Deterministic Scheduling of Periodic Messages in a Network

immediate

***Abstract*—A recent trend in mobile networks is to centralize the processing units which were attached to antennas in distant data-centers. The main challenge is to guarantee that the latency of the periodic messages sent from the antennas to their processing units and back is minimal. We show that traditional statistical multiplexing is bad at ensuring a low latency. Hence, we propose in this article to use a deterministic scheme which allows to send periodic messages without collisions in the network and thus without buffering and additional latency. We give several algorithms to compute such schemes for a common topology where one link is shared by all antennas. Some algorithm solves the problem when the routes are small or the load is small. Whe the parameters are unconstrained, and some buffering is allowed in the processing units, we propose an algorithm (PMLS) adapted from a classical scheduling method. The experimental results shows that even under full load, most of the time PMLS finds a sending scheme with no latency.**

I. Introduction

Next generations of mobile network architectures evolve toward centralized radio network architectures called C-RAN, or Cloud Radio Access Network, to reduce energy consumption costs [1] and more generally the total cost ownership. The main challenge for this type of architecture is the latency in the transfer process that is an important factor for dimensioning the network. The latency is measured between the sending of a message by the Remote Radio Heads (RRHs) on the field and the receptions of the answers, computed by real-time virtualized network functions on Baseband Units (BBUs), in the cloud. For example, even if next radio protocols are less stringent, some standards requires meeting time constraints for functions like HARQ (Hybrid Automatic Repeat reQuest) that needs to be processed in 1 to 10ms depending on targeted services [2]. The specificity of the C-RAN context is not only the latency constraint, but also the periodicity of the data transfer between RRHs and BBUs.

In this article, we study mechanisms to deal with the latency and periodicity constraints in the network located between the BUUs and the RRHs called fronthaul. Statistical multiplexing even with a large bandwidth does not comply with the latency requirements. Therefore, the current solution [3], [4] is to use dedicated circuits. Each end-point (RRH on one side, BBU on the other side) is connected through direct fiber or full optical switches. This architecture is very expensive and hardly scales in the case of a mobile network composed of about 10,000 base stations. Here we propose to use a deterministic approach to schedule our periodic messages. This approach has gained some traction recently: Deterministic Networking is under standardization in IEEE 802.1 TSN group [5], as well at IETF DetNet working group [6]. Several patents on concepts and mechanisms for DetNet have been already published, see for example [7], [8]. The aim is to operate a C-RAN on a low-cost shared switched network.

Let us expose briefly our model: the network topology is modeled by a directed weighted graph and a set of paths (routes) from source nodes (RRHs) to target nodes (BBUs). Time is discretized and a unit of time corresponds to the transmission of a minimal unit of data over the network. The process must be predictable (i.e. the date of arrival of packets is known beforehand). For simplicity, in this article we assume that the process is periodic. That is, if we look at the network at times t and $t + P$ where P is the period, all messages are at the same position in the network. Moreover, we want to avoid any buffering in internal nodes of the graph. Hence we need to design a periodic process with no collision between messages. To ensure that this property is always true we propose a deterministic process. We have two sets of parameters that we can choose when building a periodic sending process, called a **periodic assignment** in this article: the time at which each packet is sent by a RRH in each period and the waiting time in the BBU before the answer is sent back to the RRH. When building a periodic assignment, we must take into account the periodicity which makes many scheduling methods inapplicable. Not only a message 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. Moreover the latency must be minimized, that is the time between the emission of a message and the complete return of its answer. It means that the only buffering we are allowed – the waiting time before sending back the answer– must be small, in particular when the route is long.

The problem we focus on here may look like wormhole problems [9], but here we want to minimize the time lost in buffers and not just to avoid deadlocks, and we also need to deal with the periodicity. Several graph colorings have been introduced to model similar problems such as the allocation of frequencies [10], bandwidths [11] or routes [9] in a network or train schedules [12]. Unfortunately, they do not take into account the periodicity of the scheduling and the associated problems are already NP-complete. The only coloring which incorporates some periodicity is the circular coloring [13], [14], [15] but is not expressive enough to capture our problem.

