Deterministic Scheduling to Minimize Latency 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

July 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 $\{0, \ldots, n-1\}$.

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.

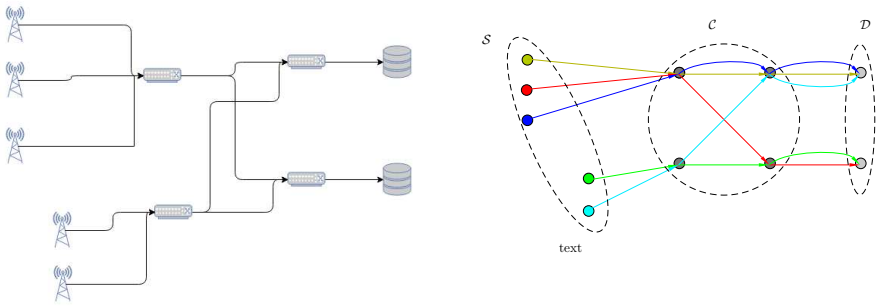

Figure 1: A C-RAN fronthaul network and its corresponding routed network

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 $\lambda(u_i, r) = \sum_{1 \leq j < i} \omega(u_j, u_{j+1})$. 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: $cl(u) = \max_{r \in \mathcal{R} \text{ and } u \in r} cl(u, r)$.

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. Several of the authors are developing a prototype based on ethernet base-X [1], using standard values for the parameters of the network: the size of an atomic data is 64 bits, the speed of the links is 10Gbps, and the length of a tic is thus about 6.4 nanoseconds.

In the process we study, a message called a **datagram** is sent on each route from the source node. The **size** of a datagram is an integer, denoted by τ, it is the number of tics needed by a node to emit the full datagram through a link. In this 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. There is no preemption: 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 available at time 0.

Let $r = (u_0, \ldots, u_l)$ 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 at vertex u_i of 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 given 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 Routed Networks

This study is motivated by a problem arising 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 is not a node of the routed network, since each RRH has its own BBU, there are no contention when going through the BBU. The datagram arrives at the BBU somewhere on an arc of the route. Once a datagram arrives at its BBU, the answer is computed and sent back over the fronthaul network. The computation time is usually the same for all messages; it is encoded into the arc corresponding to the BBU 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

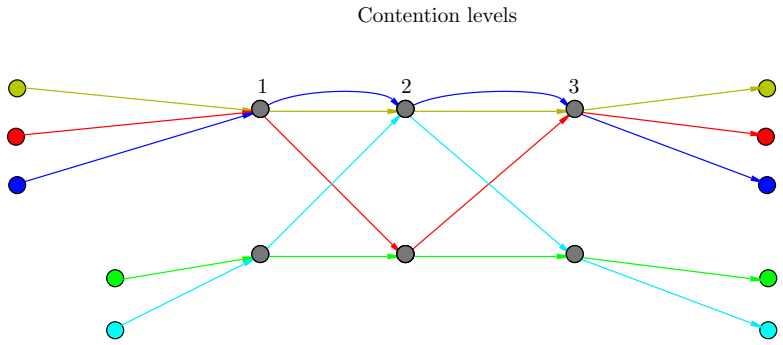

Figure 2: The routed network modeling datagrams going back and forth in the fronthaul network of figure 1.

the same. In the first case, once the messages have been scheduled to go in the datacenter, they go out in the exact same order and thus, even if all routes use the same link in the way back, it is not considerd as a contention point, as illustrated in figure 3. If the length of the links into some 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 around this vertex.

Finally, the fronthaul networks we study have **coherent** routings, a classical requirement in telecommunication networks (see e.g. [2]). It means that if the routes r and r' go through two contention points u and v, they have the same subpath between u and v. This is true for the fronthaul networks we modelize. Hence in the routed network which represents going back and forth in the fronthaul network, the coherent property is respected from the source to the arc representing the BBU and then form the arc representing the BBU to the target. As a consequence, routed networks obtained from fronthaul networks are directed acyclic multigraphs, as required in the definition of routed network.

