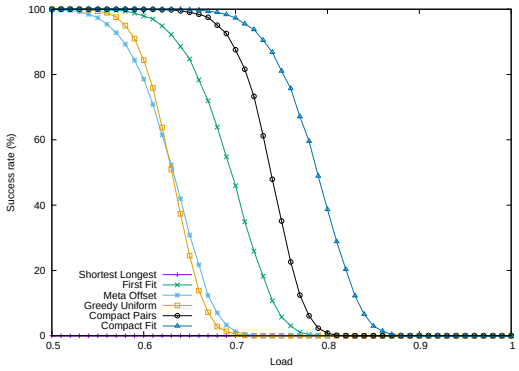

Figure 3.8 – Success rates of all algorithms for increasing loads, $\tau = 1000$, $P = 100{,}000$

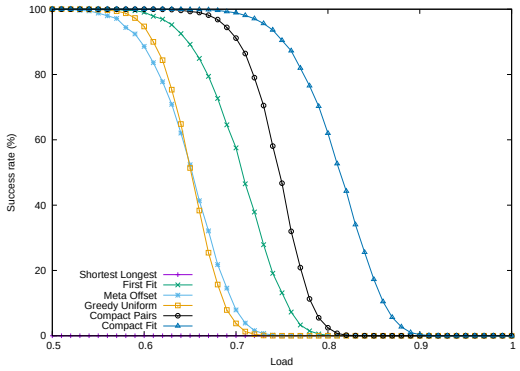

Figure 3.9 – Success rates of all algorithms for increasing loads, $\tau = 10$, $P = 1{,}000$

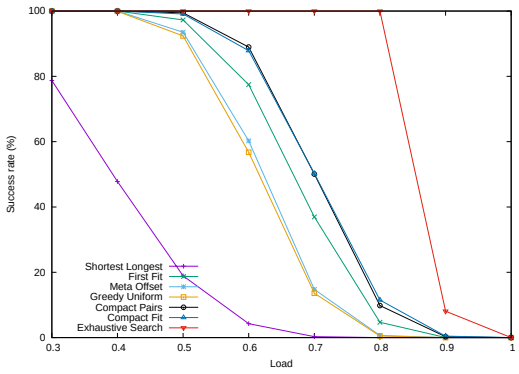

Figure 3.10 – Success rates of all algorithms for increasing loads, $\tau = 1000$, $P = 10{,}000$

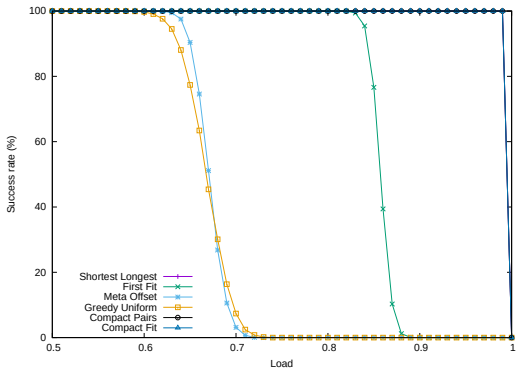

Figure 3.11 – Same parameters as in Fig. 3.8, delays uniformly drawn in $[\tau]$

`Compact Pairs` but is designed to have a good success rate on random instances is indeed better than `Compact Pairs`, when there are enough datagrams.

As illustrated by Figure 3.8 and Figure 3.9, the size of the datagrams have little impact on the success rate of the algorithms, when the number of datagrams stay the same. Comparing Figure 3.10 and Figure 3.8 shows that for more datagrams, the transition between success rate of 100% to success rate of 0% is faster. Finally, the results of `ESCA` in Figure 3.10 show that the greedy algorithm are far from always finding a solution when it exists. Moreover, we have found an instance with load 0.8 with no assignment, which gives an upper bound on the highest load for which PAZL can always be solved positively.

We also investigate the behavior of the algorithms when the delay of the datagrams are drawn in $[\tau]$ in Figure 3.11. The difference from the case of large delay is that `Compact Pairs` and `Compact Fit` are extremely efficient: they always find a solution for 99 datagrams. It is expected, since all δ'_i are equal in these settings and they will both build a 99-compact tuples and thus can only fail for load 1.

As illustrated on Figure 3.10, when the load is larger than 0.5, `ESCA` finds more solutions than the greedy algorithms, which justifies its use. However, for load larger than 0.8 there are instances for which there are no solutions to PAZL. It means that with long routes and high load, looking for a bufferless assignment is far too restrictive. This justifies the design of algorithms for the general PALL problem, which we present in the next section. We will test them on 8 long routes and a load between 1 and 0.8, parameters for which, as shown here, there are not always a bufferless assignment.

The computation time of `ESCA` is bounded by $O(4^n n!)$ as shown in Theorem 12, but it can be much better in practice, either because it finds a solutions quickly or because a large part of the tree of compact assignments is pruned during the algorithm. We study the evolution of the running time of the algorithm when n grows in the following experiment. The weights of the arcs are drawn following a uniform distribution in $[P]$ and the load is set to 0.95. The table of Figure 3.12 shows the time before the exhaustive search ends, for 8 to 16 routes, averaged on 100 random star routed networks. This shows that for less than 20 routes, which corresponds to all current topologies, the algorithm is efficient enough, but we should improve it further to work on more routes.

n	8	10	12	14	16
Time (s)	6.10^{-5}	8.10^{-4}	2.10^{-2}	0.4	11

Figure 3.12 – Running time of the exhaustive search.

3.2 Datagrams of Size One

When $\tau = 1$ and the load is less than $1/2$, *any greedy algorithm* solves PAZL positively since $Fo(A) \leq (4\tau - 2)|S| = 2|S|$ where S is the number of scheduled datagrams. We give, in this section, a method which always finds an assignment for a load larger than $1/2$.

3.2.1 Deterministic Algorithm

To go above $1/2$ of load, we optimize a potential measuring how many offsets are available for all datagrams, scheduled or not. datagrams are scheduled while possible using any greedy algorithm. Then, when all unscheduled datagrams have no available offset, we use a Swap operation defined later, which improves the potential. When the potential is high enough, it ensures that there are two datagrams whose offset can be changed so that a new datagram can be scheduled.

The algorithm is not greedy, since we allow to exchanging a scheduled datagram with an unscheduled one. It cannot work online, since it requires to know all delays of the datagrams in advance.

Definition 1. *The potential of a datagram of delay δ, for a partial assignment A is the number of integers $i \in [P]$ such that i is used in the first period and $i + \delta \mod P$ is used in the second period.*

The computation of the potential of a datagram of delay 3, is illustrated in Fig. 3.13. The potential of a datagram counts the configurations which reduce the number of forbidden offsets. Indeed, when i is used in the first period and $i + \delta \mod P$ is used in the second period, then the same offset is forbidden *twice* for a datagram of delay δ. Hence, the potential of a datagram is related to the number of possible offsets as stated in the following lemma.

Lemma 13. *Given a partial assignment A of size s, and i an unscheduled datagram of potential v, then the set $\{o \mid A(i \to o)$ has no collision$\}$ is of size $P - 2s + v$.*

For our algorithm, we need a global measure of quality of a partial assignment, that we try to increase when the algorithm fail to schedule new datagrams. We call our measure

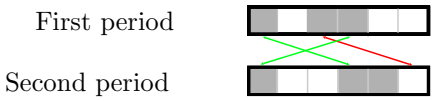

Figure 3.13 – A datagram of delay 3 has potential 2 in the represented assignment

the potential of an assignment and we denote it by $Pot(A)$, it is the sum of potentials of all datagrams in the instance.

Definition 2. *The potential of a position i, for a partial assignment A, is the number of datagrams of delay δ such that $i+\delta \mod P$ is used by a datagram scheduled by A.*

Delays (d_i)	2	3	1

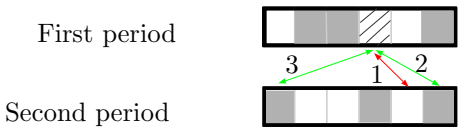

Figure 3.14 – Shaded position potential 2, in this assignment

The potential of a position is illustrated in Figure 3.14. Instead of decomposing the global potential as a sum over datagrams, it can be understood as a sum over positions, as stated in the next lemma.

Lemma 14. *The sum of potentials of all positions used in the first period by datagrams scheduled by A is equal to $Pot(A)$.*

By definition of the potential of a position, we obtain the following simple invariant.

Lemma 15. *The sum of potentials of all positions for a partial assignment with k scheduled datagrams is nk.*

As a consequence of this lemma, $Pot(A) \leq nk$. Let us define a **Swap operation**, which guarantees to obtain at last half the maximal value of the potential. Let A be some partial assignment of size s and let i be an unscheduled datagram of delay δ. Assume that i cannot be used to extend A. The Swap operation is the following: select a free position o in the first period, remove the datagram which uses the position $o+\delta$ in the second period from A and extend A by i with offset o. We denote this operation by $Swap(i,o,A)$.

Lemma 16. *Let A be some partial assignment of size k and let i be an unscheduled datagram. If i cannot be used to extend A, then either $Pot(A) \geq kn/2$ or there is an o such that $Pot(Swap(i,o,A)) > Pot(A)$.*

Proof. The positions in the first period can be partitioned into P_u the positions used by some scheduled datagram and P_f the positions unused. Let V_f be the sum of the potentials of the positions in P_f and let V_u be the sum of the potentials of the positions in P_u. By Lemma 15, since P_f and P_u partition the positions, we have $V_f + V_u = kn$. Moreover, by Lemma 14, $Pot(A) = V_u$, then $V_f + Pot(A) = kn$.

By hypothesis, i cannot be scheduled, then, for all $p \in P_f$, $p + \delta_i$ is used in the second period. Let us define the function F which associates to $p \in P_f$ the position $A(j)$ such that there is a scheduled datagram j which uses $p + d$ in the second period, that is $A(j) + \delta_j = p + d \mod P$. The function F is an injection from P_f to P_u. Remark now that, if we compare $Swap(i,p,A)$ to A, on the second period the same positions are used. Hence, the potential of each position stay the same after the swap. As a consequence, doing the operation $Swap(i,p,A)$ adds to $Pot(A)$ the potential of the position p and removes the potential of the position $F(p)$.

Assume now, to prove our lemma, that for all p, $Pot(Swap(i,p,A)) \leq Pot(A)$. It implies that for all p, the potential of p is smaller than the potential of $F(p)$. Since F is an injection from P_f to P_u, we have that $V_f \leq V_u = Pot(A)$. Since $V_f + Pot(A) = kn$, we have that $Pot(A) \geq kn/2$. □

Let us precisely describe the algorithm `Swap and Move`: datagrams are scheduled while possible by `First Fit` and then the Swap operation is applied while it increases the potential. When the potential is maximal, `Swap and Move` schedule a new datagram by moving at most two scheduled datagrams to other available offsets. If it fails to do so, `Swap and Move` stops, otherwise the whole procedure is repeated. We analyze `Swap and Move` in the following theorem.

Theorem 17. *`Swap and Move` solves positively* PAZL*, in polynomial time, for instances with $\tau = 1$ and load less than $1/2 + (\sqrt{5}/2 - 1) \approx 0{,}618$.*

Proof. We determine for which value of the load `Swap and Move` always works. We let $n = (1/2 + \varepsilon)P$ be the number of datagrams, the load is $1/2 + \varepsilon$. We need only to study the case when $n - 1$ datagrams are scheduled by A and `Swap and Move` tries to schedule the last one, since the previous steps are similar but easier.

Let δ be the delay of the unscheduled datagram. We consider the pairs of times $(o,o+d)$ for $o \in [P]$. Since the datagram cannot be scheduled, there are three cases. First, o is unused in the first period but $o + d$ is used in the second period. Since there are $n - 1$

scheduled datagrams, there are $P - n + 1$ such value of o. If a datagram using the time $o + d$ in the second period can be scheduled elsewhere, so that the unscheduled datagram can use offset o, then `Swap and Move` succeeds. Otherwise, the datagram has no possible offsets, which means its potential is equal to $2(\varepsilon P - 1)$. The second case is symmetric: o is used in the first period but $o + d$ is unused in the second period. Finally, we have the case o is used in the first period and $o + d$ is used in the second period. There are $2(\varepsilon P - 1)$ such values of o. If the two datagrams using times o and $o + d$ can be rescheduled so that offset o can be used for the unscheduled datagram, then `Swap and Move` succeeds. This is always possible when one datagram is of potential at least $2\varepsilon P - 1$ and the other of potential at least $2\varepsilon P + 1$. Since the datagrams must be of potential more than $2(\varepsilon P - 1)$ and at most $n - 1$, it is satisfied when the sum of the two potentials is at least $2(\varepsilon P - 1) + n$.

If we assume that `Swap and Move` was unable to schedule the last datagram by moving two scheduled datagrams, the previous analysis gives us a bound on twice $Pot(A)$:

$$2Pot(A) \leq 2(P - n + 1)2(\varepsilon P - 1) + 2(\varepsilon P - 1)(2(\varepsilon P - 1) + n)$$

$$Pot(A) \leq (\varepsilon P - 1)(P + n)$$

By Lemma 16, we know that $Pot(A) \geq n(n-1)/2$, hence `Swap and Move` must succeed when

$$n(n-1)/2 \geq (\varepsilon P - 1)(P + n).$$

By expanding and simplifying, we obtain a second degree inequation in ε, $1/4 - 2\varepsilon - \varepsilon^2 \geq 0$. Solving this inequation yields $\varepsilon \leq \sqrt{5}/2 - 1$.

Let us prove that `Swap and Move` is in polynomial time. All Swap operations strictly increase the potential. Moreover, when one or two datagrams are moved, the potential may decrease but a datagram is added to the partial assignment. The potential is bounded by $O(n^2)$ and the move operations all together can only remove $O(n^2)$ to the potential, hence there are at most $O(n^2)$ Swap operations during `Swap and Move`. A Swap operation can be performed in time $O(n)$, since for a given datagram, all free offsets must be tested and the potential is evaluated in time $O(1)$ (by maintaining the potential of each position). This proves that Swap and Move is in $O(n^3)$. □

Consider a partial assignment of size $n' = (1/2 + \varepsilon)P$, and a datagram of delay δ. If we consider all n' used offsets o and all times time $o + d$ in the second period, then o and $o + d$

are both used for at least $n' - (P - n') = 2\varepsilon P$ values of o. The potential of any datagram is thus larger or equal to $2\varepsilon P$. When a datagram cannot be scheduled, its potential is less or equal to $2\varepsilon P$, hence it is equal to $2\varepsilon P$.

Hence, the potential of any assignment of size n' is at least $2\varepsilon Pn$. As a consequence, the method of Lemma 16 will guarantee a non-trivial potential for $2\varepsilon Pn < nn'/2$, that is $\varepsilon < 1/6$. Any algorithm relying on the potential and the Swap operation cannot be guaranteed to work for load larger than $2/3 = 1/2 + 1/6$. However, we may hope to improve on the analysis of Lemma 16, since it is not optimal: $2\varepsilon P$ positions in P_u are not taken into account in the proof.

We conjecture that `Swap and Move` works for load up to $2/3$. On random instances, we expect the potential to be higher than the stated bound and to be better spread on the datagrams, which would make `Swap and Move` works for larger loads, as it is indeed observed in experiments (see Section 3.2.3).

3.2.2 Randomized Algorithm for Random Instances

We would like to understand better the behavior of our algorithms on instances drawn uniformly at random. To this aim, we analyze the algorithm `Greedy Uniform`, defined as follows: for each datagram in the order of the input, choose one of the offsets, which does not create a collision with the current partial assignment, uniformly at random.

We analyze `Greedy Uniform` over random instances: all datagrams have their delays drawn independently and uniformly in $[m]$. We compute the probability of success of `Greedy Uniform` over all random choices by the algorithm *and all possible instances*. It turns out that this probability, for a fixed load strictly less than one, goes to one when m grows. For a given partial assignment, we are only interested in its **trace**: the set of times which are used in the first and second period. Hence, if n datagrams are scheduled in a period of size m, the trace of an assignment is a pair of subsets of $[m]$ of size n. We now show that these traces are produced uniformly by `Greedy Uniform`.

Theorem 18. *The distribution of traces of assignments produced by `Greedy Uniform` when it succeeds, from instances drawn uniformly at random, is also uniform.*

Proof. The proof is by induction on n, the number of datagrams. It is clear for $n = 1$, since the delay of the first datagram is uniformly drawn and all offsets can be used. Assume now the theorem true for some $n > 1$. `Greedy Uniform`, by induction hypothesis has produced

uniform traces from the first n datagrams. Hence, we should prove that, if we draw delays of the $n+1^{th}$ datagram randomly, extending the trace by a random possible offset produces a random distribution on the traces of size $n+1$.

If we draw an offset uniformly at random (among all m offsets) and then extend the trace by scheduling the last datagram at this offset or fail, the distribution over the traces of size $n+1$ is the same as what produces `Greedy Uniform`. Indeed, all offsets which can be used to extend the trace have the same probability to be drawn. Since all delays are drawn independently, we can assume that, given a trace, we first draw an offset uniformly, then draw uniformly the delay of the added datagram and add it to the trace if it is possible. This proves that all extensions of a given trace are equiprobable. Thus, all traces of size $n+1$ are equiprobable, since they each can be formed from $(n+1)^2$ traces of size n by removing one used time from the first and second period. This proves the induction and the theorem. □

Since `Greedy Uniform` can be seen as a simple random process on traces by Th. 19, it is easy to analyze its probability of success.

Theorem 19. *The probability over all instances with n datagrams and period m that* `Greedy Uniform` *solves* PAZL *positively is*

$$\prod_{i=m/2}^{n-1} (1 - \frac{\binom{n}{2i-m}}{\binom{m}{i}}).$$

Proof. We evaluate Pr(m,n) the probability that `Greedy Uniform` fails at the n^{th} step assuming it has not failed before. It is independent of the delay of the n^{th} datagram. Indeed, the operation which adds one to all times used in the second period is a bijection on the set of traces of size $n-1$. It is equivalent to remove one to the delay of the n^{th} datagram. We can thus assume that the delay is zero.

Let S_1 be the set of times used in the first period by the $n-1$ first datagrams and S_2 the set of times used in the second period. We can assume that S_1 is fixed, since all subsets of the first period are equiprobable and because S_2 is independent of S_1 (Th. 19). There is no possible offset for the n^{th} datagram, if and only if $S_1 \cup S_2 = [m]$. It means that S_2 has been drawn such that it contains $[m] \setminus S_1$. By Th.19, S_2 is uniformly distributed over all sets of size $n-1$. Hence, the probability that $[m] \setminus S_1 \subseteq S_2$ is the probability to draw a set of size $n-1$ which contains $m-n+1$ fixed elements. This proves Pr(m,n) $= \frac{\binom{n}{2(n-1)-m}}{\binom{m}{n-1}}$.

From the previous expression, we can derive the probability of success of `Greedy Uniform` by a simple product of the probabilities of success $(1 - \Pr(m,i))$ at step i, for all $i \leq n$, which proves the theorem. □

If we fix the load $\lambda = n/m$, we can bound $P(m,n)$ using Stirling formula. We obtain for some constant C, that $P(m,n) \leq C\left(\frac{\lambda^{2\lambda}}{(2\lambda-1)^{2\lambda-1}}\right)^m$. We let $f(\lambda) = \frac{\lambda^{2\lambda}}{(2\lambda-1)^{2\lambda-1}}$. The derivative of f is strictly positive for $1/2 < \lambda < 1$ and $f(1) = 1$, hence $f(\lambda) < 1$ when $\lambda < 1$. By a simple union bound, the probability that `Greedy Uniform` fails is bounded by $C\lambda m f(\lambda)^m$, whose limit is zero when m goes to infinity. It explains why `Greedy Uniform` is good in practice for large m.

3.2.3 Experimental Results

In this section, the performance on random instances of the algorithms presented in Sec. 3.2 is experimentally characterized. The settings are as in Section 3.1.6, with $\tau = 1$. Because $\tau = 1$, the algorithms based on meta-offsets are not relevant anymore. The evaluated algorithms are:

- `First Fit`
- `Greedy Uniform`
- `Greedy Potential`, a greedy algorithm which leverages the notion of potential introduced for Swap. It schedules the datagrams in arbitrary order, choosing the possible offset which maximizes the potential of the unscheduled datagrams
- `Swap and Move`
- `ESCA`

In Figure 3.15, the performances of `ShortestLongest` are the same as in Figure 3.8 and Figure 3.9(10% for load 0.2, and 0% at 0.3 of load). The algorithm seems to performs better on Figure 3.16 (25% of success at load 0.5), but the algorithm is not interesting since `Greedy Potential`, `First Fit` or `Greedy Uniform` have much better success rate for a similar complexity.

For the other algorithms, as in Section 3.1.6, the success rate on random instances is much better than the bound given by worst case analysis. In the experiment presented in Figure 3.15, all algorithms succeed on all instances when the load is less than 0.64. `Greedy`

`Uniform` behaves exactly as proved in Theorem 19, with a very small variance. The performance of `Swap and Move` and of its simpler variant `Greedy Potential`, which optimizes the potential in a greedy way, are much better than `First Fit` or `Greedy Uniform`. Amazingly, `Swap and Move` always finds an assignment when the load is less than 0.95. `Swap and Move` is extremely close to `ESCA`, but for $P = 10$ and load 0.9 or 1, it fails to find some assignments, as shown in Figure. 3.16.

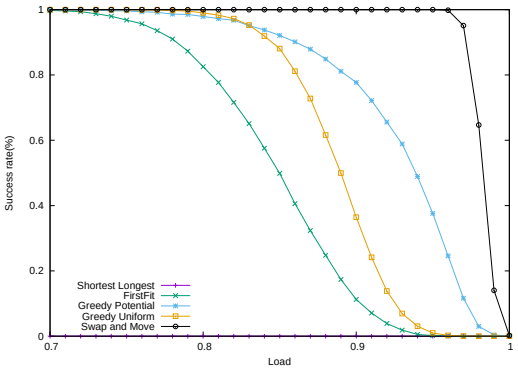

Figure 3.15 – Success rates of all algorithms for increasing loads, $\tau = 1$ and $P = 100$

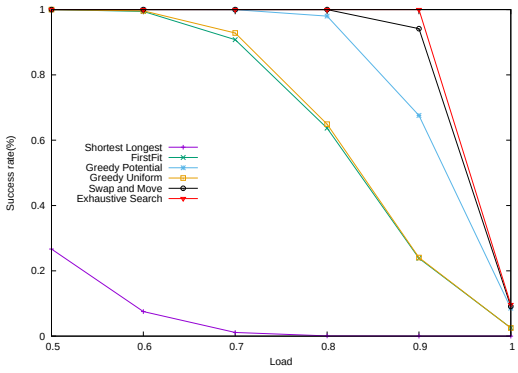

Figure 3.16 – Success rates of all algorithms for increasing loads, $\tau = 1$ and $P = 10$

Finally, we evaluate the computation times of the algorithms to understand whether they scale to large instances. We present the computation times in Fig. 3.17 and we choose to consider instances of load 1, since they require the most computation time for a given size. The empirical complexity of an algorithm is evaluated by a linear regression on the function which associates to $\log(n)$, the log of the computation time of the algorithm on n datagrams. `First Fit`, `Greedy Uniform`, `ShortestLongest` and `Swap and Move` scale almost in the same way, with an empirical complexity slightly below $O(n^2)$, while `Greedy Potential` has an empirical complexity of $O(n^3)$. The empirical complexity corresponds to the worst case complexity we have proved, except for `Swap and Move` which is in $O(n^3)$ worst case. There are two explanations: most of the datagrams are scheduled by the fast `First Fit` subroutine and most Swap operations improve the potential by more than 1, as we assume in the worst case analysis.

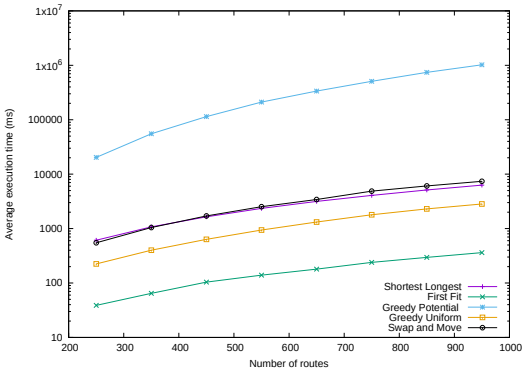

Figure 3.17 – Computation time (logarithmic scale) function of the number of datagrams of all algorithms on 10000 instances of load 1

3.3 From Large to Small Datagrams

In this section, we explain how we can trade load or buffering in the network to reduce the size of datagrams down to $\tau = 1$. This further justifies the interest of Sec. 3.2, where specific algorithms for $\tau = 1$ are given.

3.3.1 Datagram of Size One by Increasing the Load

We describe here a reduction from an instance of PAZL to another one with the same period and number of datagrams but the size of the datagrams is doubled. This instance is equivalent to an instance with $\tau = 1$, by dividing everything by the datagram size. Thus we can always assume that $\tau = 1$, if we are willing to double the load.

Theorem 20. *Let I be an instance of* PAZL *with n datagrams and load λ. There is an instance J with n datagrams of size 1 and load 2λ such that an assignment of J can be transformed into an assignment of I in polynomial time.*

Proof. From $I = (P,\tau,(\delta_0,\ldots,\delta_{n-1}))$, we build $I' = (P, 2\tau, (\delta'_0,\ldots,\delta'_{n-1}))$, where $\delta'_i = \delta_i - (\delta_i \mod 2\tau)$. The instance I' has a load twice as large as I. On the other hand, all its delays are multiples of 2τ hence solving PAZL on I' is equivalent to solving it on $I'' = (P/2\tau, 1,(\delta_0/2\tau,\ldots,\delta_{n-1}/2\tau))$, as already explained in the proof of Lemma 6.

Let us prove that an assignment A' of I' can be transformed into an assignment A of I. Consider the datagram i with offset $A'(i)$, it uses all times between $A'(i)$ and $A'(i)+2\tau-1$ in the first period and all times between $A'(i)+\delta_i-(\delta_i \mod 2\tau)$ to $A'(i)+2\tau-1+\delta_i-(\delta_i$

mod 2τ) in the second period. If $\delta_i \mod 2\tau < \tau$, we set $A(i) = A'(i)$, and the datagram i of I is scheduled "inside" the datagram i of I', see Fig. 3.18. If $\tau \leq \delta_i \mod 2\tau < 2\tau$, then we set $A(i) = A'(i) - \tau$. There is no collision in the assignment A, since all datagrams in the second period use times which are used by the same datagram in A'. In the first period, the datagrams scheduled by A use either the first half of the same datagram in A' or the position τ before, which is either free in A' or the second half of the times used by another datagram in A' and thus not used in A. □

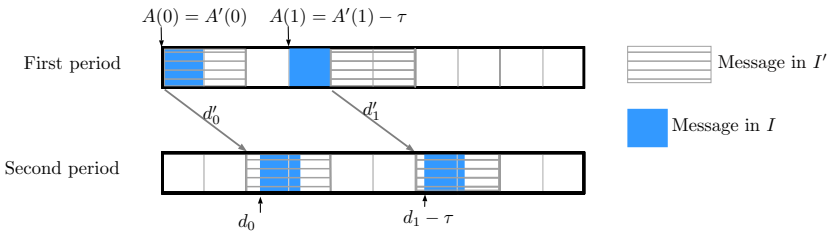

Figure 3.18 – Building I from I' as explained explained in Th. 20

Remark that combining Greedy Random and Theorem 20 allows to solve PAZL on random instances, with probability one when the number of routes goes to infinity and the load is strictly less than 1/2. This explains why we have not presented nor analyzed in details an algorithm designed for arbitrary τ on random instances, since any greedy algorithm, relying on optimizing $Fo(A)$, cannot guarantee anything for load larger than 1/2. However, in Sec. 3.1.4, we presented `Compact Fit`, a simple greedy algorithm which exhibits good performance on random instances.

3.3.2 Trade-off between Latency and Datagram Size

The problem PAZL is a simplified version of PALL, the practical problem we address, allowing a single degree of freedom for each datagram: its offset. We may relax it slightly to be more similar to what is studied in Chapter 4: we allow buffering a datagram i during a time b between the two contention points, which translates here into changing δ_i to $\delta_i + b$. The quality of the solutions obtained for such a modified instance of PAZL are worst since the buffering adds latency to the datagrams. We now describe how we can make a trade-off between the added latency and the size of the datagrams, knowing that having smaller datagrams helps to schedule instances with higher load.

The idea is to buffer all datagrams so that their δ_i have the same remainder modulo τ. It costs at most $\tau - 1$ of buffering, which is not so good, since algorithms optimizing the

latency do better on random instances, see [42]. However, it is much better than buffering for a time P, the only value for which we are guaranteed to find an assignment, whatever the instance. When all delays are changed so that δ_i is a multiple of τ, we have an easy reduction to the case of $\tau = 1$, by dividing all values by τ, as explained in the proof of Lemma 6.

We can do the same kind of transformation by buffering all datagrams, so that δ_i is a multiple of τ/k. The cost in terms of latency is then at most $\tau/k - 1$ but the reduction yields datagrams of size k. For small size of datagrams, it is easy to get better algorithm for PAZL, in particular for $\tau = 1$ as we have shown in Sec. 3.2. Here we show how to adapt `Compact Pairs` to the case of $\tau = 2$, to get an algorithm working with higher load.

Theorem 21. *`Compact Pairs` on instances with $\tau = 2$ always solves* PAZL *positively on instances of load less than* $4/9$.

Proof. We assume w.l.o.g that there are less datagram with even δ_i than odd δ_i. We schedule compact pairs of datagrams with even δ_i, then we schedule single datagram with odd δ_i. The worst case is when there is the same number of the two types of datagrams. In the first phase, if we schedule $n/2$ datagrams, the number of forbidden offsets is $(2 + 3/2)n/2 = 7n/4$. In the second phase, if we schedule $n/2$ additional offsets, the number of forbidden offsets is bounded by $(1+3/2)n/2+(1+1)n/2 = 9n/4$. Hence, both conditions are satisfied and we can always schedule datagrams when $n \leq (4/9)m$. □

We may want to add less latency to the datagram using the longest route. A natural idea is to choose the datagram with the longest route as the reference remainder by subtracting its remainder to every delay. As a consequence, this datagram needs zero buffering. However, the datagram with the second longest route may have a remainder of $\tau - 1$, thus the worst case increase of total latency is $\tau - 1$.

