Contention Management for 5G

DB,CC,MG,OM,YS

May 18, 2017

Abstract

This article treats about Contention Management for 5G.

The evolutions proposed in next generations of mobile network architectures aim at evolving toward centralized radio network architectures (C-RAN, for Cloud Radio Access Network) to reduce consumption costs and power at the base stations [1]. This type of architecture faces the challenge of mastering the latency in the transfer process between the Remote Radio Heads (RRHs) on the field and baseband units (BBUs) in the cloud. Low latency is already critical for the deployment of C-RAN approach. The standard requires meeting time constraints for functions like HARQ (Hybrid Automatic Repeat reQuest) that needs to be processed in less than from 1 to 10ms [2] (depending on targeted services). One specificity in the C-RAN context is not only the latency constraint, but also the periodicity of the data transfer between RRH and BBU (this HARQ constraints must be enforced for each frame emitted every millisecond).

Thus in this article, we work on latency and periodicity constraints in fronthaul. The best current solution is to rely on an almost full optical approach, where each end-points (RRH on one side, BBU on the other side) are connected through direct fiber or full optical switches [3]. This architecture is very expensive and hardly scales in the case of a mobile network composed of about 10,000 base stations. It is then needed to find a solution to offer low latency over commoditized packet based networks. Indeed, dynamical optical bypass and dynamical management of the emission should be considered to guarantee latency constraints, as it is considered in the french ANR project N-GREEN. This project proposes a new type of switching/routing node and a specific network architecture exploiting WDM packets thanks to a new generation of optical add/drop multiplexers (WSADM: WDM slotted add/drop multiplexer). These packets having a fixed duration close to 1μs are transported in a transparent way, to better exploit the switching matrix of the node; their headers will be transported over one dedicated wavelength at a lower bit rate, to reduce the physical constraints of the electronic processing and scheduler.

Thus, new scheduling and routing paradigms and new technologies have to be considered to guarantee delay constrained periodic data transfers. One of the most promising approaches relies on the concept of Deterministic Networking (DN) such that one get rid of statistical multiplexing. The traditional queue managements are replaced by time based forwarding. Solutions for Deterministic Networking are under standardization in IEEE 802.1 TSN group [4], as well

at IETF DetNet working group [5]. Several patents on concepts and mechanisms for DetNet have been already published, see for example [6, 7]. To make DN working over a network composed of several nodes, it is required to manage the time at which the packets of deterministic paths are crossing each nodes. The major difficulty of this problem is the periodicity of the process. Indeed, a deterministic sending for the messages between each pair BBU/RRH must not collide with the other messages sent by the others BBU/RRH in the same period, but also in the previous and following periods.

Considering a graph, modeling the network topology, and a set of routes from source nodes (modeling connections to the BBU) and destination nodes (modeling the RRH) in this graph, the purpose is to select, for each destination node a route from one source node to it and a periodic routing scheme allowing to periodically sent a packet to each base station without congestion conflicts between all such packets and to insure a minimum latency. In a slotted time model, the aim is here to minimize the duration of the period, with a constraint of the maximum length of routes to be selected. Even if the selected set of routes is given this optimization problem has been shown to be NP-hard. From an algorithmic point of view, the purpose of this project is first, to study the complexity and the approximability of this problem when the length of the routes is small (which corresponds to realistic cases), and secondly, to propose and implement some heuristics to solve this problem on realistic topologies.

This problem may look like wormhole problem [8], very popular few years ago, but here, we want to minimize the time lost in buffers and not just avoid the deadlock, and the wormhole does not treat about the periodicity. Several graph colorings have been introduced to model similar problems such as the allocation of frequencies [9], bandwidths [10][17] or routes [8][18] in a network or train schedules [11][19]. Unfortunately, they do not take into account the periodicity and the associated problems are also NP-complete. The only model which incorporates some periodicity is the circular coloring [12, 13, 14]but is not expressive enough to capture our problem.

Paper organisation

In the next section, ...

1 Model

1.1 Definitions

We consider a directed graph $G = (V, A)$ modeling a network. Each arc (u, v) in A is labeled by an integer $dl(u, v) \geq 1$ that we call the delay and which represents the number of time slots taken by a signal to go from u to v using this arc.

A **route** r in G is a sequence of consecutive arcs $a_0, \ldots, a_{k-1}$, with $a_i = (u_i, u_{i+1}) \in A$. We will often refer to the first element of the route as a source

and the last as a target.

The **latency** of a vertex u_i in r, with $i \geq 1$, is defined by

$$\lambda(u_i, r) = \sum_{0 \leq j < i} dl(a_j)$$

We also define $\lambda(u_0, r) = 0$. The latency of the route r is defined by $\lambda(r) = \lambda(u_k, r)$.

A **routing function** $\mathcal{R}$ in G associates to each pair of vertices (u, v) a route from u to v. Let $\mathcal{C}$ be an **assignment** in G, i.e., a set of couples of different vertices of G. We denote by $\mathcal{R}_\mathcal{C}$ the set of routes $\mathcal{R}(u, v)$ for any (u, v) in $\mathcal{C}$. We call $\mathcal{R}_\mathcal{C}$ a **routage graph**, it contains all the informations needed in the forthcoming problems (assignment, routes and delays of the arcs).

TODO: Si on s'en sert, ajouter ici que le routage est cohérent.

1.2 Slotted time Model

Consider now a positive integer P called the **period**. In our problem, we send messages in the network with period P. The time will thus be cut into slices of P discrete slots. Assume we send a message at the source s_i of the route r_{s_i}, at the time slot m_{s_i} in the first period, then a message will be sent at time slot m_{s_i} at each new period. We define the first time slot at which the message reaches a vertex v in this route by $t(v, r_{s_i}) = m_{s_i} + \lambda(v, r) \mod P$.

A message usually cannot be transported in a single time slot. We denote by τ the number of consecutive slots necessary to transmit a message. Let us call $[t(v, r_{s_i})]_{P,\tau}$ the set of time slots used by r_{s_i} at a vertex v in a period P, that is $[t(v, r_{s_i})]_{P,\tau} = \{t(v, r_{s_i}) + k \mod P \mid 0 \leq k < \tau\}$. Usually P and τ will be clear from the context and we will denote $[t(v, r_{s_i})]_{P,\tau}$ by $[t(v, r_{s_i})]$.