Our main contributions are the following. In Sec. II we

propose a model of the network and the periodic sending of messages along its routes and we introduce the star routed network we study in this article. We formalize the problem of finding a periodic assignment for sending messages without buffering (PAZL) or with buffering in the processing unit only (PALL) and shows that they are NP-hard even to approximate. In Sec. III, we propose two algorithms solving PAZL when the routes between RRHs and BBUs are small or when the load is light. Using a compact representation of an assignment, we show that PAZL is fixed parameter tractable when parametrized by the number of antenna on star routed networks. Finally in Sec. IV, we introduce a family of algorithms to solve PALL and our experimentation allows to select one which works extremely well even in very loaded networks. In particular we show that the deterministic communication schemes we design vastly outperform the traditional stochastic multiplexing with regard to latency.

II. Model and Problems

A. Network modeling

The network is modeled as a directed graph $G = (V, A)$. Each arc (u, v) in A is labeled by an integer weight $\omega(u, v)$ which represents the time slots taken by a signal to go from u to v using this arc. A **route** r in G is a path, that is a sequence of consecutive arcs $a_0, \ldots, a_{k-1}$, with $a_i = (u_i, u_{i+1}) \in A$. The **latency** of a vertex u_i in a path r, with $i \geq 1$, is defined by $\lambda(u_i, r) = \sum_{0 \leq j < i} \omega(a_j)$. We also define $\lambda(u_0, r) = 0$. The length of the route r is defined by $\lambda(r) = \lambda(u_k, r)$. We denote by $\mathcal{R}$ a set of routes, the pair $(G, \mathcal{R})$ is called a **routed network** and represents our telecommunication network. The first vertex of a route models an antenna (RRH) and the last one a data-center (BBU) which processes the messages sent by the antenna.

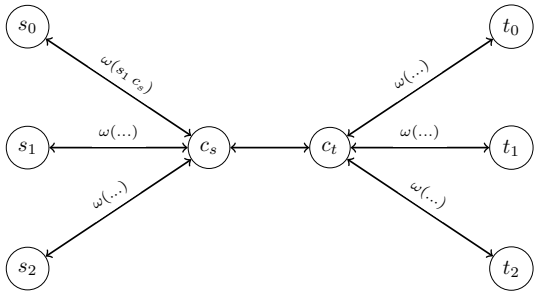
B. Messages dynamic

Consider a route r, if a message is sent at time m from s the first vertex of r then it will arrive at vertex v in r at time $m + \lambda(v, r)$. In our problem, we send one message on each route periodically and we denote the period by P. Periodicity means that if we look at the messages going through an arc between time 0 and $P - 1$, then we would observe the same messages in the same order and position between time kP and $(k + 1)P - 1$ for all k. Therefore, we will always consider a period of P discrete slots of time at each arc to characterize the working of the network. We define the first time slot at which a message sent from the first vertex of r will reach a vertex v in r by $t(v, r) = m + \lambda(v, r) \mod P$.

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 $[t(v, r)]_{P,\tau}$ the set of time slots used by route r at vertex v in a period P, that is $[t(v, r)]_{P,\tau} = \{t(v, r) + k \mod P \mid 0 \leq k < \tau\}$. Usually P and τ will be clear from the context and we will denote $[t(v, r)]_{P,\tau}$ by $[t(v, r)]$. Let r_1 and r_2 be two routes, on which messages are sent at time m_1 and m_2 in their first vertex. We say that the two routes have a **collision** if they share an arc of first vertex v and $[t(v, r_1)] \cap [t(v, r_2)] \neq \emptyset$.

C. The star routed network

Let us define a family of simple symmetric routed networks. The graph G has two sets of vertices, $S = \{s_0, ..., s_{n-1}\}$ and $T = \{t_0, ..., t_{n-1}\}$ of cardinality n and two special nodes: the central source node c_s and the central target node c_t. There is an arc between c_s and c_t and for all i, there is an arc between s_i and c_s and between t_i and c_t. All the symmetric arcs are also in the graph with the same weights. The forward routes are the directed paths $r_i = (s_i, c_s, c_t, t_i)$ and $\rho(r_i) = (t_i, c_t, c_s, s_i)$ which defines a symmetric routed network. The symmetric routed networks $(G, \{r_i, \rho(r_i)\}_{i<n})$ is called a **star routed network**.