Another aim would be to minimize the average latency rather than the worst latency. We prove that we can do the transformation yielding $\tau = 1$ while optimizing the average latency. The only degree of freedom in the presented reduction is the choice of the reference remainder since all other delays are then modified to have the same remainder. Let us define the total latency for a choice t of reference time, denoted by $L(t)$, as the sum of buffering times used for the datagrams, when t has been removed from their delay. If we sum $L(t)$, from $t = 0$ to $\tau - 1$, the contribution of each datagram is $\sum_{i=0}^{\tau-1} i$. Since there are n datagrams, the sum of $L(t)$ for all t is $n\tau(\tau - 1)/2$. There is at least one term of

the sum less than its average, hence there is a t_0 such that $L(t_0) \leq n(\tau-1)/2$. Hence, the average delay for a datagram, with t_0 as reference is less than $(\tau-1)/2$.

Conclusion

In this chapter, we study the problem PAZL that is finding periodic assignments without buffering on star routed networks, a routed network with two serial contention points. We show that for messages of arbitrary size, there is always a solution as soon as the load of the network is less than 40%. To do so, we propose several greedy algorithms of increasing sophistication. They rely on optimizing different measures of how many positions there are in the second period, given a partial assignment and any new datagram to schedule. We give a canonical representation of an assignment, whose compactness allow to derive an FPT algorithm to solve PAZL parametrized by the number of routes. When the number of routes is less than 20, we can thus find in reasonable time an optimal solution to PAZL.

For messages of size 1, we prove that it is always possible to schedule them, when the load is less than 61% using a polynomial time algorithm. The algorithm relies on optimizing a potential, which represents, given a partial assignment, how many possible positions there are for the datagrams not yet scheduled. The algorithm is not greedy, contrarily to those presented for the case $\tau > 1$, since optimizing the potential requires a procedure of local optimization which exchange a scheduled datagram with an unscheduled one. Furthermore, we study the simplest random greedy algorithm solving PAZL, and show that, for a given load lower than one, almost all instances admit a solution with high probability, explaining why most greedy algorithms work so well in practice. The study of the special case of $\tau = 1$ is then justified, by providing several reductions from $\tau > 1$ to $\tau = 1$, while increasing the load or the latency only mildly.

The performance of the presented algorithms over average instances are shown to be excellent empirically (`Compact Fit` for large τ and `Swap and Move` for $\tau = 1$) for loads up to 0.7, for tens to hundreds of datagrams. Hence, we can use the simple algorithms presented here to schedule C-RAN datagrams *without using buffer nor additional latency* in polynomial time, if we are willing to use only half the bandwidth of the shared link.

Several questions on PAZL are still unresolved, in particular its NP-hardness and the problem of doing better than load 0.5 for arbitrary τ and random instances. We could also consider more complex network topologies with several shared links, most presented algo-

rithms (`First Fit`, `MetaOffset`, `Compact Pairs`, `Swap and Move`) could easily be adapted to this context. On star routed network, our experiments show that most of the time, there is a solution for PAZL when the load is under 0.8, but not for higher loads. Hence, in the next chapter, we study the problem PALL on highly loaded star routed networks. Problem PALL gives an higher degree of liberty to build assignments by allowing a buffer in one contention point of the network, at the cost of some additional latency.

Chapter 4

Scheduling Unsynchronized Periodic Datagrams with a single Buffer

This chapter is taken from a published paper [41] and its extended version [42].

4.1 Solving PALL on Star Routed Networks

In this section, we consider the more general PALL problem on star routed networks. The datagrams are allowed to wait in the BBUs to yield more possible assignments. Hence, we allow the process time of a route to be greater than the length of the route, but it must be bounded by its deadline.

4.1.1 Simple Star Routed Networks

Often in real networks, the length of the routes are not arbitrary and we may exploit that to solve PALL easily. For instance all the weights on the arcs (c_1,c_2) are the same if all the BBUs are in the same data-center and all datagrams require the same time to be processed in the BBUs. Finding an assignment in that case is trivial: send all datagrams so that they follow each other without gaps in c_1. In the corresponding canonical routed network, one can set $o_i = i\tau$. Since all arcs (c_1,c_2) are of weight zero in this case, the interval of time used in c_2 are the same as for c_1 and there is no collision in c_2.

Another possible assumption would be that all deadlines are larger than the longest route. It may happens when, in the network we model, all RRHs are at almost the same distance to the shared link.

Proposition 5. *Let $N = (\mathcal{R}, \omega)$ be a canonical star routed network with n routes, let $P \geq n\tau$ and let d be a deadline function. Let r_{n-1} be the longest route, and assume that*

for all $r \in \mathcal{R}$, $d(r) \geq \lambda(r_{n-1})$. *Then, there is a* (P,τ)*-periodic assignment for* N *and* d *and it can be built in time* $O(n)$.

Proof. The idea is to set the waiting times of all routes so their datagrams behave exactly as the datagram of r_{n-1}. The offset of the route r_i is set to $i\tau$, which ensures that there is no collision in c_1 as soon as $P \geq n\tau$. The waiting time of the route r_i is $w_i = \lambda(r_{n-1}) - \lambda(r_i)$.

The time at which the datagrams of r_i arrives in c_2 is $t(r_i, c_2) = w_i + i\tau + \lambda(r_i)$. Substituting w_i by its value, we obtain $t(r_i, c_2) = i\tau + \lambda(r_{n-1})$. Hence, there is no collision in c_2. We denote by A the defined assignment. By definition of the transmission time, we have $TR(r_i,A) = w_i + \lambda(r_i) = \lambda(r_{n-1})$. By hypothesis, $d(r_i) \geq \lambda(r_{n-1})$, which proves that the assignment respect the deadlines.

Finally, the complexity is in $O(n)$ since we have to find the maximum of the length of the n routes and the computation of each w_i is done by a constant number of arithmetic operations. □

4.1.2 Two Stages Approach

We may decompose an algorithm solving PALL on a star routed network in two parts: first set all the offsets of routes so that there is no collision in c_1 and then knowing this information find waiting times so that there is no collision in c_2 while respecting the deadlines.

First, we give several heuristics to choose the offsets, which are experimentally evaluated in Section 4.1.6. We assume that the star routed network is in canonical form. We send the datagrams through c_1 in a compact way (no gap between datagrams). It means that for n routes, denoted by $r_0, \ldots, r_{n-1}$, the offsets are $o_i = \sigma(i) \times \tau$, for some permutation $\sigma \in \Sigma_n$. We consider the following orders σ:

- *Decreasing Margin* (DM): Decreasing order on the margin of the routes.
- *Increasing Margin* (IM): Increasing order on the margin of the routes.
- *Decreasing Arc* (DA): Decreasing order on the length of the arcs (c_1,c_2).
- *Increasing Arc* (IA): Increasing order on the length of the arcs (c_1,c_2). This sending order yields a (P,τ) periodic assignment in which the waiting times are zero, if the period is large enough (see Proposition 3).

We also propose to fix the offsets of the routes according to some random order. If we pack the datagrams as previously, we call *Random Order* (RO) the heuristic of choosing an order uniformly at random. We may also allow some time between two consecutive datagrams in c_2. The order of the routes in c_1 is still random and we consider two variations. Either the time between two datagrams in c_1 is random and we call this heuristic *Random Order and Random Spacing* (RORS) or the time between two consecutive datagrams is always the same and we call this heuristic *Random Order and Balanced Spacing* (ROBS).

We call **W**aiting **T**ime **A**ssignment or WTA the problem PALL on a star routed network, with the offsets of the routes also given as input. A solution to WTA is a valid assignment such that the offsets coincide with those given in the instance.

In the rest of the section we study different methods to solve WTA either by polynomial time heuristics or by an FPT algorithm. The methods to solve WTA are then combined with the heuristics proposed to fix the offsets of the routes to obtain an algorithm solving PALL.

4.1.3 Greedy Scheduling of Waiting Times

We now solve the problem WTA, hence we are given as input a routed network, a deadline function and an offset for each route. The **release time** of a route is defined as the first time its datagram can go through c_2: for a route r with offset o_r, it is $\lambda(r,c_2) + o_r$.

The first algorithm we propose to solve WTA is a greedy algorithm which sets the waiting times in a greedy way, by prioritizing the routes with the earliest deadline to best satisfy the constraints on the process time. We call it `Greedy Deadline`, and it works as follows. Set $t = 0$ and $U = \mathcal{R}$. While there is a route in U, find $s \geq t$ the smallest time for which there is $r \in U$ with a release time lower or equal to s. If there are several routes in U with a release time lower or equal to s, then r with the smallest deadline is selected and set $w_r = s - \lambda(r,c_2)$, $t = s + \tau$ and $U = U \setminus \{r\}$.

This algorithm does not take into account the periodicity, which may create collisions. Let r_0 be the first route selected by the algorithm, then $t_0 = t(r_0,c_2)$ is the first time at which a datagram go through c_2. Then, if all routes r are such that $t(r, c_2) \leq t_0 + P - \tau$, then by construction, there is no collision on the central arc. However, if a route r has $t(r, c_2) > t_0 + P - \tau$, since we consider everything modulo P to determine collision, it may collide with another route. Therefore we correct `Greedy Deadline` in this way: $s \geq t$ is the smallest time for which there is $r \in U$ with a release time lower or equal to s *such*

Route	0	1	2	3	4
Deadline	10	15	5	7	30
Release time	0	2	3	16	17
Waiting time	0	5	1	0	15

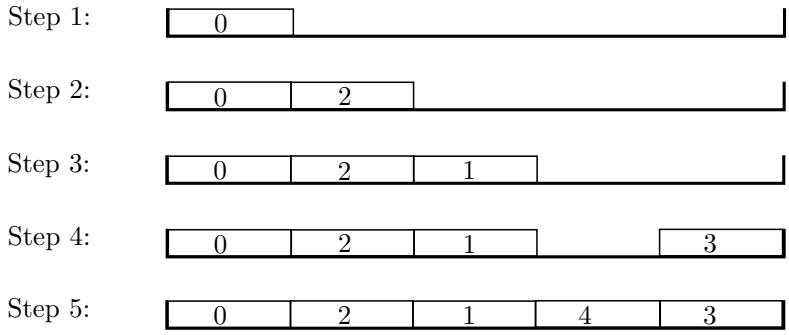

Figure 4.1 – A run of `Greedy Deadline` with $P = 20, \tau = 4$.

that there is no collision if a datagram goes through c_2 at time s. This rule guarantees that if `Greedy Deadline` succeeds to set all waiting times, it finds a solution to WTA, as illustrated in Figure 4.1. However, it can fails to find the value s at some point because the constraint on collisions cannot be satisfied. In that case `Greedy Deadline` stops without finding a solution.

The complexity of `Greedy Deadline` is in $O(n \log(n))$, using the proper data structures. The set of routes $\mathcal{R}$ must be maintained in a binary heap to be able to find the one with smallest deadline in time $O(\log(n))$. To deal with the possible collisions, one maintains a list of the intervals of time during which a datagram can go through c_2. Each time the waiting time of a route is fixed, an interval is split into at most two intervals in constant time. During the whole algorithm, each element of this list is used at most twice either when doing an insertion or when looking for the next free interval. Hence, the time needed to maintain the list is in $O(n)$.

4.1.4 Earliest Deadline Scheduling

The problem WTA is the same as a classical earliest deadline scheduling problem, if we forget the periodicity. Given a set of jobs with *release times* and *deadlines*, schedule all jobs on a single processor, that is choose the time at which they are computed, so that no two jobs are scheduled at the same time. A job is always scheduled after its release time and it must be dealt with before its deadline. Let us call n the number of jobs, the problem can be solved in time $O(n^2 \log(n))$ [35] when all jobs have the same running

time and it gives a solution which minimizes the time at which the last job is scheduled. On the other hand, if the running times are different the problem is NP-complete [36]. The polynomial time algorithm which solves this scheduling problem is similar to `Greedy Deadline`. However, when it fails because a job finishes after its deadline, it changes the schedule of the last jobs to find a possible schedule for the problematic job. The change in the scheduling is so that the algorithm cannot fail on the same job a second time except if there is no solution, which proves that the algorithm is in polynomial time.

The problem WTA is the same as this scheduling problem but adding constraints arising from the periodicity. The jobs are the routes, the size of a datagram is the running time of a job, the deadline and the release time are the same in both models. Let us call **M**inimal **L**atency **S**cheduling, denoted by `MLS`, the algorithm which transforms an instance of WTA into one of the described scheduling problem to solve it in time $O(n^2 \log(n))$ using the algorithm of [35].

Recall that $t(r,c_2)$ is the time at which the datagram of r goes through c_2. Let us denote by t_{min} and t_{max} the smallest and largest value of $t(r_i,c_2)$ for all $i \in [n]$. When `MLS` finds an assignment A, it always satisfies $PT(r) < d(r)$ for all r. Moreover, by construction `MLS` schedules the datagrams without collision if we forget about the periodicity (each route send only one datagram). Let us assume that $t_{max} - t_{min} \leq P - \tau$, then all datagrams go through c_2 during a interval of time less than P. Hence, when we compute potential collisions modulo P, all the relative positions of the datagrams stay the same which implies there is no collision. However, if $t_{max} - t_{min} > P - \tau$, then computing $t(r_i,c_2)$ modulo P for all i may reveal some collision. Since the scheduling algorithm minimizes t_{max}, it tends to find small values for $t_{max} - t_{min}$ and `MLS` may succeed in finding a valid assignment (as shown in Section 4.1.6), but not for all instances.

We now present a variant of the previous algorithm, that we call **P** **M**inimal **L**atency **S**cheduling, denoted by `PMLS`. The aim is to deal with the periodicity, by modifying the instance without changing the possible assignments, so that the chance of finding a solution with $t_{max} - t_{min} \leq P - \tau$ are larger. Remark that if an instance has a valid assignment, we can guarantee that one route has a waiting time zero in some valid assignment.

Algorithm `PMLS` runs, for each route $r \in \mathcal{R}$, the algorithm `MLS` on an instance defined as follows. Let $RT(r)$ be the release time of r, subtract it to all the release times and deadlines of the other routes. Therefore, $RT(r)$ is zero in the instance we build and the waiting time w_r is set to zero. Hence the datagram of r goes through c_2 at time 0 and $t_min = 0$. Then,

as in Proposition 2, the instance is modified so that all release times are in $[P - \tau]$. Each release time $RT(r_i)$ is replaced by $RT(r_i) \mod P$ and $d(r_i) = d(r_i) - (RT(r_i) - RT(r_i) \mod P)$. Furthermore, if the release time of a route r is between $P-\tau$ and P, we set it to 0 and $d(r) = d(r) - P$. The deadline of each route is set to the minimum of its deadline and $P - \tau$. Hence, if `MLS` finds a solution for such a modified instance, we have by construction of the instance $t_{max} \leq P - \tau$. Since $t_{min} = 0$, the assignment is valid. Algorithm `PMLS` returns the first valid assignment it finds when running `MLS` for some $r \in \mathcal{R}$.

The instance of WTA we have defined in this transformation is equivalent to the original instance, except we have fixed the waiting time of r to be zero. If there is some valid assignment, then at least one route has waiting time zero, then if `MLS` finds an assignment then `PMLS` also finds one. Algorithm `MLS` is used at most n times, thus the complexity of `PMLS` is in $O(n^3 \log(n))$. Note that `PMLS` is a heuristic and may fail to find a solution even if it exists. It is the case when, for the n modified instances, there is no solution with the times $t(r_i,c_2)$ using an interval of time less than P in c_2.

4.1.5 FPT algorithms for `WTA` and `PALL`

As a warm-up, we give a simple FPT algorithm for WTA which is practical, and then we build on it to give a more complicated FPT algorithm for PALL. Unfortunately, the dependency on n the number of routes in the second algorithm is yet too large to be useful in practice.

Theorem 22. WTA $\in$ FPT *over star routed networks when parametrized by the number of routes.*

Proof. Consider an instance of WTA, given by a release time and a deadline for each route. We show that we can build a set of instances form the original one such that one of these instances has a valid assignment if and only if the original instance has a valid assignment.

As for `PMLS`, for each route r, we consider the instance where r has release time and waiting time zero ($RT(r) = w_r = 0$). The release times and deadlines of all routes are modified so that all release times are less than P as in the transformation described for `PMLS`. If there is an assignment such that $t_{max} < P - \tau$, then the periodicity does not come into play for this assignment and the algorithm `MLS` will find the assignment as explained in Section 4.1.4.

Now, remark that if there is a valid assignment for an instance with the properties just stated, then there is a valid assignment satisfying for all i, $t(r_i,c_2) \leq 2P - \tau$. Indeed,

if there is a i such that $t(r_i,c_2) \geq 2P$ in a periodic assignment, then we have $w_i = t(r_i,c_2) - \lambda(r_i,c_2) \geq P$. Hence, we can set $w_i = w_i - P \geq 0$ and we still have a valid assignment. Moreover, for all $r_i \neq r$, it is not possible that $2P - \tau < \lambda(r_i,c_2) \leq 2P$, since it would imply a collision between r and r_i.

From an instance I, with the properties of the first paragraph, we define a new instance I' whose valid assignments are a subset of the ones of I. Moreover, one of the valid assignments of I' satisfies that for all i, $t(r_i,c_2) \leq P - \tau$ and is thus found by MLS. Let us now consider A a valid assignment of I, we can assume that $t(r_i,c_2) \leq 2P - \tau$. Let S be the set of routes r_i such that $P - \tau < t(r_i,c_2) \leq 2P - \tau$. The instance I' is defined by changing, for all route $r \in S$, $RT(r)$ and $d(r)$ to $RT(r) - P$ and $d(r) - P$. Then, by construction A is also a valid assignment of I'. Assignment A as a solution of I', satisfies $t(r_i,c_2) \leq P - \tau$ for all $i \in [n]$.

The FTP algorithm is the following: for each route r build a modified instance as in PMLS. Then, for each subset S of routes, remove P to the release time and to the deadline of each route in S and run MLS on the instance so modified. If there is a valid assignment, then we have proved that there is some S, such that the instance built from S has a valid assignment with $t(r_i,c_2) \leq P - \tau$ for all $i \in [n]$. Hence, MLS finds a valid assignment for this instance. □

The algorithm of Theorem 22 has a complexity of $O(2^n n^3 \log(n))$. If we consider some valid assignment, the routes r with $t(r,c_2) > P$, must satisfy $t(r,c_2) > P + \tau$ to avoid collision with the first route. Hence, the deadline of these routes must be larger than $P+\tau$. These routes are exactly those that must be put in S, hence we can enumerate only the subsets of routes with a deadline larger than $P + \tau$. In practice, only k routes have a deadline larger than $P + \tau$ with $k << n$, and we need only to consider 2^k subsets. Let us call this algorithm **A**ll **S**ubsets PMLS, and let us denote it by ASPMLS.

Theorem 23. PALL $\in$ FPT *over star routed networks when parameterized by the number of routes.*

Proof. Consider a star routed network, instance of PALL with a valid assignment. We characterize such a valid assignment by a set of necessary and sufficient linear equations and inequations it must satisfy. These conditions are expressed on the values $t(r,c_1)$ and $t(r,c_2)$ and setting those value is equivalent to setting the offsets and the waiting times, that is choosing an assignment.

First, we assume the star routed network is canonical. Hence, there is an assignment A, such that for all routes $r \in \mathcal{R}$, $0 \leq t(r,c_1) < P - \tau$ and $0 \leq t(r,c_2) < 2P - \tau$. By definition $t(r,c_2) = t(r,c_1) + \omega(r,c_2) + w_r$. Since a waiting time is non-negative, we have $t(r,c_2) \leq t(r,c_1) + \omega(r,c_2)$. Now, let S be the set defined as in Theorem 22, of the routes r such that $P - \tau < t(r,c_2) \leq 2P - \tau$. We want to guarantee that for $r \in \mathcal{R}$, $t(r,c_2) \in [P - \tau]$. To do that, we replace the inequation $t(r,c_2) \leq t(r,c_1) + \omega(r,c_2)$ by $t(r,c_2) \leq t(r,c_1) + \omega(r,c_2) - P$ and $d(r)$ by $d(r) - P$ for all $r \in S$. Remark that the presented linear constraints now depend on S, which itself depends on A.

Let σ and σ' be two permutations of Σ_n such that σ is the order of the routes $r_0, \ldots, r_{n-1}$ according to the value $t(r,c_1)$ and σ' according to the value $t(r,c_2)$. Since all $t(r,c_1)$ and $t(r,c_2)$ are in $[P-\tau]$, we have $t(r,c_1) = t(r,c_1)\ mod P$ and $t(r,c_2) = t(r,c_2)\ mod P$. Hence, we can express the constraints on the absence of collision between routes by adding the following equations to the ones of the previous paragraph:

- for all $i < n-1$, $t(r_{\sigma_i},c_1) \leq r_{\sigma_{i+1}},c_1 + \tau)$ (no collision in c_1)
- for all $i < n-1$, $t(r_{\sigma'_i},c_2) \leq r_{\sigma'_{i+1}},c_2 + \tau)$ (no collision in c_2)
- for all $i < n$, $t(r_i,c_2) < d(r_i)$ (deadline respected)

Consider now the system of inequations $E_{S,\sigma,\sigma'}$ we have built from A. The values $t(r,c_1)$ and $t(r,c_2)$ given by A satisfy the system by construction. Moreover, any solution to these equations yields a valid assignment, because the equations guarantee that there is no collision, that the offsets and the waiting times are non-negative and that all routes meet their deadlines. However, a solution of $E_{S,\sigma,\sigma'}$ may be rational, while offsets and waiting times must be integers. We use the following simple fact: $x + e_1 \leq y + e_2$ implies $\lceil x \rceil + e_1 < \lceil y \rceil + e_2$ when e_1 and e_2 are integers. Since all equations of $E_{S,\sigma,\sigma'}$ have this form, if we take the upper floor of the components of a solution, it is still a solution of $E_{S,\sigma,\sigma'}$ with *integer* values. As a consequence, any solution to $E_{S,\sigma,\sigma'}$ yields a valid assignment of the original instance of PALL.

The algorithm to solve PALL is the following. Build $E_{S,\sigma,\sigma'}$ for all triples (S,σ,σ'). Then, solve each linear system, and if it admits a solution, convert it back into a valid assignment of the instance of PALL by rounding. There are 2^n sets S and $n!$ orders σ. Thus, $2^n(n!)^2$ systems with $2n$ variables and a bitsize of the same order as the original instance are solved at most. Since solving each system can be done in polynomial time in

the size of the instance, it proves that the algorithm is FPT in n. Moreover, it always finds a valid assignment if there is one, since we have shown that from a valid assignment, we can find (S,σ,σ') for which the values associated to A satisfy $E_{S,\sigma,\sigma'}$.

□

4.1.6 Experimental Evaluation

Evaluating the Necessary Margin We set the number of routes to 8 to make comparisons with the results of Chapter 3 easier. We draw uniformly the weights of the arcs of the fronthaul network in $[P]$. We use *the same deadline* for all routes, which is the most common constraint, when modeling a C-RAN problem: all RRHs have the same latency constraint and all BBUs take the same time to process the answer.

We define the **margin** of an instance as the margin of the longest route (i.e. the longest value for $\lambda(r)$) . The margin represents the *logical latency* which can be used by the communication process, without taking into account the physical length of the network, since it cannot be changed. For a given star routed network, it is equivalent to set the margin or all the deadlines, since we have assumed the deadlines are equal. However, to compare different star routed networks with different sizes of routes, the margin is more relevant than the deadline. Hence, in our experiments, we test margins from 0 to 3,000 tics to understand how much logical latency is needed to find an assignment. We look at two different regimes, a medium load of 80% and a high load of 95%. Considering smaller load is not relevant since we can solve the problem using bufferless assignments, as shown in Chapter 3.

We first try to understand what is the best choice of heuristics for the first stage of the algorithm. The first stage is followed in this experiment by `Greedy Deadline`, the simplest algorithm to solve WTA. In Figure 4.2, the success rate of all possible first stage heuristics to solve PALL is given, function of the margin of the instances. The success rate is an average computed over 10,000 random star routed networks.

According to our experiments, policy IA, that is sending the datagrams on increasing order on the length of the arcs (c_1,c_2), does not work well. It corresponds to the policy of Proposition 3 which we already know to be bad for PAZL when the routes are long as in this experiment. Sending on decreasing order on the margin of the routes (DM) or on the length of the arcs (c_1,c_2) (DA) work better and it seems that DA is better than DM, especially in a loaded network.

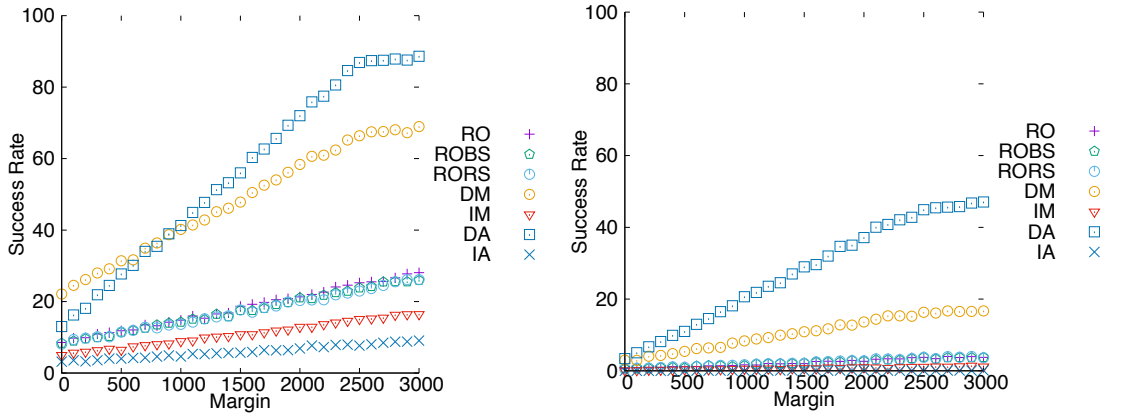

Figure 4.2 – Success rate of different sending orders, left 80% load, right 95% load.

Remark that sending the datagrams using a random order does not perform well, but better than IM and IA, which shows that the latters are a poor choice for the first stage of our algorithm. The interest of using a random order is that we can draw many of them. In Figure 4.3 the same experiment is made for the three heuristics choosing an order at random, but we now draw 1,000 different random orders and solve each induced WTA instance using `Greedy Deadline`. The algorithm is considered to succeed as soon as a valid assignment is found for one order. Each random order drawn is used for RO, RORS and ROBS to make the comparison fairer.

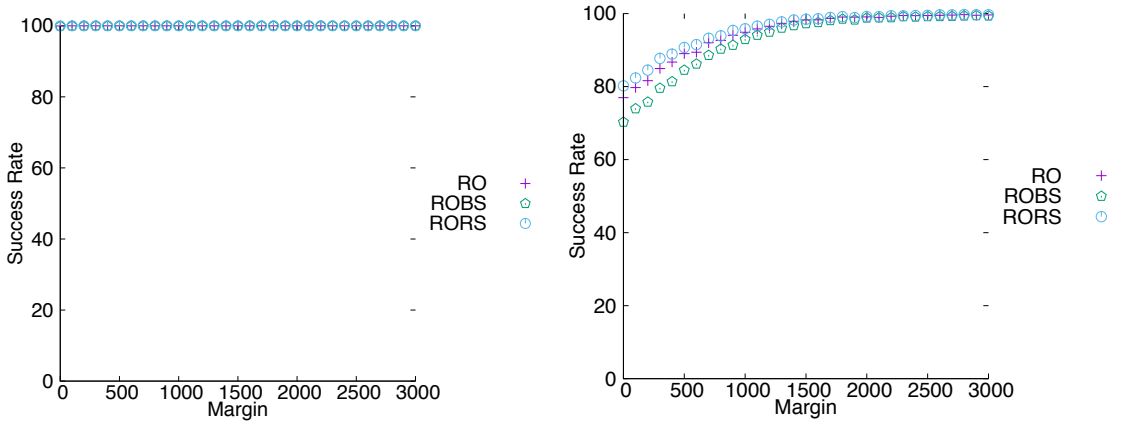

Figure 4.3 – Success rate of different sending orders with the random orders generated 1000 times, left 80% load, right 95% load.

First remark that our algorithms finds assignments with margin 0 for instances with 95% of load and long routes which was not possible when only looking for bufferless assignments (see Chapter 3). It justifies the interest of studying PALL and not only PAZL.

Using many random orders is much better than DA, the best policy using one specific

order. With a load of 95%, a solution is found with margin 0 most of the time. The three random order policies have similar performances, but RORS has slightly better success rate than the two others ones, under high load and small margin. Hence, in the following experiments, we draw 1,000 random orders using the policy RORS to set the offsets of the assignments.

We now compare the performances of the four different algorithms used in the second stage to set the waiting times. Since `Greedy Deadline` already finds assignments with margin 0 on mild loads, it is more interesting to focus on the behavior of the algorithms with high load. In Figure 4.4, we represent the success rate of the four algorithms with regards to the margin, computed over 10,000 random star routed networks generated with the same parameters as previously.

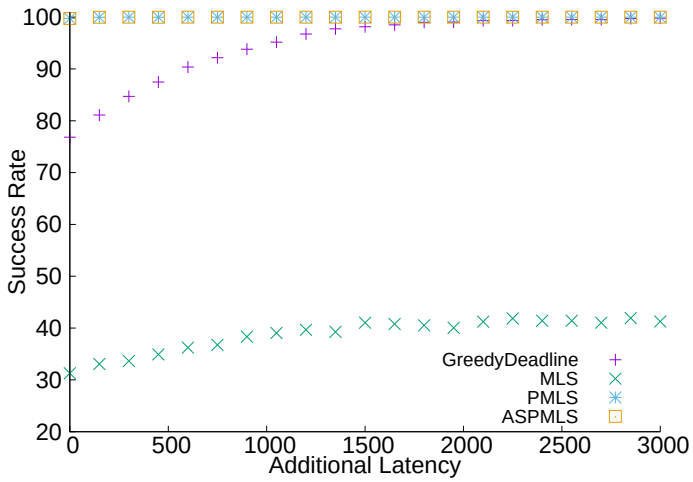

Figure 4.4 – Success rate of four algorithms solving PALL, 95% load

The `MLS` algorithm performs poorly, worst than `Greedy Deadline`, `PMLS` and `ASPMLS`, which shows that *taking into account the periodicity* is fundamental. Algorithm `Greedy Deadline` is close to 100% success rate for margins larger than 1,500 while `PMLS` and `ASPMLS` algorithms find a solution for more than 99% of the random instances, even *with a margin* 0. In other words, for very high load and no margin, there are very few instances for which we do not find an assignment. With a margin of 300, which corresponds to about 15μs of additional delay with the chosen parameters, we always find a solution.