A (P, τ)**-periodic affectation** of a routage graph $\mathcal{R}_\mathcal{C}$ is a sequence $\mathcal{M} = (m_{s_0}, \ldots, m_{s_{n-1}})$ of n integers that we call **offsets**, with n the number of routes in $\mathcal{R}_\mathcal{C}$. The number m_{s_i} represents the index of the first slot used in a period by the route $r_{s_i} \in \mathcal{R}_\mathcal{C}$ at its source s_i. A (P, τ)-periodic affectation must have no **collision** between two routes in $\mathcal{R}_\mathcal{C}$, that is $\forall (r_{s_i}, r_{s_j}) \in {\mathcal{R}_\mathcal{C}}^2, i \neq j$, we have

$$[t(u, r_{s_i})] \cap [t(u, r_{s_j})] = \emptyset.$$

As an example of a $(2, 1)$-periodic affectation, let consider a routage graph with routes $\{r_{s_i}\}_{i=0,\ldots,n-1}$, such that all pairs of routes intersect at a different edge. We set $\tau = 1$ and the delays are chosen so that if r_{s_i} and r_{s_j} have v as first common vertex then we have $\lambda(v, r_{s_i}) - \lambda(v, r_{s_j}) = 1$.There is a $(2, 1)$-periodic affectation by setting all m_{s_i} to 0.

1.3 Problems

We want to ensure that there is an affectation which allows to send periodic messages from sources to target without collisions. The problem we need to solve is thus the following:

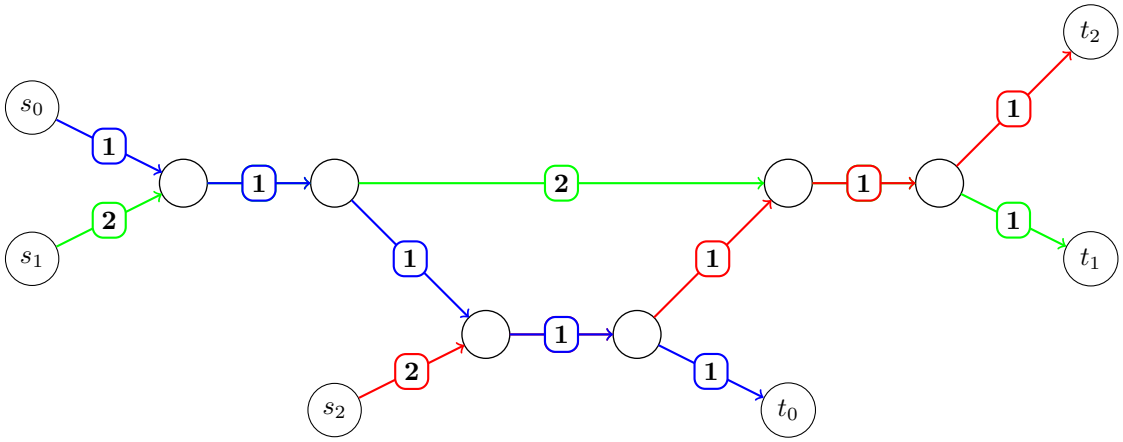


Figure 1: A routage graph with $(0, 0, 0)$ as a $(2, 1)$-periodic affectation

Periodic Routes Assignment (PRA)

Input: a routage graph $\mathcal{R}_\mathcal{C}$, an integer τ and an integer P.

Question: does there exist a (P, τ)-periodic affectation of $\mathcal{R}_\mathcal{C}$?

We will prove in Sec. 2 that the problem PRA is NP-complete, even in restricted settings. Even approximating the smallest value of P for which there is a (P, τ)-periodic affectation is hard.

An unusual property of affectation is that given a routage graph, we may have a (P, τ)-periodic affectation but no (P', τ)-periodic affectation with $P' > P$: the existence of an affectation is not monotone with regards to P.

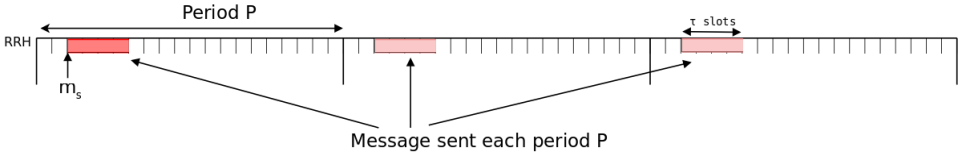
Lemma 1. *For any odd P, there is a routage graph such that there is $(2, 1)$-periodic affectation but no $(P, 1)$-periodic affectation.*

Proof. Consider the routage graph $\mathcal{R}_\mathcal{C}$ given in the previous subsection. We change the delays so that for v, the first vertex which belongs to r_i and r_j, we have $\lambda(v, r_i) - \lambda(v, r_j) = P$, where P is an odd number smaller than n, the number of routes in $\mathcal{R}_\mathcal{C}$. In such a graph, there is no (P, τ)-periodic affectation, since the problem reduces to finding a P-coloring in a complete graph with $n > P$ vertices.
If we consider a period of 2, for all $i \neq j$, $\lambda(v, r_i) - \lambda(v, r_j) \mod 2 = 1$. Therefore $(0, \ldots, 0)$ is a $(2, 1)$-periodic affectation of $\mathcal{R}_\mathcal{C}$.

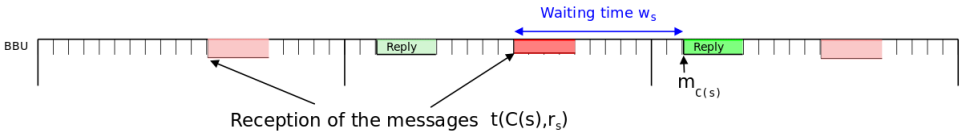
□

In the context of cloud-RAN applications, we consider here the digraph $G = (V, A)$ modeling the target network and two disjoint subsets of vertices S and T of equal cardinality, where S is the set of RRHs and T is the set of BBUs. A **symmetric** assignment $\mathcal{C}$, is an involutive function from S to T, which maps each element $s \in S$ to $\mathcal{C}(s)$ and $\mathcal{C}(s)$ to s. It can also be seen as the set of pairs $(s, \mathcal{C}(s))$ and $(\mathcal{C}(s), s)$.
The routing function $\mathcal{R}$ associates to each couple $(s, \mathcal{C}(s))$ a route called r_s and

to each couples $(\mathcal{C}(s), s)$ a route called $r_{\mathcal{C}(s)}$.
We are given a period P, a routing function $\mathcal{R}$, and we consider a (P, τ)-periodic affectation of $\mathcal{R}_{\mathcal{C}}$ which associates m_s to the route r_s which begins by s, and $m_{\mathcal{C}(s)}$ to the routes $r_{\mathcal{C}(s)}$ which begins by $\mathcal{C}(s)$. This affectation represents the following process: first a message is sent at s, through the route r_s, at time m_s.



