Scheduling algorithm to avoid contention in meshed networks

Dominique Barth[1], **Maël Guiraud**[1,2], **Brice Leclerc**[2], **Olivier Marcé**[2], and **Yann Strozecki**[1]

[1]David Laboratory, UVSQ
[2]Nokia Bell Labs France

June 17, 2020

1 Introduction

TODO: doner le contexte CRAN, puis l'objectif d'optimisation de latence. Dire qu'on a déjà fait désynchronisé mais que dans les architectures actuelles il faut plutot faire synchronisé. Comparer à nos travaux antérieurs et à d'autres qu'on a déjà cité (voir les papiers précédents, surtout les citations du dernier).

2 Scheduling of Periodic Datagrams over a Network: the problem SPALL

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

2.1 Modeling the Network and its Contention Points

We study a communication network with pairs of source-destination nodes between which messages are sent periodically. The routing between each pair of such nodes is given. The network is represented by a directed acyclic multigraph $G = (V, A)$. The set of vertices is composed of three disjoint subsets: $\mathcal{S}$ the set of sources of the messages, $\mathcal{D}$ the set of destination of the messages, and $\mathcal{C}$ the set of contention points in the network. Indeed, some links of the network are shared between several pairs of source-destination nodes. A contention point represents the beginning of a shared link, i.e. the physical node of the network which sends the messages into the link. An arc in G can represent several physical links or nodes, which do not induce contention points. Each arc (u, v) in A is labeled by an integer weight $\omega(u, v)$ which represents the time elapsed between the sending time of the message in u and the reception time of this message in v using this arc.

A **route** r in G is a directed path, that is, a sequence of adjacent vertices $u_1, \ldots, u_l$, with $(u_i, u_{i+1}) \in A$. The **weight of a vertex** u_i in a route $r = (u_0, \ldots, u_l)$ is defined by $\lambda(u_i, r) = \sum\limits_{1 \leq j < i} \omega(u_j, u_{j+1})$. It is the number of tics 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 $\lambda(r) = \lambda(u_l, r)$. We denote by $\mathcal{R}$ a set of routes of the graph G, the pair $(G, \mathcal{R})$ is called a **routed network** and represents our telecommunication network.
Let $r \in \mathcal{R}$, with $r = (u_0, u_1, \ldots, u_l)$, then we say that u_i is of **contention level** i for the route r, and we denote it by $cl(u_i, r) = i$. The contention level of a node u is the maximum of its contention level over all routes going through itself: $cl(u) = \max\limits_{r \in \mathcal{R} \text{ and } u \in r} cl(u, r)$.

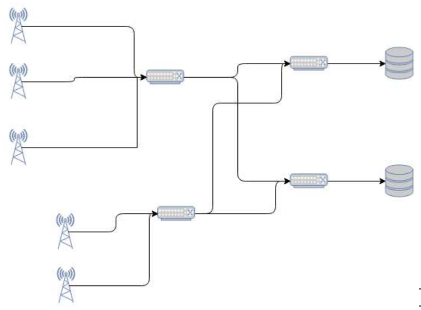

Figure 1: A C-RAN fronthaul network

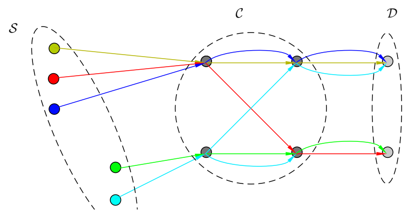

Figure 2: The corresponding routed network

The **contention depth** of a routed network $(G, \mathcal{R})$ is equal to the maximum of the contention level over all vertices. It is the number of contention points on the longest route of the network. In all the article, $(G, \mathcal{R})$ is the routed network, $\mathcal{C}$ is the set of its contention vertices, n denotes $|\mathcal{R}|$ the number of routes and d is the contention depth.

2.2 Dynamic of Datagrams Transmissions

In this article, 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. In practice, the size of an atomic data is usually 64B, the speed of the links is considered to be 10Gbps, and the length of a tic is thus TODO: rajouter durée d'un tic.

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 paper, 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. Once a datagram has been emitted, it cannot be fragmented during its travel in the network. We assume that the first datagram sent on each route is ready to be sent at time 0.

Let $r = (u_0, \dots, u_l)$ be a route. In order to avoid contention, it is possible to buffer datagrams in contention points. The function A, called an **assignment**, associates an integer value, greater or equal to 0, to each couple (route,vertex) of the routed network $(G, \mathcal{R})$. Those integers represent the buffering time of the datagrams in the nodes of the network: a datagram of route r waits $A(r, u)$ tics in the buffer of vertex u.