2.4 Periodic Emission of Datagrams

The process we model 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 they are not made explicit in the assignment (contrarily to previous work [3]). 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.
Output: assignment 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. We evaluate the arithmetic complexity of our algorithms, that is arithmetic operations are considered to be in constant time. Then, the complexity will surprisingly not depend on P, τ or the weights of the routed network. Given a contention point u, let $\mathcal{R}_u \subseteq \mathcal{R}$ be the set of routes which contain u. The **load** of u is $\tau|\mathcal{R}_u|/P$, 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 crucial in the theoritical study of a simpler variant of SPALL [4], and is also important to the practical 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 component, hence the case with a single contention point is equivalent to contention depth one. If we remove the periodicity from SPALL on a single contention point, it can be seen as a single processor scheduling problem, where datagrams are jobs of the same size, their release times are given by the length of the arc to the contention point, and the deadline of each job is computed from the length of the route. The scheduling problem is as follows: each job must be scheduled such that no two jobs uses the machine at the same time. Each job must be scheduled after its release time and must satisfy its deadline. The makespan of a solution is the time elapsed between the beginning of the computation of the first job and the end of the last job. There is a polynomial time algorithms to find a scheduling that minimizes the makespan [5].

Solving the same problem, but taking into account periodicity is solving SPALL over a routed network with a single contention point. This problem has already been defined and studied in [6] under the name BRA. We proved that it can be solved by reducing it to at most 2^n instances (much less in practice) of the non periodic variant. While BRA may be in P, when messages have different sizes, the non periodic scheduling problem is known to be NP-hard []. It implies NP-hardness for BRA with different message sizes, since the periodic problem is equivalent to the non-periodic one by taking a large enough period, say the maximum of the lengths of the routes plus the number of routes times τ.

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 destination 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 centers). This topology, called *star shaped network* in previous work, is classical in fronthaul networks, and a non synchronized version of SPALL has been studied on star shaped networks [3, 6, 4].

Theorem 1. *The problem* SPALL *on star shaped network is* NP*-hard*

Proof. The two flow shop problem studied in [7] is shown to be NP-hard. The problem is as follows, a set of jobs have to be processed in sequence on two machines; a job must be processed on machine 1 before beeing processed on machine 2. There is, for each job, a delay between the end of the processing on the first machine and the begining of the processing on the second one. The objective is to minimize the makespan. The time needed to process each job is the same.
We reduce an instance of SPALL in an instance of the two flow shop proplem: A message is a job, the delay of the jobs are equivalent to the lenght of the routes between the two contention points. We then set the makespan value to P, and the instance of SPALL becomes an instance of two flow shop. □

Contention Depth Two and More However, star shaped networks may not be the most general topology with regard to problem SPALL. Que peut-on dire des topologies de profondeur 2 en général ? Expliquer que si elles modélisent un aller-retour, ce sont forcément des star shaped networks. Introduire les réseaux plus généraux de contention depth 3.Dire comment ce qu'on veut modéliser est un cas particulier de contention depth 3: profondeur 2 dans le réseau, même distance de tous les datacenters dans les BBUs ce qui vire un point de contention. Donner un exemple illustré.

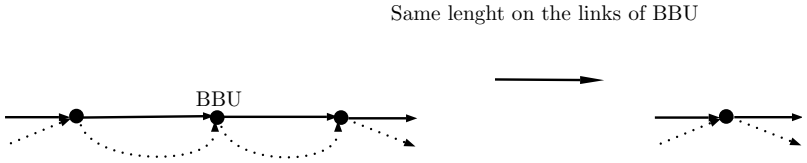

Figure 3: One or several contention points around the BBU according to the lenght of the links

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 a Valid Assignment to its Compact Representation

Let us define a function which maps a valid assignment A to a compact assignment, called the compact representation of A, denoted by $CR(A)$. Let assume that for all contention vertices u, there is a route $r \in \mathcal{R}_u$ such that $A(r, u) = 0$.

We assume that routes in $\mathcal{R}$ are indexed by the integers in $[n]$. Say w.l.o.g. that r_0 is the route of smallest index such that $A(r_0, u) = 0$. The datagram of r_0 arrives, and goes to the next contention vertex, 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 $ns(r_0, r, u) <$

$nt(r_0, r, u)$. Represent the time as 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 it has been available in.

Fig. 4 illustrates how a compact representation is computed from an assignment on a single node u. On top, the datagrams are represented by sending time $s(r_i, u)$ while the bottom of the figure shows the datagrams in a single period, represented by normalized sending times $ns(r_0, r_i, u)$.