This message is received by $\mathcal{C}(s)$ at time $t(\mathcal{C}(s), r_s)$. It is then sent back to s in the same period at time $m_{\mathcal{C}(s)}$ if $m_{\mathcal{C}(s)} > t(\mathcal{C}(s), r_s)$, otherwise at time $m_{\mathcal{C}(s)}$ in the next period. The time between the arrival of the message and the time it is sent back is called the **waiting time** and is defined by $w_s = m_{\mathcal{C}(s)} - t(\mathcal{C}(s), r_s)$ if $m_{\mathcal{C}(s)} > t(\mathcal{C}(s), r_s)$ and $w_s = m_{\mathcal{C}(s)} + P - t(\mathcal{C}(s), r_s)$ otherwise.



When a BBU receives a message, it must compute the answer before sending it back to the RRH. This time can be encoded in the last arc leading to the BBU and thus we need not to consider it explicitly in our model.

Thus, the whole process time for a message sent at vertex s is equal to

$$PT(s) = \lambda(r_s) + w_s + \lambda(r_{\mathcal{C}(s)}).$$

In the process time, we count the time between the first time at which a message is emitted and the first time at which the message comes back. Alternatively we could consider the time between the emission of the first slot and the reception of the last slot of the message, which would add τ to the process time. However, since all messages are of size τ, it will not change the problem we solve in the rest of the article and that we now introduce.

The **maximum process time** of the (P, τ)-periodic affectation $\mathcal{M}$ is defined by $MT(\mathcal{M}) = \max_{s \in S} PT(s)$. The problem we want to solve is the following.

Periodic Assignment for Low Latency(PALL)

Input: A routage graph $\mathcal{R}_{\mathcal{C}}$ with $\mathcal{C}$ a symmetric assignment, a period P, an integer τ, an integer T_{max}.

Question: does there exist a (P, τ)-periodic affectation $\mathcal{M}$ of $\mathcal{R}_{\mathcal{C}}$ such that $MT(\mathcal{M}) \leq T_{max}$?

TODO: Si on veut, on peut parler du temps moyen aussi ici, seulement si on fait quelque chose dessus dans la suite

In this article, we works on a given routage, but we could imagine a problem, in which, considering a given graph $G = (V, A)$ and a period P, the question is to find, if it exists, a routage graph $\mathcal{R}_\mathcal{C}$, in which there is (P, τ)-periodic affectation.

TODO: Dire si on regle le probleme de trouver un bon routage dans un cas particulier après

2 Solving PRA

2.1 NP-Hardness

In this section we assume that the size of a message τ is equal to one. We will prove the hardness of PRA and PALL for this parameter, which implies the hardness of the general problems. Consider an instance of the problem PRA, i.e., a routage graph $\mathcal{R}_\mathcal{C}$ and a period P.
The **conflict depth** of a route is the number of other routes which share an edge with it. The conflict depth of a routage graph $\mathcal{R}_\mathcal{C}$ is the maximum of the conflict depth of the routes in $\mathcal{R}_\mathcal{C}$.
The **load** of a routage graph is the maximal number of routes sharing the same arc. It is clear that a $(P, 1)$-periodic affectation must satisfy that P is larger or equal to the load.

We give two alternate proofs that PRA is NP-complete. The first proof works already for conflict depth two. Remark that for conflict depth one, the graph can be seen as a set of disjoint pair of routes, on which PRA and PALL can be solved in linear time. The second proof reduces the problem to graph coloring and implies inapproximability when one tries to find the smallest possible P.

Proposition 1. *Problem PRA is NP-complete, when the routing is of conflict depth two.*

Proof. The problem PRA is in NP since given an offset for each route in an affectation, it is easy to check in linear time in the number of edges whether there are collisions.

Let $H = (V, E)$ be a graph and let d be its maximal degree. We consider the problem to determine whether H is edge-colorable with d or $d + 1$ colors. The edge coloring problem is NP-hard [15] and we reduce it to PRA to prove its NP-hardness. We define from H an instance of PRA as follows. For each v in V, the graph G has two vertices v_1, v_2, and for each $(u, v) \in E$,the graph G has two vertices $s_{u,v}, t_{u,v}$. Set an arbitrary orientation of the edge (u, v), such that the following route is directed from u to v.

For each edge $(u, v) \in E$, there is a route $s_{u,v}, u_1, u_2, v_1, v_2, t_{u,v}$ in $\mathcal{R}$. All these arcs are of weight 0. The set of arcs of G is the union between all the arcs of the previous routes. The affectation $\mathcal{C}$ is the set of pair of vertices $(s_{u,v}, t_{u,v})$.

Observe that the existence of a d-coloring of H is equivalent to the existence of a $(d, 1)$-periodic affectation of $\mathcal{R}_\mathcal{C}$. Indeed, a d-coloring of H can be seen as a labeling of its edges by the integers in $\{0, \ldots, d - 1\}$ and we have a bijection

between d-colorings of H and offsets of the routes of $\mathcal{R}_\mathcal{C}$. By construction, the constraint of having no collision between the routes is equivalent to the fact that no two adjacent edges have the same color. Therefore we have reduced edge coloring to PRA which concludes the proof. □

TODO: Faire un dessin d'illustration ?

Remark that we have used zero weight in the proof. If we ask the weights to be strictly positive, which makes sense in our model since they represent the latency of physical links, it is easy to adapt the proof. We just have to set them so that in any route the delay at u_1 is equal to d and thus equal to 0 modulo d. We now lift this hardness result to the problem PALL.

Corollary 1. *Problem PALL is* NP*-complete for routing of conflict depth two.*

Proof. We consider $(\mathcal{R}_\mathcal{C}, P, \tau)$ an instance of PRA. We assume that no vertex appears both in the first and second position in a pair of $\mathcal{C}$. Remark that this condition is satisfied in the previous proof, which makes the problem PRA restricted to this kind of instances NP-complete. Let us define $T_{max} = 2 \times \max_{r\in\mathcal{R}} \lambda(r) + P$. We consider $\mathcal{C}'$ and $\mathcal{R}'_{\mathcal{C}'}$ symmetrized versions of $\mathcal{C}$ and $\mathcal{R}_\mathcal{C}$ where for every route there is a symmetric route with new arcs and the same weights. The instance $(\mathcal{R}'_{\mathcal{C}'}, P, \tau, T_{max})$ is in PALL if and only if $(\mathcal{R}_\mathcal{C}, P, \tau)$ is in PRA. Indeed the waiting time of each route is by definition less than P and thus the maximal process time is always less than T_{max}. Moreover a (P, τ)-affectation of $\mathcal{R}_\mathcal{C}$ can be extended into a (P, τ)-affectation of $\mathcal{R}'_{\mathcal{C}'}$ in the following way. For each route $r_{u,v}$, the time at which the message arrives is $t(v, r_{u,v})$, then we choose as offset for $r_{v,u}$, $-t(v, r_{u,v}) \mod P$. The symmetry ensures that each new route $r_{v,u}$ in $\mathcal{R}'_{\mathcal{C}'}$ uses the same times slot as $r_{u,v}$ and thus avoid collisions. □

