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

Chapter 1

Scheduling Unsynchronized Periodic Datagrams with a single Buffer

This chapter is taken from a published paper [1] and its extended version [2].

1.1 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.

1.1.1 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 1. *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. □

1.1.2 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 1.1.6. 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 **??**).

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.

1.1.3 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

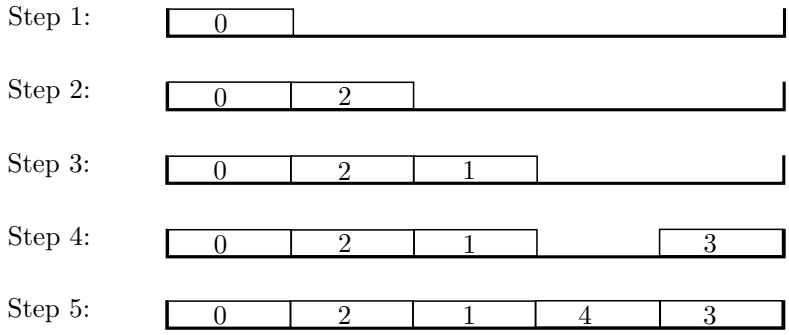

Figure 1.1 – A run of `Greedy Deadline` with $P = 20, \tau = 4$.

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 1.1. 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 $O(n)$.

1.1.4 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 $O(n^2 \log(n))$ [3] 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 [4]. 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 [3].

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 1.1.6), 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 **??**, 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.

1.1.5 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 1. 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 1.1.4.

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 1 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 2. PALL ∈ 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 1, 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 $E_{S,\sigma,\sigma'}$.

□

1.1.6 Experimental Evaluation

Evaluating the Necessary Margin We set the number of routes to 8 to make comparisons with the results of Chapter **??** 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 **??**.

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 1.2, 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 **??** 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.

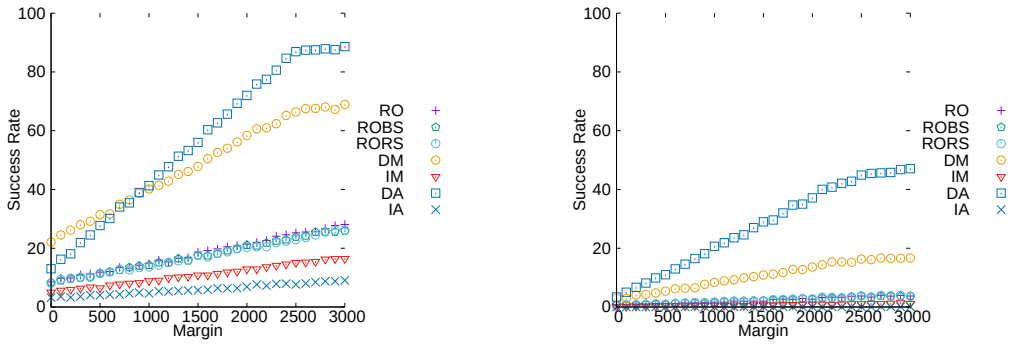

Figure 1.2 – Success rate of different sending orders, left 80% load, right 95% load.

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 1.3 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 1.3 – 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 **??**). 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 1.4, 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.

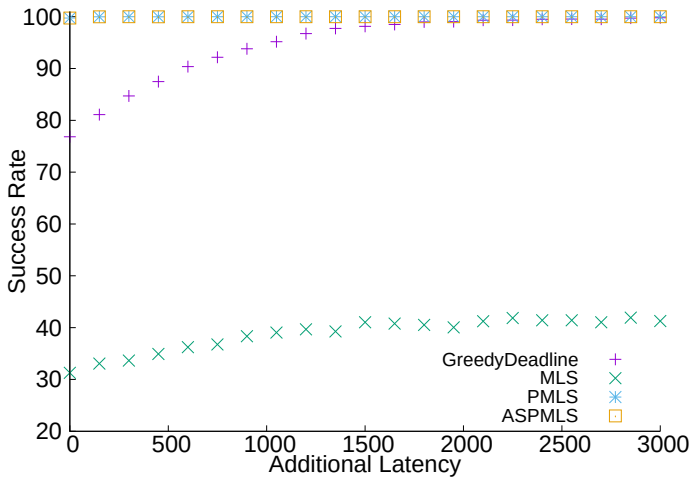

Figure 1.4 – Success rate of four algorithms solving PALL, 95% load

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 1.5 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 1.5 – 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 1.5, 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 1.6 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 1.6 – 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 1, 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 1.7 and 1.8 show the probability of success of PMLS over 10,000 instances as a function of the margin. In Figure 1.7 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 1.8, 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.

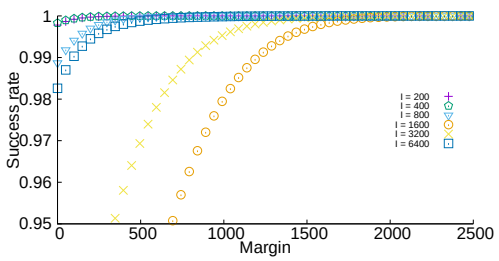

Figure 1.7 – Success rate of `PMLS`, with length of arcs drawn in $[I]$

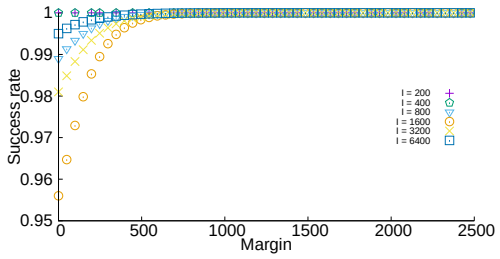

Figure 1.8 – Success rate of `PMLS`, with length of arcs drawn in $[I]$ and $[P/2, P/2 + I[$

1.1.7 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.

In Figure 1.9, we represent the probability of success of statistical multiplexing and `PMLS`

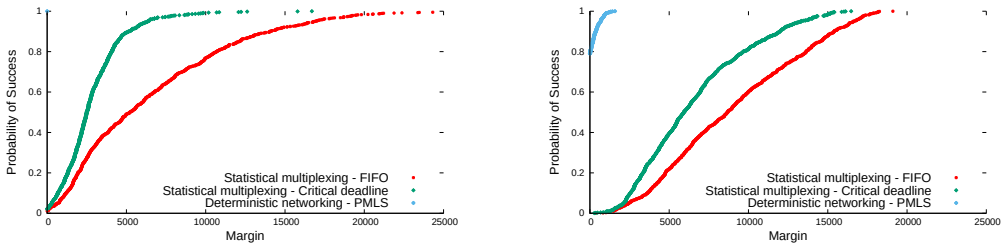

Figure 1.9 – Probability of success of statistical multiplexing and PMLS for several margins on random topologies when the size of the routes are distributed either on P (left) or on a small range of values (right).

for different margins. The success rates are computed from 10,000 star routed networks for each margin. On the left part of Figure 1.9, 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.

Bibliography

[1] 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.

[2] 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.

[3] Barbara Simons. "A fast algorithm for single processor scheduling". In: *Foundations of Computer Science, 1978., 19th Annual Symposium on*. IEEE. 1978, pp. 246–252.

[4] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. "Complexity of machine scheduling problems". In: *Annals of discrete mathematics* 1 (1977), pp. 343–362.