The **arrival time** of a datagram in vertex u_i of r, is the first time at which the datagram sent on r reaches u_i, and is defined by $t(r, u_i) = \lambda(u_i, r) + \sum_{k=0}^{i-1} A(r, u_k)$. 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 in vertex u_i in r, is the first time at which the datagram is sent by u_i. It is defined by $s(r, u_i) = t(r, u_i) + A(r, u_i)$. This is the arrival time of the datagram plus the buffer time fixed by A.

If u_l is the last vertex of the route r, the transmission time of the datagram on r is denoted by $TR(A, r)$ and is equal to $t(r, u_l)$. We define the **transmission time** of an assignment A as $TR(A) = \max_{r \in \mathcal{R}} TR(A, r)$. This is the time elapsed before the reception of the beginning of the last datagram.

2.3 From C-RAN Networks to Contention Graphs

This study is motivated by a problem from Cloud RAN: messages containing local radio information are sent by a radio antena (called RRH) to a BaseBand Unit (BBU) in a datacenter through a fronthaul network. Then, the datacenter sends back an answer to the RRH. The routed network presented in this article is used to modelize both ways of this communication, from an RRH to the corresponding BBU and back to the RRH. Hence, each RRH corresponds to the first and the last vertex of a route. The BBU does not appear explicitely in the routed network, the message arrives at the BBU somewhere on an arc of the route. Once a message arrives to a BBU, the answer is computed and sent back in the network. The computation time is usually the same for all messages; it is encoded into the arc by adding it to the physical delay of transmission over the arc.

In this article, we consider that the links of the fronthaul network are full duplex, that is the messages can go through the links in both ways, without interacting. Usually the route from the RRH to the BBU is the same (with the orientation reversed) as the route from the BBU to the RRH. This property does not need to be enforced in our theoritical modeling, but it matches real fronthaul network and we will use such examples for our experiments.

Several BBUs are 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 the routes uses the same node in the way back, it is not considerd as a contention point. In the case the length of the links into the datacenter are different, the contention point before the datacenter is also a contention point at the exit of the datacenter and the routes are symmetrical.

TODO: faire un dessin pour illustrer tout ca

TODO: Parler de la propriété de routage cohérent, ça implique qu'on obtient bien des DAG.

2.4 Periodic Emission of Datagrams

The process we modelize in this article is *periodic*: for each period of P tics, a datagram is ready to be sent from each source node in the network at time zero of the period. The process is assumed to be infinite, since it should work for an arbitrary number of periods. We chose to always use the same buffering in all periods, for simplicity of implementation in real networks but also to make our problem more tractable from a theoritical perspective. In other words, at the same time of two different periods, all messages are at the same position in the network: the assignments we build 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. Let us call $[r,u]_{P,\tau}$ the set of tics used by a datagram on r at vertex u in a period P, that is $[r,u]_{P,\tau} = \{s(r,u) + i \mod P \mid 0 \leq i < \tau\}$.

Let us consider two routes r_1 and r_2, they have a **collision** at the contention point u if and only if $[r_1,u]_{P,\tau} \cap [r_2,u]_{P,\tau} \neq \emptyset$.

An assignment A of a routed network $(G,\mathcal{R})$ is said to be **valid** if *no pair of routes has a collision*. The validity of an assignment depends on P the period and τ the size of the messages, but in this article these values are fixed and we do not recall them (contrarily to previous work [1]). Note that the period P, as well as the size of a message τ is fixed in our $C-RAN$ application, but not the buffering policy. Hence, the aim of our work is to find a valid assignment which minimizes the latency of transmissions over the network, that is $TR(A)$.

Synchronized Periodic Assignment for Low Latency (SPALL)
Input: Symmetric routed network $(G,\mathcal{R})$, period P, datagram size τ.
Question: find A which minimizes $TR(A)$.

Several parameters are important for the study of SPALL. The algorithms we present have a complexity depending on n the number of routes and d the contention level, but not on P or G. Given a contention point u, let $\mathcal{R}_u \subseteq \mathcal{R}$ be the set of routes which contain u. We define the load of u to be $|\mathcal{R}_u|\tau/P$, it measures the percentage of the bandwidth used by datagrams going through u. The load of $(G,\mathcal{R})$ is the maximum of the load of u, for all contention vertices in G. This parameter is important for theoritical study of problems simpler than SPALL [], but is also critical for the performance of algorithms presented in this article.

2.5 Contention Depth and SPALL

Contention Depth One Each contention vertex of a routed network of contention depth one induces a connected component. Problem SPALL can be independently solved on each connected componentes, hence the case with a single contention point is equivalent. It has already been defined as a useful subproblem in [] under the name BRA TODO: mettre le bon style.

TODO: Donner la version non périodique et lui doner un nom et donner les deux résultats algorithmique-> simmons et notre adaptation pour Bra

If the messages are allowed to have different size, then the problem is known to be NP-hard for the non periodic variant. It also implies hardness for BRA, by taking a large enough period so that it is equivalent to the non periodic case.