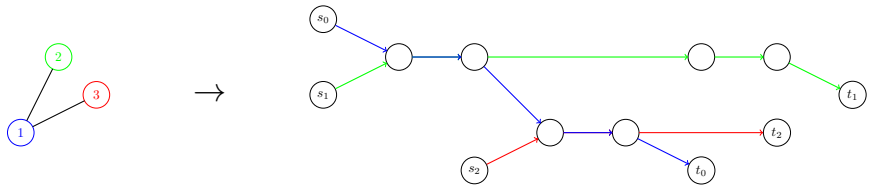
Let MIN-PRA be the problem, given a routage graph and an assignment, to find the minimal period P such that there is a P-periodic affectation.

Theorem 2. *The problem MIN-PRA cannot be approximated in polynomial time within a factor $n^{1-o(1)}$, with n the number of routes, unless* P = NP *even when the load is two.*

Proof. We reduce graph coloring to PRA. Let H be a graph instance of the k-coloring problem. We define $\mathcal{R}$ in the following way: for each vertex v in H, there is a route r_v in $\mathcal{R}$. Two routes r_v and r_u share an arc if and only if (u, v) is an edge in H; this arc is the only one shared by these two routes. All arcs are of delay 0.

Observe that the existence of a k-coloring of H is equivalent to the existence of a $(k, 1)$-periodic affectation in G, by converting an offset of a route into a color of a vertex and reciprocally. Therefore if we can approximate the minimum value of P within some factor, we could approximate the minimal number of colors needed to color a graph within the same factor. The proof follows from the hardness of approximability of finding a minimal coloring [16]. □

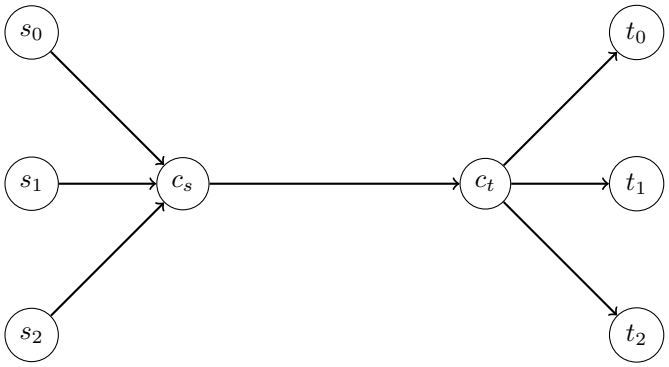
In particular, this reduction shows that even with small maximal load, the minimal period can be large.



H G

3 The Star Topology

In this paper, we consider graphs with a very simple topology that we call the **star topology**. First, for each arc (u, v), there is also an arc (v, u) with the same weight. Moreover, there is a special arc, the central arc, which is shared by all routes. All routes consist from an arc from its source to the central source node, denoted by c_s, then the central arc to c_t, the central target node, and an arc to its target. In addition to those central nodes, there are two sets of vertices, $S = \{s_0, ..., s_{n-1}\}$ and $T = \{t_0, ..., t_{n-1}\}$, of cardinality n and $\mathcal{C}$ a symmetric assignment from S to T. The routes are the directed paths of the form s_i, c_s, c_t, t_i and t_i, c_t, c_s, s_i.



We assume the weight of the central arc to be 0 since it appears in every route, its value does not matter when considering affectations.
Collisions between messages can only appear at node c_s on the way forward and at node c_t on the way back. Thus, we only have to check if there is no collisions in a period at those two points. We consider two periods P at these vertices, the the **forward period** at c_s, and the **backward period** at c_t. A message issued by the source s_1 at time m_{s_1} will reach c_s at time $m_{s_1} + t(c_s, r_{s_1}) \mod P$ in the forward period, and c_t at time $m_{t_1} + t(c_t, r_{t_1}) \mod P$ in the backward period.

A (P, τ)-periodic affectation is the choice for all $i < n$, of m_i and $m_{\mathcal{C}(s_i)}$ such that, for any couple (j, k), we have $[t(c_s, r_{s_j})] \cap [t(c_s, r_{s_k})] = \emptyset$ and $[t(c_t, r_{t_j})] \cap [t(c_t, r_{t_k})] = \emptyset$.

3.1 Solving PALL without waiting times

In this subsection, we deal with a simpler version of the problem PALL. We ask for a (P, τ)-periodic affectation **with all waiting times equal to** 0 and we call this restricted problem **Periodic Assignment for Zero Latency** or PAZL. In that case $MT(\mathcal{M})$ is equal to twice the delay of the longest route, thus T_{max} is not relevant anymore. Since $w_i = 0$, choosing m_i, the offset of the route from s_i to t_i, also sets the offset of the route from t_i to s_i to $m_i + \lambda(r_i) \mod P$. Remark that there is a bijection between the affectation of a star topology and the the affectation of the same star topology where all arcs from s_i to c_s have weight equal to 0 by changing m_i into $m_i - t(c_s, r_i) \mod P$. Therefore we assume from now on that the *weight of the arcs from s_i to c_s are all equal to* 0.

TODO: Dire qu'on donne deux algos qui trouvent toujours une solution quand certains condtions sont remplies et un dernier qui sait décider si il y a une solution oui ou non mais qui est exponentiel avec des optimisations pour le rendre pratique

3.1.1 Shortest-longest policy

We present a simple policy, which works when the period is large with regards to the delay of the routes. The message are sent in order from the shortest route to the longest route, without any gap between two messages in the forward period. In other words, we assume that the route r_i are sorted by increasing $\lambda(r_i)$ and we set $m_{s_i} = i\tau$. We call this algorithm **Shortest-Longest**.

By definition, there are no collision in the forward period and if the period is long enough, it is easy to see that in the backward period the order of the messages are the same as in the forward period and that no collision can occur.

Proposition 2. *When $n\tau + 2(\lambda(r_{s_{n-1}}) - \lambda(r_{s_0})) \leq P$, the algorithm Shortest-Longest solves positively PAZL in time $O(n \log(n))$.*

Proof. Since $m_{s_i} = i\tau$, $[t(c_s, r_{s_i})] = \{i\tau, \ldots, (i+1)\tau - 1\}$ and there are no collision on the forward period.