It turns out that the performances of `PMLS` and `ASPMLS` are almost identical. Even with a load of 100% and a margin of 0, we have to draw 100,000 random instances before

finding one which can be solved by ASPMLS and not by PMLS. Since ASPMLS is of exponential complexity in n, it is not relevant to use it within the parameters of this experiment. To verify that, we present the computing time of PMLS and ASPMLS for different instance sizes. To stress the algorithms, we set the margin to 0 and the load to 95%. The table of Figure 4.5 shows the computation times of PMLS and ASPMLS, averaged on 1,000 instances.

# routes	8	12	16	20	24
ASPMLS (ms)	1.88	5.98	47.75	209.2	1815
PMLS (ms)	0.07	0.08	0.09	0.10	0.12
Ratio	27	78	523	2122	14882

Figure 4.5 – Computation time for PMLS and ASPMLS function of the number of routes

The complexity of both these algorithm depends on the number of routes. As shown in Figure 4.5, the time complexity of PMLS seems linear on *average*, while its theoretical worst case complexity is cubic. ASPMLS scales exponentially with the number of routes as expected. Both algorithms are usable for instances of 20 routes, but for 40 routes or more ASPMLS becomes too slow. Since ASPMLS almost never finds a solution when PMLS does not and is much slower, one should prefer to use PMLS.

When evaluating the computing time of our method, we should take into account how many random orders are drawn. In previous experiments, we have drawn 1,000 random orders which may be 1,000 time slower than using a single fixed order. There is a trade-off between the number of random orders and the success rate. We investigate the success rate of our algorithms with regards to the number of random orders drawn, a load of 95% and a margin 0. The table of Figure 4.6 presents the success rate for different numbers of sending orders, averaged over 10,000 instances, for Greedy Deadline, PMLS and ASPMLS.

# orders	1	10	100	1,000	10^4	10^5
Greedy Deadline	0.55	6.05	35.44	77.43	90.1	92.4
PMLS	82.04	98.84	99.71	99.80	99.83	99.83
ASPMLS	91.33	99.17	99.72	99.80	99.83	99.83

Figure 4.6 – Success rates function of the number of random orders drawn

First, observe that the better the algorithm to solve WTA is, the less random orders it needs in stage one to achieve its best success rate. In particular, ASPMLS has better results than PMLS for less than 1,000 random orders, but not beyond. This further justifies our choice to draw 1,000 random orders, to obtain the best success rate within the smallest

time.

The number of different orders is $7! = 5{,}040$ since we have 8 routes and the solutions are invariant up to a circular permutation of the order. Hence, for 8 routes it is possible to test every possible order. However the computation time of this exhaustive method scales badly with n. The fact that PMLS and ASPMLS have already high success rates for 10 random orders hints that even for a larger number of routes, drawing 1000 random orders is sufficient to obtain good assignments.

Harder Topologies Previous experiments use instances with weights of arcs uniformly drawn in a large interval. However, it is quite natural to consider that most routes are of roughly the same length or can be arranged in two groups of similar lengths, when the fronthaul network involves one or two data-centers.

By Proposition 5, there is an assignment with margin equal to the maximum difference between the sizes of the routes. Hence, if all routes have almost the same size, the needed margin is small. If the routes are drawn uniformly in a large interval, then the expected difference between the longest route and the second longest is large. This difference can be seen as a free waiting time for most routes, hence we expect to need little margin in this regime too. As a consequence, the harder instances should be for routes with length drawn in an interval of moderate size compared to the period.

Figures 4.7 and 4.8 show the probability of success of PMLS over 10,000 instances as a function of the margin. In Figure 4.7 the length of arcs are drawn in $[0,I]$, where I goes from 0 to 6400. As expected the success rate decreases when the size of the interval increases, until $I = 1600$, and then increases again. In the most difficult settings, only 78% of the instances can be solved with margin 0, and we need a margin of 1,900 to ensure that PMLS always finds a solution. Unfortunately, ASPMLS does not gives better results on these hard instances.

In Figure 4.8, we do the same experiment, except that the weights of arcs of half of the routes is drawn in $[I]$ and the length of the other half is drawn in $[P/2,P/2+I[$. The situation is the same as for the previous experiment but with better success rates, hence the case of two data centers is simpler to deal with.

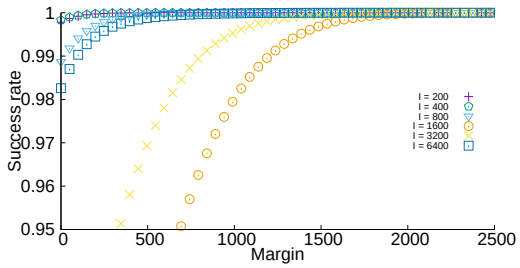

Figure 4.7 – Success rate of `PMLS`, with length of arcs drawn in $[I]$

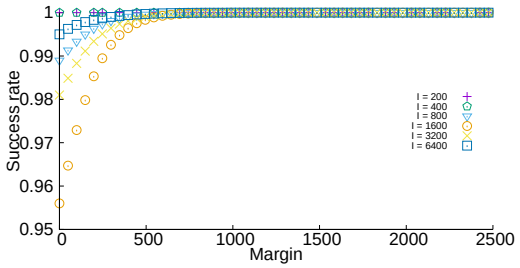

Figure 4.8 – Success rate of `PMLS`, with length of arcs drawn in $[I]$ or $[P/2, P/2 + I[$

4.1.7 Performance of Statistical Multiplexing

Now that we have designed and tuned `PMLS` to solve PALL efficiently, we compare its performances against the actual way to manage the messages in a network: *statistical multiplexing*, with a `FIFO` buffer in each node of the network to resolve collisions. For statistical multiplexing, the time at which the datagrams are sent in the network is not managed by the user as in our approach, thus we assume the offsets of each route is fixed to some random value, and they stay the same over time. We consider a second policy to manage buffers called `CriticalDeadline`. In a buffer with several datagrams, this policy sends the one with the smallest remaining margin, which is the time it can wait before missing its deadline.

We have implemented a statistical multiplexing simulator, to evaluate the performance of these two policies and to compare them to finding assignments with small margin by solving PALL. For statistical multiplexing, both contention points have a buffer. The process is not periodic: even if the offset of a route is the same each period, it is possible that some datagram do not arrive at the same time in a contention point in two consecutive periods because of buffering. Therefore we must measure the process time of each route over several periods if we want to compute the maximum latency of the network. We choose to simulate it for 1,000 periods but we have observed that the process time usually stabilizes in less than 10 periods. The **margin**, for statistical multiplexing, is defined as the maximum process time, computed as explained, minus the size of the longest route of the star routed network.

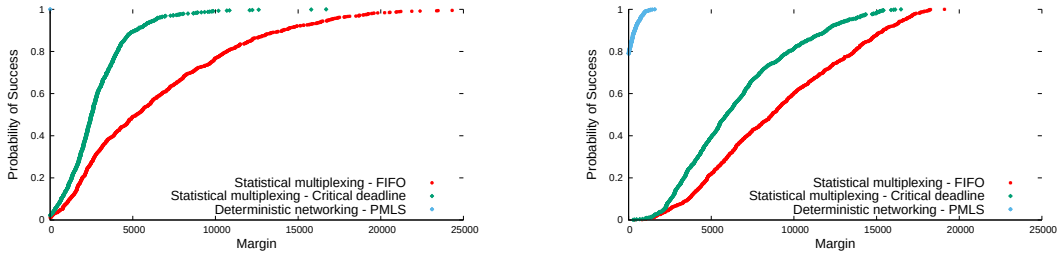

Figure 4.9 – Probability of success of statistical multiplexing and PMLS for several margins on random topologies when the size of the routes are distributed either on P (left) or on a small range of values (right).

In Figure 4.9, we represent the probability of success of statistical multiplexing and PMLS for different margins. The success rates are computed from 10,000 star routed networks for each margin. On the left part of Figure 4.9, the arcs of the network are uniformly drawn in $[P]$, while on the right part, the arcs of the network are uniformly drawn in [1600] (the hard case of the previous section). The others parameters of the experiences are the same as previously. We represent the distribution under high and light load for statistical multiplexing and under high load only for PMLS since under light load the margin is always 0.

The experiment clearly shows that statistical multiplexing does not ensure a minimal latency. For random topologies, the latency is extremely high when the load is high, with a margin of about 10,000 for the worst 10% which corresponds to half the period, that is 0.5ms. Even when the network is lightly loaded, 20% of the instances have a margin of more than 2,000 while PMLS finds an assignment with margin 0 in a highly loaded network 99% of the time!

For hard topologies, statistical multiplexing is slightly worst for small margins and the same for high margins. The settings are stressful for PMLS, we find an assignment in only 78% of the instances with margin 0, and it needs a margin of 2,000 to be sure to find an assignment. However, PMLS still vastly outperforms the statistical multiplexing both for the average margin and for the worst margin.

For each 1,000 tics of latency we save from the periodic process, we are able to lengthen the routes of 10km, which has a huge economical impact. We feel that it strongly justifies the use of a deterministic sending scheme for latency critical applications such as our

C-RAN motivating problem.

Conclusion

In this chapter, we propose solutions for the PALL problem. We decompose the problem in two steps. The first one consists in arbitrary set the offsets of the routes, and the second one is to solve the problem WTA in order to compute the waiting times. Several heuristics to choose the offset have been proposed and experimentally verified, and the best one is to generate a large number of random offsets for every route, to solve WTA on them and to keep the best solution. We propose polynomial time heuristics and an exact FPT algorithm that solve WTA when parametrized by the number of routes. This latter is built from a scheduling algorithm (that we call `MLS`) of the literature. We adapted `MLS` for periodicity under the name `PMLS`. In `PMLS`, we set one datagram to be the first in the period, and we set the deadline of the other datagrams according to the sending time of the first message and the period. We repeat this operation with every datagram at the first position on the period, and we keep the best solution. Because we reinforce the deadline constraint, `PMLS` does not ensure to find a solution if it exists. Thus, we proposed `ASPMLS`, an FPT algorithm that solves WTA if there is a solution. `ASPMLS` is based on the canonical form of the assignments and explores all subset of routes in order to not forget valid instances. We also propose an FPT algorithm to solve PALL when parametrized by the number of routes, but it consists of a list of constraints for linear programming and its combinatorial complexity is too high to be programmed, even on instances with few routes.

We show that `ASPMLS` and `PMLS` have excellent performances for Cloud-RAN parameters. They find a solution with 0 additional latency for 99.9% of the instances, even for load 0.95 on random instances. We also show that our solution largely outperforms statistical multiplexing, even using a buffering policy taking into account the latency.

Even if the performance are excellent on practical instances, the algorithm we propose here focuses on solving the problem WTA of computing the waiting time when the offsets are chosen, but not the entire problem : PALL. Furthermore, neither `Greedy Deadline`, `PMLS` nor `ASPMLS` are easily adaptable for more complex topologies than star routed networks, studied in next chapter for a synchronized version of PALL.

Chapter 5

Scheduling Synchronized Periodic Datagrams in Arbitrary Networks

In this chapter, we consider a problem similar to PALL with an additional constraint: the sending of the messages in all the sources of the routes must be **synchronized**. We need to add buffering on the first contention point of each route, otherwise the synchronization constraint makes collisions unavoidable. Since it is much harder to find assignments with low latency in this context, we allow buffering in all contention points of the routed network (and not only in the ones corresponding to BBUs). Hopefully, this higher degree of freedom to schedule the datagrams helps decrease the process time of the assignments.

We modify the model in order to take into account the buffers, and we define the problem MINSTRA, the synchronized version of MINTRA, studied in previous chapters. The algorithms presented in this section solve MINSTRA on the star routed network as in Chapters 3 and 4, but also on any directed acyclic multigraphs representing a routed network. We first show that greedy algorithms similar to those used for star shaped networks are not efficient in routed network of higher contention depth. Then, we present several local search heuristics (Hill Climbing, Simulated Annealing, Tabu Search) that improve on the greedy algorithms and find low latency assignments. Finally, we present an FPT Branch and Bound algorithm that gives the optimal solution, which allows to assess the performances of previous algorithms on small networks.

5.1 Model changes

5.1.1 A synchronized version of MINSTRA

The fronthaul network is still represented in this chapter by a routed network $N = (\mathcal{R}, \omega)$ but the set $\mathcal{B}$ of vertices with possible buffering is omitted. Indeed, in this chapter, $\mathcal{B}$ is

equal to $\mathcal{C}$, the set of contention points, that is buffering is allowed in all vertices.

In Chapter 3 and 4, the routed network is assumed to be a star routed network, a restriction that we now lift. We define by $\mathcal{R}_c$ the subset of routes in $\mathcal{R}$ containing c. Let $r \in \mathcal{R}$, with $r = (s,c_0,\ldots,c_l,t)$, then we say that c_i is of **contention depth** i for the route r, and we denote it by $cd(r,c_i) = i$. The contention depth of a contention point c is the maximum of its contention depth over all routes going through c: $cd(c) = \max_{r\in\mathcal{R}_c \text{ and } c\in r} cd(r,c)$. As a reminder, the **contention depth** of a routed network $N = (\mathcal{R},\mathcal{B},\omega)$ is the maximal number of contention points on a route in the routed network.

Let $r = (s,c_0,\ldots,c_l,t)$ be a route. As mentioned above, all sources emit a datagram at the same date. This means that, w.l.o.g. $o_r = 0$. In order to avoid contention, it is possible to buffer datagrams in all contention points. An **assignment**, denoted by A, is a function which associates a non negative integer value $A(r,c)$ to each couple (r,c) with $r \in \mathcal{R}$ and c a vertex of r. The values $A(r,c)$ represent the buffering times: a datagram of route r waits $A(r,c)$ tics in the buffer of c.

The **arrival time** of a datagram in vertex c_i of r, is the first time at which the datagram sent on r reaches c_i, and is defined by $t(r,c_i) = \lambda(r,c_i) + \sum_{k=0}^{i-1} A(r,c_k)$. The date at which a datagram reaches a vertex u_i is decomposed into a *physical delay* due to the time to go through the links before u_i and a *logical* delay caused by the use of buffers as determined by assignment A. The **sending time** of a datagram at vertex c_i of r, is the first time at which the datagram is sent by c_i. It is defined by $s(r,c_i) = t(r,c_i) + A(r,c_i)$. This is the arrival time of the datagram plus the buffering time given by A.

Consider v the last vertex of the route r, the transmission time of the datagram on r is denoted by $TR(r,A)$ as in Chapter 4 and is equal to $t(r,v)$. Then, the **transmission time** of an assignment A is defined as $TR(A) = \max_{r\in\mathcal{R}} TR(r,A)$. This is the time elapsed before the reception of the beginning of the last datagram. We denote by $TR(N)$ the best possible transmission time for the routed network N, that is the minimum of $TR(A)$ over all A valid assignments.

As in Chapter 4, given a network N, the objective is to minimize $TR(A)$, we thus define MINSTRA, the problem of computing A a (P,τ)-periodic valid assignment such that $TR(A) = TR(N)$.

Minimizing Synchronized TRansmission time of Assignments (MINSTRA)

Input: A routed network $N = (\mathcal{R},\omega)$, a period P, a datagram size τ.

Problem: Find A with $TR(A) = TR(N)$.

We evaluate the arithmetic complexity of our algorithms to solve MINSTRA, that is arithmetic operations are considered to be in constant time. Surprisingly, the complexity of the presented algorithms do not depend on P, τ or the weights of the routed network, but only on n the number of routes and d the contention depth.

5.1.2 Fronthaul networks modeling

Contention Depth One Each contention point of a routed network of contention depth one induces a connected component. Problem MINSTRA can be independently solved on each connected component of the network, hence the case with a single contention point is equivalent to contention depth one. Problem MINSTRA over a routed network with a single contention point is equivalent to WTA, a problem already studied in Chapter 4.

Star routed network The simplest case of contention depth 2 is a routed network with two contention points. This is enough to modelize our process of sending a datagram from an RRH to a BBU and back when there is a single contention point (a shared link between the RRHs and the data centers). This topology is the star routed network on which we have solved PAZL in Chapter 3 and PALL in Chapter 4.

Theorem 24. *The problem* MINSTRA *is* NP*-hard when restricted to star routed networks.*

Proof. The two flow shop problem studied in [37] is shown to be NP-hard. The problem is defined as follows: a set of n jobs have to be processed in sequence on two machines. Each job must be processed on machine 1 before being processed on machine 2. All jobs can be processed from time 0 on machine 1, then for a job i, there is a delay d_i between the end of the processing on machine 1 and the time at which it can be processed machine 2. The time needed to process a job is the same for all jobs and both machines. The objective is to minimize the makespan, that is the time at which the last job is scheduled.

We reduce an instance of the two flow shop problem to an instance of MINSTRA on a star routed network (cf Section 2.1.5): A job is a route, the time to process a job is τ the size of a datagram and the delay of the job i is the length of the arc (c_1,c_2) in the route r_i. If all first datagrams of a route can go through the routed network before the end of a period, then the periodicity of MINSTRA does not come into play. In other word, we want to ensure that there is an assignment A such that for all $r \in \mathcal{R}$, $TR(A,r) \leq P$. We let $P = \sum_{1 \leq i \leq n} \lambda(r_i)$ and for all $i \leq n$, we let $A(r_i,c_1) = \sum_{1 \leq j < i} \lambda(r_j)$ and $A(r_i,c_2) = 0$. By construction, A is

a (P,τ)-periodic assignment since there is always only one datagram moving through the network at some point in time and it satisfies for all $r \in \mathcal{R}$, $TR(A,r) \leq P$ by construction. Solving MINSTRA on the instance we have defined is equivalent to finding the minimal makespan in the two flow shop problem, which proves the theorem. □

The fronthaul networks we study have **coherent** routings, a classical requirement in telecommunication networks (see e.g. [20]). It means that if the routes r and r' go through two contention points u and v, they have the same subpath between u and v. This is true for the fronthaul networks we modelize. The coherent property is respected from the source to the arc representing the BBU and then from the arc representing the BBU to the target. As a consequence, routed networks obtained from fronthaul networks are directed acyclic multigraphs, as required in the definition of routed network.

Contention depth larger than one In this chapter, we deal with more general routed networks than star routed networks. The algorithms proposed here solves MINSTRA on every routed network which are directed acyclic graphs. We focus our study on symmetric fronthaul networks; networks in which the routes between the RRH and the BBU use the same links in both ways, but this property does not need to be enforced. We say that a routed network modeling a symmetric fronthaul network is a **symmetric routed network**.

Star routed networks, considered when solving PAZL and PALL in previous chapters are symmetric routed network: they are symmetric around the central arc (c_1,c_2). More generally, if a symmetric fronthaul network is of contention depth 2, then all routes which contain the same contention point of depth one also contain same contention point of depth two. Thus, every symmetric fronthaul network of depth 2 can be represented by several disjoint star routed networks.

Symmetric routed networks of higher contention depth are specifically studied in this chapter. For higher contention depth, a reasonable simplifying assumption is to consider that the length of the links in the datacenter are the same for all routes. Then, there is no contention on the link going out of the BBU, as explained in Section 5.3.4 and the routed networks are symmetrical around the contention point preceding the BBU.

5.2 Compact Representation of an Assignment

We define $\prec$, the pointwise order on assignments: $A_1 \preceq A_2$ if for all $r \in \mathcal{R}$, $TR(A_1,r) \leq TR(A_2,r)$. Moreover, we say that $A_1 \prec A_2$ if $A_1 \preceq A_2$ and there is an $r \in \mathcal{R}$ such that $TR(A_1,r) < TR(A_2,r)$. Remark that assignments which minimize $TR(A)$ are also minimal for $\prec$. Hence, it is enough to consider minimal assignments for $\prec$ to solve MINSTRA.

We explain in this section how to represent most assignments in a compact way, forgetting about the precise buffering time by only considering informations about the order of the datagrams in each contention point. All minimal assignments have a compact representation, which implies that we do not need to consider assignment without a compact representation when solving MINSTRA. It allows to design an FPT algorithm for MINSTRA by going through all compact representations, but also to design good polynomial time heuristics using Tabu Search or Simulated Annealing, since one can easily define the neighborhood of a compact representation.

Definition 3 (Compact assignment)**.** *Let $(G,\mathcal{R})$ be a routed network. A compact assignment CA is a function which maps to each contention point c in G a pair (O_c,S_c), where O_c is an order on $\mathcal{R}_c$ and S_c is a subset of $\mathcal{R}_c$.*

5.2.1 From a Valid Assignment to its Compact Representation

Let us define a function which maps a valid assignment A to a compact assignment, called the compact representation of A, denoted by $CR(A)$. We assume that for all contention points u, there is a route $r \in \mathcal{R}_u$ such that $A(r,u) = 0$. The routes in $\mathcal{R}$ are indexed by the integers in $[n]$. Say w.l.o.g. that r_0 is the route of smallest index such that $A(r_0,u) = 0$. The datagram of r_0 arrives, and goes to the next contention point, at time $t(r_0,u)$. Let us define the **normalized arrival time** of r at u: for all $r \in \mathcal{R}_u$, $nt(r_0,r,u) = (t(r,u) - t(r_0,u)) \mod P$. It is the time at which the datagram of r arrives at u, in a period normalized so that the datagram of r_0 goes through u at time 0. Similarly, we define the **normalized sending time** as $ns(r_0,r,u) = (s(r,u) - t(r_0,u)) \mod P$.

We define O_u as the order on the routes of $\mathcal{R}_u$ induced by the values $ns(r,u)$. The set S_u is defined as the set of routes going through u such that $ns(r_0,r,u) < nt(r_0,r,u)$. Intuitively, the time being seen as cut into periods $[t(r_0,u) + iP, t(r_0,u) + (i+1)P[$ with $i \in \mathbb{N}$, then S_u represents the set of routes with a datagram going through u in the period *after* the one it has been available in.

Fig. 5.1 illustrates how a compact representation is computed from an assignment on a single node u. On top, the datagrams are represented by sending time $s(r_i,u)$ while the bottom of the figure shows the datagrams in a single period, represented by normalized sending times $ns(r_0,r_i,u)$.

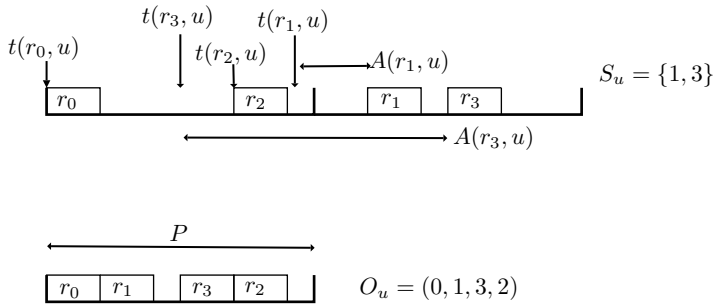

Figure 5.1 – A compact representation of an assignment in which O_u = (0,1,3,2) and $S_u = \{1,3\}$

Remark that for $CR(A)$ to be defined, we need that, on each contention point, at least one datagram is not buffered. We call such an assignment a **canonical assignment**. It turns out that any assignment A can be made canonical without increasing $TR(A)$, hence we can only consider canonical assignments when solving MINSTRA.

Lemma 25. *Let A be a valid assignment, then there is a valid canonical assignment A' such that $A' \preceq A$.*

Proof. Consider a vertex u of contention depth 1, such that for all $r \in \mathcal{R}_u$, $A(r,u) > 0$. Let us define m as the minimum of these values, we define $A'(r,u) = A(r,u) - m$. Assignment A' has no collision on u, since all departure times have been shifted by the same value and A has no collision. Moreover, if v is the vertex after u in a route r, we define $A'(r,v) = A(r,v) + m$. Hence, all departure times for vertices of contention depths larger than one are the same in A and A', which implies that there are no collisions in these vertices. We have proven that A' is still valid. Since all departure times of A' are less or equal to those induced by A, we have $A' \preceq A$. Moreover, if r_0 is the route with $A(r_0,u) = m$, then $A'(r_0,u) = 0$.

We apply this transformation by increasing contention depth. Since, the transformation applied at some contention depth do not change A' for smaller contention depths, a trivial induction proves that A' is valid, canonical and that $A' \preceq A$. □

5.2.2 From a Compact Assignment to its Realization

We now explain how to transform a compact representation into a canonical assignment. Moreover, we show that the obtained assignment is the smallest among all assignments of same representation. We first explain how to do the transformation on a routed network with a single contention point u.

Recall that the datagram of a route r is available at time $t(r,u)$ in the vertex u. Let us consider a compact assignment CA, which maps u to the pair (O_u,S_u). The assignment $Real(CA)$ is built inductively from CA, it is called the realization of CA. If the construction of $Real(CA)$ fails, then $Real(CA)$ is undefined and we say that CA is not realizable. In the next paragraph, we build an assignment A by setting the buffering time of the routes in the order (O_u). If the construction succeeds, we set $Real(CA) = A$

Let say that the order O_u is $(r_0, \ldots, r_l)$. We fix $A(r_0,u)$ to zero, that is the first datagram in the period has no buffering time. Then, in each period beginning by the first datagram, the datagrams will be in order O_u. When the first datagram of the period is chosen, we use it to define normalized arrival times and normalized sending times. Assume that $A(r_i,u)$ have been set for $i \leq l$, let us explain how to set $A(r_{i+1},u)$. If $r_{i+1} \notin S_u$, then $A(r_{i+1},u)$ is chosen so that $ns(r_1,r_{i+1},u)$ is the maximum of $ns(r_1,r_i,u) + \tau$ and $nt(r_1,r_{i+1},u)$. If $ns(r_1,r_{i+1},u) > P - \tau$, then CA is not realizable. If $r_{i+1} \in S_u$, then $A(r_{i+1},u)$ is chosen so that $ns(r_1, r_{i+1},u) = ns(r_1,r_i,u)+\tau$. In both cases, if $ns(r_1, r_{i+1},u) \geq nt(r_1,r_{i+1},u)$, then CA is not realizable (the sending time is in the wrong period with regard to S_u).

Figure 5.2 shows how an assignment $Real(CA)$ is built from a compact assignment CA on a single contention point u. We have $O_u = (2,1,0,3)$ and $S_u = \{1\}$. First, the datagram 2 is fixed, that is, $A(r_2,u) = 0$. Then, since $r_1 \in S_u$, we set $A(r_1,u)$ such that $ns(r_2,r_1,u) = ns(r_2,r_2,u) + \tau$. Finally, since r_0 and $r_3 \notin S_u$, we set $A(r_0,u)$ and $A(r_3,u)$ such that $ns(r_2,r_0,u) = nt(r_2,r_0,u)$ and $ns(r_2,r_3,u) = ns(r_2,r_0,u) + \tau$.

The function $Real$ can easily be generalized to any routed network. Indeed, one can first consider all vertices of contention depth 1, the routes going through them form disjoint sets. Hence, we can define $Real$ independently on each vertex of contention depth 1. Then using the buffering computed for these vertices, one can compute the arrival time of each route in vertices of contention depth 2 and compute $Real$ for these vertices in the exact same way, and so on for all contention depths. In the following lemmas and theorems, we always consider a single contention point, since it is trivial to extend any property for one

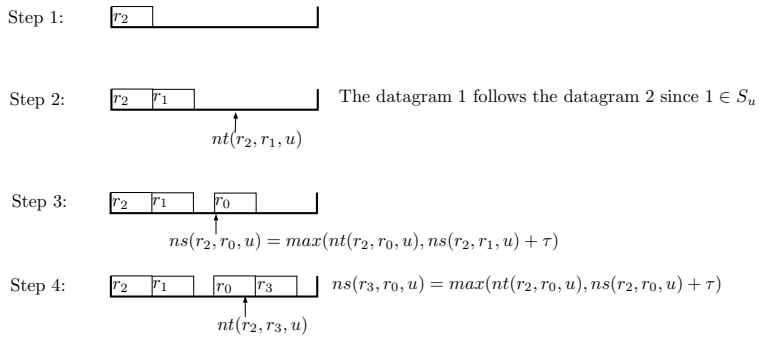

Figure 5.2 – Inductive construction of $Real((2,1,0,3),\{1\})$ from CA on a single contention point u.

contention point to the whole routed network as we just explained.

Lemma 26. *The assignment $Real(CA)$ can be computed in time $O(nd)$, where d is the contention depth of the network. If CA is realizable, then $Real(CA)$ is a valid canonical assignment.*

Proof. In the inductive construction of $Real(CA)$, only a constant number of comparisons and additions are needed to compute the buffer time of a route from the previous one. Hence, the time spent in a vertex u is linear in $|\mathcal{R}_u|$. A route can go through only one vertex of a given contention depth, hence the time spent computing buffers for all vertices of a contention depth is in $O(n)$ and for the whole graph it is in $O(nd)$.

To prove that there is no collision between pair of routes for a given assignment, it is enough to prove it for any interval of time of size P. Hence, it is enough to consider the normalized sending time and to verify they do not induce a collision. By construction, $ns(r_1,r_{i+1},u)$ is always larger than $ns(r_1,r_i,u) + \tau$ and less than $P - \tau$, which proves the absence of collision. Finally, $Real(CA)$ is canonical, since by definition $Real(CA)(r_1,u) = 0$, where r_1 is the first route in O_u. □

We can define the following equivalence relation over canonical assignments: A and B are equivalent if and only if $CR(A) = CR(B)$. We say that a compact assignment $CA = (O_u,S_u)_{u\in V(G)}$ is *canonical* if it is a realizable compact assignment, $CR(Real(CA)) = (O'_u,S'_u)_{u\in V(G)}$ and if for all vertices u, the first routes of O_u and O'_u coincide. This notion of canonicity is defined so that the function CR always sends a canonical assignment on a canonical compact assignment. It is just restrictive enough (by fixing the first element in each order), that the function CR is the inverse of $Real$ over canonical compact assignments. It implies that $Real(CA)$ can be chosen as the representative of the equivalence

class of the assignments having CA as a representation.