Since the central arc appears in every route, its value does not matter when considering the process time of an assignment. Moreover an assignment without collisions is also an assignment without collisions if the weight of the central arc is set to 0, therefore we assume from now on it is indeed 0.

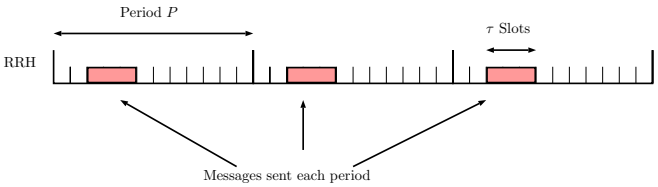
Collisions between messages can only appear in the arc (c_s, c_t) on the way forward and in the arc (c_t, c_s) on the way back. The flow of messages in a star routed network is completely described by their repartition in two time windows of size P that we call the **forward period** (at (c_s, c_t)), and the **backward period** (at (c_t, c_s)).

D. Periodic assignment for low latency

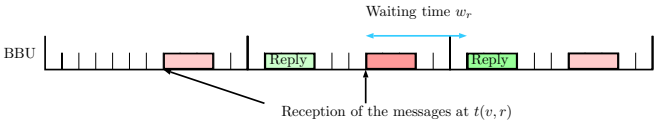
In the context of cloud-RAN applications, we need to send message from a RRH u to a BBU v and then we must send the answer from v back to u. We say that a routed network $(G, \mathcal{R})$ is **symmetric** if the set of routes is partitioned into the sets F of forward routes and B of backward routes. Moreover there is a bijection ρ between F and B such that for any forward route $r \in F$ with first vertex u and last vertex v, the backward route $\rho(r) \in B$ has first vertex v and last vertex u. In all practical cases the routes r and $\rho(r)$ will be the same with the orientation of the arcs reversed, which corresponds to bidirectional links in the networks, but we need not to enforce this property.

A (P,τ)**-periodic assignment** of a routed network $(G,\mathcal{R})$ is a set of two sequence $\mathcal{M} = \{(m_0,\dots,m_{n-1}),(m_{\rho_0},\dots,m_{\rho_{n-1}})\}$ of n integers that we call **offsets**, with n the number of routes in $\mathcal{R}$. The number m_i (respectively m_{ρ_i}) represents the time at which a message is emitted at the first vertex of the route r_i (respectively r_{ρ_i}).A (P,τ)-periodic assignment of a $(G,\mathcal{R})$ symmetric routed network represents the sending of a message and of its answer. *No pair of routes must have a collision* in a (P,τ)-periodic assignment. This assignment represents the following process: first a message is sent at u, through the route $r \in F$, at time m_r.

TODO: mettre le lemme sur la monotonie sans preuve?



This message is received by v the last vertex of r at time $t(v,r)$. It is then sent back to u on the route $\rho(r)$ in the same period at time $m_{\rho(r)}$ if $m_{\rho(r)} > t(v,r)$, otherwise at time $m_{\rho(r)}$ 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 $w_r = m_{\rho(r)} - t(v,r)$ if $m_{\rho(r)} > t(v,r)$ and $w_r = m_{\rho(r)} + P - t(v,r)$ otherwise.



Thus, the whole process time for a message sent on the route r is equal to $PT(r) = \lambda(r) + w_r + \lambda(r)$. 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 problems we consider in the rest of the article. Finally, we could add to the process time the computation time a BBU needs to deal with one message, but it can be encoded in the weight of the last arc leading to the BBU and thus we need not to consider it explicitly in our model.

The **maximum process time** of the assignment $\mathcal{M}$ of $(G,\mathcal{R})$ is defined by $MT(\mathcal{M}) = \max_{r\in\mathcal{R}} PT(r)$. We consider the following decision problem.

Periodic Assignment for Low Latency (PALL)

Input: A symmetric routed network $(G,\mathcal{R})$, the integers P, τ and T_{max}.

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

We show in the long version of this article that this problem is NP-hard. In Sec. IV we will study heuristics which try to solve the search version of PALL (computing an assignment), also denoted by PALL for simplicity.