We can further restrict the problem, by asking that all weigths of arcs from sources to the contention point are the same or equivalently all weigths from the contention point to the destionation are the same. Then, it is enough to consider all datagrams and their time of avalabality in the contention point ... TODO: décrire proprement l'algo greedy GD qui marche ici, puisqu'on s'en sert pas mal après

Star Shaped Network The most simple 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 center). This topology, called star shaped network in previous work, is common in real networks, and a non synchronized version of SPALL has been studied in this case [?]. The problem SPALL on star shaped network is NP-hard, since it is a periodic version of the two flow shop problem studied in []. TODO: mettre un peu plus de détails et pourquoi la périodicité ne change rien

Contention Depth Two and More However, star shaped networks may not be the most general topology with regard to problem SPALL. TODO: Que peut-on dire des tpologies de profondeur 2 en général ? Introduire les réseaux plus généraux de contention depth 3. Dire comment ce qu'on veut modéliser est un cas particulier et lequel (profondeur 2 dans le réseau, même distance de tous les datacenters dans les BBUs). Donner un exemple illustré.

3 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 SPALL.

We explain in this section how to represent most assignment 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 minimimal assignments have a compact representation, which implies that we do not need to consider assignment without a compact representation when solving SPALL. It allows to design an FPT algorithm for SPALL by going through all compact representations, but also to design good polynomial time heuristics using taboo search or simulated annealing, since one can easily define the neighborood of a compact representation.

Definition 1 (Compact assignment)**.** *Let $(G, \mathcal{R})$ be a routed network. A compact assignment CA is a function which maps to each contention point u in G a pair (O_u, S_u), where O_u is an order on R_u and S_u is a subset of R_u.*

3.1 From Valid Assignments to Compact Representation

We first define a method to obtain a compact assignment from some valid assignment A, that we call the compact representation of A and that we denote by $CR(A)$. Assume all routes are indexed by an integer in $[n]$ and let u be a contention point of G. We also assume that for all vertices u, there is a route $r \in \mathcal{R}_u$ such that $A(r_i, u) = 0$.

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 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 $ns(r_0, r, u) < nt(r_0, r, u)$. Imagine the time is cut into periods $[t(r_0, u) + iP, t(r_0, u) + (i+1)P[$, with $i \in \mathbb{N}$. Intuitively, S_u represents the set of routes with a datagram going through u in the period *after* the one they have been available in.

Figure 3 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, normalized by arrival time $ns(r_0, r_i, u)$.

Note that for $CR(A)$ to be defined, we need that, on each contention point, at least a 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 $TR(A)$.

Lemma 1. *Let A be a valid non canonical assignment, then there is a valid canonical assignment A' such that $A' \preceq A$.*

Proof. Consider a vertex u of contention level 1, such that for all $r \in \mathcal{R}_u$, $A(r, u) > 0$. Let us define m as the minimimum 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

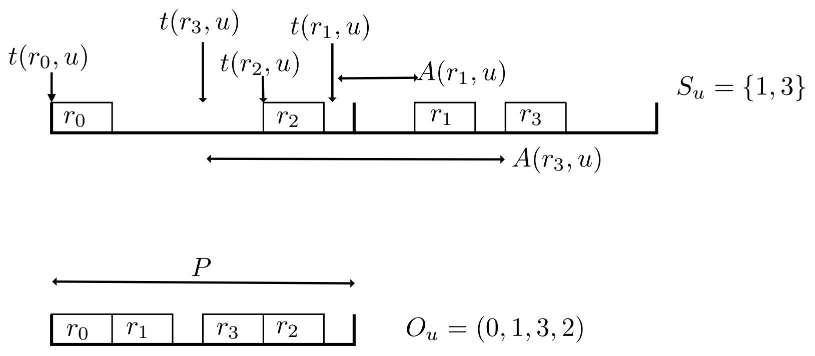

Figure 3: A compact representation of an assignment in which $O_u = (0, 1, 3, 2)$ and $S_u = \{1, 3\}$

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 levels 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 level. Since, the transformation applied at some contention level do not change A' for smaller contention levels, a trivial induction proves that A' is valid, canonical and that $A' \preceq A$. □

3.2 From a Compact Assignment to its Realization

We now explain how to transform a compact representation into a canonical assignment. Moreover, we will 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). We define inductively how to build an assignment $Real(CA)$ from CA, that we call the realization of CA. Note that we can fail to build a realization from a compact representation, then $Real(CA)$ is undefined and we say that CA is not realizable. In the next paragraphs, we build an assignment A by setting the buffering time of the routes in the order (O_u). If the construction suceeds, we set $Real(CA) = A$

Let say that the order O_u is $(r_1, \ldots, r_l)$. We fix $A(r_1, u)$ 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. Now the first datagram of the period is fixed, we use it to consider normalized arrival times and normalized sending times. Assume that $A(r_i, u)$ have been set for

$i \leq l$, we explain how to set $A(r_{i+1}, u)$. If $r_{i+1} \notin S_u$, then we fix $A(r_{i+1}, u)$ 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 we fix $A(r_{i+1}, u)$ 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 4 shows how an assignment *Real*(CA) is built from a compact assignment CA on a single contention vertex 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 $ns(r_2, r_3, u) = ns(r_2, r_0, u) + \tau$.