In fact, as implied by the following Lemma, we can be more precise on $Real(CA)$: it is minimal for $\prec$ in its equivalence class.

Lemma 27. *Let A be a valid assignment, then $Real(CR(A)) \preceq A$.*

Proof. Given a vertex u and a route $r \in \mathcal{R}_u$, we prove by induction that $Real(CR(A))(r,u) \leq A(r,u)$. Let (O_u,S_u) be the pair associated to u by $CR(A)$, with $O_u = (r_1,\ldots,r_l)$. By definition of CR, r_1 the first route in O_u, is such that $A(r_1,u) = 0$. By definition of $Real$, we have that $Real(CR(A))(r_1,u) = 0 = A(r_1,u)$. Now assume that $Real(CR(A))(r_i,u) \leq A(r_i,u)$ for some i.

First, consider the case $r_{i+1} \notin S_u$. By definition of CR, $ns(r_1,r_{i+1},u)$ must be larger than $ns(r_1,r_i,u) + \tau$ and because $r_{i+1} \notin S_u$ it must also be larger than $rs(r_1,r_{i+1},u)$. Since $Real(CR(A))(r_{i+1},u)$ is the minimum value so that both constraints are true for $Real(CR(A))$, using the induction hypothesis, we have $Real(CR(A))(r_{i+1},u) \leq A(r_{i+1},u)$. The case $r_{i+1} \in S_u$ is similar and left to the reader. □

5.3 Greedy Algorithms

In the next section, we propose several local search algorithms to explore the compact assignments in order to find a compact assignment CA with the smallest possible $TR(Real(CA))$. A realizable compact assignment is needed to initialize these local search algorithms. To find such initial compact assignment, we propose in this section three greedy algorithms which try to build canonical valid assignments, which can be turned into a compact representation by the CR function.

5.3.1 Greedy Deadline

We first present a simple algorithm, which is the natural approach in a context without *periodicity*. The contention points are sorted by contention depth, and the contention depths are dealt with in ascending order. The assignment, on contention points of the same contention depth, is computed independently. The `Greedy Deadline` algorithm consists in selecting among the routes of arrival time less than the current time the one with the longest transmission time. If no route are available at the current time, select the one with the smallest arrival time.

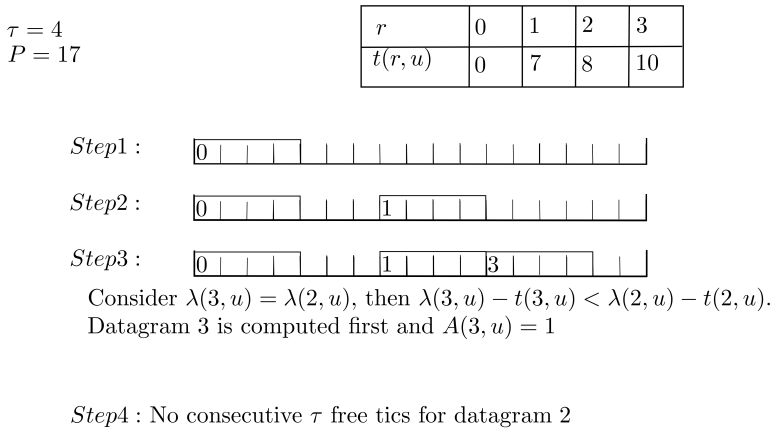

Figure 5.3 – An instance for which `Greedy Deadline` fails to build an assignment

More precisely, `Greedy Deadline` works as follow. For a vertex u, select the route r such that the arrival time $t(r,u)$ is minimal and fix $A(r,u) = 0$. Assume that some datagrams have now been scheduled, the last one on the route r at time $s(r,u)$, we explain here how to schedule the next route. If there are several routes r' for which $t(r',u) < s(r,u) + \tau$, we need to select one of those. For each r' with the previous property, we compute the value $\lambda(r,u) - t(r',u)$ and select the one which minimizes this value. Then, the selected datagram r' is sent with a delay $A(r',u) = t(r,u) + \tau - t(r',u)$. If no route satisfies $t(r',u) < s(r,u) + \tau$, the route with the lowest $t(r,u)$ is sent without delay ($A(r',u) = 0$). Due to the periodicity, once the route r' has been selected and $s(r',u)$ computed, it is possible that there is a collision. If so, $s(r',u)$ is increased to the first time such that there is no collision. If there is no such time, the algorithm fails.

5.3.2 Greedy Normalized

We present here a variant of `Greedy Deadline`: select as first datagram the one with minimal $t(r,u)$, then select the datagrams by lowest normalized arrival times instead of arrival times. Let us call this algorithm `Greedy Normalized`. In practice, it performs better than `Greedy Deadline`.

Both `Greedy Deadline` and `Greedy Normalized` may fail to find a valid assignment for some routed networks, for which there exist a valid assignment. The way we select departure times for the routes can create unused interval of time of size less than τ. These intervals are not usable to schedule datagram of size τ. If too much time is wasted in this way, the algorithms will fail, while there is always a valid assignment when the load is less

or equal to 1. Since each datagram forbids at most $2\tau - 1$ tics in the period to the other datagrams, by a pigeonhole argument, all routes can be scheduled by greedy algorithms considering all departure times, when the load is less than 0.5 (see Chapters 3 for similar arguments).

5.3.3 Greedy Packed

A compact assignment is needed to initialize the local search algorithms presented in the next section. Hence, we propose the `Greedy Packed` algorithm that is guaranteed to find an assignment, even if the transmission time may be worse on average than what is found by the two previous greedy algorithms. The contention points are still managed level by level. For a vertex u, we explain how to build the pair (O_u,S_u). First, the route with the lowest arrival time is selected, say r_0 and we say that 0 is the first element of O_u and $0 \notin S_u$. From now on, r_0 is used to define the normalized arrival times of the other routes. Assume that $(r_0,\ldots,r_i)$, the first i routes of O_u are chosen, let us explain how to choose the $i+1$-th route. If there are routes with a normalized arrival time lower or equal to $ns(r_0,r_i,u)+\tau$, the route r with the smallest value of $\lambda(r,u)-t(r,u)$ is chosen (as in `Greedy Normalized`). If no route satisfy this property, then let r be the route which minimizes $\lambda(r,u)-t(r,u)-nt(r_0,r,u)$, choose it, and $S_u = S_u \cup \{r\}$. In other words, select the route with the smallest transmission time if scheduled without creating gap in the period.

5.3.4 Random generation of routed network

This chapter presents several algorithms that each have several variants or parameters to tune. Thus, each section or subsection describing a new algorithm also provide some experimental results. We describe here how the instances are generated for every experiment of the chapter until Section 5.6 that present more general performance evaluations.

First, remark that, contrary to our choice for star routed networks, we consider that the physical length of the links into the datacenter are the same for all routes. Indeed, several BBUs are often gathered in one or several datacenters. The length of the links between the entrance of the datacenter and all BBUs may or may not be the same. In the first case, once the messages have been scheduled to go in the datacenter, they go out in the exact same order and thus, even if all routes use the same link in the way back, it is not considered as a contention point (see figure 5.4), and the routes are symmetrical

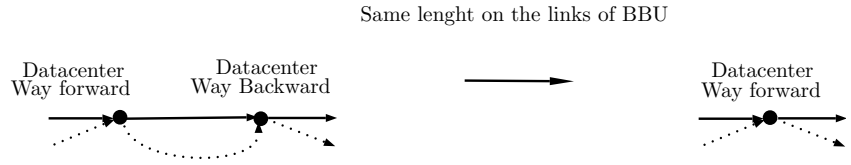

Figure 5.4 – One or several contention points around the BBU according to the length of the link

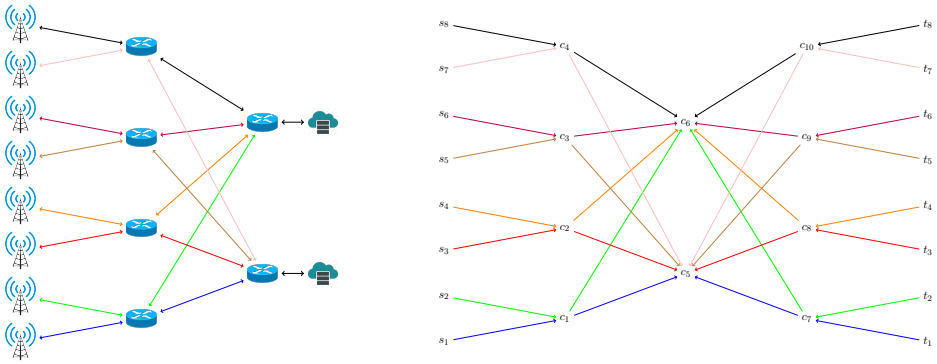

Figure 5.5 – Left, a physical fronthaul network and right, the routed network modeling a round trip in the fronthaul network. Each route is represented by the arcs of the same color.

around the vertex representing the entrance in the BBU. If the length of the links into some datacenter are different, the aggregation node before the datacenter is represented by two contention points. The datagrams may collide in both the entrance and the exit of the datacenter and the routes are symmetrical around the arc between the two contention points.

We propose several experiments to assess the practical performance (in speed and quality) of the proposed algorithms. We present here the instances on which we test our algorithms, which are derived from our application to Cloud-RAN. We consider networks of contention depth three, as illustrated in Figure 5.19, in which each dotted arc represents the arcs of two routes.

To generate random routed networks, several parameters must be chosen: The load of the network, the number of routes, the distribution of the length of the arcs, and the topology of the routed network. We would like to understand the impact of those parameters, in terms of computation time and quality of found assignment, for each of the algorithms studied. In order to reduce the number of experiences presented here, we fix the topology of the routed network to the one shown in Figure 5.19. The difference between the performances of the presented algorithms are not significantly impacted if we

change the number of contention points.

The impact of the **load** on the quality of the results has been investigated: When the load is increased, the relative quality of solutions found by the local search algorithms does not changes significantly. Hence, we choose to fix the load to 0.8, which is an already high load. This means that $P = \frac{\tau \times n}{0.8}$, with n the maximal number of routes over a contention point. The size of the C-RAN traffic depends of the service requirement [8]. Here, we fix $\tau = 2500$ tics.

In a C-RAN context, the number of route is low. In the network we study, there is $n = 8$ routes on the graph. This kind of graphs with few routes allows us to use the Branch and Bound algorithm to find the optimal solution, which helps to interpret the performances of the other algorithms. We study the impact of the number of routes in the graph in Section 5.6. The length of the arcs is drawn uniformly between 0 and P. This choice makes the periodicity of our problem impactful, and does not allow us to reuse algorithms from a non periodic setting.

Since the notion of transmission time has changed in this chapter, compared to Chapter 4, the notion of **margin** is also slightly different. For a given routed network in which r_n is the longest route (i.e. the route for which $\lambda(r)$ is the largest), the margin of an assignment A is equal to $TR(A) - \lambda(r)$, that is, the difference between the transmission time and the physical delay of the route. In other words, this represent the time used for logical delays, that are set by the assignment. When solving MINSTRA, we want to minimize $TR(A)$ down to $TR(N)$, which is equivalent to the minimization of the margin of A. By definition, the margin measures the additional latency given by an assignment and we will express the performance of our algorithms as the value of the margin of the assignment found.

5.3.5 Success Rate and Performance of the Greedy Algorithms

We want to compare the success rate and the performance of the different algorithms presented in this section. First, we consider the impact of the load of the network on the success rate of the three greedy algorithms. We have explained that all greedy algorithms succeed when the load is less than 0.5 and that `Greedy Packed` always succeeds. Figure 5.6 shows the success rate of `Greedy Deadline` and `Greedy Normalized` on 1000 random instances for loads from 0.7 to 1. `Greedy Deadline` fails less than `Greedy Normalized` on highly loaded networks, while `Greedy Normalized` seems more robust on loads between

Success \ Load	0.7	0.8	0.9	1
`Greedy Deadline`	99.5%	92.4%	43.4%	15.7%
`Greedy Normalized`	99.3%	93.2%	51.2%	0%

Figure 5.6 – Success rate of the greedy algorithms for different loads

0.8 and 0.9.

We now want to compare the quality of the solution found by these algorithms. Figure 5.7 shows the margin needed by the assignments given by the algorithms, when there is one. As expected, `Greedy Packed`, that trades margin for success rate, performs worse than `Greedy Deadline` and `Greedy Normalized` when they are able to find an assignment. `Greedy Normalized` performs better than `Greedy Deadline` when it finds an assignment. On vertices with high load, the three algorithms almost always find the same assignment (or fail). On vertices of small load, the constraint of packing the datagram imposed by `Greedy Deadline` worsen the latency.

We propose an improved version of `Greedy Deadline` and `Greedy Normalized` that always find a solution. For each contention point, we first try `Greedy Deadline` (or `Greedy Normalized`), and if the algorithm fails, we apply `Greedy Packed`. Let us call `Hybrid Greedy Deadline` and `Hybrid Greedy Normalized` those two algorithms. Figure 5.8 shows the performances of `Hybrid Greedy Deadline`, `Hybrid Greedy Normalized`, and `Greedy Packed` on 1000 routed networks. Here, the load is of 0.9 to emphasize the difference between algorithms.

Algorithm `Hybrid Greedy Normalized` seems much better than the other two. Hence, in the rest of the paper `Hybrid Greedy Normalized` serve as a baseline of assignment quality since it can be obtained in very short time. It is also used to initialize local search algorithms with a first assignment of sufficient quality.

5.4 Local Search Heuristics

The number of compact assignments CA grows extremely quickly with n. Hence, to find one which minimizes $TR(Real(CA))$, we propose several classical local search algorithm: Hill Climbing, Tabu Search and Simulated Annealing. These methods work as long as a relevant notion of neighborhood of a solution is proposed. The neighborhood relation

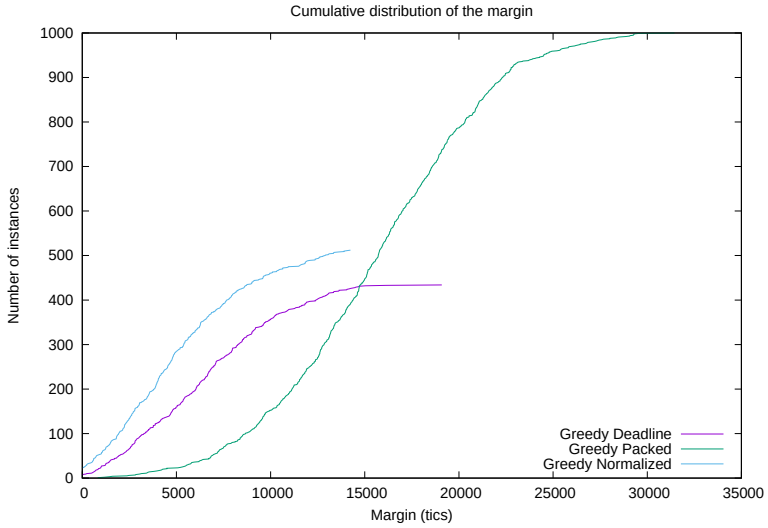

Figure 5.7 – Performance of `Greedy Deadline`, `Greedy Normalized` and `Greedy Packed`. Curves of `Greedy Deadline` and `Greedy Normalized` are incomplete because only the instances for which a solution is found are represented here.

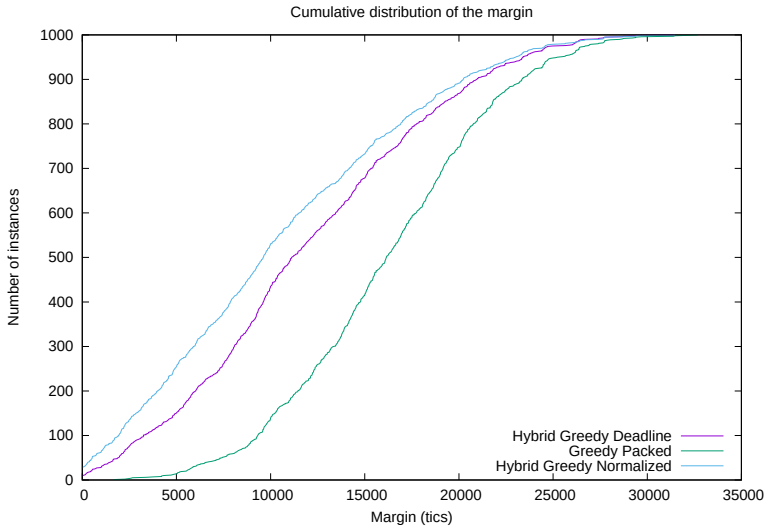

Figure 5.8 – Performance of the updated greedy algorithms that always gives an assignment

must satisfy two properties: it must be quick to compute (hence not too large) and the implicit graph of solutions defined by the neighborhood relation should be connected. We now propose a simple neighborhood relation over compact assignments.

Let u be a contention point of a network, and let CA be a compact assignment for this network, which associates the pair (O_u,S_u) to u. Let $O_u = (r_1,\ldots,r_l)$ and let $\triangle$ denotes the symmetric difference. Let $r_i \in \mathcal{R}_u$, the *r_i-neighborhood* of (O_u,S_u) is the set of pairs (O,S) such that:

1. $O = O_u$ and $S_u = S$ or $S\triangle\{r_i\}$
2. $O = (r_1,\ldots,r_{i-2},r_i,r_{i-1},\ldots,r_l)$ and $S_u = S$ or $S\triangle\{r_i\}$ or $S\triangle\{r_{i-1}\}$ or $S\triangle\{r_i,r_{i-1}\}$

Informally, a compact assignment is in the r-neighborhood of another one if it can be obtained by moving down r once (or not changing it) in the order and adding or removing r and the previous route from the set. Remark that the r-neighborhood of any pair (O_u,S_u) has at most 6 members (it can be 4 when the route r is in first position and cannot be exchanged with the previous one). Figure 5.9 represent the r-neighborhood of a pair (O_u,S_u).

The *r-neighborhood* of a compact assignment CA is the set of all compact assignments $CA' = (O'_u,S'_u)_{u\in V(G)}$, such that (O'_u,S'_u) is in the r-neighborhood of (O_u,S_u). Finally, the *neighborhood* of a compact assignment CA is the union for all $r \in \mathcal{R}$ of the r-neighborhoods of CA.

Let us denote by $k_1,\ldots,k_n$ the number of contention points on the n routes of a routed network. Then, a compact assignment has at most $\sum_{i=1}^{n} 6^{k_i}$ neighbors. Since the networks we consider are of bounded contention depth (2 or 3 in practice), the size of a neighborhood is linear in the number of routes. We further restrict the notion of neighborhood to realizable compact assignments. Indeed, the unrealizable compact assignments do not yield a real assignment, their transmission time is not defined and we cannot use them in our local search algorithms. We call the graph defined by the neighborhood relation over realizable compact assignments of a routed network the **transposition graph** of the routed network. All algorithms presented in this section will do a walk in the transposition graph, trying to find a vertex with optimal transmission time.

Lemma 28. *There is a path from a realizable compact assignment CA, with $CA(u) = (O_u,S_u)$ to CA', such that CA' is equal to CA except on u where it is equal to $(O_u,S_u\cup E)$.*

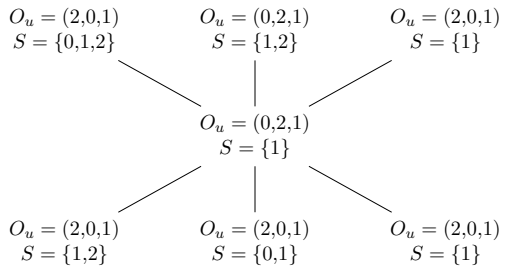

Figure 5.9 – Neighborhood of a pair $O_u = (0,2,1)$, $S_u = \{1\}$ for one contention point.

Proof. The path is by adding elements in E one by one. To prove the existence of the path, it is enough to prove that for $E = \{v\}$. By definition, $(O_u,S_u \cup v)$ is in the neighborhood of (O_u,S_u). However, one should also prove that CA' is realizable. Since the order in which the buffers are fixed by the algorithm of *Real* is the same for (O_u,S_u) and $(O_u,S_u \cup v)$, it is easy to prove by induction that the normalized sending times of $(O_u,S_u \cup v)$ are less than the normalized sending times of (O_u,S_u). Thus, CA realizable implies CA' realizable. Indeed, a compact assignment is realizable if and only if the last normalized sending time is less than $P - \tau$. □

Theorem 29. *The transposition graph of a routed network is connected.*

Proof. We prove the result for a routed network with a single contention node u, it can be generalized to any routed networks by applying the proof contention node by contention node. Let (O_u,S_u) and (O'_u,S'_u) be two realizable compact assignments, we show there is a path between them. Let r be the first element of O_u and let $E = \mathcal{R}_u \setminus \{r\}$. By Lemma 28, there is a path from (O_u,S_u) to (O_u,E). Consider now O''_u, the order O'_u with r placed in first position. There is a path from (O_u,E) to (O''_u,E). Indeed, any order is realizable, when all elements but the first are in E because there are no constraints on their normalized sending time. Now, let r' be the first element of O'_u. By definition, $(O'_u,E\triangle\{r,r'\})$ is in the r' neighborhood of (O''_u,E). Moreover, $(O'_u,E\triangle\{r,r'\})$ is realizable because $E\triangle\{r,r'\}$ is equal to all routes but the first in O'_u. Finally, using Lemma 28 once again prove there is a path between $(O'_u,E\triangle\{r,r'\})$ and (O'_u,S'_u) since (O'_u,S'_u) is realizable and $S'_u \subseteq E\triangle\{r,r'\}$, which proves the theorem. □

5.4.1 Hill Climbing

The most simple local search heuristic is Hill Climbing. This algorithm starts from a compact assignment CA, explores the entire neighborhood of CA, and selects the realizable

compact assignment CA' of minimal transmission time. Then, we set $CA = CA'$ and repeat this step until there are no CA' such that $TR(CA') < TR(CA)$. Then, the algorithm stops and returns $Real(CA)$ which is a local minimum.

The quality of Hill Climbing depends on the the initial compact assignment. A first choice is to consider the compact representation $CR(A)$ of the assignment A given by `Hybrid Greedy Normalized` (`HGN`). We can also choose a random compact assignment. Since a compact assignment does not always give a valid assignment nor a good one, we should draw many random compact assignments and return the best assignment found by Hill Climbing using these initial solutions.

Figure 5.10 shows the difference between initializing Hill Climbing with `Hybrid Greedy Normalized`, or with one or several random compact assignments. Those results are computed from 1000 random instances, with load of 0.8.

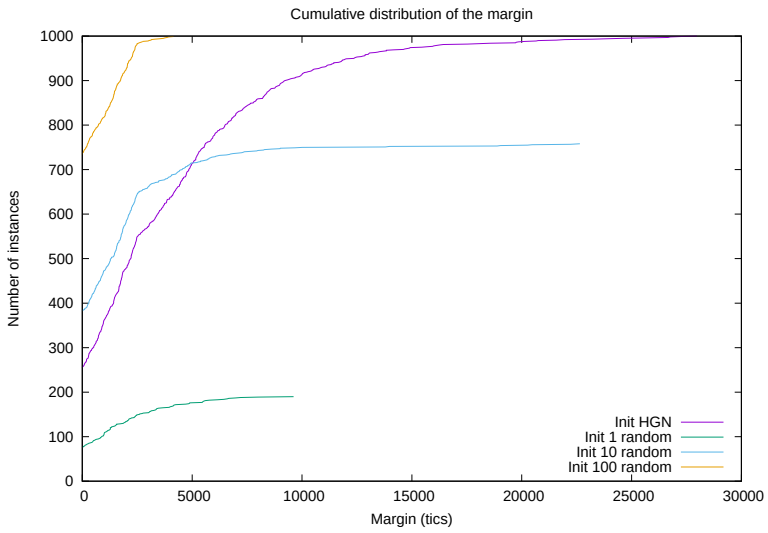

Figure 5.10 – Margin of solutions found by Hill Climbing, initialized with `HGN`, 1, 10 or 100 random compact assignments.

Initializing Hill Climbing with 100 random compact assignments seems to give better results. However, choosing 100 random compact assignments can still fail to produce one valid assignment. We investigate this issue in experiments presented in Figures 5.11 and 5.12. We represent the probability of drawing at least one compact assignment that gives a valid assignment, when drawing 1, 10 or 100 random compact assignments. In Figure 5.11, we fix the number of routes to 8, and we change the load from 0.8 to 1. In Figure 5.12, we fix the load to 0.8 and the number of routes goes from 8 to 12 in the routed network.

Load	0.8	0.9	1
HGN	100%	100%	100%
1 random	19%	12%	4%
10 random	75%	56%	37%
100 random	99%	98%	96%

Figure 5.11 – Success rate of Hill Climbing for several initializations, increasing the load with 8 routes.

#routes	8	10	12
HGN	100%	100%	100%
1 random	19%	6%	2%
10 random	75%	38%	21%
100 random	99%	92%	72%

Figure 5.12 – Success rate of Hill Climbing for several initializations, increasing the number of routes. Load 0.8.

Each value is computed from 1000 random instances.

Those experiences show that Hill Climbing computed on 100 random instances performs well when the number of routes and the load are low. However, this is not sufficient when the load or the number of routes increases. Indeed, higher loads makes valid solutions harder to find, and increasing the number of routes also increase the size of the neighborhood, and thus, the number of compact assignments which are not valid. Furthermore, the computation time required by executing Hill Climbing on many random compact assignments instead of one (using HGN) makes it less effective.

We thus propose an hybrid initialization scheme for Hill Climbing: Between the assignments given by initializing the Hill Climbing either with HGN, or with k random compact assignments, return the one that minimize $TR(A)$. We call this initialization hybrid k. Figure 5.13 shows the margin needed by the best solution given by Hill Climbing, with different initializations. The results are computed on 1000 random instances.

We now focus on how many steps Hill Climbing does before ending in a local optimum. Tabular 5.14 shows the average number of steps done by Hill Climbing, for the initial solution giving the best solution. The results in Table 5.14 are taken from the experiment done to produce Figure 5.13.

	Init HGN	Hybrid 1	Hybrid 10	Hybrid 100
Average number of steps Step	1.16	1.23	1.77	3.52

Figure 5.14 – Average number of steps needed by Hill Climbing to reach a local optimum.

The more steps Hill Climbing does, the more the initial solution is improved. When Hill Climbing starts from the result of `Hybrid Greedy Normalized`, it does not improve much the solution. When drawing a large number of random compact assignments, the probability of drawing one that can be improved a lot is better and it turns out that compact assignments improved many times are often the one with the best margin.

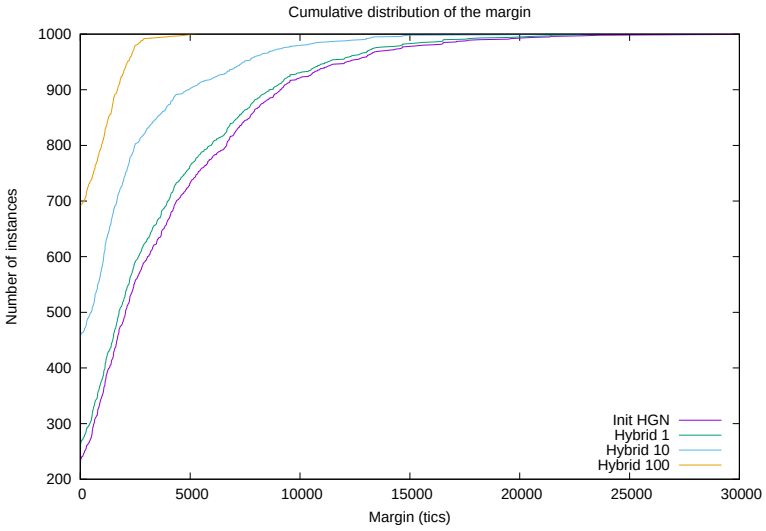

Figure 5.13 – Margin needed to find a solution for Hill Climbing, initialized with `HGN`, hybrid 1, hybrid 10 or hybrid 100. Only the instance for which a solution is found are represented here.

The idea of drawing a large number of random compact assignments to initialize Hill Climbing is a naive version of Simulated Annealing. The next two presented meta-heuristics are designed to explore the compact assignments, even if a local optimum is reached. Tabu Search remembers the explored solutions, in order to avoid them, and Simulated Annealing browses the compact assignments with a stochastic approach.

5.4.2 Tabu Search

Tabu Search is a variation on Hill Climbing using memory. We start from a compact assignment CA given by `Hybrid Greedy Normalized`. Then, at each step, from the current compact assignment CA, we select the compact assignment CA' which minimizes $TR(CA')$, even when $TR(CA') < TR(CA)$ is not satisfied. To avoid looping around a local minimum, we keep in memory the last M solutions explored and we forbid to visit them again. This algorithm can still loop on a solution cycle larger than M, hence we must fix some integer N and stop the algorithm after N steps. The parameters N and M must be chosen appropriately to minimize the computation time of Tabu Search, while maximizing the quality of the solutions found.

We first investigate on Figure 5.15 the impact of N alone. To do so, we fixe $N = M$ (infinite memory) and we compute Tabu Search on an instance, with $N = 100, 500, 1000$

and 2000. Those simulations have been made with 8 routes and a load 0.8, and results are similar for 20 routes. It appears that with $N = M$, the more steps Tabu Search computes, the better is the solution. For most instance, Tabu Search finds the optimal solutions in the first steps, however for some instances the solution is improved after a large number of steps, which impacts strongly the average margin. At each step, Tabu Search explores the entire neighborhood of the current compact assignment, which is of the same size for any compact assignment, hence the computation time is linear in N. For $N > 500$, the computation time may not be worth the improvement of the solution, as shown in Figure 5.15.