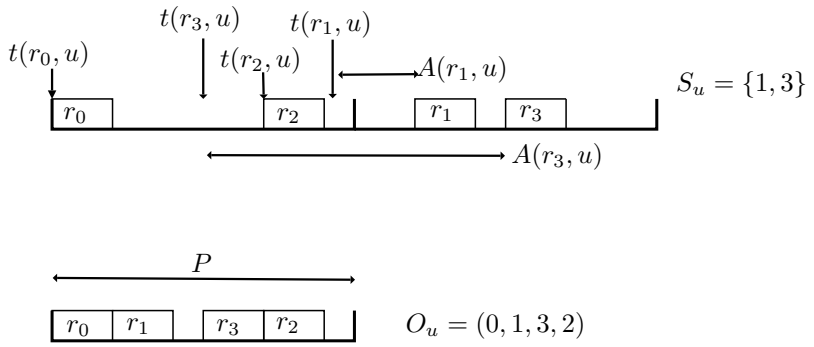

Figure 4: A compact representation of an assignment in which $O_u = (0, 1, 3, 2)$ and $S_u = \{1, 3\}$

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

Proof. Consider a vertex u of contention level 1, such that for all $r \in \mathcal{R}_u$, $A(r, u) > 0$. Let us define m as the minimum of these values, we define $A'(r, u) = A(r, u) - m$. Assignment A' has no collision on u, since all departure times have been shifted by the same value and A has no collision. Moreover, if v is the vertex after u in a route r, we define $A'(r, v) = A(r, v) + m$. Hence, all departure times for vertices of contention 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 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 vertex u.

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

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

Figure 5 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 $ns(r_2, r_3, u) = ns(r_2, r_0, u) + \tau$.

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 3. *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

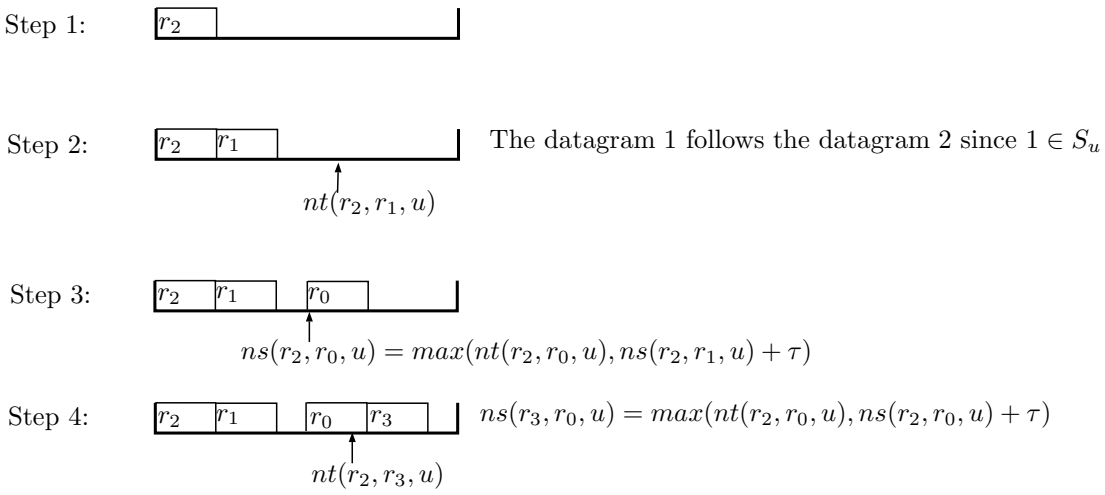

Figure 5: Inductive construction of $Real((2,1,0,3),\{1\})$ from CA on a single contention vertex u.

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

We can define the following equivalence relation over canonical assignments: A and B are equivalent if and only if $CR(A) = CR(B)$. We say that a compact assignment $CA = (O_u, S_u)_{u \in V(G)}$ is *canonical* if it is a realizable compact assignment, $CR(Real(CA)) = (O'_u, S'_u)_{u \in V(G)}$ and if for all vertices u, the first 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 for $\prec$ in its equivalence class.

Lemma 4. *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 5. *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 3 and we keep the compact assignment for which this value is minimal. Because of Lemma 4, 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 vertex 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 $\{u_1, \ldots, u_l\}$, with $s_i = |\mathcal{R}_{u_i}|$, there are $\prod_{1 \leq i \leq l} s_i!2^{s_i}$ compact assignments. On a given contention level, all routes use at most 1 vertex, hence $\sum_{1 \leq i \leq l} s_i \leq n$. Since $\prod_{1 \leq i \leq l} s_i! \leq (\sum_{1 \leq i \leq l} s_i)!$, we have $\prod_{1 \leq i \leq l} s_i!2^{s_i} \leq n!2^n$. There are d contention levels, thus we have at most $(n!2^n)^d$ compact assignments which proves the theorem. □

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 $\{u_1, \ldots, u_l\}$, we only need to consider $(n!2^n)^{d-1}(\sum_{1 \leq i \leq l} s_i!2^{s_i})$ 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. TODO: Mettre ça plutot dans la partie branch and bound et dire qu'on peut encore plus faire baisser la complexité en utilisant not simmons FPT à ce niveau. Enfin analyser le nombre de solutions qu'on voit en ne regardant que les canoniques, (minimales ?)