We assume may that $\lambda(r_{s_0}) = 0$, since removing $\lambda(r_{s_0})$ to every route does not change the order on the length of the routes and thus the result of this algorithm. We have $[t(c_s, r_{t_i})] = \{2\lambda(r_{s_i}) + i\tau, \ldots, 2\lambda(r_{s_i}) + (i+1)\tau - 1\}$ since $2\lambda(r_{s_{n-1}}) + n\tau \leq P$. Since $\lambda(r_{s_i}) \leq \lambda(r_{s_{i+1}})$ by construction, we have $2\lambda(r_{s_i}) + (i+1)\tau - 1 < 2\lambda(r_{s_{i+1}}) + (i+1)\tau$ which proves that there are no collision on the backward period.

The complexity of the algorithm is dominated by the sorting of the routes. □

If the period is slightly smaller that the bound of Proposition 2, a collision will occur on the route in the backward period, making this policy not useful even as a heuristic for longer routes.

3.1.2 Greedy Algorithm

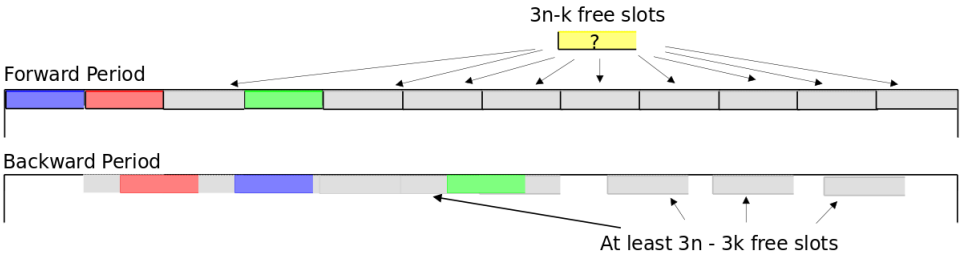
We propose a greedy algorithm to build a (P, τ)-periodic affectation, which provably works when the period is large enough with regards to the number of routes and size of messages. In other words, we can always find a solution when the network is far from saturated.

Proposition 3. *Let n be the number of routes and let $3n\tau \leq P$, then there is an algorithm which positively solves PAZL in time $O(n^2)$.*

Proof. We consider the forward period and cut it into consecutive interval of size τ that we call macro-slots. The algorithm works by choosing an offset for each route in the following way: try all offsets which put the message in a yet not used macro-slot in the forward period. The choice of an offset also fixes the position of the message in the backward period, chose the first one which does not create a collision. We now prove that this algorithm always finds a (P, τ)-periodic affectation without waiting time when $P \geq 3n\tau$.

Assume we are choosing the offset of the route r_{k+1}, we have at least $P-k \geq 3n-k$ free macro-slots in the forward period, since $P \geq 3n\tau$. Each of these $3n-k$ possible offset values translates into $3n-k$ positions of messages in the backward period. All these positions are separated by at least τ slots. There are already k messages of size τ in the backward period. One such message can intersect at most 2 potential positions since they are disjoint intervals. Therefore amongst the possible $3n-k$ positions, there are at least $3n-k-2k$ which are without collision. Since $k < n$, $3n-k-2k \geq 1$, which proves that the algorithm terminates and find a (P, τ)-periodic affectation.

This algorithm works with a complexity $O(n^2)$, since for the k^{th} route we have to try at most $2k$ offsets before finding a correct one. We can test the $2k$ offsets of the backward period in time $O(k)$ by maintaining an ordered list of the intervals used by already set routes. □



This algorithm, contrarily to the previous one, may work well, even when the condition $P \geq 3n\tau$ is not true. In fact, experimental data in Subsection 3.1.4 suggests that the algorithm finds a solution on average when $P \geq 1.5n\tau$. Note that we also experimented with other greedy algorithms which do not use macro-slots, they work even better in practice but their theoretical upper bound is worse.

3.1.3 Exhaustive search

We now present an exhaustive search algorithm, which tries to set the offsets in all possible ways until it has found a (P, τ)-periodic affectation. Contrarily to the two previous algorithms, when it fails to find a solution, then it certifies there are no solution to PAZL.

We have n routes denoted by $\{0, \ldots, n-1\}$. A partial solution S is a partial function from $[n]$ to $\{0, 1, \ldots, P-\tau-1\}$ which sets a starting time for a subset of the routes $R(S) \subseteq [n]$, such that there are no collisions for these routes. A partial solution S' extends S, if S' is defined over one more route than S and this route has a larger starting offset than all routes of S: $R(S') = R(S) \cup \{r'\}$ and for all $r \in R(S)$, $S(r) + \tau \leq S'r'$. We define a tree whose nodes are partial solutions and such that the root is the empty partial solution and the children of a partial solution are the partial solutions which extend it. The solutions to our problem will be the leaves of depth n in the tree, therefore our algorithm is a depth-first search of the tree.

Remark that a node S with $|R(S)| = k$ can have as many as $(n-k)P$ children. Since the tree is of depth n, the tree may have as many as $n!P^n$ elements and while n is small, P may be large which makes its traversal intractable. Therefore we have to find cuts in the tree to avoid to explore it entirely and henceforth make the algorithm practical. Cuts correspond to the detection of subtrees which contain no solutions and can thus be skipped. We now propose three cuts, the first two being particularly useful when the network is loaded ($n\tau$ is not far from P).

1. We consider the number of slots which can be used by routes not yet fixed by a partial solution in the *forward period*. When we extend a solution into S by a route at offset m, then at most $(P-m)/\tau$ routes can still be set without collisions in the forward period. If that value is less than the number of routes which are not in $R(S)$, it is a failure and the algorithm backtracks.

2. For the next two cuts, we need to define the notion of the useful slots of a partial solution S in the *backward period*: a slot is said to be useful, if it is not used by a message set by S in the backward period and it belongs to an interval of at least τ such slots. Useful slots are positions which can be used when extending S. We will denote by $([a_i, b_i[)_{i \leq l}$ the ordered sequence of intervals of useful slots of S. Without loss of generality we can assume that all $a_i, b_i \leq l$. The number of messages of size τ which can be placed in the useful slots of S is thus $\sum_{i=0}^{l} (b_i - a_i)/\tau$. If that value is less than the number of routes which are not in $R(S)$, it is a failure and the algorithm backtracks. Notice that the list of intervals of useful slots and the value $\sum_{i=0}^{l} (b_i - a_i)/\tau$ can be maintained in constant time, since each time a route is added, we only need to split an interval of useful slots into at most two such intervals.