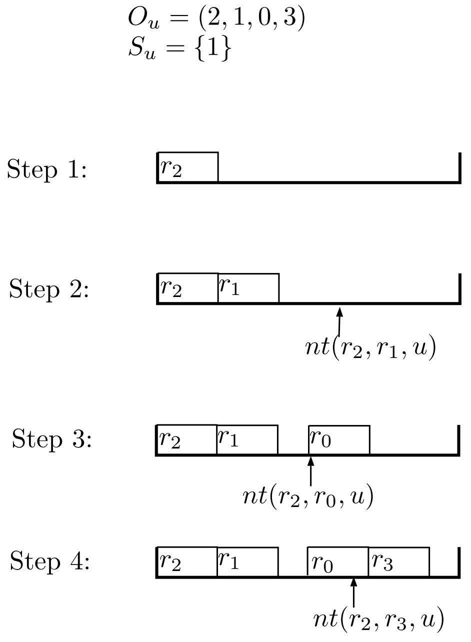

Figure 4: Inductive contruction of *Real*(CA) from CA on a single contention point u.

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

Lemma 2. *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 level, hence the time spent computing buffers for all vertices of a contention level is in $O(n)$ and for the whole graph it is in $O(nd)$.

To prove that there are 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 route of O_u and O'_u coincide. This notion of canonicity is useful because 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 in its equivalence class for $\prec$.

Lemma 3. *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. □

To solve SPALL, we need to find an assignment for which $TR(A)$ is minimal. To do so, because of the previous Lemmas, it is enough to enumerate all canonical compact representations, to compute their realization and the corresponding transmission time.

Theorem 4. *For routed networks of fixed contention depth d, the problem* SPALL *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 SPALL is the following: all compact assignments CA are generated, for each of them $TR(Real(CA))$ is computed in time $O(nd)$ by Lemma 2 and we keep the compact assignment for which this value is minimal. Because of Lemma 3, to compute the minimum of $TR(A)$, it is enough to compute the minimum of $TR(Real(CA))$.

Now, we need to evaluate the number of compact assignments. On a single contention point u with $s = |\mathcal{R}_u|$ 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 $\mathcal{R}_u$. On a given contention level consisting in the vertices $\{u_1, \dots, u_l\}$, with $s_i = |\mathcal{R}_{u_i}|$, there are $\prod_{1 \le i \le l} s_i!2^{s_i}$ compact assignments. On a given contention level, all routes use at most 1 vertex, hence $\sum_{1 \le i \le l} s_i \le n$. Since $\prod_{1 \le i \le l} s_i! \le (\sum_{1 \le i \le l} s_i)!$, we have $\prod_{1 \le i \le l} s_i!2^{s_i} \le n!2^n$. There are d contention levels, thus we have at most $(n!2^n)^d$ compact assignments which proves the theorem. $\square$

Note that for the vertices of the last contention level, compact assignments can be considered independently, since they do not interact. Hence, if the last level contains the vertices $\{u_1, \dots, u_l\}$, we only need to consider $(n!2^n)^{d-1}(\sum_{1 \le i \le l} s_i!2^{s_i})$ assignments. This makes a large difference in our target application and the experiments presented in this paper, since in this context d is two or three and the s_i's are pretty balanced.

4 Greedy Algorithms

4.1 Random generation of routed network

In this paper, we focus on Cloud-RAN networks. For the results presented here, we consider that the networksTODO: dessin have a meshed shape. Also, we consider that each datacenter is preceded by a switch in the network, and that the lenght of the routes comming to this datacenter is the same between this switch and each BBU contained in the datacenter. Also, we consider that the contention level of the networks studied here is 3. Indeed, there is two contention point in the way forward for the messages, and one in the way backward, since once the datagrams have been scheduled to go in the datacenter, they go out of it in the exact same order because the size of the arcs is the same for all routes of a same datacenter.