4 Greedy Algorithms

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

4.1 Greedy Deadline

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

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 $\lambda(u,r)-t(r',u)$ 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, τ
Ensure: An assignment $A(G, \mathcal{R})$, or FAILS
  $budget[|\mathcal{R}|]$ integer table.
  for all route r in $\mathcal{R}$ **do**
    $budget[r] \leftarrow \lambda(u, r) - t(r, u)$
  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

4.2 Greedy Normalized

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

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

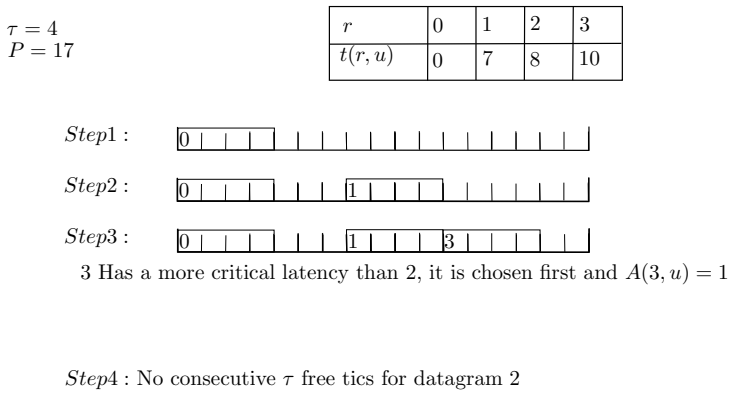

Figure 6: An instance for which `Greedy Deadline` fails to build an assignment

is always a solution when the load is less or equal to 1. Since each datagram forbids at most $2\tau - 1$ tics in the period to the other datagrams, by a pigeonhole argument, all routes can be scheduled when the load is less than 0.5 by greedy algorithms considering all departure times (see [6, 4] for similar arguments).

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

4.3 Greedy Packed

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

4.4 Random Generation of Routed Network

We propose several experiments to assess the practical performance (in speed and quality) of the poposed algorithms. We present here the instances on which we test our algorithms, which are derived from our application to Cloud-RAN. We consider networks of contention depth three, as illustrated in figure 7, in which the dotted arcs represents each the arcs of two routes. As explained is

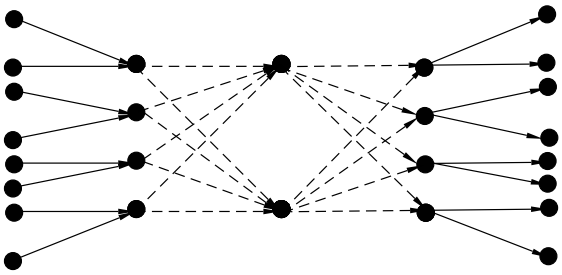

Figure 7: Shape of the randomly generated routed networks: 4 vertices of contention level 1, each with two routes going through, which go to the two different data centers in contention level 2

section 2.5, there is only one contention vertex for each datacenter, they are the two vertices of contention level 2.

To generate random routed networks, several parameters must be chosen: The load of the network, the number of routes, the distribution of the length of the arcs, and the topology of the routed network. We would like to understand the the impact of those parameters, in terms of computation time and quality, on the algorithms studied. In order to reduce the number of experiences presented here, we fix the topology of the graph as in Figure 7. The results are not significantly impacted if we change those values.

TODO: dire que par défaut on fixe 80pour cent, mais qu'on va montrer l'effet

The impact of the load on the quality of the results was investigated: 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. This means that $P = \frac{\tau \times n}{0.8}$, whith n the number of routes on the graph. The size of the C-RAN traffic depends of the service requirement [?]. Here, we fix $\tau = 2500$ tics.

In a C-RAN context, the number of route is low. In the network we study, there is $n = 8$ routes on the graph. This kind of graphs with few routes allows us to use the branch and boud algorithm as a lower bound for evaluating the performances of the other algorithms. We will study the impact of the number of routes in the graph at the end of this section. The length of the arcs is drawn uniformly between 0 and P. This choice makes the periodicity of our problem impactful, and does not allow for algorithms reused from a non periodic setting.