3. Let S_1 and S_2 be two partial solutions with $R(S_1) = R(S_2)$. Let US_1

(respectively US_2) be the set of useful slots of S_1 (resp. S_2). We say that S_1 *dominates* S_2 if there are more useful slots both in the forward and backward periods for S_1 than for S_2. Formally, the largest offset fixed in S_1 is smaller than the one in S_2 and $US_2 \subseteq US_1$. Remark that any valid sequence of extensions of S_2 (choosing offsets of routes in the complementary of $R(S_2)$) is also a valid sequence of extensions of S_1. Therefore if the tree rooted at S_2 contain a solution, then S_1 contains one too. Hence, in our exhaustive search of the tree of partial solutions, we can skip the tree rooted at S_2.

We now explain how we can detect some partial solutions which are dominated so that we do not explore their subtrees. Consider a partial solution S which we extend into S' by setting the offset of the route r to be the smallest possible. The offset of r in the backward period is $S'(r) + \lambda$ and we denote the end of the message before before by a. Hence all extensions of S into S'' such that $S'(r) < S''(r) < a + \tau - \lambda$ are dominated by S'. Therefore when computing the extension of S, we first build S' and then S'' with $S''(r) = a + \tau - \lambda$, skipping all values in between.

The third cut works well in conjunction with the first one since it makes the offsets grow quickly and which makes the first cut more likely to apply. A last cut could be implemented: compute for every route not in $R(S)$ the set of possible positions in the backward period and verify whether at least one is contained in the useful slots of S.

3.1.4 Experimental Results

The following experimental results help compare the three presented algorithms. Notice that both Greedy algorithm and the Shortest-Longest are polynomial time algorithm but are not always able to find a solution. On the other hand, the exhaustive search will find an optimal solution, if it exists, but works in exponential time. We will compare the performance of the algorithms in two different regimes: the routes are either short with regards to τ and P or unrestricted. We choose the parameter realistic with regard to our C-RAN context. The number of routes is at most $n = 20$, τ is equal to 2500 and the period is at most $3n\tau$ (otherwise the greedy algorithm always returns a solution).

TODO: Dire où on peut trouver le code (pas la peine de donner des infos sur la machine/compilateur car one ne fait pas d'étude de performances.

Short routes First we consider routes which are shorter than τ, that is a message cannot be contained completely in a single edge. This is typical in our context and we will consider graphs in which the values $\lambda(c_t, r_{t_i})$ are drawn uniformly between 0 and 700 slots. It corresponds to messages of approximately 1Mb, links of bandwidth 10Gbps and length less than 5km between the BBU and the RRH.

Our aim is to understand how well the algorithms are working when the network has a high load (when the load is low, the greedy algorithm always returns a solution). To do that we try to evaluate the minimal period for which

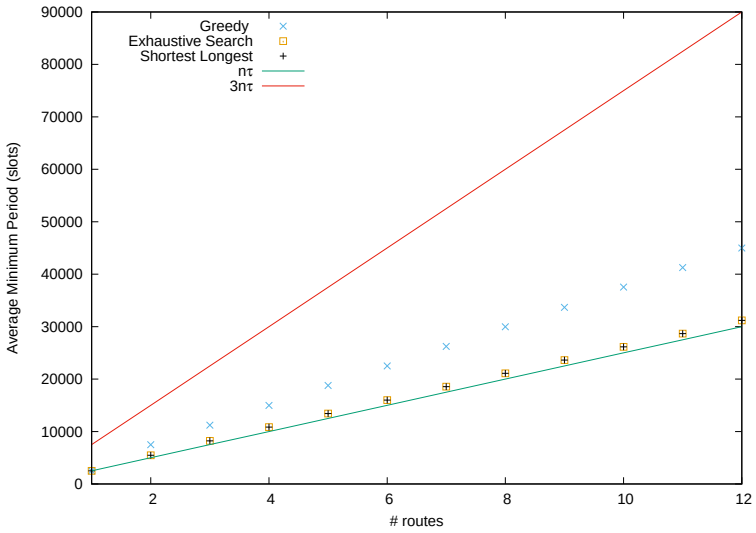


Figure 2: Minimum period averaged over 1000 random instances

a (P, τ) periodic affectation can be found by each algorithm. To that end we do a linear search on P, since a dichotomy would not work because of Lemma 1.

In our experiment, we generated 1000 random instances of PAZL for 1 to 12 routes. We measured the average minimal period, for each algorithm. We needed to stop our experiments at 12 routes since the exhaustive search was taking more than a second to finish on many instances. Note that the exhaustive search is optimal, that is it finds the best possible period for a given instance. We also drew the function $f(n) = n\tau$, which is an absolute lower bound on the period and $g(n) = 3n\tau$ which is the theoretical upper bound of the greedy algorithm.

First, we remark that the period found by the exhaustive search is only very slightly above the lower bound of $n\tau$, which means that, in this regime, it is very well justified to look for a solution without waiting time even for a highly loaded network.

The period needed by Shortest-Longest to find an optimal solution is determined by the difference between the longest and the shortest route and could be easily determined theoretically. The algorithm was expected to be good since the routes are short, but it turns out it always finds the optimal solution in our experiments. Therefore, we should use it in practical application, in this regime, instead of the exhaustive search which is much more computationally expensive.

Finally we remark that the greedy algorithm have an average period much lower than the theoretical upper bound of $3n\tau$. We made a linear regression on this value and with a correlation coefficient greater than $0,999$ we find a slope

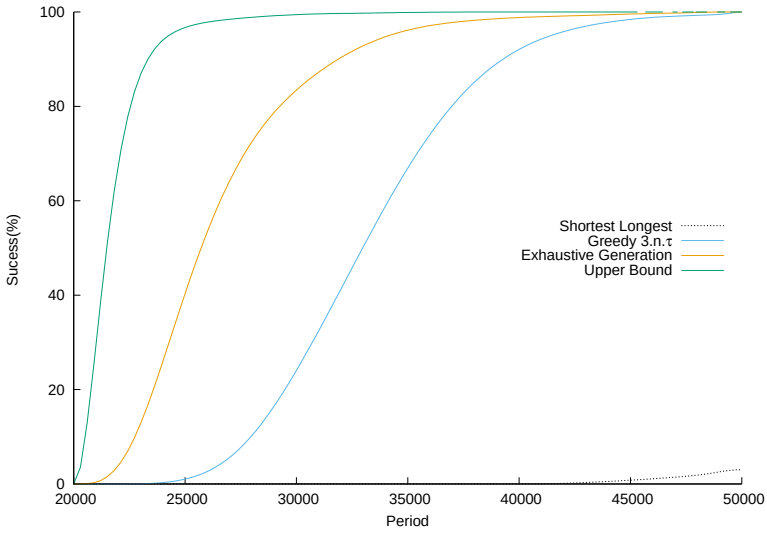


Figure 3: Success rate for 8 routes over 1000 random instances

of $1.53n\tau$.