To generate a routed network, several synthetic datas might be fixed: The load of the network, the number of routes, the length of the routes, the distribution of the length of the routes, and the number of contention points. We

Range	0 and $\tau/3$	0 and P	$P - 0.1 \times P$ and P
Branch and bound	4273	222	2704
GNS	12177	10628	9648

Figure 5: Difference between the optimal solution of the greedy algorithms, for different kind of instances generated.

first want to determine the impact in terms of computation time and quality of the results of those synthetic datas on the algorithms studied. In order to lighten the number of experiences presented here, we consider that the number of datacenter (i.e. the number of contention points of level 2) is set to 2, and the number of contention points of level 1 (and 3, since the network is symetric) is set to 4. The results are not significantly impacted if we change those values. Also, in a C-RAN context, the number of route is low. In the network we study, there is 8 routes on the graph. This kind of graphs with few routes allows us to use the branch and boud algorithm as a lower bound for evaluating the performances of the other algorithms. We will study the impact of the numer of routes in the graph at the end of this section.

Once all those synthetic datas are set, we focus on the length of the arcs in the grap. We look at three ways to draw some values for the length of the arcs: uniformly between 0 and $\tau/3$ (with τ the size of a datagram), uniformly between 0 and P (the size of the period), and uniformly between $P - 0.1 \times P$ and P. TODO: expliquer pourquoi ces valeurs

Figure 5 shows the average additional latency needed by the branch and bound and the greedy algorithm that we call GNS presented in next section (used to intialize the local search algorithms) for the different ways to draw the length of the arcs. Those values are draw for 100 instances of each kind.

The objective is to find the instances in which a solution given can be the most improoved by the local search heuristics. As observed, when drawing the size of the arcs betwenn 0 and P, the lower bound (found by the branch and bound algorithm) is clearly lower than with the other kinds of instances while the additional latency needed by GNS remains in the same order of magnitude.

Also, we investigated the impact of the load on the quality of the results. When the load is increased, the performances of the local search algorithm does not changes significantly. Otherwise, we choose to fix the load to 80%, which is, in practice, an already high load.

Given a routed network, the local search algorithms explore all compact assignments in order to find a compact assignment CA such that $TR(Real(CA))$ is as small as possible. A first compact assignment is needed to intialize any local search algorithm. We present in this section three greedy algorithms which try to build canonical valid assigments, which can then turned into the corresponding compact representation by the CR function.

4.2 Greedy Deadline

We first present an intuitive algorithm, which is the natural approach in a context witout *periodicity*. The contention vertices are sorted by contention level, and the contention levels are computed in ascending order. Each assignment on contention vertices of the same contention level is computed independently. The *Greedy Deadline* algorithm or GD consists in selecting all routes of arrival time less than the current time and among them, select the one with the worst transmission time. If no route are available at the current time, select the one with the smallest arrival time.

GD works precisely as follow. For a vertex u, first, select the route r such that the arrival time $t(r,u)$ is minimal. and fix $A(r,u)=0$. Consider that some datagrams have 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', we compute the value $2 \times \lambda(u,r) - t(r',u)$ 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 satifies $t(r',u) < s(r,u)+\tau$, 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.

Algorithm 1 Greedy deadline

```
Require: $(G, \mathcal{R})$, $P$, $\tau$
Ensure: An assignment $A(G, \mathcal{R})$, or FAILS
  $budget[|\mathcal{R}|]$ integer table.
  for all route $r$ in $\mathcal{R}$ do
    $budget[r] \leftarrow 2 \times \lambda(u, r) - t(r, u)$
  end for
  Let $first$ be the route such that $t(first, u)$ is minimal
  $A(first, u) \leftarrow 0$
  $offset \leftarrow t(first, u) + \tau$
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \{first\}$
  while $\mathcal{R} \neq \emptyset$ do
    if $\exists i \in \mathcal{R}, t(i, u) \leq offset$ then
      Choose $i$ with the lowest $budget[i]$
      $A(i, u) \leftarrow offset - t(i, u)$
      $offset \leftarrow offset + \tau$
    else
      Choose $i \in \mathcal{R}$ with the lowest $t(i, u)$
      $A(i, u) \leftarrow 0$
      $offset \leftarrow t(i, u) + \tau$
    end if
    if $A$ is not valid then
      if $add\_delay \leftarrow FIRST\_VALID(A, P, i)$ then
        $A(i, u) \leftarrow A(i, u) + add\_delay$
      else
        Return FAIL
      end if
    end if
  end while
  Return SUCCESS
```

TODO: ajouter la routine FIRST VALID qui donne la premiere position a laquelle on peut placer le message a partir de l'offset donné

4.3 Greedy Normalized

A variant of the Greedy deadline assignment algorithm is as follows: select for first datagram the one with minimal $t(r, u)$, then select the datagrams by lowest normalised arrival times instead of arrival times. We will show that in practice, this algorithm performs better than GD.

However, GD and GN may fail to give a compact assignment for all routed networks. The way we select departure times for the routes can make sequences of times of size less than τ unusable for unscheduled routes. If too much time is wasted, the algorithms will fail while there is always a solution when the load is less or equal to 1. Each datagram wastes at most $2\tau - 1$ tics in the period,

Load / Sucess	70%	80%	90%	100%
GD	100%	95.1%	56.3%	11.2%
GN	99.9%	95.5%	68.3%	0%

Figure 6: Success rate of the greedy algorithms for different loads

hence by a simple pigeonhole argument, all routes can be scheduled when the load is less than 0.5 by any greedy algorithm considering all departure times.