TODO: rendre très clair que n est fixé à 8 la plupart du temps. Il manque la valeur de τ. Avec la charge à 80 pour cent, ça donne tous les paramètres du problème

Load / Sucess	70%	80%	90%	100%
`Greedy Deadline`	100%	95.1%	56.3%	11.2%
`Greedy Normalized`	99.9%	95.5%	68.3%	0%

Figure 8: Success rate of the greedy algorithms for different loads

4.5 Success Rate and Performance of the Greedy Algorithms

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 8 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 9 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 `Greedy Deadline` and `Greedy Normalized` when they are able to find an assignment. `Greedy Normalized` performs better than `Greedy Deadline` when it finds an assignment. On vertices with high load, the three algorithms almost always find the same assignment (or fail). On vertices of small load, the constraint of packing the datagram imposed by `Greedy Deadline` worsen the latency.

We propose improved version of `Greedy Deadline` and `Greedy Normalized` that always find a solution. For each contention point, we first try `Greedy Deadline` (or `Greedy Normalized`), and if the algorithm fails, we apply `Greedy Packed`. Let us call `Hybrid Greedy Deadline` and `Hybrid Greedy Normalized` those two algorithms. Figure 10 shows the performances of `Hybrid Greedy Deadline`, `Hybrid Greedy Normalized`, and `Greedy Packed` on 1000 routed networks with a load of 90%.

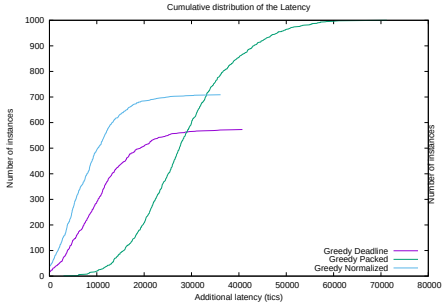

Figure 9: Performance of `Greedy Deadline`, `Greedy Normalized` and `Greedy Packed` over 1000 instances on graphs with 90% load.

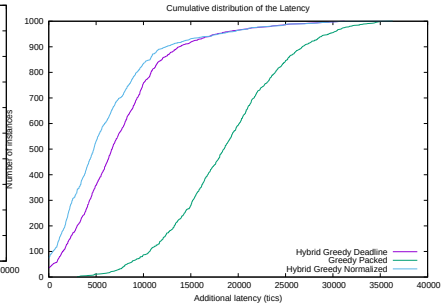

Figure 10: Performance of the updated greedy algorithms that always gives an assignment

Algorithm `Hybrid Greedy Normalized` seems much better than the other two. Hence, in the rest of the paper `Hybrid Greedy Normalized` 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 $O_u = (r_1, \ldots, r_l)$. Let $r_i \in \mathcal{R}_u$, the *r_i-neighborood* of (O_u, S_u) is the set of pairs (O, S) such that:

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

Informally, a compact 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 and the previous route from the set. Remark that the

r-neighborood of any pair (O_u, S_u) has at most 6 members (it can be 4 when the route r is in first position and cannot be exchanged with the previous one).

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 $k_1, \ldots, k_n$ the number of contention vertices on the n routes of a routed network $(G, \mathcal{R})$. Then, a compact representation has at most $\sum_{i=1}^{n} 6^{k_i}$ neighbors. Since the networks we consider are of bounded contention depth (2 or 3 in practice), the size of a neighborhood is linear in the number of routes. We further restrict the notion of 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. We call the graph defined by the neighborood relation over realizable compact assignments of a routed network the **transposition graph** of the routed network. All algorithms presented in this section will do a walk in the transposition graph, trying to find a vertex with optimal transmission time.

TODO: Faire un dessin d'un routed network simple

Lemma 6. *There is a path from a realizable compact representation CR, with $CR(u) = (O_u, S_u)$ to CR', such that CR' is equal to CR except on u where it is equal to $(O_u, S_u \cup E)$.*

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

Theorem 7. *The transposition graph of a routed network is connected.*

Proof. For simplicity, we assume that the routed network has a single contention node u. Let (O_u, S_u) and (O'_u, S'_u) be two realizable compact assignments, we show there is a path between them. Let r be the first element of O_u and let $E = \mathcal{R}_u \setminus \{r\}$. By Lemma 6, there is a path from (O_u, S_u) to (O_u, E). Consider now O''_u, the order O'_u whith r placed in first position. There is a path from (O_u, E) to (O''_u, E). Indeed, any order is realizable, when all elements but the first are in E because there are no constraints on their normalized sending time. Now, let r' be the first element of O'_u. By definition, $(O'_u, E \triangle \{r, r'\})$ is in the r' neighborood of (O''_u, E). Moreover, $(O'_u, E \triangle \{r, r'\})$ is realizable because $E \triangle \{r, r'\}$ is equal to all routes but the first in O'_u. Finally, using Lemma 6 once again prove there is a path between $(O'_u, E \triangle \{r, r'\})$ and (O'_u, S'_u) since (O'_u, S'_u) is realizable and $S'_u \subseteq E \triangle \{r, r'\}$, which proves the theorem. □