Long routes We now want to understand the performance of these algorithms when the size of the routes are unbounded. When the routes are long, the cuts in the exhaustive search are less efficient and thus even for a small number of routes the algorithm may be impractical. To make the experimentations short enough, we have bounded the number of nodes of the tree of partial solutions the exhaustive search can visit by 10^9.

In this experiment we will fix the number of routes to 8 and the values $\lambda(c_t, r_{t_i})$ are drawn following an uniform distribution between 0 and 30000 slots (in the same range as the period). We measure for each algorithm its percentage of success, for periods from 20000 which is the minimum possible to 50000.

Since the exhaustive search can fail because there are no solutions or because it does not have enough time to finish its calculation. We represent both the success rate of the algorithm, and an upper bound on the success rate of any algorithm solving PAZL by combining its successes and its failures because of lack of time.

In this regime, the performances of Shortest-Longest are abysmal since it depends on the difference between the longest and the smallest route which is large here. On the other hand, the greedy algorithm has a performance not very different from the case of short routes, which is expected since it does not depends on the size of the route. TODO: Ajouter la moyenne pour la comparer aux routes courtes

When the period is less than 40000, the exhaustive search find many more solutions than the greedy algorithm which justifies its use. However for small period, less than 30000 the computation time is really important and without improvement it would not be possible to tackle larger problems with 10 to 20 routes. Moreover, examining the upper bound curve for small periods, we see that there are very few instances with a solution for PAZL. It means that with long routes and high load, looking for an affectation without waiting may be too restrictive. That is why we present algorithms for the general PALL problem in our next section. We will test them on 8 long routes and a period between 20000 and 30000 where, as we have shown here, there are often no easy to compute affectation without waiting time.

3.2 Allowing waiting times

3.2.1 Intro

In the previous section, we described some algorithm to solve an easier version of PALL, in which there is no waiting times. As we conclude, this problem cannot be solved when the period is too small. The C-RAN context gives us a fixed period, thus, some topologies with a lot of routes might be impossible to schedule with the previous algorithms. We propose to allow some message to wait in the target vertices (BBU), with the aim of giving an higher degree of freedom for the scheduling. In this section, the waiting times w_i are not necessarily null anymore. Thus, $MT(\mathcal{M})$ might be superior to T_{max} and it is necessary to minimize $MT(\mathcal{M})$.

3.2.2 A first approach

First, we imagined that prioritize the messages one the longest route could be a good idea. Thus, the following heuristic is suggested:

On the forward period, for the first way, the messages are scheduled so that they are following each others, from the one using the longest route, to the one using the shortest route. Thus, we set the m_{s_i} of the routes such that $t(c_s, r_{s_i}) = \tau.i$, if the routes r_i are ordered from the longest to the shortest.

On c_t, the algorithm is the following one:

Algorithm 1 Eligible route

1. To be eligible, a message needs to be able to come back on c_t at the current clock (if clock = 0, or no messages are able to come back at this date, take the first message able to come back).

2. Between the eligible messages, schedule first the one which has the lowest deadline.

3. For any route i, the deadline correspond to $m_{s_i} + T_{max} - t(c_s, r_{s_i})$. This deadline corresponds to the latest date at which a message is able to quit the target switch without exceed T_{max}.

Algorithm 2 Longest Shortest Greedy (LSG)

Input: routage graph $\mathcal{R}_\mathcal{C}$, period P, packet size τ
Output: $(P-\tau)-$periodic affectation of $\mathcal{R}_\mathcal{C}$

```
clock ← 0
for all route i in r_{s_i} sorted from the longest to the shortest do
    m_{s_i} ← clock − t(c_s, r_{s_i}) mod P
    clock ← clock + τ
end for
take i such that r_{t_i} is the eligible route (Algorithm 1)
w_i ← 0
clock = λ(r_i) + t(c_t, r_{t_i}) + τ
while there is a route which has not been scheduled do
    take i such that r_{t_i} is the eligible route (Algorithm 1)
    if the messages is able to come back before clock then
        w_i = clock − t(c_t, r_{t_i}) − (m_i + λ(r_i))
        clock ← clock + τ
    else
        w_i ← 0
        clock = λ(r_i) + t(c_t, r_{t_i}) + τ
    end if
end while
```

This policy is split in two steps. First, we choose arbitrary a sending order for the routes, and we set the offsets m_i such that the messages are following each others in c_s. Then, the way back is determined considering the time at which the messages reach their target. This greedy policy determines the w_i, such that the messages does not collide on c_t. One can expect that this policy prioritise the longest route, and thus helps to minimise $PT(s_i)$. Indeed, the messages on the longest routes will be less buffered in their target than the messages using the shortest ones. The goal is to balance the process times such that if there is a very long route, the message on it will not be buffered at all, and if there is a very short route, the message can be buffered, but not enough to have a process time greater than twice the delay on the longest route. In other words, if i is the very long route, and j the very short one, we want $w_i = 0$, and $PT(s_i) = 2.\lambda(r_{s_i}) \geq 2.\lambda(r_{s_j}) + w_j = PT(s_j)$. This greedy policy does not ensure that, but this is the main idea of the algorithm. Furthermore, this heuristic does not take into account the period at all, thus, a solution given by this algorithm can satisfy the constraint $MT(\mathcal{M}) \leq T_{max}$, but this solution might need a too large period.

3.2.3 Sending order

We tried to look at the behaviour of the greedy policy with different kind of sending order for the messages.

We tried the five following sending policy, each ones determines an order in which the messages will be sent, but the messages are still following each others in c_s.

The different sending policy that we tested are the following :

1. Generate X random permutations for the routes and tries the greedy heuristic on the way back until we get an available (P, τ) periodic affectation.

2. Send the messages from the one using longest to the one using shortest route (this is the sending order used in the previous heuristic).

3. Send the messages from the one using shortest to the one using longest route

4. Send the messages from the one having the longest arc between c_t an the target to the one having the shortest one.

5. Send the messages from the one having the shortest arc between c_t an the target to the one having the longest one.

The fifth order gives us the same policy than Shortest-Longest, the algorithm described in last section. This sending order gives us some (P, τ) periodic affectation in which all the $w_i = 0$, but needs a huge period if the difference between the longest and the shortest route is large (see proposition 2).