TODO: donner un exemple ou ça fail pour chacun des deux algorithmes

4.4 Greedy Packed

A compact assignment is needed to initialize the local search algorithms presented in the next section. Hence, we propose the following greedy algorithm that is guaranteed to find an assignment, even if the transmission time may be worse on average. The contention vertices are still managed level by level. For a vertex u, we explain how to determine the pair (O_u, S_u). First, the datagram d_0 is fixed and sent without buffer. This is the datagrams with the lowest arrival time. This datagram is also the reference to compute the normalized arrival and sending times. Once i datagrams are fixed, if serveral datagrams have a normalized arrival time lower or equal than $ns(d_0, d_i, u) + \tau$, the datagram $i + 1$ is selected in the same way as in GN, that is, choose the datagram on the route r with the lowest value of $2 \times \lambda(u, r) - t(r, u)$. If no datagram can be selected, then the datagram on the route r which maximize the value $2 \times \lambda(u, r) - t(r, u) - (ns(d_0, d_r, u) + \tau + P - nt(d_0, d_r, u)$ is selected to be the $i + 1^{th}$ in the order O_u and $S_u = S_u + (i + 1)$. In other word, select the datagram which is the less impacted by the additional latency needed to send the datagram without gap in the period.

4.5 Success Rate and Performance of the Greedy Algorithms

TODO: Mettre les settings expérimentaux ici, sans s'attarder sur leur justification pour l'instant. Définir la additionnal latency qui va servir de mesure de performance.

We want to compare the succes rate and the performance of the different algorithms presented here. First, we consider the impact of the load of the network on the succes rate of the three greedy algorithms. The more we increase the load, the less margin the algorithms will have to schedule the datagrams. We have seen that all greedy algorithms succeeds when the load is less than 0.5 and that GP will always succeed. Figure 6 shows the success rate of Greedy Deadline and Greedy Normalized on 1000 random instances for loads from 70% to 100%.

Greedy Deadline fails less than Greedy Normalized on highly loaded networks, while Greedy Normalized seems more robust on loads between 80% and 90%. We now want to compare the performance of those algorithms. Figure 7 shows the additional latency induced by the assignments given by the algorithms, when there is one. As expected, the additional latency increases when the load increases and Greedy Packed, that trades additional latency for success rate, performs worst than GD and GN 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 algorithm almost always find the same assignment (or fail). On vertices of small load, the constraint of packing the datagram imposed by GD worsen the latency.

We propose improved version of GD and GN that always find a solution. For each contention point, we first try GD (or GN), and if the algorithm fails, we apply GP. Let us call Hybrid Greedy Deadline (HGD) and Hybrid Greedy Normalized (HGN) those two algorithms. Figure 8 shows the performances of HGD, HGS, and GP on 1000 routed networks with a load of 90%.

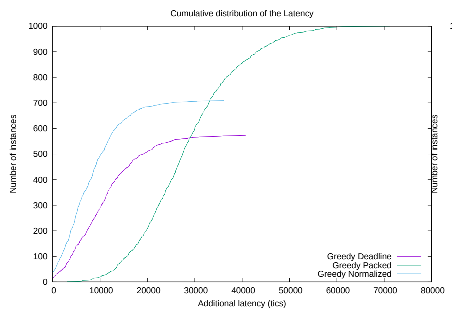

Figure 7: Performance of GD, GN and GP over 1000 instances on graphs with 90% load.

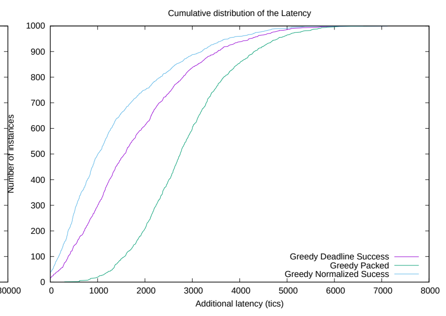

Figure 8: Performance of the updated greedy algorithms that always gives an assignment

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

5 Local Search Heuristics

The number of compact representations grows extremely quickly with n. Hence, to find one which minimizes $TR(A)$, we propose several classical local search algorithm: hill climbing, tabu search and simulated annealing. These methods works as long as a relevant notion of neighborood of a solution is proposed. The neighborood relation must satisfy two properties: it must be quick to compute (hence not to large) and the implicit graph of solutions defined by the neighborood relation should be connected. We now propose a simple neighborood relation over compact assignments.