5.1 Hill Climbing

The simplest local search heuristic is hill climbing. This algorithm starts from a compact assignment CA, explores the entire neighborhood of CA, and selects CA' the realizable compact assignment 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 on the the intial compact assignment. Since we have already designed `Hybrid Greedy Normalized` which is always able to produce a decent solution, we use the compact representation of its result to initialize the hill climbing algorithm. Another possibility is to use a random compact assignment, by drawing uniformly the order and the set for each contention vertex. Since a random compact assignment does not always yields a valid assignment, or may give an assignment of low quality, we execute hill climbing starting from several random compact assignments, and return the best assignment given.

Figure 11 shows the difference between initalizing the hill climbing with `Hybrid Greedy Normalized`, one or several random compact representations. Those results results are computed from 1000 random instances, as explained in Sec. 4.4.

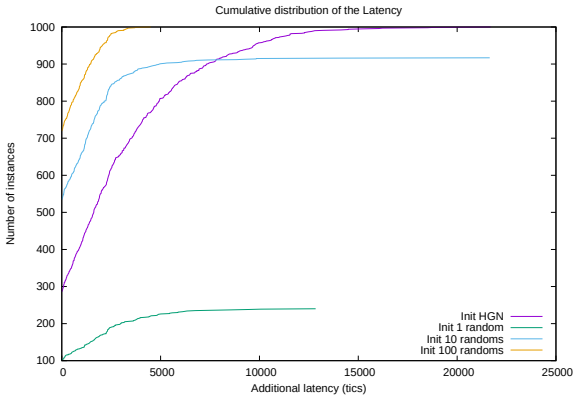

Figure 11: Additional latency of the solutions given by hill climbing, intialized with HGN, 1, 10 or 100 random compact representations.

Initializing hill climbing with 100 random compact representations gives better results. Nevertheless, it may fail to find a *valid* compact representation in the initialization, making the whole algorithm fail. When the load grows, it is harder to find a valid compact representation. Moreover, at fixed load, when the number of routes and thus of compact representations grows, the probability of finding a valid one at random decreases. Figures 12 and 13 illustrates

this phenomenon, they present the probability of drawing at least one compact representation that gives a valid assignment, when drawing 1, 10 or 100 random compact representations. Each result is computed from 1000 random instances.

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

Figure 12: Success rate of hill climbing when increasing the load.

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

Figure 13: Success rate of hill climbing when increasing the number of routes.

Those experiences shows that the choice of initialization by many random instances, while good for our typical network with 8 routes, do not scale to a larger number of routes or higher loads. We could compensate for this phenomenon, by increasing the number of random compact representation used to initialise hill climbing. However, at high load, we expect the ratio of valid representations to decrease exponentially fast, which makes the algorithm to slow.

We propose again an hybrid approach to initialise hill climbing: compute the best solution given by hill climbing intialized by HGN and k random compact representations. This algorithm is called k-Hybrid Hill Climbing or k-HHC. Figure 14 shows the additional latency incurred by the solution found by k-HCC, for different k, to illustrate the tradeoff between quality and time.

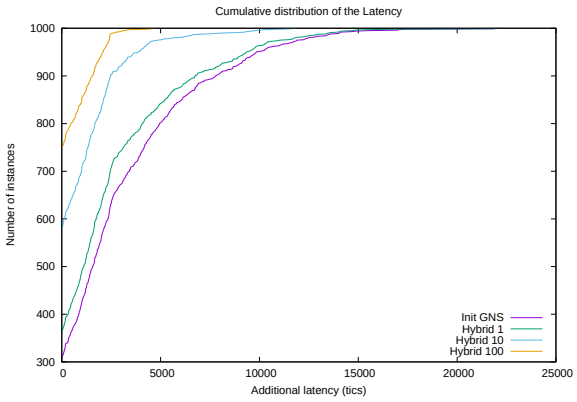

Figure 14: Cumulative distribution of the additional latency of solutions given by k-HHC computed from 1000 random instances).

We now investigate how many steps hill climbing computes before ending. Tab. 15 shows the average number of steps made by different hill climbing algorithm on its run computing the best solution. The number of steps are

taken from the experiment of Fig. 14.

Init	HGN	Hybrid 1	Hybrid 10	Hybrid 100
Average Nb Step	1.14	1.26	1.92	3.38

Figure 15: Average number of step needed by hill climbing to reach a local optimum, for the run giving the best solution.