N	100	500	1000	2000
Average number of step	6.80	10.75	10.75	29.36
Largest Number of step	89	257	257	1864
Average margin	2318	2297	2297	2295
Computation time (s)	2.0	10.9	27.8	81.7

Figure 5.15 – Average and largest number of step needed by Tabu Search to reach a local optimum and average value of the margin of this local optimum with infinite memory.

We now study the choice of the parameter M. Note that increasing may not necessarily decrease the margin of the solution found by Tabu search. Indeed, a large value for M could restrict the Tabu Search to some component of the transposition graph while a small value of M may allow loops.

We fix $N = 500$ and we compute Tabu Search with M equals 10, 50, 100, 200 or 500. Figure 5.16 shows the average and number of steps and margin and the largest number of steps over all instances needed by Tabu Search to find it's local optimum with different values for M. The values are computed on 100 random instances.

M	10	50	100	200	500
Average number of steps	2.54	4.97	7.42	11.38	11.38
Largest Number of step	26	67	165	165	165
Average margin	3064	2810	2722	2510	2510

Figure 5.16 – Average and largest number of steps needed by Tabu Search to reach a local optimum and average value of the margin of this local optimum.

It seems that the more steps Tabu Search remembers, the smaller is the margin of the assignment. In more of 60% of the cases, Tabu Search finds its best assignment before 10

steps. When increasing the memory, the average number of steps needed by Tabu Search to find the best solution increases. Nevertheless, remark that for $M > 100$, the maximal number of steps to find the best solution is 165. It seems that the memory size has a very small effect on hard instances.

5.4.3 Simulated Annealing

In this section, we study the Simulated Annealing method which works as follow. An initial *t*emperature is set and an initial compact valid assignment CA is computed. At each step, we try to replace the current valid compact assignment CA, by CA' drawn uniformly at random in the neighborhood of CA. Then, in function of the temperature t and Δ the difference between $TR(Real(CA))$ and $TR(Real(CA'))$, CA' is either accepted or rejected. More precisely, CA' is accepted with probability $e^{-\frac{\Delta}{t}}$. After a given number of steps, the temperature is decreased by multiplying it by some constant less than one. The lower the temperature, the lower the chance to accept a compact assignment that worsen the solution. This algorithm is an answer to the exploration/exploitation paradox: in the beginning of the algorithm the whole solution space is explored but as the temperature decreases, the search becomes more and more local around a good solution.

When using Simulated Annealing, we need to fix the following parameters: initial solution, initial temperature, number of steps before decreasing the temperature, factor by which the temperature is decreased, number of steps without improvement before ending the process. In order to fix the initial temperature t_0, we follow [51]:

1. Initiate 100 disturbances at random; evaluate the average $\bar{\Delta}$ of the corresponding variations Δ
2. Choose an initial rate of acceptance τ_0 of the "degrading perturbations" according to the assumed "quality" of the initial configuration; for example:
   - "poor" quality: $\tau_0 = 50\%$ (starting at high temperature)
   - "good" quality: $\tau_0 = 20\%$ (starting at low temperature)
3. Deduce t_0 from the relation: $e^{-\frac{\bar{\Delta}}{t_0}} = \tau_0$

Tabular 5.17 shows the average margin of the solutions produced by Simulated Annealing when initialized with temperatures computed from the previous routine and the computation time. The initial solution used by Simulated Annealing is the solution given

by Hill Climbing initialized by `Hybrid Greedy Normalized`. The experiment is made on 100 random instances. We experimentally observed that increasing the initial temperature does not significantly improve the quality of the solution, but does increase the computation time. Hence, we assume from now on that the initial solution given by Hill Climbing can be considered as "good" to fix the initial temperature.

Quality of initial configuration	Good	Poor
t_0	1788	4153
Average margin	4212	4217
Computation time (ms)	2817	4035

Figure 5.17 – Comparison of two initial temperatures, considering the quality of the initial configuration

In Simulated Annealing, the temperature should decrease slowly. At each **level**, N compact assignments are drawn. At the end of a level, the temperature is decreased by 1% and the algorithm stops if less than 1% of the compact assignments drawn are accepted during two consecutive steps. Hence, drawing too few compact assignments in a level decreases the temperature too fast and reduces the efficiency of Simulated Annealing. However, drawing too much compact assignments during a level increases the computation time of the algorithm, there is tradeoff between time an quality and length of a level should be set carefully.

When N is low ($N = 10$ or $N = 20$), the probability of drawing no compact assignment that will be accepted is high and Simulated annealing stops too fast. To fix this issue, we force Simulated Annealing to continue during 10 consecutive levels for which less than 1% of the compact assignment are accepted. This increases the computation time for higher values for N, even though Simulated Annealing does not exhibit the problem for these values. Since the neighborhood of a solution is composed of a large number of solutions of the same value, the acceptance rate is greater than 1% even under low temperatures. Hence, we set a minimal temperature under which Simulated Annealing stops.

Figure 5.18 shows the margin needed by Simulated Annealing with different values of N. Those results are computed on 1,000 random instances, in which the initial temperature is set by the routine of [51] presented before and the load is 0.8

N	10	20	50	100	200	500	1000
Average Margin	656	378	270	257	255	249	249
Computation time (s)	0.11	0.33	1.3	2.8	5.9	15.2	30.1

Figure 5.18 – Average Margin of best solution found by Simulated annealing, with a different number of compact assignments drawn at each level.

As expected, the computation time is roughly linear in the number of steps N. The higher N is, the better is the average margin of the best solution found. It appears that drawing more than 100 compact assignments at each level does not significantly improve the solution related to the additional computation time.

5.5 Branch and Bound

5.5.1 Brute-forcing Compact Assignments

Solving MINSTRA, means finding an assignment for which $TR(A)$ is minimal. Given an instance of MINSTRA, the local search algorithms presented in the previous sections explore a few compact assignments CA and return one which minimize $TR(Real(CA))$. We begin by providing a bruteforce algorithm testing all compact assignments, then we show how a large number of compact assignments can be avoided using a Branch and Bound algorithm, which allows to solve MINSTRA optimally in practice for small number of routes.

Theorem 30. *For routed networks of fixed contention depth d, the problem* MINSTRA *parametrized by n the number of routes is FPT: it can be solved in time $O(nd(n!2^n)^d)$.*

Proof. The algorithm to solve MINSTRA is the following: all compact assignments CA are generated, for each of them $TR(Real(CA))$ is computed in time $O(nd)$ by Lemma 26 and we keep the compact assignment for which this value is minimal. Because of Lemma 27, to compute the minimum of $TR(A)$ over all assignments A, it is enough to compute the minimum of $TR(Real(CA))$ over all compact assignment CA.

Let us evaluate the number of compact assignments. On a single contention point c with $s = |\mathcal{R}_c|$ routes going through, there are $s!2^s$ possible restrictions of a compact assignment by counting the number of pairs of set and order over $\mathcal{R}_c$. On a given contention depth consisting in the vertices $\{c_1, \ldots, c_l\}$, with $s_i = |\mathcal{R}_{c_i}|$, there are $\prod_{1 \leq i \leq l} s_i!2^{s_i}$ compact assignments. On a given contention depth, all routes use at most 1 vertex, hence $\sum_{1 \leq i \leq l} s_i \leq n$. Since $\prod_{1 \leq i \leq l} s_i! \leq (\sum_{1 \leq i \leq l} s_i)!$, we have $\prod_{1 \leq i \leq l} s_i!2^{s_i} \leq n!2^n$. Since the

contention depth is d, we have at most $(n!2^n)^d$ compact assignments which proves the theorem. □

Note that for the vertices of the largest contention depth, compact assignments can be considered independently, since they do not interact. Let $\{u_1,\ldots,u_l\}$ be the vertices of maximal contention depth, and let $s_1,\ldots,s_l$ be their width, then we need only to consider $(n!2^n)^{d-1}(\sum_{1\leq i\leq l} s_i!2^{s_i})$ compact assignments. This makes a large difference in our target application and the experiments presented in this chapter, since in this context d equals three and the s_i's are pretty balanced.

5.5.2 Compact Assignment Tree

From now on, we denote the contention points by $\mathcal{C} = \{c_1,\ldots,c_m\}$, and we assume they are indexed by contention depth, that is if $i < j \leq m$ then $cd(c_i) \leq cd(c_j)$.

A **Partial Compact Assignment** CA is a compact assignment defined on a subset $\mathcal{C}_i = \{c_1,\ldots,c_i\}$ of $\mathcal{C}$. For a compact assignment CA, we denote by CA_i the restriction of CA to $\mathcal{C}_i$. Let CA be a partial compact assignment defined on $\mathcal{C}_{i-1}$, then $CA[(O,S)]$ is an extension of CA to $\mathcal{C}_i$, defined by $CA[(O,S)](c_i) = (O,S)$. In this section, we build a compact assignment CA by extending partial compact assignments incrementally from c_1 to c_m.

The bruteforce algorithm of Theorem 30 can be seen as going through a tree of partial compact assignments, whose leaves are the compact assignments. We call this tree the **compact assignment tree**. Each vertex v is labeled by a couple $l(v) = (O,S)$ except the root. There is a bijection between a vertex of the tree and a partial compact assignment, such that a vertex at depth i is a partial assignment defined over $\mathcal{C}_i$. We define this bijection recursively: let v be a vertex and u its parent, then if u is mapped to CA, and v is at depth i, then v is mapped to $CA[l(v)]$.

5.5.3 The Branch and Bound Algorithm

We now explain how to cut the compact representation tree while exploring it by two different means: the computation of a lower bound of the transmission time over a subtree, to cut the whole subtree and several simple rules which eliminate solutions which are dominated by others or which do not yield a valid assignment.

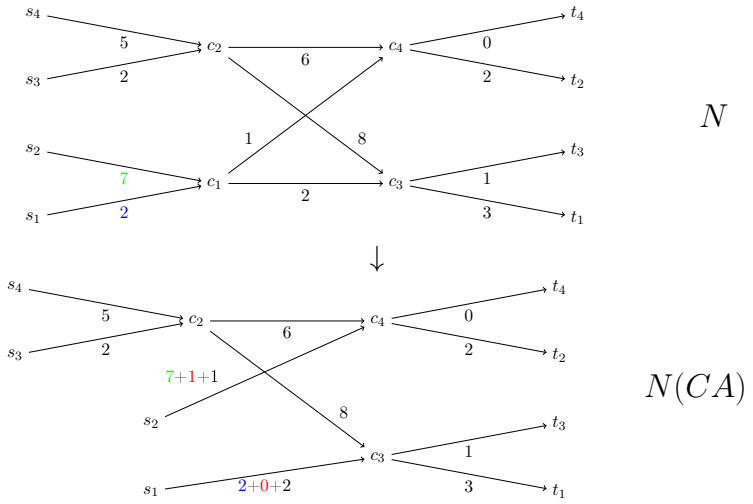

Figure 5.19 – A restricted routed network $N(CA)$ obtained from N and the partial compact assignment CA, defined over $\{c_1\}$, whith $A(1,c_1) = 0$ and $A(2,c_1) = 1$.

Lower Bounding the Transmission Time Once a partial compact assignment is set, one can discard the nodes of the routed network on which it is defined and work with a simpler routed network, called a **Restricted Routed Network**. Let N be a routed network and CA a partial compact assignment defined over $\{c_1\}$, we define the restricted routed network $N(CA)$ by defining a new set of routes $\mathcal{R}'$ and a new weight function ω' as follows.

A route $r \in \mathcal{R}$ going through c_1, that is equal to $(s,c_1,c_j,\ldots,t)$, is replaced by the route $(s,c_j,\ldots,t)$ in $\mathcal{R}'$ while the routes not going through c_1 stay the same in $\mathcal{R}'$. We define $\omega'(r,s) = \omega(r,s)+\omega(r,c_1)+A(r,c_1)$ where $A(r,c_1)$ is computed from the function $Real(CA_1)$ over the contention point c_1, as explained in Section 5.2.2. To define a restricted routed network obtained from a partial compact assignment defined over $\mathcal{C}_i$ and a network N, we define recursively $N_j = N_{j-1}(CA \restriction_{\{c_j\}})$, with $N_0 = N$. Remark that $CA \restriction_{\{c_j\}}$ is the restriction of CA to the singleton $\{c_j\}$ which is the first contention point of N_{j-1}, which makes $N_{j-1}(CA \restriction_{\{c_j\}})$ well defined.

The restricted routed network represents the problem which is left to solve when fixing a partial compact assignment.

Lemma 31. *Let N be a routed network, CA a partial compact assignment of N, and $\mathcal{CA}$ the set of compact assignments which are extensions of CA. Then, $TR(N(CA)) = \min_{A\in\mathcal{CA}} TR(Real(A))$.*

Proof. Let $\mathcal{C}_i$ be the domain of CA, by definition of $\mathcal{CA}$, for all $CA' \in \mathcal{CA}$, $CA'_i = CA$.

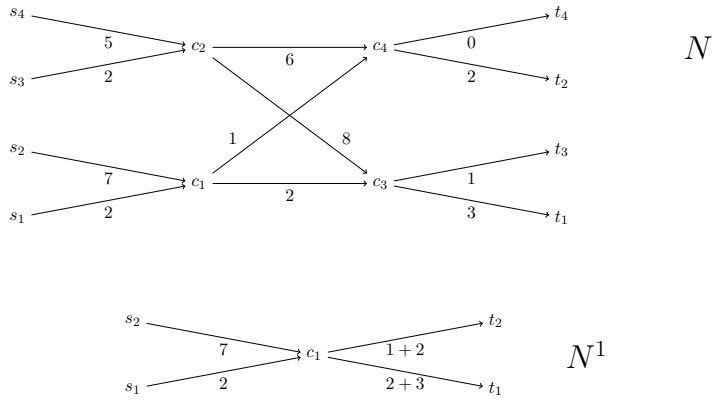

Figure 5.20 – Problem MINSTRA relaxed to one contention point.

Let us define the function Ext from the compact assignments of $N(CA)$ to $\mathcal{CA}$. The assignment $Ext(CA')$ is defined as equal to CA over $\mathcal{C}_i$ and equal to CA' over the other contention points. The function Ext is a bijection and $TR(CA') = TR(Ext(CA'))$, by construction of the restricted routed network $N(CA)$, which proves the lemma. □

Let v be a vertex of the compact representation tree representing the partial compact assignment CA. If we want to ignore the subtree rooted at v while exploring the partial compact assignment tree, we must know a lower bound on the transmission time of the compact assignments in this subtree. Lemma 31 shows that it is given by the transmission time of the restricted routed network $N(CA)$, that is solving MINSTRA on a simpler network. Since this value is still too expensive to compute, we provide a relaxation of the problem of solving MINSTRA over $N(CA)$, which is practical to solve.

Relaxation of MINSTRA To lower bound the value of $TR(N)$, we propose to transform a routed network into a network with a single contention point, with a lower transmission time (but as large as possible). The problem MINSTRA over a single contention point is the problem WTA, that we have solved in Chapter 4, and that we can compute efficiently.

Let N be any network and let c_j be a contention point. Let N^j be the routed network with the single contention point c_j, a set of routes $\mathcal{R}'$ and a weight function ω' defined as follows. For each route r of N which goes through c_j, there is a route $r' = (s_r,c_j,t_r) \in \mathcal{R}'$. We define $\omega'(s_r,c_j) = \lambda(r,c_j)$ and $\omega'(c_j,t_r) = \lambda(r) - \lambda(r,c_j)$. The network N^j represents all the routes going through c_j and forget all constraint before and after c_j. Figure 5.20 shows a routed network N transformed into N^1.

$A(i,c_j)$

i	c_1	c_2	c_3	c_4
1	0	-	0	-
2	1	-	-	0
3	-	0	0	-
4	-	2	-	1

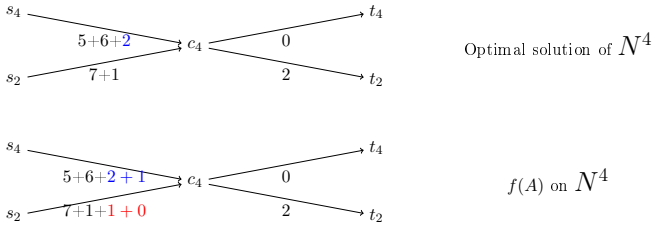

Figure 5.21 – A network N^4, obtained from N of Figure 5.20, and the optimal assignment A of N. On the first representation of N^4, $f(A)$ the image of A on N^4 and on the second representation of N^4, an optimal assignment.

Lemma 32. *Let N be a routed network and c_j one of its contention point, then $TR(N) \geq TR(N^j)$.*

Proof. We associate to any valid assignment A of N, the assignment $f(A)$ of N^j which is defined as $f(A)(r,c_j) = s(r,c_j) - \lambda(r,c_j)$, that is the sum of waiting times in c_j and in contention nodes before c_j. By construction of N, the transmission time of a route r using A in N is larger than the transmission time of the route r using $f(A)$ in N^j, since all the waiting times are the same up to c_j and they are 0 for the contention nodes following c_j. Hence, we have for all assignments A of N, $TR(A) \geq TR(f(A))$ which proves the lemma. □

Figure 5.21 illustrates Lemma 32. From the routed network N of Figure 5.20, we build N^4 and its optimal assignment A', defined as $A'(4,c_4) = 2$, which implies that $TR(N^4) = 13$. We also give the assignment A in the table of Figure 5.21, which is the optimal solution for N. We have $TR(f(A)) = 14$, which is indeed higher than the optimal solution of N^4. Remark that, on N^3, $f(A)$ is optimal.

Let $SCB(N)$, for *Single Contention Bound*, be the function which associates to N the integer $\max_j TR(N^j)$. By Lemma 32, we have for all j, $TR(N) \geq TR(N^j)$, hence $SCB(N) \leq TR(N)$. We can compute $SCB(N)$ in time $m2^k k^3 log(k)$ where k is the contention width of the network and m the number of contention nodes. Indeed, for each j, solving MINSTRA on N^j is equivalent to solving WTA, which can be done in time $2^k k^3 log(k)$

by ASPMLS, where k is the number of routes in N^j, which is equal to the contention width of c_j in N. In the example of Figures 5.20 and 5.21, $SCB(N) = 13$ while $TR(N) = 14$.

Description of the Branch and Bound Algorithm We now describe precisely how the proposed branch and bound algorithm works. It starts by computing a solution using Simulated Annealing, as described in Section 5.4.3, to initialize an upper bound on the transmission time. Then, it does a depth first traversal of the compact assignment tree. When it enters a node v, which represents a partial compact assignment CA, it computes $SCB(N(CA))$ and if it is larger or equal to the upper bound, it backtracks, that is it goes back to the parent of v without visiting the subtree rooted at v. When the algorithm reaches a leaf representing some compact assignment CA, it computes $TR(Real(CA))$ and updates the upper bound. Lemmas 31 and 32 proves that the leaves which have not been explored correspond to compact assignments with higher transmission time than the one found by the branch and bound algorithm, proving that it solves MINSTRA.

Additional Cuts Even with the cuts due to the evaluation of $SCB(N(CA))$, the compact assignment tree is still too long to traverse. We propose here several additional cuts that improve the computation time of the algorithm.

Assume that the branch and bound algorithm reach vertex v in the compact assignment tree, representing the partial compact assignment CA over $\mathcal{C}_i$. Let u be a child of v, it represents $CA[(O,S)]$, some extension of CA to $\mathcal{C}_{i+1}$. Then, (O,S) is a compact assignment for c_{i+1} in $N(CA)$ and we would like it to be *valid*, *canonical* and *minimal* for $\prec$. Indeed, if (O,S) is not valid for c_{i+1}, then none of the extensions of $CA[(O,S)]$ will be valid and we can discard the subtree rooted at u. If (O,S) is not a canonical compact assignment, then by Lemma 27, there is a a compact assignment (O',S') which is smaller for $\prec$. In the same way, if (O,S) is not minimal for $\prec$, then there is a compact assignment (O',S') which is smaller. In both cases, the transmission time of the extensions of $CA[(O,S)]$ will always be larger than the transmission time of the extensions of $CA[(O',S')]$ and again we can discard the subtree rooted at u.

The cut consisting in verifying whether (O,S) is a compact assignment for c_{i+1} in $N(CA)$ is simple to implement in linear time by computing $Real((O,S))$.

To guarantee that we only consider canonical compact assignments, we must guarantee that the buffering of the first route is zero. It is the same as requiring for a compact assignment (O,S), that r_{O_1}, the first route in O, is not in S. Hence, the cut consists in

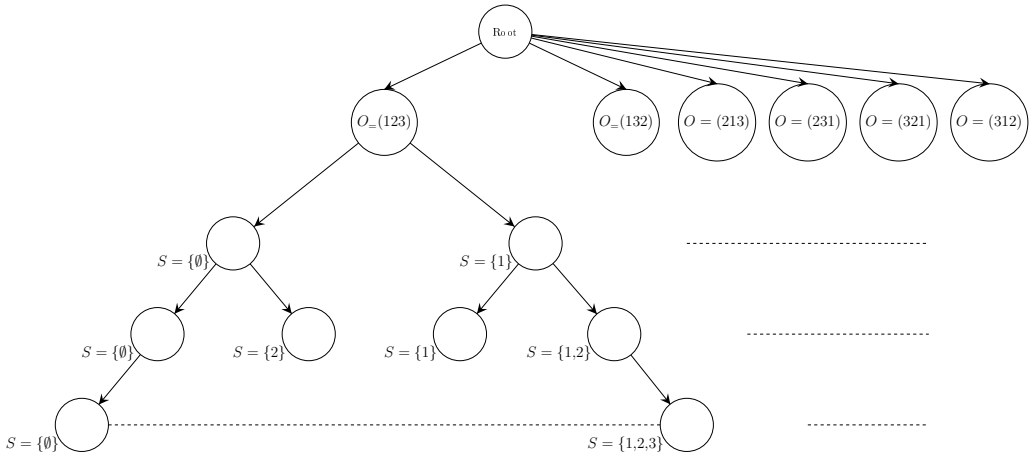

Figure 5.22 – Expansion of a vertex of the compact assignment tree, corresponding to a contention vertex of width 3.

discarding assignments with $r_{O_1} \in S$. This allows to compute normalized sending times for all routes, with r_{O_1} as a reference.

We would like to cut any subtree rooted at a non minimal assignment, but we are not yet able to decide whether a compact assignment is minimal in polynomial time. Hence, we propose several easy to compute heuristics to detect when a compact assignment is not minimal. The first one is to consider the assignment (O,S) of $N(CA)$ and for each route $r_i \in S$, we consider $Real((O,S))$ where r_i has been removed. Then, we try to add back r_i, with $r_i \notin S$ but no constraint on its position in the order. If we manage to do so, we have found a compact assignment $(O',S \setminus \{r_i\}) \prec (O,S)$. Indeed, given a fixed first element in the order, routes in S have their sending time larger than when they are not in S, regardless of the order.

For the next cuts, we need to consider that the set S is built incrementally as shown in Figure 5.22. We expand a vertex of the compact assignment tree, so that all orders on routes of the contention node are children of the node, then a complete binary tree for each of these orders represents all possible subsets of routes.

- Let us write O_i for the ith element of the order i. Assume we are traversing the expansion of a vertex corresponding to the contention vertex c. In the binary tree, a vertex has two children of depth i representing the fact that $r_{O_i} \notin S$ and $r_{O_i} \in S$. We build the partial assignment over $r_{O_1}, \ldots, r_{O_i}$ and its realization for $r_{O_i} \notin S$. If $ns(r_{O_1}, r_{O_{i-1}}, c) + \tau = ns(r_{O_1}, r_{O_i}, c)$, then the two datagrams of the routes $r_{O_{i-1}}$ and r_{O_i} follow one another without gap in the period.

Hence, if we fix $r_{O_i} \in S$, it will not change $ns(r_{O_1},r_{O_i},c)$ but $s(r_{O_i},c)$ is increased by P tics. Hence, any extension of this compact assignment are dominated by the extension of the same compact assignment with $r_{O_i} \notin S$ Thus, we cut the branch $r_{O_i} \in S$ as in Figure 5.23.

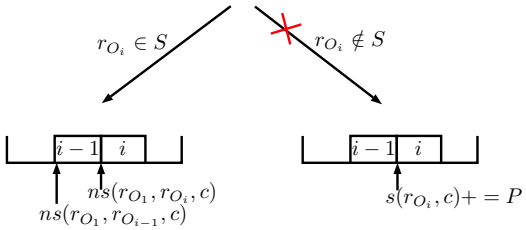

Figure 5.23 – When $ns(r_{O_1},r_{O_{i-1}},c)+\tau = ns(r_{O_1},r_{O_i},c)$, every extension in the branch with $r_{O_i} \in S$ is dominated by the corresponding extension in the branch $r_{O_i} \notin S$.

- When considering a route r_{O_i} (with $r_{O_i} \notin S$ or $r_{O_i} \in S$), we consider the realization of the partial assignments over the elements $r_{O_1},\ldots,r_{O_{i-1}}$. If it is possible to fix the normalized sending time of r_{O_i} before the normalized sending time of $r_{O_{i-1}}$, without collision, then there is a compact assignment $(O',S) \prec (O,S)$ (with O_i at a new, smaller position in O', see Figure 5.24). It corresponds to a compact assignment, with a gap in the period which can be used by some route which should be placed after the gap because of the order. This cut can be computed in linear time, by comparison of the normalized sending times.

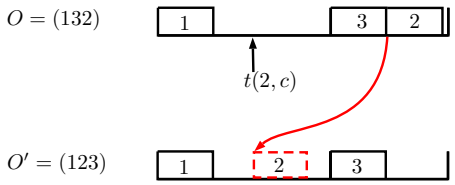

Figure 5.24 – An example of two orders O and O' for which $(O',S) \prec (O,S)$, with $S = \emptyset$.

- To ensure that we go through only canonical assignment, we force the first route r_{O_1} to have zero buffering time, by setting $r_{O_1} \notin S$. But there can be several datagrams with zero buffering time and we refine the notion of canonicity by requiring that the one of smallest id is the first in the order as shown in Figure 5.25. Hence, when we

consider O_i with $O_i \notin S$ and that in $Real((O,S))$, O_i as no buffering, then $O_i > O_1$ otherwise we cut the subtree from $O_i \notin S$.

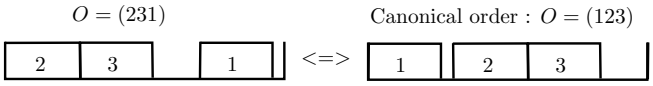

Figure 5.25 – An example of order in which O_1 and O_3 have no buffering and in which $O_1 = 2 > O_3 = 1$. The assignment with O is thus non-canonical.

5.5.4 Scalability of Branch and Bound Algorithm

The complexity of the Branch and Bound algorithm depends on both the contention depth of the routed network and the number of routes. We focus on networks of contention depth 3, which is realistic in our C-RAN context, and we want to investigate how much the computation time increases in practice when the number of routes grows. Note that the running time of the algorithm is also very sensitive to the width of the network: for the same number of routes, if the routes are well spread out over the contention nodes, there are less compact assignment than if they are concentrated on some contention vertices. In previous experiments, the maximal width is 4 in the vertices of depth 2, as shown in Figure 5.19. We generalize the topology of Figure 5.19 by considering any number of contention vertex at depth 1 instead of 4. Each of theses vertex is of width 2 and the 2 routes going through a vertex of depth 1 are distinct from the other routes and goes to the two different vertices of depth 2.

Tabular 5.26 shows the average computation time of Branch and Bound on 8, 10, and 12 routes. The results are computed on 100 instances in which the length of the arcs have been drawn in $[P]$ and the load is 0.8.

Number of routes	8	10	12
Average computation time	2.2s	22s	47m

Figure 5.26 – Average computation time of Branch and Bound with different number of routes.

When the number of routes is lower than 10, we can find a solution in a reasonable time. For 12 routes, the time needed to compute a solution is in the range of one hour. By increasing the quality of the cuts or even by generating only minimal compact assignments, with a perfectly optimized implementation we could solve problems with 14 routes, at most

16. Beyond this value, the sheer number of compact assignments that Branch and Bound must traverse is too large to compute a solution.

5.6 Experimental Evaluation

In this section, we compare the performance of all algorithms presented in this chapter.

We use the settings described in Section 5.3.4: There are 8 routes in the routed network, the length of the arcs are drawn uniformly in $[P]$, and the load is 0.8. The algorithm compared here are:

- `Hybrid Greedy Normalized`.
- Hill climbing initialized by `HGN`.
- Hybrid Hill climbing, initialized by 100 random compact assignments and `HGN`.
- Tabu Search, with infinite memory and 500 steps.
- Simulated Annealing using 100 steps by level.
- Branch and Bound

Figure 5.27 shows the cumulative distribution of the margin of the solution found by each algorithm. Tabular 5.28 shows the average computation time and the average margin of the algorithms computed over 1000 instances.

Algorithm	HGN	Hill Climbing	Hybrid HC	Tabu Search	Simulated Annealing	Branch and Bound
Margin (tics)	5968	3666	372	2240	294	284
Computation time (s)	0.005	0.035	1.39	10.76	2.28	0.22

Figure 5.28 – Average margin and average computation time of each algorithm for 8 routes, length of arcs drawn in $[P]$.