Let u be a contention vertex of a network, and let CA be a compact assignment for this network, which associates the pair (O_u, S_u) to u. Let $r \in \mathcal{R}_u$, the *r-neighborood* of (O_u, S_u) is the set of pairs (O, S) such that:

1. $O_u = O$ or $O_u = (r_1, \dots, r_l)$ with $r_i = r$ and $O = (r_1, \dots, r_{i-2}, r_i, r_{i-1}, \dots, r_l)$
2. $S_u = S \cup \{r\}$ or $S_u = S \setminus \{r\}$

Informally, a compact representation is in the r-neighborood 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 from the set. Remark that the r-neighborood of any pair (O_u, S_u) has at most 4 members (it can be 2 when the route r is in first position and cannot be exchanged with the previous one).

The *r-neighborood* 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-neighborood of (O_u, S_u). Finally, the *neighborood* of an assignment CA is the union for all $r \in \mathcal{R}$ of the r-neighboroods of CA.

Let us denote by $k_1, \dots, k_n$ the number of contention vertices on the n routes of a routed network $(G, \mathcal{R})$. Then, a compact representation has at most $\sum_{i=1}^{n} 4^{k_i}$ neighbors. Since we always work with networks 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 neighborood 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. All algorithms presented in this section will traverse the graph defined by the neighborood relation.

TODO: un nom pour ce graphe ?

Proposition 5. *The graph defined over realizable compact solutions by the neighborood relation is connected.*

Proof. Sketch (should prove a fact before on solutions with the same order but included sets)

Consider two realizable compact solutions (O_u, S_u) and (O'_u, S'_u). First set S_u to be equal to $\mathcal{R}_u$. This can be done by adding repeateadly elements in $\bar{S_u}$. This always leads to realizable compact assignment, since the constraints becomes weaker. (to check)

Then, when $S_u = \mathcal{R}_u$, all orders yield a realizable compact assignment. Hence there is a path from $(O_u, \mathcal{R}_u)$ to $(O'_u, \mathcal{R}_u)$. Finally, we can remove elements from $\mathcal{R}_u$ to obtain S'_u, since (O'_u, S'_u) is realizable, the intermediate solutions are also realizable (less constraints) $\square$

5.1 Hill Climbing

The most simple local search heuristic is hill climbing. This algorithm starts from a compact assignment CA, explore the entire neighborhood of CA, and select 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 algorithm stops and returns $Real(CA)$ which is a local minimum.

The quality of hill climbing depends of the the intial compact assignment. A first natural intialisation is to consider the compact representation $CR(A)$ of the assignment A given by the Greedy Normalized Success algorithm. Also, a random compact assignment can be taken. Since a compact assignment does not always give a valid assignment, the last idea is to execetue several hill climbing starting from a random compact assignment, and to return the best assignment given.

Figure 9 shows the difference between initalizing the hill climbing with the greedy normalized success algorithm, or with one or several random compact representation. Those resultas are drawn on 100 random instances, with 80% of load. Initializing the hill climbing with 100 randoms compact representation

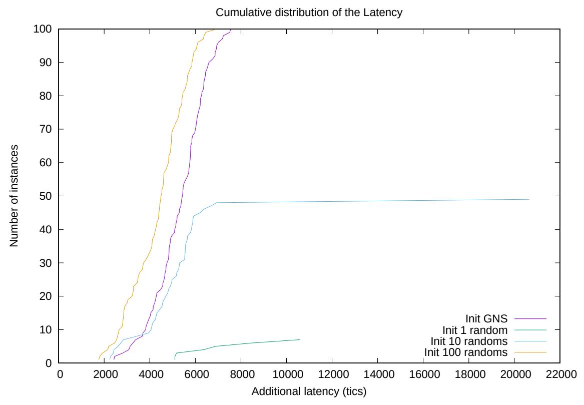

Figure 9: Additional latency needed to find a solution for hill climbing, intialized with GNS, 1,10 or 100 random compact representation.

seems to give better results. Nevertheless remember that the set of compact representation does not always give a valid assignment. Drawing some random compact representation can be a limited option when the set of compact representation grows, because the chance of drawing a non valid, or bad random compact representation grows too.

Figures 10 and 11 shows the success rate of the hill climbing when drawing 1,10 or 100 random compact representations. Each result is computed on 1000 random instances.

Load	80%	90%	100%
GNS	100%	100%	100%
1 rand	7%	4%	6%
10 rand	49%	42%	28%
100 rand	100%	97%	92%

Figure 10: Success rate of hill climbing with different initialization, varying the load.

Nb routes	8	10	12
GNS	100%	100%	100%
1 rand	7%	2%	1%
10 rand	49%	23%	5%
100 rand	100%	86%	56%

Figure 11: Success rate of hill climbing with different initialization, varying the number of routes on the graph.