For the fourth order, in the case in which all the weight on the arcs between c_s and the sources are the same, the algorithm gives optimal solutions. Since we can simplify the weight on the arcs between c_s and the sources, we consider only the delay between c_t and the targets. Order the routes such that $\lambda(r_{s_i}) > \lambda(r_{s_j}), i, j \in n, i < j$, i.e. the route 1 is the longest one, and the route n is the shortest. The messages are sent in the forward period from 0 to $n-1$, that is $m_{s_i} = \tau * n$. The first message has totally reach the backward period at time $2.\lambda(r_{s_0}) + \tau$. The process time of the first message is $PT(s_0) = 2.\lambda(r_{s_0})$. The second message is eligible in the backward period at time $\tau + 2.\lambda(r_{s_1})$. Thus, the waiting time of the second route is $w_{s_1} = 2.\lambda(r_{s_0}) + \tau - (\tau + 2.\lambda(r_{s_1})) = 2.(\lambda(r_{s_0}) - \lambda(r_{s_1}))$. Thus $PT(s_1) = 2.\lambda(r_{s_1}) + 2.(\lambda(r_{s_0}) - \lambda(r_{s_1})) = 2.\lambda(r_{s_0}) = PT(s_0)$. By induction, $PT(s_i) = PT(s_0), \forall i \in n$. Consequently, for this case, the heuristic is optimal, because the longest route has no waiting time all the other routes has the exact same process time, thus, $MT(M) = PT(s_0) = 2.\lambda(r_{s_0})$.

Particular case corresponding to C-RAN One of the closest approach to real networks is the case in which all the weight on the arcs going from c_t to targets are the, this algorithm gives an optimal solution. Indeed, all the same weight can be simplified by 0, thus the scheduling in the forward and backward period is the same, that is, all message following each others, so the period is minimal : $n.\tau$, where n is the number of routes. Furthermore, all the waiting times are set to 0 i.e. $w_i = 0, \forall i \in n$, thus, $MT(\mathcal{M})$ is equal to twice the longer delay of the routes, that is not alterable.

3.2.4 A better Algorithm : periodicity

For the rest of the section, we look at the different algorithm suggested to improve the second part of the scheduling, after having set the sending order. The previous algorithm is a greedy heuristic, which does not take the period

into account. Thus, some solutions given by this algorithm might need a too large period an thus, not be available. We propose an improved version of the algorithm, still greedy, but which consider the period P.

We first remember the date at which the first messages starts to come back through c_t. Let us note bp this date. The greedy policy is the same until the message are scheduled in the period between bp and $bp + P$. When a message is scheduled, we update a list of free intervals in the backward period. Once $clock > bp + P$, we consider that we are in the second period, and thus search a free interval after $clock$, large enough to put a message.

This policy can significantly increase w_i, but ensures a periodic results. This policy is strictly better than the previous one.

3.2.5 Another better Algorithm than LSG: optimal on PT

The problem PALL is very similar to a scheduling problem, combined to the periodic aspect. Thus, the article "A fast algorithm for single processor scheduling", by Barbara Simons [17] gave us an optimal algorithm to schedule some jobs which has a release time, a deadline and the same execution time. We consider that the messages are the jobs, the release times are the date at which the messages are able to come back on c_t, and the deadlines are the latest date at which a message have to leave c_t to comeback before $m_i + T_{max}$. This algorithm gives in $\mathcal{O}(n^2 \log n)$ a solution with the earliest possible completion time, if it exists.

This algorithm uses the exact same greedy scheduling than LSG. Nevertheless, when it tries to schedule a message that is not able to totally pass c_t before it's deadline (let us call *crisis message this message), the algorithm call a subroutine that reschedule the previous message such that the crisis message will be scheduled earlier.*

3.2.6 Mix : Simons periodic

3.2.7 A fixed parameter tractable solution

In this part we try to give a fixed parameter tractable algorithm to solve the PALL problem with parameter n the number of routes. This is justified since n is always very small (less than 20*) while P, or τ may be very large.*

Theorem 3. *There is an algorithm which solves PALL on star topologies in* $O()$.

Proof. In this algorithm we will guess a certain number of values related to an affectation and then compute in polynomial time an affectation which satisfies these properties if it exists. Then the FPT algorithm will be to try every possible set of values and use the polynomial time algorithm.

We assume we are given the order at which the routes goes through the and the order at which th □

4 Conclusion

References

[1] *C. Mobile, "C-RAN: the road towards green RAN,"* White Paper, ver, *vol. 2, 2011.*

[2] *Y. Bouguen, E. Hardouin, A. Maloberti, and F.-X. Wolff,* LTE et les réseaux 4G. *Editions Eyrolles, 2012.*

[3] *G. P. A. W. Group, "View on 5G architecture,"* White Paper, ver, *vol. 1.0, 2016.*

[4] *N. Finn and P. Thubert, "Deterministic Networking Architecture," Internet-Draft draft-finn-detnet-architecture-08, Internet Engineering Task Force, Sept. 2016. Work in Progress.*

[5] *"Time-sensitive networking task group."* `http://www.ieee802.org/1/pages/tsn.html`. *Accessed: 2016-09-22.*

[6] *W. Howe, "Time-scheduled and time-reservation packet switching," Mar. 17 2005. US Patent App. 10/947,487.*

[7] *B. Leclerc and O. Marcé, "Transmission of coherent data flow within packet-switched network," June 15 2016. EP Patent App. EP20,140,307,006.*

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

[9] *R. Borndörfer, A. Eisenblätter, M. Grötschel, and A. Martin, "Frequency assignment in cellular phone networks,"* Annals of Operations Research*, vol. 76, pp. 73–93, 1998.*

[10] *T. Erlebach and K. Jansen, "The complexity of path coloring and call scheduling,"* Theoretical Computer Science*, vol. 255, no. 1, pp. 33–50, 2001.*

[11] *C. Strotmann,* Railway scheduling problems and their decomposition*. PhD thesis, PhD thesis, Universität Osnabrück, 2007.*

[12] *B. Zhou, "Multiple circular colouring as a model for scheduling," 2013.*

[13] *X. Zhu, "Circular chromatic number: a survey,"* Discrete mathematics*, vol. 229, no. 1, pp. 371–410, 2001.*

[14] *X. Zhu, "Recent developments in circular colouring of graphs," in* Topics in discrete mathematics*, pp. 497–550, Springer, 2006.*

[15] *I. Holyer, "The np-completeness of edge-coloring,"* SIAM Journal on computing*, vol. 10, no. 4, pp. 718–720, 1981.*

[16] *D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," in* Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 681–690, ACM, 2006.*

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