The more steps the hill climbing does before ending, the more the initial solution is improved. When hill climbing starts from `Hybrid Greedy Normalized`, it does not improve the solution much. As expected, drawing more random representation as initialisation, allows to find better solution and this solution is obtained by more work from the initial random representation.

The idea of drawing a large number of random compact representations to initialize is a naive approach to explore better the solution space without being blocked in a local optimum. We then introduce taboo search and simulated annealing, to realize the same objective of exploring the solution space more widely but also more efficiently.

5.2 Tabu Search

Tabu search is a variation on hill climbing using memory, in order to escape local optimum and hopefully find better solutions. At each step, we select the compact assignment CA' which minimizes $TR(CA')$, even when $TR(CA') < TR(CA)$ is not satisfied. To avoid loops in local minimums, the last N solutions explored are memorized and we forbid to visit them again (they are not considered when computing the minimum of the neighborood). 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 solutions in order to not visit them again. Varying the value of x can improve or worsen the performance of the tabu search. Figure 16 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.

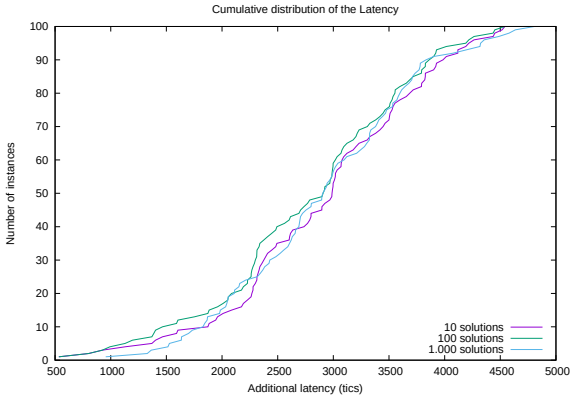

Figure 16: Difference between the tabu search whith a memory of 10,100 or 1000 solutions

Also, it appears that, in most of 60% of the cases, tabu search finds it's best 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. The value Δ represent the difference $TR(REAL(CA')) - TR(REAL(CA))$ where CA' is a compact assignment in the neighboorhood of CA. CA' is accepted or rejected with the probability $e^{-\frac{\Delta}{t}}$, where t is the temperature. This means that $\frac{\Delta}{t}$ must be close to 0 at the begining, and must approaches infinity at the end of the execution.

In order to fix the initial temperature t_0, we compute a routine explained in [8]:

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

Tabular 17 shows the additional latency needed by simulated annealing when intialized with two intial temperatures computed from the previous routine. The first solution considered by simulated annealing is the solution given by hill climbing. The experiment is made over 100 instances. During the computations 1000 compact representation are drawn before decreasing the temperature.

Quality of initial configuration	Good	Poor
t_0	2200	13000
Additional latency	4212	4217
Computation time (ms)	2.817	4.035

Figure 17: Comparison of two intial temperatures, considering the quality of the initial configuration

As observed, the initial solution given by hill climbing can be considered as "good". Indeed, increasing the initial temperature does not affect the quality of the solution, but increase the computation time.
In simulated annealing, the temperature decrease slowly. Each **level**, several compact representation are drawn. At the end of a level, the temperature is decreased. Drawing too few compact representation per level induce decrasing the temperature too fast and thus, reducing the efficiency of simulated annealing. In an ohter hand, drawing too much compact representation increases the computation time of the algorithm.
We investiage the impact of drawing a large number of compact representation before decreasing the temperature. Figure 18 shows the additional latency needed by simulated annealing with different number of compact representation draw for each level. Those results comes from 1000 random instances, in which the temperature is set to 2200.
It appears that drawing more than 100 compact representation per level does not improove the quality of the solution, while it increase the computation time.
TODO: differents profils de temperature

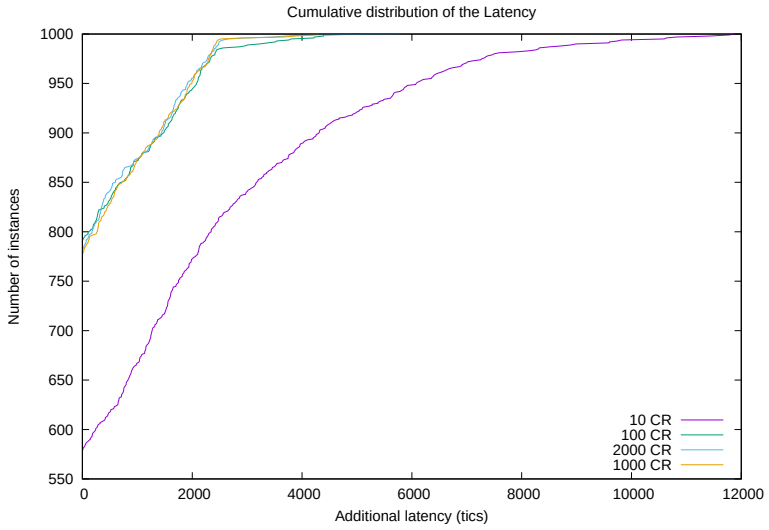

Figure 18: Additional latency needed to find a solution for simulated annealing, with a different number of compact representation drawn to each level.