First, remark that `Hybrid Greedy Normalized` is far from finding an optimal solution, even when it is followed by a Hill Climbing algorithm, but then the improvement is important. However, computing Hybrid Hill Climbing from 100 random compact assignments yields solutions which are very close to the optimal. In these simple settings, it seems to be enough to draw 100 random solutions to cover efficiently the whole compact assignment space. As expected, Tabu Search is better than Hill Climbing computed from the solution

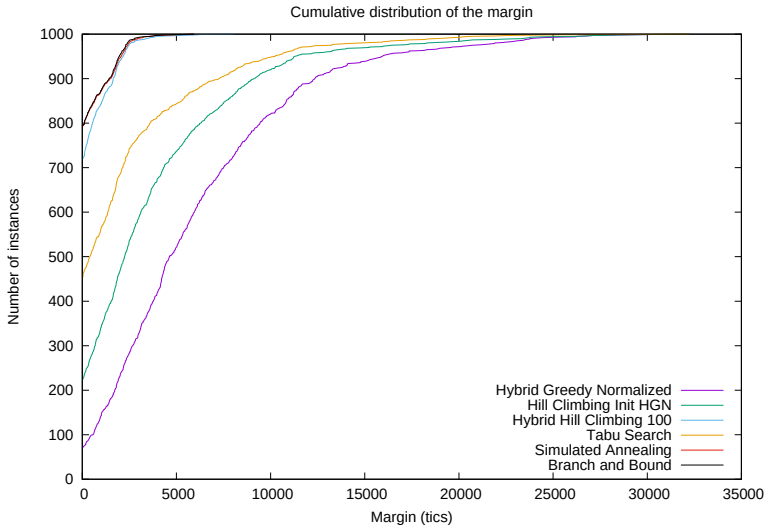

Figure 5.27 – Cumulative distribution of the margin for 8 routes, length of arcs drawn in $[P]$.

of `Hybrid Greedy Normalized`, but since it is extremely slow to compute a solution, it is not possible to use it on 100 different random compact assignments. Simulated Annealing is able to compute a good solution, very close to the optimal in the fifth of the time needed by Tabu Search. Branch and Bound is far better than the other algorithms: its computation time, in this network with few routes and a small contention width, is 10 times smaller than Simulated Annealing, while it finds the global optimum.

We now want to compare the performance of those algorithms when the routes are drawn in a small range of values. Thus, we set the length of the arcs to be drawn in $[0.9P,P]$. We do the same experiment as previously, but from now on, simulated annealing computes 1000 compact assignment at each level. As we observed in section 5.4.3, increasing the number of compact assignment considered at each level increases the quality of the solution. While drawing 100 compact assignments was enough for the simple routed network of the previous experiment, drawing 1000 random compact assignments at each level in the two following experiments improves dramatically the performances of Simulated Annealing, while still requiring a lower computation time than tabu search.

Figure 5.29 shows the cumulative distribution of the margin of the algorithms, while Tabular 5.30 shows the average margin and the computation time of each algorithms on 1000 random instances.

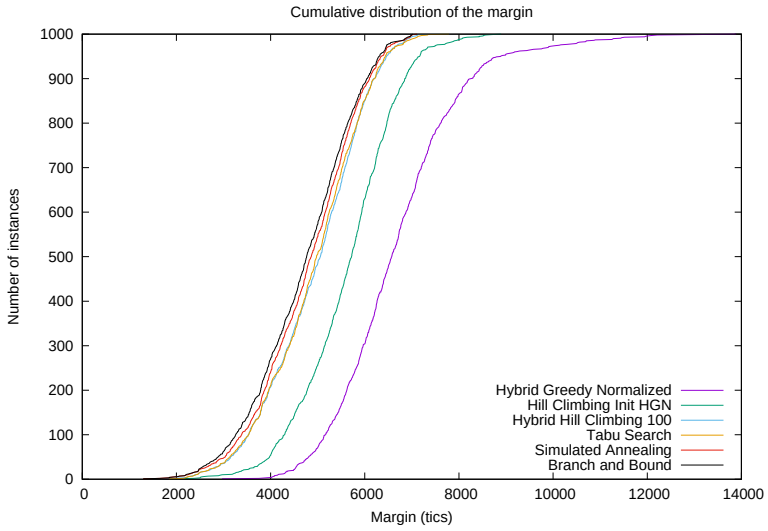

Figure 5.29 – Cumulative distribution of the margin for 8 routes, length of arcs drawn in $[0.9P,P]$.

Algorithm	HGN	Hill Climbing	Hybrid Hill Climbing	Tabu Search	Simulated Annealing	Branch and Bound
Margin (tics)	6700	5636	4933	4914	4789	4703
Computation time (s)	0.004	0.048	0.965	10.71	6.60	0.104

Figure 5.30 – Average margin and average computation time of each algorithm for 8 routes drawn in $[0.9.P,P]$.

We observe that the relative performance of the algorithm does not change. Instances where the length of the arcs are drawn in the same range of value needs more margin to be solved. We made the same observation in Chapters 3 3 for different networks and constraints on the assignments.

The following experiment shows the performance of the algorithms when increasing the number of routes. In the instance generated, there are 24 routes. To do so, we replace each route of the routed network of Figure 5.19 by three distinct routes.

This number of routes is too large to use Branch and Bound. Thus, we also represent SCB in the graph, the lower bound on $TR(N)$ introduced to design Branch and Bound algorithm. Figure 5.31 and Table 5.32 show respectively the cumulative distribution of the margin of the algorithms and the average margin and computation times for the same experiment. The results are computed on 1000 instance, in which the length of the arcs are drawn in $[P]$.

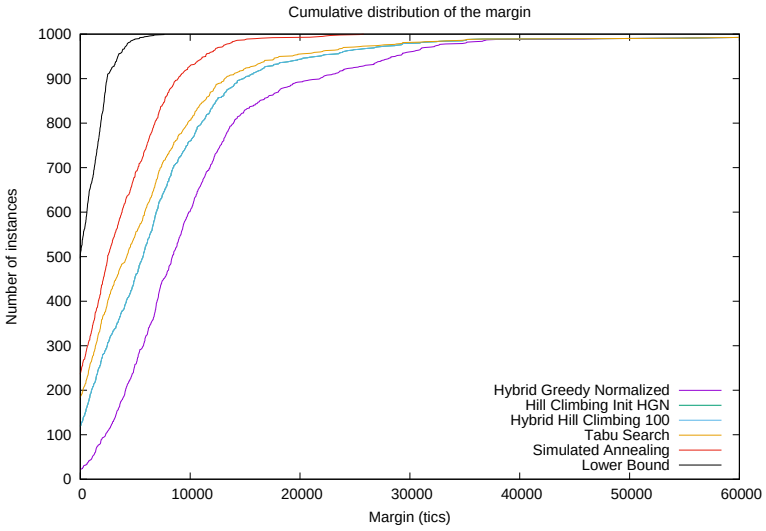

Figure 5.31 – Cumulative distribution of the margin for 24 routes, length of the arcs drawn in $[P]$.

Algorithm	HGN	Hill Climbing	Hybrid Hill Climbing	Tabu Search	Simulated Annealing
Margin (tics)	10572	7402	7402	6311	3725
Computation time (s)	0.014	0.425	3.89	55.67	46.32

Figure 5.32 – Average margin and average computation time of each algorithm for 24 routes, length of the arcs drawn in $[P]$.

First, remark that Hill Climbing has the same performance when it is initialized with 100 random compact assignments and the solution given by `HGN` or only the solution given by `HGN`. As explained in Section 5.4.1, the chances to draw a realizable compact assignment is low when the number of routes is large, and we should draw much more random compact assignment to find one which is realizable. It is not reasonable to do so, since we already have Simulated Annealing, which does a random search in the space of all compact assignments in a much smarter way. While the average margin is higher for 24 routes, than for 8 routes, Simulated Annealing is still twice better than Tabu Search, its closest competitor, while requiring less computation time.

5.6.1 Performance against Statistical Multiplexing

We now compare the performance of our best algorithm with the current way to manage networks: Statistical Multiplexing. We use the simulator presented in Section 4.1.7. As

a reminder, we propose two policies to deal with buffer in statistical multiplexing. The first one `FIFO`, sends the messages in a buffer following the First In First Out policy. The second one, `CriticalDeadline` computes the transmission time of the message in the buffer, if we assume they will not be buffered anymore, and send the message with the largest transmission time first.

We first compare the performances of our best algorithm, Branch and Bound, to statistical multiplexing. Figure 5.33 shows the cumulative distribution of the margin of the solutions produced by Branch and Bound, and statistical multiplexing using `FIFO` or `CriticalDeadline` policy. The experiment has been made on 1000 instances of load 0.8, with the length of the arcs drawn in $[P]$.

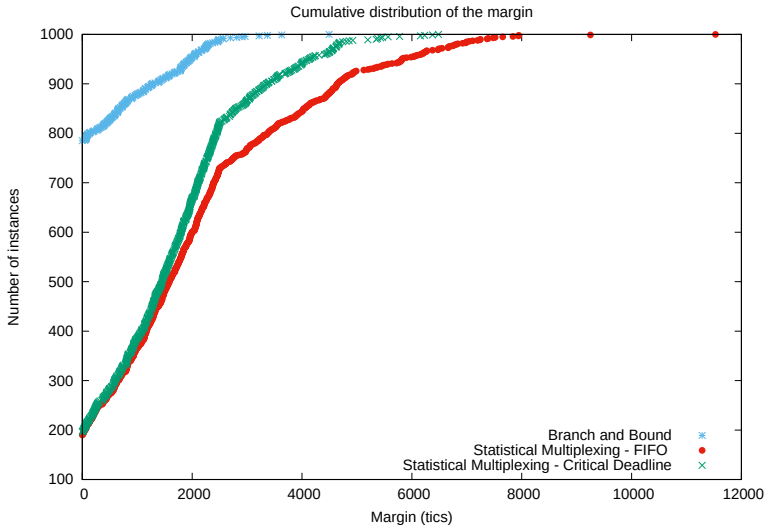

Figure 5.33 – Number of instances for which there is a solution less than a given margin, for Branch and Bound and Statistical Multiplexing, in a routed network of depth 3, load 0.8 and 8 routes.

Statistical multiplexing using the `CriticalDeadline` policy has a better margin than using `FIFO`, which is expected and similar to Chapter 4. However, it needs up to 2000 tics of margin to deal with the 80% most favorable instances, while Branch and Bound finds a solution with margin 0 for the same instances.

We now investigate the performance of Simulated Annealing compared to statistical multiplexing when the number of route is too large to execute Branch and Bound. On

Figure 5.33 are represented the cumulative distribution of the margin of the solutions found by by Simulated Annealing, and statistical multiplexing using `FIFO` or `CriticalDeadline` policy. The experiment has been made on 100 instances of load 0.8. Here, simulated annealing has been allowed to do 5000 generations of random compact assignment at each level, because it increases the quality of the results, with a reasonable computation time (a few minutes for each instance).

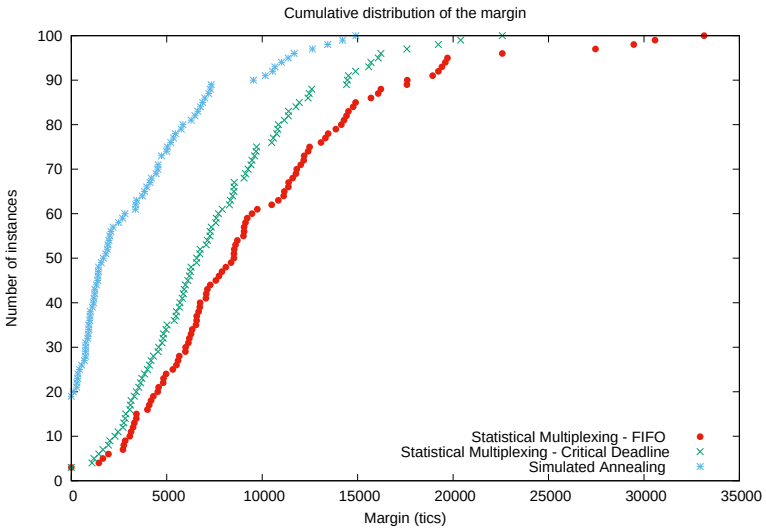

Figure 5.34 – Performance of Simulated Annealing Against Statistical Multiplexing for 24 routes.

Here, the number of instance for which there is an assignment with 0 margin is lower than in previous experiment, because the instance are harder to solve. Nevertheless, we observe that Simulated Annealing finds solution with about half the margin of statistical multiplexing using `CriticalDeadline` policy. The average margin is 3329 for Simulated Annealing, 7516 for `CriticalDeadline` and 9744 for `FIFO`.

Conclusion

This chapter present a variant of the problems studied in previous chapters. Here, we consider that the antennas send their message at the same date, which more faithfully represents the current C-RAN context. By setting all offsets to 0 and considering only one

contention vertex in which it is possible to delay the message, it seems hard to solve PALL, even if we have not investigated this problem in detail. We propose to allow buffering at every node of the network, and we study MINSTRA, the synchronized version of the minimization problem MINTRA. In both MINTRA and MINSTRA, the objective is to minimize $TR(A)$. In the first case, the value of $TR(A)$ is not impacted by the offsets. Thus, it is a local value of each route. In MINSTRA, since all offsets are set to 0, we want to minimize the global value of the process.

We propose greedy algorithms for MINSTRA, that are able to find solutions, whatever the load. However, those solutions are far from the optimal solution. We define a compact representation of the assignment that allows to reduce the number of assignment we must explore to find the optimal solutions for MINSTRA. We then define a neighborhood using this compact representation, and we optimize the solutions found by greedy algorithms with classical Hill Climbing, Tabu Search and Simulated Annealing. We also show that MINSTRA is FPT when parametrized by the number of routes, and we propose a Branch and Bound algorithm that find the optimal solution for a reasonable number of routes by enumerating all compact assignments. We reduce the number of compact assignment generated by Branch and Bound with several cuts using the properties on the compact form. The performance evaluations let us to conjecture that Simulated Annealing finds good solutions, even for instances with many routes.

One can still improve the computation time of Branch and Bound by computing only solutions which are minimal for MINSTRA or by further optimizing our implementation, but it does not seem possible to compute a solution with Branch and Bound for a large number of routes. Simulated Annealing can also be optimized by choosing a different neighborhood, or better analyzing the setting of the parameters, or choosing a more refined temperature cooling schedule.

Chapter 6

Mixing Periodic Datagrams and Stochastic Datagrams

In previous chapters, we presented algorithms that solve the problems of scheduling deterministic traffic in the network. In practice, networks are shared between deterministic and stochastic traffics. This is possible in practice using Time Sensitive Networking technology, that allows to manage traffics independently. The objective of this chapter is to study the impact on stochastic traffic of the algorithms we have designed to minimize the latency of deterministic traffic. We propose a method using the algorithms we have designed, to improve the latency of *all* traffics of the network.

6.1 Periodic Assignment and Random Traffic on Star Routed Networks

This section is taken from [42]. The algorithms proposed in this thesis are designed to manage deterministic periodic flows in dedicated networks. In this section, the objective is to determine the effect of adding in the network non-deterministic flows (internet traffic, best-effort) managed by statistical multiplexing.

The algorithms solving PALL are not designed to take into account best-effort traffic. In particular, they often build very compact assignments, with all messages following one another in a contention point, which is bad for the latency of best-efforts packets trying to go through the same contention point. Thus, we propose an adaptation of any algorithm solving PALL, to find assignments where the unused tics are as evenly spaced as possible to minimize the maximal latency of any random packet trying to go through the contention point.

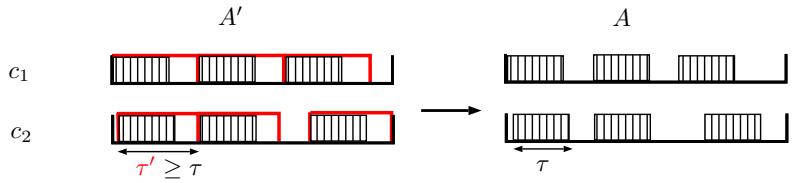

Figure 6.1 – A (P,τ')-assignment interpreted as a (P,τ)-assignment

6.1.1 Spaced Assignments

Most algorithms for PALL, when determining the waiting times, send datagrams as early as possible and thus create long sequences of datagrams in c_2, without free tics between them. We propose to modify any algorithm solving PALL on an instance with datagram size τ as follows: compute a (P,τ') assignment using the algorithm, for the largest possible $\tau' \geq \tau$.

Lemma 33. *Let $I' = (N,P,\tau',d)$ be an instance of* PALL*, for which there is an assignment, and let $\tau \leq \tau'$, then there is also an assignment for $I = (N,P,\tau,d)$.*

Proof. Let A be the assignment of I', the absence of collision is the absence of intersection between intervals $[r_i,c_1]_{P,\tau'}$ (and $[r_i,c_2]_{P,\tau'}$). If we consider A as an assignment of I, then the intervals are $[r_i,c_1]_{P,\tau}$ and are strictly included in $[r_i,c_1]_{P,\tau'}$, hence they do not have an intersection either. □

Lemma 33 gives a way to obtain a solution to the original instance from the instance with a larger message size as illustrated in Figure 6.1, with the additional property that all datagrams are separated by at least $\tau' - \tau$ free tics in each contention point. We are interested in finding the maximal τ' for which there is an assignment. Since the property of having an assignment is monotonous with regards to τ, we can do so by a dichotomous search.

We call `SPMLS`, for **S**paced `PMLS`, the adaptation of `PMLS` which finds an assignment for the largest possible τ by dichotomous search on τ. We now investigate how large are the τs for which `SPMLS` finds an assignment. In Figure 6.2, we represent the probability to find a (P,τ')-assignment function of τ'. The star routed networks are generated as in Chapter 4, with 8 routes and length of the arcs drawn in $[P]$. The network has a load of 60% of C-RAN traffic, hence the period is set to 33,333 for $\tau = 2500$. The network is less loaded with C-RAN traffic than in the previous sections because it will also support non deterministic traffic, incurring an additional load.

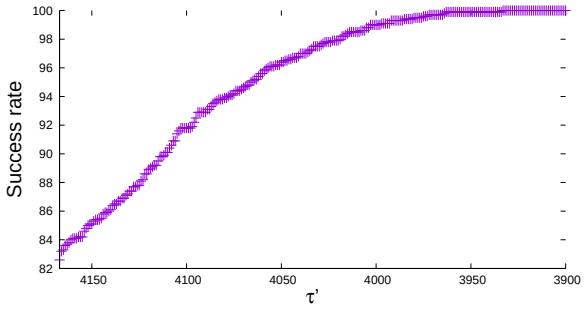

Figure 6.2 – Probability of finding a (P,τ')-assignment over 10,000 instances

For more than 80% of the instances, there is an assignment for the maximal size of a message $\tau' = \frac{P}{n} = 4166$. This means that `SPMLS` perfectly balances the free tics in the period. In the worst case, a solution with $\tau' = 3925$ is found, which still yields $3925 - 2500 = 1425$ unused tics between datagrams. Hence, we expect `SPMLS` to work well in conjunction with random traffic. The excellent performance of `PMLS` when the load is high explains this result and justifies the work we have done to solve PALL efficiently under high load rather than just requiring a mild load in applications.

6.1.2 Performance Evaluation

We evaluate in this section different ways to manage both statistical and deterministic traffics together in the same network.

6.1.2.1 Best-effort datagrams generation

Let us denote best-effort by BE. The BE traffic is generated as follows. The size of a BE datagram is small in practice, and set to 50 tics in our experiments. We generate 20% of average load of BE traffic in our experiment, to obtain a total load of 80%. The BE datagrams do not make a round trip in the network as the C-RAN datagrams, they go through a single contention point. We simulate that, by generating 20% of average load of BE datagrams for each of the two contention points c_1 and c_2. The **latency** of a BE datagram is defined as the time it must wait before going through its contention point.

On each contention point, the generation is split in two exponential distributions which

give the time before the next arrival of datagrams. The first one models background traffic, it has an average load of 15% by generating one BE datagram every 333 tics on average. The second models a burst of BE datagrams with an average load of 5%, that is, a generation of 10 datagrams, but every 10,000 tics on average.

6.1.2.2 Statistical multiplexing policy

We test several policies to deal with all traffics using statistical multiplexing. The BE traffic is managed using `FIFO`, and we propose two policies to deal with C-RAN. First, all datagrams, BE or C-RAN, are stored in the same buffer and dealt with the `FIFO` policy regardless of their type. We call this policy `FIFO`.

In order to minimize the latency of C-RAN traffic, we can store the two types of datagrams in two different buffers, managed each with `FIFO`, but we prioritize the C-RAN datagrams which are always sent first. It can be technically implemented by using TSN 802.1Qbu [16], that allows to define priority class in the traffic to schedule first the traffic with the highest priority, here the C-RAN traffic. We call this policy `FramePreemption`.

We also consider the case of C-RAN traffic scheduled by `PMLS` or `SPMLS`. Then, we need to forbid the transit of a BE datagram which collides with a C-RAN datagram. Thus, in each contention point, we reserve 50 tics (the size of a BE datagram) before the arrival of a C-RAN message. Observe that it wastes some ressources and thus slightly decreases the maximal throughput and may worsen the latency of BE datagrams.

Figure 6.3 shows the cumulative distribution of the logical latency of BE datagrams, that is the probability that a BE datagram has a latency less than some value. The distribution is computed over 1000 random instances, and for each the traffic is simulated for ten periods.

If we compare `FIFO` and `FramePreemption`, we see that the latency of BE datagrams is better (1977 tics on average) with `FIFO`. It is expected, since in `FramePreemption` the C-RAN datagrams are prioritized and thus the latency of the BE datagrams is strictly worse, 3256 tics on average. However, this is a trade-off with the margin of the C-RAN datagrams, which is strictly better for `FramePreemption`: 1919 tics on average versus 5265 tics for `FIFO`.

Using a deterministic approach for C-RAN with `PMLS`, the trade-off is even stronger: the CRAN margin is down to 0, but the BE traffic is more impacted, at a latency of 4909 tics on average. This can be explained by both reservation of tics to deal with the

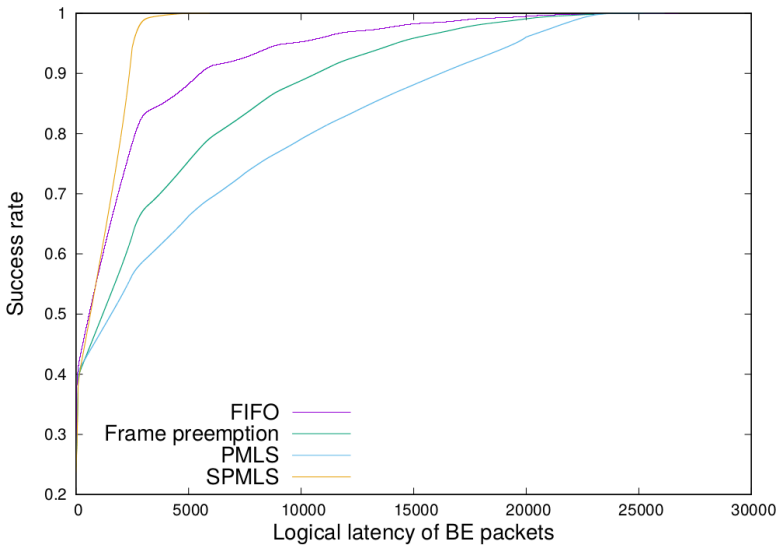

Figure 6.3 – Cumulative distribution of the latency of BE datagrams for several network management schemes

periodic sending scheme and the long sequences of CRAN datagrams without free time in contention points.

In `SPMLS`, the C-RAN traffic is smoothed over the period, in order to regularly leave some free tics for BE traffic. By construction, we still have CRAN margin of 0 but it improves the latency of BE datagrams to 949 tics on average, which is even better than with `FIFO`. This result shows that managing deterministic traffic deterministically is also good for the other sources of traffic on the network. We have already observed such a phenomenon in [52], a similar problem on an optical ring, that we describe in the next section.

6.2 Both Traffics On Optical Ring : An Industrial product

In this section, taken from [52], we study a C-RAN application based on an optical ring. We work on an industrial product which was developed in the ANR project N-GREEN described in [53, 54]. In contrast with the previous chapters, finding emission timings so that different periodic sources do not use the same resource is easy in the context of the N-GREEN optical ring with a single data-center. However, we deal with two additional difficulties arising from practice: the messages from RRHs are scattered because of the

electronic to optic interface and there are other traffics whose latency must be preserved. It turns out that the deterministic management of CRAN traffic we propose reduces the latency of CRAN traffic to the physical delay of the routes, while reducing the latency of the other traffics by smoothing the load of the ring over the period. To achieve such a good latency, our solution needs to reserve resources in advance, which slightly decreases the maximal load the N-GREEN optical ring can handle. Such an approach of reservation of the network for an application (CRAN in our context) relates to network slicing [55] or virtual-circuit-switched connections in optical networks [56, 57].

In Section 6.2.1, we model the optical ring and the traffic flow. In Section 6.2.2, we experimentally evaluate the latency when using stochastic multiplexing to manage packets insertion on the ring, with or without priority for C-RAN packets. In Section 6.2.3, we propose a deterministic way to manage C-RAN packets without buffers, which guarantees to have zero latency from buffering. We propose several refinements of this deterministic sending scheme to spread the load over time, which improves the latency of best-effort packet, or in Section 6.2.3.3, to allow the ring to support a maximal number of antennas at the cost of a very small latency for the C-RAN traffic.

6.2.1 Model of C-RAN traffic over an optical ring

N-GREEN Optical ring The unidirectional optical ring is represented by an oriented cycle. The vertices of the cycle represent the nodes of the ring, where the traffic arrives. The arcs (u,v) of the cycle have an integer weight $\omega(u,v)$ which represents the time to transmit a unit of information from u to v. By extension, if u and v are not adjacent, we denote by $\omega(u,v)$ the size of the directed path from u to v. The **ring size** is the length of the cycle, that is $\omega(u,u)$ and we denote it by RS. A **container**, of capacity C expressed in bytes, is a basic unit of data in the optical ring.

The time is discretized: a unit of time corresponds to the time needed to fill a container with data. As shown in Figure 6.4, the node u can fill a container with a data packet of size less than C bytes at time t if the container at position u at time t is *free*. If there are several packets in a node or if a node cannot fill a container, because it is not free, the remaining packets are stored in the **insertion buffer** of the node. A container goes from u to v in $\omega(u,v)$ units of time. The ring follows a **broadcast and select scheme with emission release policy**: When a container is filled by some node u, it is freed when it comes back at u after going through the whole cycle.

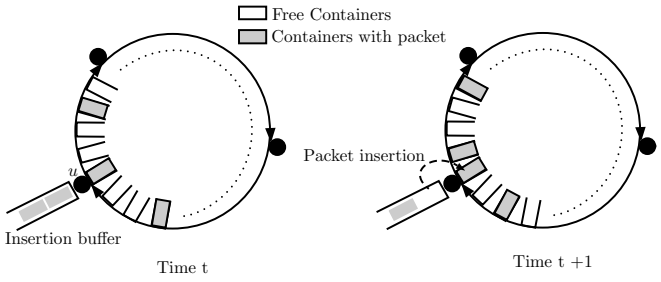

Figure 6.4 – Dynamic behavior of the ring.

C-RAN traffic The RRHs are the source of the **deterministic and periodic** C-RAN traffic. There are k RRHs attached to the ring and several RRHs can be attached to the same vertex. An RRH is linked to a node of the ring through an electronic interface of bit rate R Bps. The ring has a larger bit rate of $F \times R$ Bps. The integer F is called the **acceleration factor** between the electronic and the optical domains. A node aggregates the data received on the electronic interface during F units of time to create a packet of size C and then puts it in the insertion buffer. In each period P, an RRH emits data during a time called **emission time** or ET. Hence the RRH emits ET/F packets, i.e. requires a container of size C each F units of time during the emission time, as shown in Figure 6.5.

At each period, the data of the RRH i begins to arrive in the insertion buffer at a time o_i called **offset**. The offsets can be determined by the designer of the system and can be different for each RRH but must remain the same over all periods. We assume that all BBUs are contained in the same data-center attached to the node v. The data from u is routed to its BBU at node v through the ring and arrives at time $o_i + \omega(u,v)$ if it has been inserted in the ring upon arrival. Then, after some computation time, which w.l.o.g. is supposed to be zero, an answer is sent back from the BBU to the RRH. The same quantity of data is emitted by each BBU or RRH during any period.

The **latency** of a data packet is defined as the time it waits in an insertion buffer. Indeed, because of the ring topology, the routes between RRHs and BBUs are fixed, thus we cannot reduce the physical transmission delay of a data which depends only on the size of the arcs used. Moreover, there is only one buffering point in the N-GREEN optical ring, the insertion buffer of the node at which the data arrives. Hence, in this context, to minimize the end-to-end delay, we need to minimize the (logical) latency. More precisely, we want to reduce the latency of the C-RAN traffic to **zero**, both for the RRHs (uplink)

and the BBUs (downlink). In Section 6.2.3 we propose a deterministic mechanism with zero latency for C-RAN which also improves the latency of other data going through the optical ring. We shortly describe the nature of this additional traffic in the next paragraph.

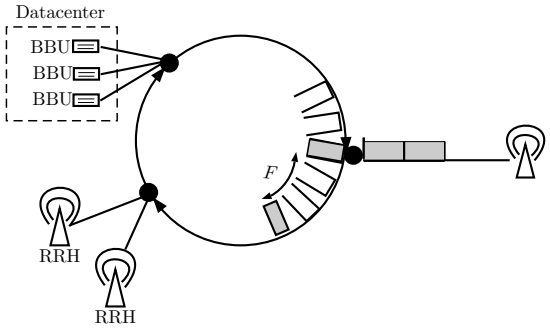

Figure 6.5 – Insertion of C-RAN traffic in the N-GREEN optical ring.

Best-Effort traffic The optical ring supports other traffics, corresponding to the internet flow. We call this traffic **Best-Effort** (BE). We want it to have the best possible distribution of latency, but since BE traffic is less critical than C-RAN traffic, we impose no hard constraint on its latency. At each node of the ring, a **contention buffer** is filled by a batch arrival process of BE data. This batch arrival process consists in generating, at each unit of time, a quantity of data drawn from a bimodal distribution to model the fact that internet traffic is bursty. Then, according to the fill rate of the contention buffer and the maximum waiting time of the data, a packet of size at most C may be created by aggregating data in the contention buffer. This packet is then put in the insertion buffer of the node. Hence, the arrival of BE messages can be modeled by a temporal law that gives the distribution of times between two arrivals of a BE packet in the insertion buffer. The computation of this distribution for the parameters of the contention buffer used in the N-GREEN optical ring is described in [58]. We use this distribution in our experiments to model arrivals of BE packets in the insertion buffer.