Those experiences shows that, even if initializing the hill climbing with 100 random instances performs better when the number of routes and the load are low, this initialization is not robust when the load of the number of routes increases. Furthermore, the computation time induced by executing the hill climbing on a lot of random compact representation instead of only one (for GNS) makes it less effective.

TODO: rajouter un tableau sur le nombre de pas de descente

5.2 Tabu Search

Tabu search is a variation on hill climbing using memory. We also select the compact assignment CA' which minimizes $TR(CA')$, even when $TR(CA') < TR(CA)$ is not satisfied. To avoid loop in local minimum, we use a memory of the last N solutions explored and we forbid to visit them again. This algorithm can still loop on a cycle larger than N, hence we must fix some integer M and stop the algorithm after M steps.

One can change the size N and M in tabu search -> explore how to fix these parameters.

After reaching a local optimum, tabu search continues it execution untill it has executed a given number of steps. Thus, we investigate the average number of steps needed by the tabu search to find the assignment it returns. Remember that, to each step, the search explores the entire neighborhood of a solution. The computation of a large number of steps is then long.

Tabu search has a memory, it remembers the compact assignment for which it explored the neighborhood for the x last steps in order to not visit them again. Varying the value of x can improve or worsen the performance of the tabu search. Figure 12 shows cumulative distribution of the additional latency needed by tabu search with different size of memory. The number of steps computed by the tabu search is 1000. It appears that, the more steps tabu search remember, the better the assignment given is. In order to understand what happends during the tabu search, we investigate the distribution of the step at which the tabu search finds its best solution. For 10 steps of memory, tabu search finds the best assignment in average in 3.8 steps, with a memory of 100 steps, the average step that gives the best assignment is 18.1, and for 1000 steps, tabu search needs in average 84.3 steps to find the best solution. Also, it appears that, in most of 60% of the cases, tabu search finds it's best

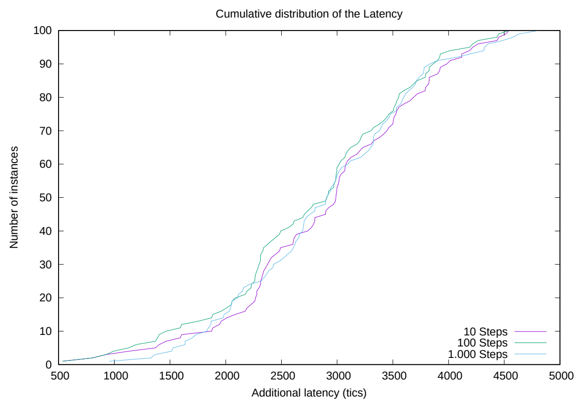

Figure 12: Difference between the tabu search whith a memory of 10,100 or 1000 steps

assignment before 10 steps, but some computation finds a better assignment up to the 900^{th} step. This means that running the tabu search longer could improove the quality of the result. Nevertheless, this the computation is really expensive, using simulated annealing is more interesting.

5.3 Simulated annealing

A simulated annealing search works as follow. A $t$emperature is set. Considering a compact assignment CA, we draw in its neighborhood a compact assignment CA'. Then, considering the temperature, $TR(Real(CA))$ and $TR(Real(CA'))$, the compact assignment CA' is or accepeted or rejected. If it is accepted, the neighborhood of CA' is then explored. During the simulated annealing, the temperature decrease. The lower is the temperature, the lower are the chance to accept a compact assignment that worsen the solution.

Simulated annealing needs two parameters to run. A temperature, and a number of steps. The temperature must be fixed in order to accept to worsen the solution at the begining of the execution, must decrease slowly. When the temperature is low, the simulated annealing rejects bad compact representations. Accepting or rejecting a solution is a stochastic process. Consider CA, the best compact assignment found during the exection of the simulated annealing. An compact assignment CA' is accepted or rejected with the probability $e^{-\frac{TR(REAL(CA'))-TR(REAL(CA))}{t}}$, where t is the temperature. This means that $\frac{TR(REAL(CA'))-TR(REAL(CA))}{t}$ must be close to 0 at the begining, and must approaches infinity at the end of the execution.

5.4 Branch and Bound

définition des différentes coupes.
Montrer avec des courbes (nombre de représentation générées et pourcentage de minimales en fonction du nombre de routes) que ces coupes sont très efficace, elles permettent de générer uniquement des représentations compactes canoniques, et presque aucunes dont la réalisation n'est pas minimale. très peu de représentation compacte en

5.5 Results comparison

6 Long routes

7 Conclusion

C'est vraiment génial comme travail, on va tuer le game du DETNET.

References

[1] D. Barth, M. Guiraud, B. Leclerc, O. Marce, and Y. Strozecki, "Deterministic scheduling of periodic messages for cloud RAN," in *2018 25th International Conference on Telecommunications (ICT) (ICT 2018)*, (Saint Malo, France), June 2018.