6 Branch and Bound

6.1 Partial Compact Representation and Relaxation

D'abord introduire la notion de solution partielle avec seulement les temps d'attente fixée sur seulement une partie des sommets de contention (dans l'ordre des niveaux de contention).
Puis définir le graphe associé à une solution partielle, en enlevant les sommets déjà traités et en mettant à jour les délais. Un exemple serait le bienvenu.
Donner la relaxation du problème général au problème sur un point de contention, c'est à dire remplacer tous les chemins vers et depuis le sommet par un arc de la même longueur. Expliquer que la solution optimale pour le problème relaxé est meilleure (borne supérieure) de ce qu'on cherche. On définit ensuite une borne supérieure meilleure en faisant le max des relaxations pour tous les points de contention. Expliquer que cette borne se calcule facilement en pratique (algo Simmons FPT vs algo branch and bound simplifié).

6.2 The Branch and Bound Algorithms

Définir l'arbre de recherche exhaustif sur les solutions partielles et expliquer les coupes qu'on peut faire grâce à la borne sup qu'on vient de définir (en gardant la valeur de la meilleur solution et en initialisant cette valeur par un algo de

bonne qualité, genre hill clibming ou carrément recuit simulé).

6.3 Locally minimal representation

On voudrait énumérer uniquement les solutions qui sont minimales pour $\prec$ pour un sommet donné et canoniques, ce qu'on appelle des solutions locally minimal. On propose un ensemble de coupes faciles à calculer qui permettent d'éliminer la plupart des solutions non minimales. Proposer une caractérisation des solutions localement minimales (à vérifier).
Montrer avec des expériences (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 valides, et presque aucunes dont la réalisation n'est pas minimale.

6.4 Last Contention Level

Mettre ici le traitement spécifique du dernier niveau -> analyse de la complexité différente à mettre ici, certaines coupes à ne pas faire car seul l'objectif de temps global nous importe (à vérifier c'est un tradeoff, mais certaines sont évidemment à retirer).

7 Experimental Evaluation

Comparer les temps de calcul et la qualité des solutions trouvées. Le faire pour plusieurs régimes: -8 routes vs bcp de routes (branch and bound impossible) -charge 40 80 et 100 pour cent -routes dont les arcs sont tous petits (tirés dans [tau] ou moins) ou grands (tirés dans [P]). -plus de 2 data centers, 3 ou 4 ?
Enfin comparer aux méthodes de multiplexage statistique avec FIFO et Greedy Deadline. Expliquer les rapports/différences de ces méthodes avec les algos greedys.

8 Autres trucs à raconter ?

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 19 shows the average additional latency needed by the branch and bound and the greedy algorithm that we call HGN 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.

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

Figure 19: Difference between the optimal solution of the greedy algorithms, for different kind of instances generated.

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 HGN remains in the same order of magnitude.

9 Conclusion

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

References

[1] "IEEE 802.3 10 gb/s."

[2] L. Schwiebert and D. Jayasimha, "A necessary and sufficient condition for deadlock-free wormhole routing," *Journal of Parallel and Distributed Computing*, vol. 32, no. 1, pp. 103–117, 1996.

[3] 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.

[4] M. Guiraud and Y. Strozecki, "Scheduling periodic messages on a shared link," *arXiv preprint arXiv:2002.07606*, 2020.

[5] B. Simons, "A fast algorithm for single processor scheduling," in *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, (Ann Arbor, MI, USA), pp. 246–252, IEEE, Oct. 1978.

[6] D. Barth, M. Guiraud, and Y. Strozecki, "Deterministic scheduling of periodic messages for cloud ran," *arXiv preprint arXiv:1801.07029*, 2018.

[7] W. Yu, H. Hoogeveen, and J. K. Lenstra, "Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard," *Journal of Scheduling*, vol. 7, no. 5, pp. 333–348, 2004.

[8] I. H. Osman and J. P. Kelly, "Meta-heuristics theory and applications," *Journal of the Operational Research Society*, vol. 48, no. 6, pp. 657–657, 1997.