6.2.2 Evaluation of the latency on the N-GREEN optical ring

We first study the latency of the C-RAN and BE traffics when the ring follows an opportunistic insertion policy: When a free container goes through a node, it is filled with a packet of its insertion buffer, if there is one. Two different methods to manage the insertion buffer are experimentally compared. First, the **FIFO** rule, which consists in managing the C-RAN and BE packets in the same insertion buffer. Then, when a free container is

available, the node fills it with the oldest packet of the insertion buffer, without distinction between C-RAN and BE. This method is compared to a method called **C-RAN priority** that uses two insertion buffers: one for the BE packets, and another for the C-RAN packets. The C-RAN insertion buffer has the priority and is used to fill containers on the ring while it is non empty before considering the BE insertion buffer.

We compare experimentally these two methods in the simplest topology: The lengths of the arcs between nodes are equal and there is one RRH by node. The experimental parameters are given in Table 6.1 and chosen following [53]. In each experiment, the offsets of the RRHs are drawn uniformly at random in the period. The results are computed over 1,000 experiments in which the optical ring is simulated during 1,000,000 units of time. Fig. 6.6 gives the cumulative distribution of both C-RAN and BE traffics latencies for the FIFO and the C-RAN priority methods. The source code in C of the experiments can be found on the webpage [49].

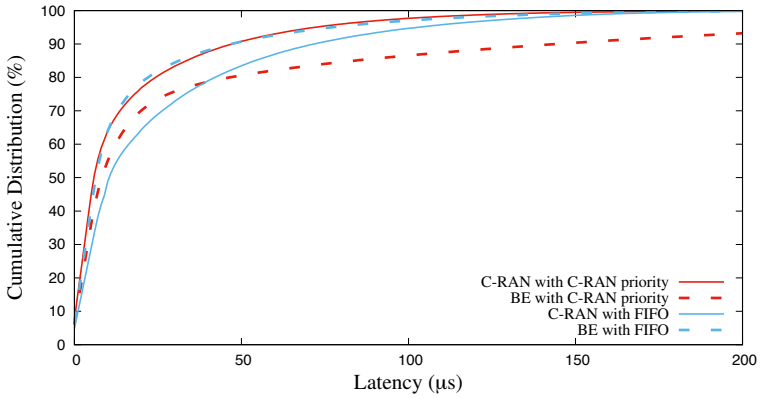

Figure 6.6 – Distribution of latencies for FIFO and C-RAN first

Bit rate of an electronic interface R	10 Gbps
Optical ring bit rate $F \times R$	100 Gbps
Acceleration factor F	10
Container size C	100 kb
Unit of time (UoT) $C/(F \times R)$	1 μs
Length traveled during one UoT	200 m

Time to go through the cycle RS	100 UoT
Emission time ET	500 UoT
Period P	1,000 UoT
Number of RRH	5
Number of nodes k	5
Load induced by C-RAN traffic	50%
Load induced by BE traffic	40%

Table 6.1 – Parameters of the N-GREEN architecture.

Unsurprisingly, the latency of the C-RAN traffic is better when we prioritize the C-RAN messages, while the BE traffic is heavily penalized. Furthermore, there is still 10% of the C-RAN traffic with a latency higher than 50μs, a problem we address in the next section.

Remark that, due to the broadcast and select mode, a message coming from any node induces the same load for all the nodes of the ring. Hence the latency of the traffics coming from any RRHs or from the BBUs are the same, which may seem couterintuitive knowing that all BBUs share the same node on the ring. This is why in Fig. 6.1 we do not ditinguish between uplink C-RAN traffic (RRH to BBU) and downlink C-RAN traffic (BBU to RRH).

6.2.3 Deterministic approach for zero latency

6.2.3.1 Reservation

Finding good offsets for the C-RAN traffic is a hard problem even for simple topologies and without BE traffic, as we have shown in previous chapters. In this section, we give a simple solution to this problem in the N-GREEN optical ring, and we adapt it to minimize the latency of the BE traffic.

Let u be the node to which is attached the RRH i. To ensure zero latency for the C-RAN traffic, the container which arrives at u at time o_i must be free so that the data from the RRH can be sent immediately on the optical ring.

To avoid latency between the arrival of the data from the RRH and its insertion on the optical ring, we allow nodes to **reserve** a container one round before using it. A container which is reserved cannot be filled by any node except the one which has reserved it (but it may not be free when it is reserved). If u reserves a container at time $o_i - RS$, then it is guaranteed that u can fill a free container at time o_i with the data of the RRH i. In the method we now describe, the C-RAN packets never wait in the node: The message sent by the RRH i arrives at its BBU at node v at time $o_i + \omega(u,v)$ and the answer is sent from the BBU at time $o_i + \omega(u,v) + 1$.

Recall that an RRH fills a container every F units of time, during a time ET. Thus if we divide the period P into **slots** of F consecutive units of time, an RRH needs to fill at most one container each slot. If an RRH emits at time o_i, then we say it is at **position** $o_i + \omega(u,v) \pmod F$. The position of an RRH corresponds to the position in a slot of the container it has emitted, when it arrives at v, the node of the BBU. If an RRH is at position p, then by construction, the corresponding BBU is at position $p+1 \pmod F$. For now, we do not allow waiting times for C-RAN traffic, hence each RRH uses a container at *the same position during all the emission time.*

Given a ring, a set of RRH's, a period and an acceleration factor F, the problem we solve here is to find an **assignment** of values of the offsets o_i's which is **valid**: two RRHs

must never use the same container in a period. Moreover we want to preserve the latency of the BE traffic. It means that the time a BE packet waits in the insertion buffer must be minimized. To do so, we must minimize the time a node waits for a free container at any point in the period, by spreading the C-RAN traffic as uniformly as possible over the period.

Figure 6.7 represents an assignment of two couples of RRH and BBU by showing the containers going through the node of the BBU during a period. Each slot has a duration of F unit of times, and, since an RRH/BBU emits a packet each F UoT during ET UoT, if we take the granularity of a slot to represent the time, the emission of a BBU/RRH is continuous in our representation, during ET/F slots. A date t in the period corresponds in Figure 6.7 to the slot t/F and is at position $t \mod F$.

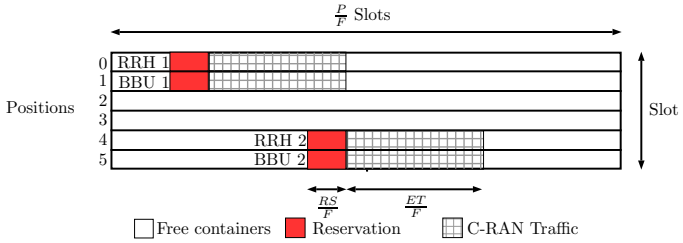

Figure 6.7 – A valid assignment with $F = 6$.

6.2.3.2 Building valid assignment with zero C-RAN latency

Remark that two RRHs which are not at the same position never use the same containers. Moreover, if we fix the offsets of the RRHs to even positions so that they do not reserve the same containers, then, because the answers of the BBU are sent without delay in our model, it will fix the offsets of the BBUs to odd positions which do not reserve the same containers. Hence, we need to deal with the RRHs only. The next proposition gives a simple method to find an assignment.

Proposition 6. *There is a valid assignment of the offsets $o_1, \ldots, o_k$ on the same position if $kET + RS \leq P$.*

Proof. W.l.o.g we fix o_1 to 0 and all the other offsets will then be chosen at position 0. Let $u_1, \ldots, u_k$ be the nodes attached to the RRHs $1, \ldots, k$. We assume that $u_1, \ldots, u_k$ are in the order of the oriented cycle. The last message emitted by the RRH 1 arrives at u_2 at time $ET - 1 + \omega(u_1, u_2)$. Therefore we can fix $o_2 = ET + \omega(u_1, u_2)$. In general we can

set $o_i = (i-1) \times ET + \omega(u_1,u_i)$ and all RRHs will use different containers at position 0 during a period. By hypothesis $k \times ET + \omega(u_1,u_1) \leq P$, thus the containers filled by the k-th RRH are freed before P. Hence, when the RRH 1 must emit something at the first unit of time of the second period, there is a free container. □

Remark that reserving free containers make them unusable for BE traffic which is akin to a loss of bandwidth. However, with our choice of emission times of the RRHs in the order of the cycle, most of the container we reserve are used by the data from some RRH. If all containers at some position are used, that is $kET + RS = P$, then there are only RS free containers wasted. In the worst case, less than $2RS$ containers are wasted by the assignment of Proposition 6.

It is now easy to derive the maximal number of antennas which can be supported by an optical ring, when using reservation and the same position for an RRH for the whole period.

Corollary 1. *There is a valid assignment with* $\lfloor \frac{P-RS}{ET} \rfloor \times \frac{F}{2}$ *antennas and zero latency.*

Proof. Following Proposition 6, the maximal number of antennas for which there is an assignment on the same position is $k = \lfloor \frac{P-RS}{ET} \rfloor$. In such an assignment, we need a second position to deal with the traffic coming from the BBUs coming back to those k antennas. Since we got F positions in the slot, the number of antennas supported by the ring is thus equal to $k \times \frac{F}{2}$. □

With the parameters of the N-GREEN ring given in Figure 6.1, we can support 5 antennas, while stochastic multiplexing can support 10 antennas albeit with extreme latency. There are two sources of inefficiency in our method. The first comes from the reservation and cannot be avoided to guarantee the latency of the C-RAN traffic. The second comes from the fact that an RRH must emit at the same position during all the emission time (to guarantee zero latency). We relax this constraint in Section 6.2.3.3 to maximize the number of antennas supported by the ring, while minimizing the loss of bandwidth due to reservation.

We now present an algorithm using reservation as in Proposition 6 to set the offsets of several RRHs at the same position. In a naive assignment, we put each RRH in an arbitrary position, for instance one RRH by position. We then propose three ideas to optimize the latency of the BE traffic, by spacing as well as possible the free containers in a period.

Balancing inside the period With the parameters of the N-GREEN ring given in Figure 6.1 ($ET = \frac{P}{2}$, $F = 10$ and $n = 5$), there are no unused position. Any assignment has exactly one BBU or RRH at each position. If all the RRHs start to emit at the first slot, then during ET there will be no free container anywhere on the ring, inducing a huge latency for the BE traffic. To mitigate this problem, in a period, the time with free containers in each position must be uniformly distributed over the period as shown in Fig. 6.8.

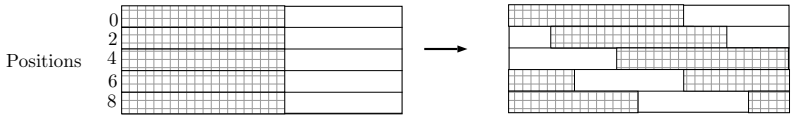

Figure 6.8 – Balancing inside the period.

Compacting positions For each position which is used by some RRH, and for each period, at least RS free containers are reserved which decreases the maximal load the system can handle. Therefore to not waste bandwidth, it is important to put as many RRHs as possible on the same position as shown in Fig. 6.9. Indeed, for any position which is not used at all, no container needs to be reserved. This strategy is also good to spread the load during the period since it maximizes the number of unused positions and for each unused position there is a container free of C-RAN traffic each F unit of times.

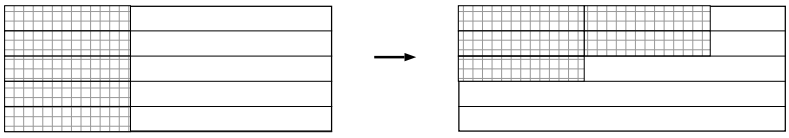

Figure 6.9 – Compacting positions.

Balancing used positions The free positions can be distributed uniformly over a slot, to minimize the time to wait before a node has access to a container from a free position, as shown in Fig. 6.10. To do so, compute the number of needed positions $x = \lceil k \times \frac{ET}{P-RS} \rceil$, with k the number of antennas using the previous strategy. Then, set the x used positions in the following way: $\lfloor \frac{F}{x} \rfloor - 1$ free positions are set between each used positions. If $\frac{F}{x}$ has a reminder r, then we set the r free remaining positions uniformly over the interval in the same way and so on until there are no more free position. It is a small optimization, since

it decreases the latency by at most $F/2$.

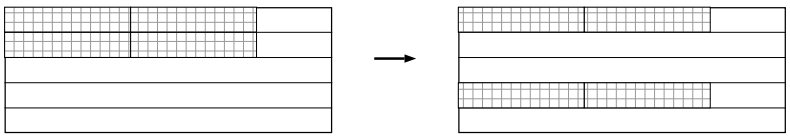

Figure 6.10 – Balancing used positions.

Experimental evaluation Our algorithm *combines the three methods* we have described to spread the load over the period. In order to understand the interest of each improvement, we present the cumulative distribution of the latency of the BE traffic using them either alone or in conjunction and we compare our algorithm to stochastic multiplexing with C-RAN priority.

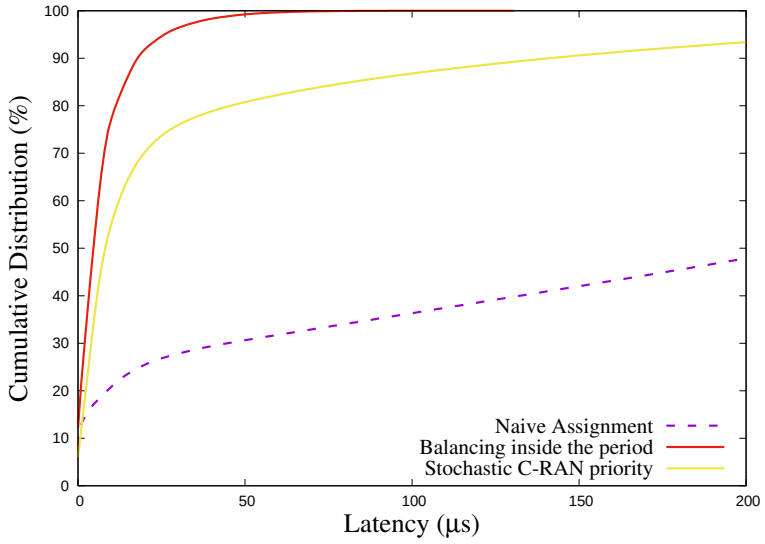

Figure 6.11 – BE latencies with a naive assignment and balancing inside the period for 5 antennas.

Figure 6.11 shows the performance of balancing the C-RAN traffic inside the period against a naive assignment in which all the RRH begin to emit at the same slot. We keep the same parameters as in Section 6.2.2 (see Table 6.1). As expected, the BE traffic latency is much better when we balance the C-RAN traffic inside the period and already much better than stochastic multiplexing.

To show the interest of compacting the positions, we must be able to put several RRHs at the same position. Hence, we change the emission time to $ET = 200$ and the number of antennas to $k = 12$ to keep the load around 90% as in the experiment of Figure 6.6. This is not out of context since the exact split of the C-RAN (the degree of centralization of the computation units in the cloud) is not fully determined yet [8].

As shown in Figure 6.12, the performance of the naive assignment is really bad. Compacting the RRHs on a minimal number of positions decreases dramatically the latency. If in addition, we balance over a period, we get another gain of latency of smaller magnitude: the average (respectively maximum) latency for BE traffic goes from 4.76μs (respectively 48μs) to 3.28μs (resp. 37μs). We did not represent the benefit of balancing used positions because the reduction in latency it yields is small as expected: the average (respectively maximum) latency for BE traffic goes from 4.76μs (resp. 48μs) to 4.43μs (resp. 44μs).

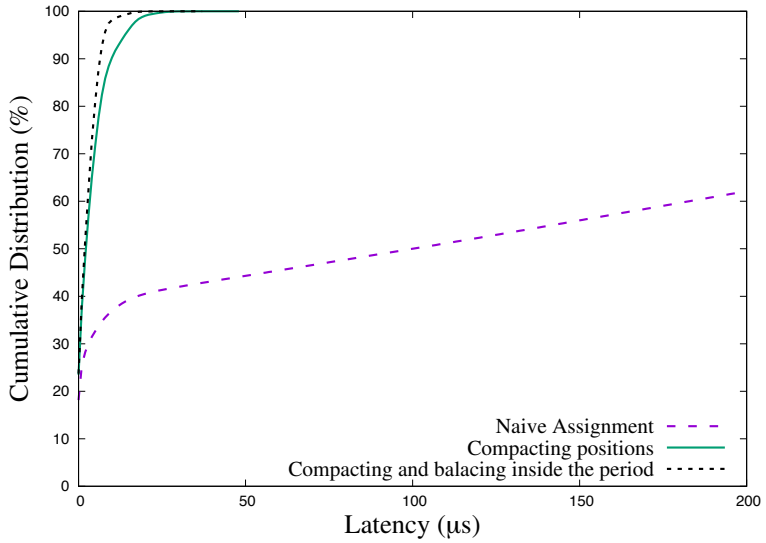

Figure 6.12 – BE latencies of compacting positions and balancing inside the period for 12 antennas.

In Figure 6.13, we compare the cumulative distribution of the latency of the BE traffic using the FIFO rule to our reservation algorithm with the three proposed improvements. The parameter are the same as in the previous experiment. The performance of our reservation algorithm is excellent, since the C-RAN traffic has *zero latency* and the BE traffic has a *better latency* than with the FIFO rule despite the cost of reservation. It is due to the balancing of the load of the C-RAN traffic over the period, that guarantee a more regular bandwidth for the BE traffic.

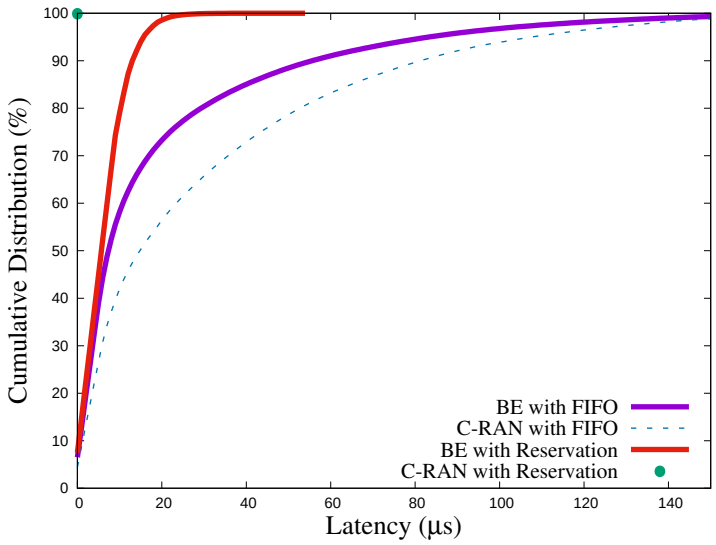

Figure 6.13 – FIFO buffer compared to the best method with reservation for 12 antennas.

6.2.3.3 Building Valid Assignments with Additional C-RAN Latency

The previous approach limits the number of antennas supported by the ring when $P - RS \mod ET \neq 0$, which is the case with N-GREEN parameters. The method we present in this section enables us to support more antennas and improves the latency of BE traffic (it reserves less free containers) by *allowing the data from an RRH to use two positions*. It is at the cost of a slightly worse latency for C-RAN traffic and it also requires in practice to implement some buffering for the C-RAN packets.

In order to support as much antennas as possible on the ring, we use *all* containers in a given position, improving on the compacting position heuristic.

Proposition 7. *There is a valid assignment for k antennas when $k \leq \lfloor \frac{P-RS}{ET} \times \frac{F}{2} \rfloor$.*

Proof. We consider the RRHs in the order of the ring. Let $l = \lfloor \frac{P-RS}{ET} \rfloor$, then we set the offsets of the first l RRHs as in Proposition 6. These RRHs are at position zero and the $(l+1)$th RRH first emits at position zero, with offset $o_{l+1} = l * ET + \omega(u_0,u_{l+1})$.

The $(l+1)$th RRH emits up to time $P - \omega(u_{l+1},u_0)$ at position zero, so that there is no conflict with RRH 0 during the next period. Hence, it has used the position zero during $x = P - \omega(u_{l+1},u_0) - l * ET - \omega(u_0,u_{l+1}) = P - l * ET - RS$. From time $P - \omega(u_{l+1},u_0) + 2$, the $(l+1)$th RRH emits at position 2 and during a time $ET - x$. Then the next RRH in

the order is assigned to position 2, and begins to emit at time $P - \omega(u_{l+1},u_0) + ET - x$ instead of zero. The rest of the assignment is built in the same way filling completely all first positions, until there are no more RRH. □

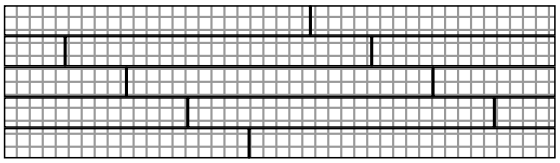

Figure 6.14 – Valid assignment for 9 antennas and the N-GREEN parameters.

Figure 6.14 illustrates the construction of Proposition 7 for the N-GREEN parameters. The loss due to reservation is exactly RS containers by used positions. Hence, it is possible to support 9 antennas (but no BE traffic in this extreme case), rather than 5 with the method of Section 6.2.3.2.

We call this new reservation algorithm **saturating positions** since it improves on compacting positions of the previous subsection. Moreover, there are no free slots in used positions, hence the idea of balancing into the period is not relevant. The only possible optimisation would be to balance the used positions, but it is not worth it since it adds additional latency for the RRHs using two different positions.

Figure 6.15 represents the cumulative distribution of the latency of BE traffic for the FIFO rule, saturating position, and balancing into the period using the N-GREEN parameters. Saturating positions reduces the BE traffic latency more than balancing into the period. This is easily explained by its lesser use of reservation. It is at the cost of a maximal latency of 2 μs for C-RAN traffic, so the designer can choose any of the two algorithms, according to the desired latency for C-RAN and BE traffic.

Conclusion

The concept of Cloud-RAN is to use non-dedicated networks, that is, networks shared with other applications. In this chapter, we study the impact of the scheduling of C-RAN flows on the latency of Best-Effort flows.

Our algorithm `ASPMLS` used to solve PALL on star routed networks tend to create long sequence of contiguous messages that monopolize the ressources during a long time. Hence, the latency of the Best-Effort flows is consequently worsen. To solve this issue, we virtually

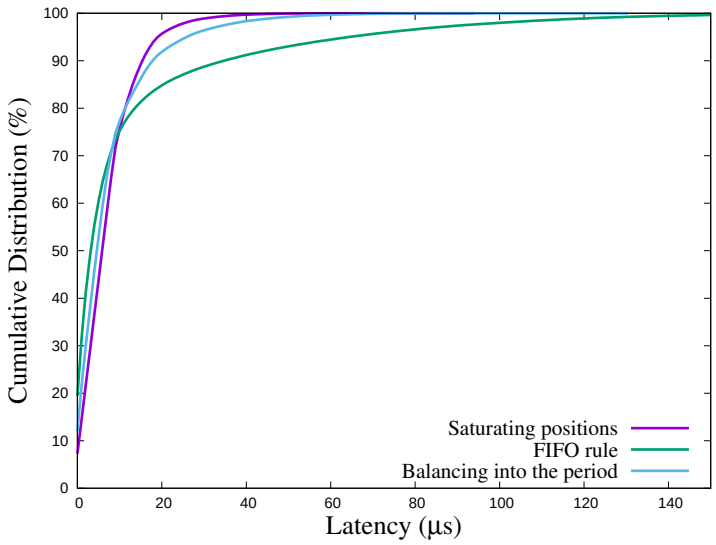

Figure 6.15 – Latencies of saturating positions, balancing into the period and FIFO rule for 5 antennas.

change the size τ of the datagram to the largest possible value τ' for which `ASPMLS` finds a solution. Then, we use the computed scheduling with the size of datagram τ, that leaves $\tau' - \tau$ free tics of time between every datagrams. Such an approach is possible because we are free to chose any offset in the period without impacting the latency. When solving SPALL, this approach would not be reasonable in term of latency, and we did not investigate yet the impact of our algorithms on Best-Effort latency.

We also present similar results on optical ring, developed for ANR project N-Green. In N-Green optial ring, the technical conception of the equipments makes the problem of scheduling the C-RAN flows trivial. We developped several techniques to smooth the load of the C-RAN flow over the period in order to let regular free tics for the Best-Effort traffic.

In both case, in order to ensure that a ressource scheduled for a route is free at the exact moment it is needed, we propose some reservation mechanisms. Reservation creates artificial use of bandwidth, which should result in lower latency for Best-Effort flows. In fact, it appears that Best-Effort latency is better when the C-RAN traffic is managed while smoothing the load on the period, even with the reservation than when all flows follows statistical multiplexing laws.

Mixing several kinds of flow and following a scheduling for a part of them is one of the major technical issue currently studied for deterministic networking. We detail in next

chapter how released standards leads us to deterministic management of the flows.

Chapter 7

Proof of Feasibility

The additional latency induced by contention buffers is one of the major manageable source of delay in a switched packets network. In this thesis, we propose algorithms that minimize contention in several topologies, either by removing the contention buffers or computing traffic organization to minimize buffering time. The objective is to delay datagrams as little as possible in the network.

Our approach of the network consists in deterministically managing datagrams in each node in order to prevent contention. When it is not possible to get rid of the contention buffers, we want to fix the duration each datagram is buffered. The contention buffers are not anymore a consequence of the traffic but a tool to manage it. This approach may look similar to the concept of Deterministic Networking. A working group from IETF called DetNet [59] works in collaboration with TSN (Time Sensitive Networking) [16], a task group of IEEE, to develop technical solutions for deterministic networking. The main difference between DetNet and TSN is the layer it focuses on. While DetNet works on Layer 3, TSN develops solutions for Layer 2. Whatever the case, using the term "deterministic" is inappropriate since all works related to DetNet or TSN are still based on statistical models. The latency guarantee given by those approaches are an upper bound on the latency, while we aim to minimize it.

In Section 7.1, we introduce the IEEE standards for TSN that allows for a better network management based on controlled management of flows in the switches. While TSN standards are designed to drive stochastic flows in network, we show that going further and managing deterministic traffic enable us to remove some technical constraint, and leads us to a new technology, that we call Hyper-TSN. Section 7.2 presents a prototype of a switch that goes beyond TSN, by delivering datagrams at exact planned dates. With Hyper-TSN, the additional latency due to contention buffers is minimized, even at full loads. Furthermore, we get rid of the synchronization constraint, with an innovative alignment mechanism. With Hyper-TSN, the latency is at physical limits. [10]

7.1 Customized Management of the Network

The model we present in Chapter 2 is based on several assumptions:

- The algorithms we develop are based on a central knowledge of the network and suppose all nodes are able to follow instructions. There must be an entity that centralizes and manages the configuration of the nodes.
- We assume the nodes are able to distinguish and manage different flows.
- Because all nodes follows a global scheduling , we assume they are able to dynamically rectify the shift between the clocks.

We explain in this chapter how the recent standards developed by TSN task group is close from such a model, and we show the limits of TSN for deterministic networking that leads us to Hyper-TSN. A detailed survey of all TSN standards can be found in [60].

7.1.1 Overview of TSN Standards

Centralized vision of the network The standard IEEE 802.1Qat SRP [61] (Stream Reservation Protocol) provides a central management framework that allows a centralized entity to collect data about the flows. It has been improved by IEEE 802.1Qcc [62]. In these standards, a centralized entity called the **controller**, collect all required information needed by users about the network (the routing, the periodicity and the size of the datagrams). This controller proposes a user interface that enable any user to collect all network information in order to compute the configuration of the network. Figure 7.1 shows a network managed by a controller, communicating via its user interface with algorithms, and able to send requirement to the nodes. Such an approach is related to Software Defined Network (SDN) [63]. We can find in [64] an example of an SDN for TSN.

Individual management of flows Standard 802.1Qbv [65] allows us to manage different flows in the nodes by a gate mechanism. Every output port of the switch is organized as follows. The flows are stored in traffic queues, and for each traffic queue, a gate is ordered to be open or closed. To do so, the switch needs a Gate Control List (GCL). This GCL is computed by the user, and sent to each switch of the network by the controller. It is a list of dates, and for every date of the list are specified the output ports (gates) of the switch which are open or closed.

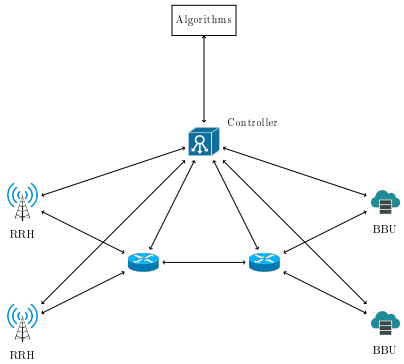

Figure 7.1 – A TSN network managed by a controller, able to collect network informations, and control the nodes behavior.

Figure 7.2 from [66] shows the mechanism of a switch using 802.1Qbv technology. Considering a given period (T_{cycle} in the figure), the switch selects at each time ($T_1, T_2, \ldots$ the queues that must be open to transmit datagrams. In figure 7.2, at time T_1, all gates except the one for scheduled traffic are open, at time T_2, all gates are closed and at time T_3 only the gate for scheduled traffic is open.

With such a mechanism, it is possible to organize the flows in order to control the latency. The GCLs are computed upstream considering the traffic (size of the datagrams, periodicity, or average throughput of each flow for non-deterministic traffic). Several works on Time Aware Shaping have been developed on this topic: [67] introduce how to manage one scheduled flow and one best-effort flow (non-periodic generation of data, stochastic model). This paper shows that by correctly setting the GCL of the nodes, it is possible to ensure no contention for one scheduled flow. Nevertheless, works about managing several scheduled flows together are mainly based on linear programming [31, 32, 30, 33], which has an high complexity and does not scale well with number of routes and contention depth of the networks.

Synchronization To be efficient, the components of the network must be completely synchronized. Such an hypothesis seems unrealistic. Standards like IEEE 802.1AS [68], or IEEE 1588 [69] propose good solutions for clock synchronization, but this problem is still difficult to solve. Indeed, even if those protocols propose good solution to re-synchronize clocks, there may be some shift between clocks of different switches of the network. The major source of shift is the need to precisely evaluate the length of a link between two components. This value can be deduced from the travel time of datagrams between two

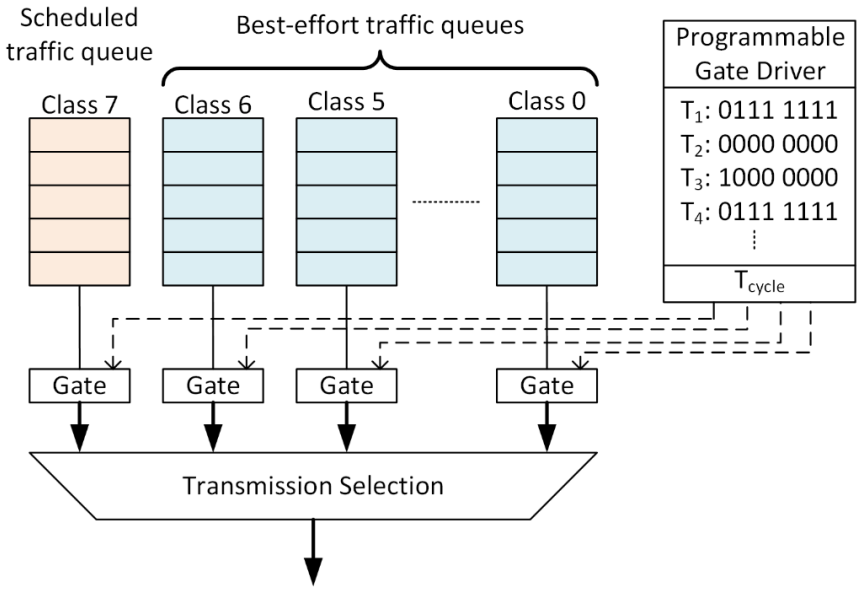

Figure 7.2 – IEEE 802.1Qbv mechanism ([66])

switches, but this value highly depends on the temperature of the link which is not constant. Network designers use *Guard Time* to prevent this problem. The GCL are computed such that a gate is open before and after the theoretical arrival of datagrams. Such a mechanism induce an artificial use of bandwidth, because a gate is open even when no datagram is being transmitted.

7.1.2 Limits of TSN when managing Deterministic flows

All the previously mentioned standards are designed for a statistical management of the flows. In this thesis, we work on deterministic flows: our models and algorithms do not rely to a statistical approach like current traffic shaping, but a deterministic approach. We compute the exact date at which each datagram is able to reach each node, without loss of bandwidth due to guard time. Furthermore, a statistical approach does not allow to control the contention and the exact forwarding time of the datagram in the network. This implies that a datagram sent periodically does not always have the same latency over time. The variation of the latency is called **jitter**. The jitter is an indicator of the stability of the network. The higher the jitter, the higher the maximal value of the latency is.

A deterministic approach requires to rethink the network management. We control the position of a datagram at every moment in order to let them pass through the nodes

without contention thanks to a gate mechanism similar to TSN Qbv. We thus need the nodes to be perfectly synchronized otherwise it could be counterproductive. Indeed, if a datagram of a flow arrives in a node before the planned date, the gate is closed and the datagram is buffered, that induces additional latency due to contention buffer. In the worst case in which a datagram arrives after the planned date, it is buffered until the gate get open. In our context of periodic flows, the datagram will be buffered between up to one period.

Also, switches in the physical layer of the network induce an additional latency due to physical buffering. In store-and-forward concept [21], datagrams are stored at reception of a node before being forwarded. However, solution like cut-through [70] allows to reduce storage size and corresponding delay to the header size only. But this is effective if the egress port is available to forward the datagram at the same time only. If not, the datagram is buffered until the port is free. Even if it is possible to adapt our model to take into consideration the physical buffering cost, it still induces additional latency, which is not desirable.

Next section present a new kind of switch, called Hyper-TSN switch, that allows to overcome all the above limits.

7.2 Deterministic management for a deterministic latency

Deterministic Networking are mentioned in the same survey cited ahead for TSN [60]. The researches about DetNet are until now limited either to linear programming for finding scheduling policies for the network -as mentioned above- or traffic shaping (see [71] for a comparison of actual traffic shaping methods). Traffic shaping methods are based on stochastic models, with bounds on the arrival of the flows, that allows to bound the maximal latency. Nevertheless, as we mentioned ahead, Deterministic Networking do not propose a deterministic management of deterministic flows in order to ensure a minimal latency and 0 jitter.

In this thesis we remove the contention buffer, or, if this is not possible, we use the contention buffers as a tool, by controlling the buffering time of each datagrams while minimizing it. Furthermore, we remove jitters in network, which is a deeper aspect of deterministic networking. Here, because of our desire to manage deterministic flows, multiple standards of TSN are not useful yet. Indeed, since the exact date of arrival of all

datagrams are computed, we can get rid of several tools designed to manage the traffic on a statistical manner.

We present in this section an Hyper-TSN switch that solves all the problematic of TSN when managing deterministic traffic. This technology is under an advanced phase of research, and a prototype has been experimented in Nokia Bell Labs [72].

7.2.1 Hyper-TSN switch

A 2x2 Hyper-TSN switch has been developed. It is composed of a switching matrix connected to a deterministic scheduler and two 10 Gbps ethernet two ways transceivers. The switching matrix includes also a monitoring circuitry. The deterministic scheduler controls the switching matrix and is configured with a timing table. This table is similar to a 802.1 Qbv Gate Control List (GCL). It defines the periodicity of the scheduling and, for each egress ports, the planned date of arrival of the frames which are part of deterministic flows. At each of these dates the deterministic scheduler sets the switching matrix to transmit data incoming on a specified ingress port. Figure 7.3 shows an example of scheduling both deterministic (but not periodic) and stochastic traffic. In such a switch, the buffers are totally absent for managed traffic: when a datagram arrives in the switch, it is instantly transmitted to the egress port, and there is no buffering operation. This process allows us to reduce the physical delay to its lowest for scheduled traffic, while it is still possible to buffer the Best-Effort traffic, which has not critical latency constraints.

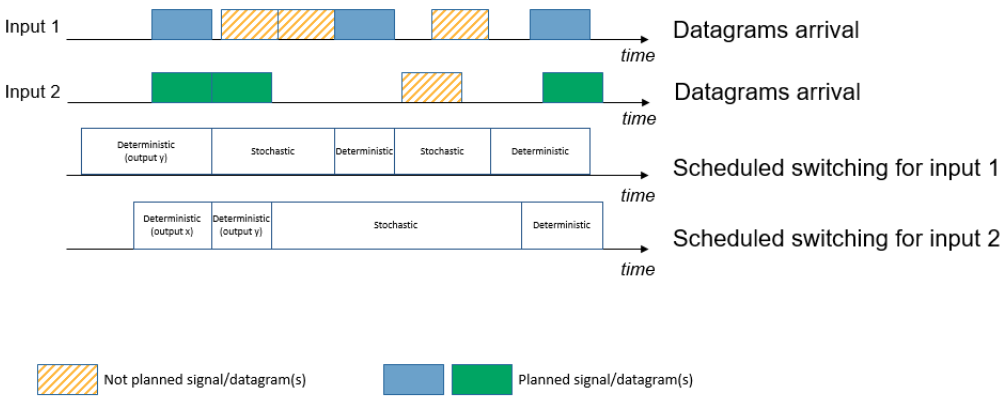

Figure 7.3 – The scheduling of a 2x2 switch on which both deterministic and stochastic traffics arrives. The deterministic traffic is forwarded without contention.

To coordinate switches, Hyper-TSN considers the flow as a reference [73] as we now explain. Since the exact arrival time of a datagram is known, if one datagram arrives before or after this expected date, this means the physical transmission delay between the sender of the datagram and the switch known by the controller is false. The monitoring circuitry of the switch detects this issue and an alert signal is sent to the controller that is able to reschedule correctly the GCL. This ensures a dynamical clock alignment between the nodes, and it is possible because the switches are developed on components with industrial clocks (Xilinx FPGA boards : Zynq UltraScale MPSoC zcu102, Zynq-7000 SoC zc706). The switch also includes a frame analyzer that enables to check that the switched frames are not corrupted and none is missing.

7.2.2 Implementation and reliability tests

To perform experiments, a generator of deterministic flows has been developed. This generator set the dates it sends the frames according to the controls received from the monitoring circuitry. The period are defined in the timing table. The size of the frames is set to fully load the ethernet links (i.e. 100% load). When starting, the monitoring circuitry detects that frames do not arrive at the planned date and sends control commands to the generator. These first frames are lost. Then, the generator corrects the dates it sends the frames, and no more shifting has been observed during the running of a 2 hours experiment. 100% of the frames are correctly switched without being corrupted or lost. The switching of each frame from the ingress port to the planed egress port is performed introducing only one clock cycle delay (here 3,87 ns). Figure 7.4 shows the Hyper-TSN 2x2 switch used for our experiments.

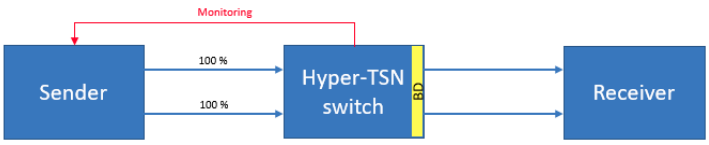

Figure 7.4 – An Hyper-TSN switch with a 2x2 switching matrix.

Conclusion

The algorithms developed in this thesis are based on several assumptions. First, we consider that a central entity is able to collect informations about the flows and the routes of the network. This is possible in practice by using SDN, which is one of the major solution for dynamic programmable networks.

The nodes need to be able to differentiate flows and to follow a scheduling to forward them. The norm TSN 802.1Qbv allows such a mechanism. We also assume that the nodes are synchronized with the precision of a tic. Even with advanced synchronization protocols, it is hard to synchronize devices with such a precision, and at least the header of the messages needs to be buffered to be read in classical packet switched networks.

As a solution to all these requirements, we present Hyper-TSN switches, which are currently in development. Those switches are based on a new vision of the network, by considering the flow as a reference. Because we manage the arrival time of every datagram in the nodes, we are able to detect a time shift if a datagram does not arrive a expected date. This allow a precise clock alignment of the devices. Furthermore, those switches are able to forward the datagrams without any buffer due to physical operations, which is an innovation in packet switched network.

Conclusion

In this thesis, we presented the problem of minimizing latency of periodic flows in a packet switched network. Current networks in use for telecommunication are based on statistical multiplexing: the links are dimensioned considering the average bit rate of the flows so that the flows can share a link most of the time or be buffered until enough capacity is available. Statistical multiplexing is a low-cost solution to deploy a network, but it does not guarantee the latency of the packets using it. If a burst of data is sent by one flow, shared resources become critical and some packets are buffered while waiting for their availability. These buffers are called contention buffers, and are a major source of latency.

We study the Cloud-RAN application case. In C-RAN, radio antennas periodically send packets to datacenters, that compute an answer and send it back to the antennas. The packets must have an end-to-end latency lower than a maximal value, required by the protocols. Statistical multiplexing is not able to guarantee an end-to-end latency for packets, and because of contention buffers, the more loaded is a network, the largest is the latency. In our C-RAN use case, the flows are periodic and a large amount of data is sent at each period by the antennas and the datacenters. Thus, managing the packets with statistical multiplexing is not appropriate.

Several working groups (DetNet, TSN, see Chapter 7), have developed standards and mechanisms ensuring an upper bound on the latency in packet switched networks. The network devices can reserve a port during a given time to forward the traffic of a given flow without contention buffer. The arrival date of the packets in a device of the network must then be known and precise. To do so, a scheduling of every output port of the devices is computed ahead. Current approaches to compute this scheduling are based on stochastic laws, since most of the internet traffic follows a stochastic behavior. In such a situation, it is impossible to completely get rid of contention buffers. Nevertheless, since our flows are deterministic (the amount of data and the periodicity of the packets remain the same all over time), we show that deterministic scheduling guarantees a minimal latency of deterministic flows and it also helps to reduce the latency of stochastic best-effort flows. Furthermore, TSN and DetNet mechanisms induce several sources of additional latency, like guard time around packets to prevent the time shift between clocks or buffering time of the header of the packets in every switch to read the destination. In this thesis, we go further by

proposing solutions to reduce the end-to-end latency of packets to its physical transmission time by proposing a new generation of switch that get rid of technical constraints imposed by a statistical vision of the network.

This thesis focuses on the problem of computing a deterministic scheduling for periodic flows, a problem that we prove to be NP-hard for arbitrary networks. We study in Chapters 3 and 4 this problem when the flows are unsynchronized, that is, we can choose the emission date of the packets in the sources of the flows. This case does not perfectly match with Cloud-RAN, but it corresponds to various use cases, like Industry 4.0, autonomous vehicle... In Chapter 3, we give several greedy algorithms and one FPT algorithm that allows to reduce the latency to the physical transmission time (i.e. without any contention) on a common network topology with a single shared link, when the load induced by the deterministic traffic is low enough. We experimentally show using the exact FPT algorithm, that on very loaded networks (when the load is greater than 80%), it is not possible to get rid of contention buffers. We then propose solutions that buffer packets in nodes of the network, but we try to minimize this additional latency. Remark that in such an approach, the buffers are not anymore a consequence of contention that we cannot control or predict, but a tool to organize the packets. In Chapter 4, we study the problem of organizing flows in a network with a single shared link (as in Chapter 3), and allowing one buffer (positioned in the datacenters) on the route for every packet. We propose several greedy algorithms and one FPT algorithm based on a classical scheduling algorithm that we adapted for periodicity. The performances of our algorithm are excellent, we show it is possible to reduce latency to the physical transmission time of the longest route in 99,9% of the cases, while statistical multiplexing, even prioritizing critical flows adds a latency to the flows, due to contention buffers, equal to 1/4 of the period.

We study in Chapter 5 the C-RAN use case, in which all antennas send their messages at the same date, on arbitrary networks. We propose a compact form of the solutions to our problem, that allows to define a neighborhood of a solution. Then, several local search heuristics are designed using this notion of neighborhood. A branch and bound algorithm based on the compact form of the solutions is also proposed and run efficiently for small C-RAN network with ten to twenty routes. Then, we experimentally show that our approach dramatically over performs statistical multiplexing in terms of latency.

We then show in Chapter 6 how to adapt our algorithm not to impact best-effort flows latency while scheduling the C-RAN traffic. We explain how to adapt all our algorithms,

by solving instances with artificially increased size of messages, in a way which does not impact the latency of C-RAN datagrams, and smooth the load of C-RAN traffic all over the period. We show that, even if our approach induces an additional use of bandwidth due to resource reservation, we are able to improve the average latency of best-effort traffic while minimizing the C-RAN traffic latency. We also show similar results in an industrial optical ring, in which scheduling the C-RAN packets is trivial because of the multiplexing of the electronic signals over the optical ring.

We have proposed solutions to minimize the end-to-end latency in various use cases: Cloud-RAN, Industry 4.0, motion control, autonomous vehicle, etc... Reducing the transmission latency allows to:

- Respect latency constraints required by protocols
- Increase the Network Quality of Service
- Give more time to the other components of the chain (computation in datacenters for C-RAN example)
- Augment the maximum physical links length, which means, for C-RAN; a wider area of development and thus lower exploitation and development costs (CAPEX, OPEX).

Limits and Further researchs

Several questions remain open in our work. We conjecture that PAZL and PALL are NP-hard on star routed networks, but are not yet able to prove it. The algorithms we developed for PAZL and PALL are designed for star routed networks and most of them are not easy to extend to general networks. The greedy algorithms we proposed to solve PAZL can be adapted for arbitrary networks respecting the coherent routing property, but the they can be proved to work only for small loads. These algorithms are not usable in their current state, and we need to study them carefully. Furthermore, the FPT algorithm we give to solve MINSTRA can be adapted to solve PAZL and PALL on arbitrary networks. An experimental study has to be done to see if the approach is promising and the algorithm could be optimized for the case of unsynchronised messages.

This thesis arises in the context of SDN which aims to develop dynamical and programmable networks. In C-RAN for example, the radio network aims to be able to turn off antennas when the number of connected devices is low. Our algorithms for PAZL and

PALL compute the scheduling for all flows, and must reschedule the entire solution if a flow is removed or added to the network to ensure a minimal latency. In the case of the algorithms presented for MINSTRA, the compact form of the solutions we presented allows us to efficiently add a flow to the best solution, but not to quickly re-compute the best solution when a flow is removed. A challenge is thus to design a dynamic algorithm, which can produce a new solution quickly after a local change in the network.

The measure we want to minimize is the maximal end-to-end latency of all flows. This often means that the flow using the longest route is never delayed in contention buffer, but the other flows are more or less impacted by the solutions we give. One can imagine use-cases in which the constraint on latency is not as strict, and where it makes more sense to minimize the average latency of the flows. For this variant of the problem, the study of PAZL is still relevant, but it may be simpler from a complexiy point of view and the algorithms may be quite different from the ones presented in this thesis. We can also expand our model to allow flows in the networks with different periods or sizes of message. Already several methods fail for mixing different message sizes, since solving the problem WTA becomes NP-hard in this context and mixing several periods require to completely change our algorithms.

In Section 6.2 we introduce the fact that links of the networks may have different capacity. This induces additional latency due to the physical conversion. It could allow use cases requiring low latency to be developed on metropolitan networks. It is possible to adapt our model to take into account this issue, and to consider this physical conversion time while computing the solutions.

Last, we consider that the routing is given in our model. It could be interesting to compute the routing along with the assignment to further improve the quality of the results. We plan to study this question on star routed networks, where the number of routings is limited and an exhaustive approach is conceivable.

Bibliography

[1] Erik Dahlman, Stefan Parkvall, and Johan Skold. *5G NR: The next generation wireless access technology*. Academic Press, 2018.

[2] Giovanni Romano. "IMT-2020 Requirements and Realization". In: *Wiley 5G Ref: The Essential 5G Reference Online* (2019), pp. 1–28.

[3] 5G Alliance for Connected Industries and Automation. *White paper: 5G for Connected Industries and Automation , second edition*. Tech. rep. 5G Alliance for Connected Industries and Automation. URL: https://www.5g-acia.org/publications/5g-for-connected-industries-and-automation-white-paper/.

[4] Matthew Baker and Miikka Poikselkâ. *White paper: 5G Releases 16 and 17 in 3GPP*. Tech. rep. Nokia.

[5] Ravi Krishnamurthy et al. *Latency-based statistical multiplexing*. US Patent 6,665,872. 2003.

[6] Chitra Venkatramani and Tzi-cker Chiueh. "Supporting real-time traffic on Ethernet". In: *Real-Time Systems Symposium, 1994., Proceedings*. IEEE. 1994, pp. 282–286.

[7] Yannick Bouguen, Eric Hardouin, and François-Xavier Wolff. *LTE pour les reseaux 4G*. Editions Eyrolles, 2012.

[8] China Mobile. "C-RAN: the road towards green RAN". In: *White Paper, ver* 2 (2011).

[9] Federico Boccardi et al. "Five disruptive technology directions for 5G". In: *IEEE Communications Magazine* 52.2 (2014), pp. 74–80.

[10] *ALTO Performance Cost Metrics draft-ietf-alto-performance-metrics-12*. https://tools.ietf.org/html/draft-ietf-alto-performance-metrics-12. Accessed: 2021-02-11.

[11] *Understanding Cisco Express Forwarding (CEF)*. https://www.cisco.com/c/en/us/support/docs/routers/12000-series-routers/47321-ciscoef.html. Accessed: 2021-02-10.

[12] Anna Pizzinat et al. "Things you should know about fronthaul". In: *Journal of Lightwave Technology* 33.5 (2015), pp. 1077–1083.

[13] Z Tayq et al. "Real time demonstration of the transport of ethernet fronthaul based on vRAN in optical access networks". In: *Optical Fiber Communications Conference and Exhibition (OFC), 2017*. IEEE. 2017, pp. 1–3.

[14] B. Leclerc and O. Marcé. *Transmission of coherent data flow within packet-switched network*. EP Patent App. EP20,140,307,006. 2016. URL: https://www.google.com.gt/patents/EP3032781A1?cl=en.

[15] B. Leclerc and O. Marcé. *Signaling for transmission of coherent data flow within packet-switched network*. EP Patent App. EP20,140,306,186. 2016. URL: http://www.google.com.gt/patents/EP2978180A1?cl=en.

[16] *Time-Sensitive Networking Task Group*. http://www.ieee802.org/1/pages/tsn.html. Accessed: 2021-02-10.

[17] Time-Sensitive Networking Task Group of IEEE 802.1. "Time-Sensitive Networks for Fronthaul". In: (2016). IEEE P802.1/D0.4.

[18] Lionel M. Ni and Philip K. McKinley. "A survey of wormhole routing techniques in direct networks". In: *Computer* 26.2 (1993), pp. 62–76.

[19] Richard J Cole, Bruce M Maggs, and Ramesh K Sitaraman. "On the benefit of supporting virtual channels in wormhole routers". In: *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*. ACM. 1996, pp. 131–141.

[20] L. Schwiebert and D. Jayasimha. "A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing". In: *J. Parallel Distributed Comput.* 32 (1996), pp. 103–117.

[21] Elbert G Tindell and Kyle Crawford. *Store and forward video system*. US Patent 5,130,792. 1992.

[22] Thomas Erlebach and Klaus Jansen. "The complexity of path coloring and call scheduling". In: *Theoretical Computer Science* 255.1 (2001), pp. 33–50.

[23] Ralf Borndörfer et al. "Frequency assignment in cellular phone networks". In: *Annals of Operations Research* 76 (1998), pp. 73–93.

[24] Xuding Zhu. "Circular chromatic number: a survey". In: *Discrete Mathematics* 229.1 (2001), pp. 371–410. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(00)00217-X. URL: https://www.sciencedirect.com/science/article/pii/S0012365X0000217X.

[25] Bing Zhou. "Multiple Circular Colouring as a Model for Scheduling". In: (2013).

[26] Alan Tucker. "Coloring a Family of Circular Arcs". In: *SIAM Journal on Applied Mathematics* 29.3 (1975), pp. 493–502. ISSN: 00361399. URL: http://www.jstor.org/stable/2100446.

[27] Vijay Kumar. "Approximating circular arc colouring and bandwidth allocation in all-optical ring networks". In: *Approximation Algorithms for Combinatiorial Optimization*. Ed. by Klaus Jansen and José Rolim. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 147–158. ISBN: 978-3-540-69067-2.

[28] Richard M Lusby et al. "Railway track allocation: models and methods". In: *OR spectrum* 33.4 (2011), pp. 843–883.

[29] Paolo Serafini and Walter Ukovich. "A mathematical model for periodic scheduling problems". In: *SIAM Journal on Discrete Mathematics* 2.4 (1989), pp. 550–581.

[30] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. "Incremental flow scheduling and routing in time-sensitive software-defined networks". In: *IEEE Transactions on Industrial Informatics* 14.5 (2017), pp. 2066–2075.

[31] Wilfried Steiner, Silviu S Craciunas, and Ramon Serna Oliver. "Traffic planning for time-sensitive communication". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 42–47.

[32] Silviu Craciunas, Ramon Serna Oliver, and Wilfried Steiner. "Formal Scheduling Constraints for Time-Sensitive Networks". In: (Dec. 2017).

[33] Naresh Nayak, Frank Dürr, and Kurt Rothermel. "Time-sensitive Software-defined Network (TSSDN) for Real-time Applications". In: Oct. 2016, pp. 193–202. DOI: 10.1145/2997465.2997487.

[34] Aellison Cassimiro T dos Santos, Ben Schneider, and Vivek Nigam. "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking". In: *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE. 2019, pp. 69–77.

[35] Barbara Simons. "A fast algorithm for single processor scheduling". In: *Foundations of Computer Science, 1978., 19th Annual Symposium on*. IEEE. 1978, pp. 246–252.

[36] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. "Complexity of machine scheduling problems". In: *Annals of discrete mathematics* 1 (1977), pp. 343–362.

[37] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. "Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard". In: *Journal of Scheduling* 7.5 (2004), pp. 333–348.

[38] Jan Korst et al. "Periodic multiprocessor scheduling". In: *PARLE'91 Parallel Architectures and Languages Europe*. Springer. 1991, pp. 166–178.

[39] Eugene Levner et al. "Complexity of cyclic scheduling problems: A state-of-the-art survey". In: *Computers & Industrial Engineering* 59.2 (2010), pp. 352–361.

[40] Claire Hanen and Alix Munier. *Cyclic scheduling on parallel processors: an overview*. Université de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique, 1993.

[41] Dominique Barth et al. "Deterministic Scheduling of Periodic Messages for Cloud RAN". In: *25th International Conference on Telecommunications, ICT 2018, Saint Malo, France, June 26-28, 2018*. IEEE, 2018, pp. 405–410. DOI: 10.1109/ICT.2018.8464890. URL: https://doi.org/10.1109/ICT.2018.8464890.

[42] Dominique Barth, Maël Guiraud, and Yann Strozecki. "Deterministic Scheduling of Periodic Messages for Cloud RAN". In: *CoRR* abs/1801.07029 (2018). arXiv: 1801.07029. URL: http://arxiv.org/abs/1801.07029.

[43] *IEEE 802.3 10 Gb/s*. URL: http://www.ieee802.org/3/10GMMFSG/ (visited on 07/08/2020).

[44] Fabien De Montgolfier, Mauricio Soto, and Laurent Viennot. "Treewidth and hyperbolicity of the internet". In: *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*. IEEE. 2011, pp. 25–32.

[45] Ian Holyer. "The NP-completeness of edge-coloring". In: *SIAM Journal on computing* 10.4 (1981), pp. 718–720.

[46] David Zuckerman. "Linear degree extractors and the inapproximability of max clique and chromatic number". In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. ACM. 2006, pp. 681–690.

[47] Alex J Orman and Chris N Potts. "On the complexity of coupled-task scheduling". In: *Discrete Applied Mathematics* 72.1-2 (1997), pp. 141–154.

[48] Maël Guiraud and Yann Strozecki. "Scheduling periodic messages on a shared link". In: *CoRR* abs/2002.07606 (2020). arXiv: 2002.07606. URL: https://arxiv.org/abs/2002.07606.

[49] *Maël Guiraud's website*. https://mael-guiraud.github.io/.

[50] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[51] Ibrahim H Osman and James Patrick Kelly. "Meta-heuristics theory and applications". In: *Journal of the Operational Research Society* 48.6 (1997), pp. 657–657.

[52] Dominique Barth, Maël Guiraud, and Yann Strozecki. "Deterministic Contention Management for Low Latency Cloud RAN over an Optical Ring". In: *Optical Network Design and Modeling - 23rd IFIP WG 6.10 International Conference, ONDM 2019*. Vol. 11616. Lecture Notes in Computer Science. Springer, 2019, pp. 479–491.

[53] Dominique Chiaroni. "Network Energy: Problematic and solutions towards sustainable ICT". In: *invited paper, International Commission of Optics (ICO-24)* (2017).

[54] Bogdan Uscumlic et al. "Scalable deterministic scheduling for WDM slot switching Xhaul with zero-jitter". In: *2018 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2018, pp. 100–105.

[55] Menglan Jiang, Massimo Condoluci, and Toktam Mahmoodi. "Network slicing management & prioritization in 5G mobile systems". In: *European wireless*. 2016, pp. 1–6.

[56] Christian Cadéré et al. "Virtual circuit allocation with QoS guarantees in the ECOFRAME optical ring". In: *Optical Network Design and Modeling (ONDM), 2010 14th Conference on*. IEEE. 2010, pp. 1–6.

[57] Ted H Szymanski. "An ultra-low-latency guaranteed-rate Internet for cloud services". In: *IEEE/ACM Transactions on Networking* 24.1 (2016), pp. 123–136.

[58] Youssef Ait el mahjoub, Hind Castel-Taleb, and Jean-Michel Fourneau. "Performance and energy efficiency analysis in NGREEN optical network". In: *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) (WiMob 2018)*. Limassol, Cyprus, Oct. 2018.

[59] Norman Finn and Pascal Thubert. *Deterministic Networking Architecture*. Internet-Draft draft-finn-detnet-architecture-08. Work in Progress. Internet Engineering Task Force, 2016. 32 pp. URL: https://tools.ietf.org/html/draft-finn-detnet-architecture-08.

[60] A. Nasrallah et al. "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research". In: *IEEE Communications Surveys Tutorials* 21.1 (2019), pp. 88–145. DOI: 10.1109/COMST.2018.2869350.

[61] Daniel Bujosa, Inés Álvarez, and Julian Proenza. "CSRP: an Enhanced Protocol for Consistent Reservation of Resources in AVB/TSN". In: *IEEE Transactions on Industrial Informatics* PP (Aug. 2020), pp. 1–1. DOI: 10.1109/TII.2020.3015926.

[62] "IEEE Draft Standard for Local and Metropolitan Area Networks: Overview and Architecture". In: *IEEE P802-REV/D1.7* (2014), pp. 1–68.

[63] Yong Li and Min Chen. "Software-defined network function virtualization: A survey". In: *IEEE Access* 3 (2015), pp. 2542–2553.

[64] N. G. Nayak, F. Durr, and K. Rothermel. "Software-defined environment for reconfigurable manufacturing systems". In: *2015 5th International Conference on the Internet of Things (IOT)*. 2015, pp. 122–129. DOI: 10.1109/IOT.2015.7356556.

[65] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic". In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)* (2016), pp. 1–57. DOI: 10.1109/IEEESTD.2016.8613095.

[66] Frank Dürr and Naresh Ganesh Nayak. "No-wait packet scheduling for IEEE time-sensitive networks (TSN)". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 2016, pp. 203–212.

[67] Mohamad Kenan Al-Hares et al. "Modeling time aware shaping in an Ethernet fronthaul". In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6.

[68] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks". In: *IEEE Std 802.1AS-2011* (2011), pp. 1–292. DOI: 10.1109/IEEESTD.2011.5741898.

[69] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". In: *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (2008), pp. 1–300. DOI: 10.1109/IEEESTD.2008.4579760.

[70] Parviz Kermani and Leonard Kleinrock. "Virtual cut-through: A new computer communication switching technique". In: *Computer Networks (1976)* 3.4 (1979), pp. 267–286.

[71] S. Thangamuthu et al. "Analysis of Ethernet-switch traffic shapers for in-vehicle networking applications". In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2015, pp. 55–60. DOI: 10.7873/DATE.2015.0045.

[72] Maël Guiraud, Olivier Marcé, and Brice Leclerc. "An Experimental Platform for Hyper TSN Flows Scheduling and Control Automation". In: (). To be published.

[73] Brice Leclerc et al. *Optical packet switching based on traffic properties*. US Patent 10,701,466. 2020.