One can consider a simpler version of PALL. We ask for a (P,τ)-periodic assignment **with all waiting times equal to** 0 and we call this restricted problem **Periodic Assignment for Zero Latency** or PAZL. We consider PAZL since it is simpler to get theoretical results and good algorithms than for PALL. Moreover, as we show in our experimentations of Sec.III-B, this problem can often be solved positively (albeit less often than the general problem). Finally, a solution to PAZL is simpler to implement in real telecommunication networks, since we do not need to implement any buffering at all.

One may consider a variant of PAZL where the central link is unidirectional, that is collisions can happen between messages going from c_s to c_t and messages going from c_t to c_s. This problem can be shown to be NP-complete by a reduction from the subset sum problem as it is done for a similar problem of scheduling pair of tasks [16]. We conjecture that the problem PAZL is also NP-complete, but we have yet not been able to prove it. On the other hand we show positive results: when the period is large or when the routes are short there is always a solution to PAZL and it can be found in polynomial time. We then give an exponential time exhaustive search algorithm which always finds a solution if there is one.

III. The Star Routed Network: No Waiting Time

A. Solving PAZL

In this subsection, we deal with the problem PAZL. When the waiting times are zero, $MT(\mathcal{M})$ is equal to twice the length of the longest route, thus T_{max} is not relevant anymore. For a route r, choosing the offset m_r also sets the offset of the route $\rho(r)$ to $m_{\rho(r)} = m_r + \lambda(r) \mod P$. Consider an assignment $(m_0,\dots,m_{n-1})$ of a star routed network and let $m'_i = m_i - t(c_s,r_i) \mod P$. Then $\{m'_0,\dots,m'_{n-1}\}$ is an assignment of the same star routed network where the weights of the arcs from s_i to c_s are 0 for all i. Therefore we assume from now on that the *weight of the arcs from s_i to c_s are all equal to* 0.

Shortest-longest policy: We first present a simple policy, which works when the period is large with regard to the weight of the routes. The messages 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_i the offset of r_i to $i\tau$. 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 1. *Let $(G, \mathcal{R})$ be a symmetric routed network, and let $n\tau + 2(\lambda(r_{n-1}) - \lambda(r_0)) \leq P$, then there is (P, τ)-periodic assignment of $(G, \mathcal{R})$ with all waiting times 0 which is given by Shortest-Longest in time $O(n \log(n))$.*

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

We may assume that $\lambda(r_0) = 0$, since removing $\lambda(r_0)$ from every arc (c_t, t_i) does not change the order on the length of the routes nor the collisions between messages. Since $\lambda(r_0) = 0$, by hypothesis we have $n\tau + 2\lambda(r_n) \leq P$ which implies that $[t(c_t, r_i)] = \{2\lambda(r_i) + i\tau, \ldots, 2\lambda(r_i) + (i+1)\tau - 1\}$. Since $\lambda(r_i) \leq \lambda(r_{i+1})$ by construction, we have $2\lambda(r_i) + i\tau - 1 < 2\lambda(r_{i+1}) + (i+1)\tau$ which proves that there are no collision on the backward period.

The complexity of the algorithm is dominated by the sorting of the routes in $O(n \log(n))$. □

If the period is slightly smaller that the bound of Proposition 1, a collision will occur on the first route in the backward period. Hence, this policy is not useful even as a heuristic for longer routes as confirmed by the experimental results of Subsection III-B.

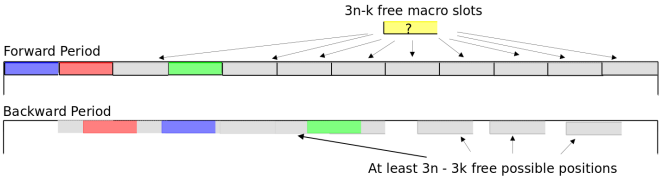
Greedy algorithm: We propose a greedy algorithm to build a (P, τ)-periodic assignment, which provably works when the period is large enough with regard to the number of routes and size of messages. In other words, we can always find a solution when the network is far from saturated. This algorithm defines an upper bound for the period. As a reminder, the lower bound is $n\tau$, that is, the size taken by the messages to be emitted.

Proposition 2. *Let $(G, \mathcal{R})$ be a symmetric routed network, and let $3n\tau \leq P$, then there is (P, τ)-periodic assignment of $(G, \mathcal{R})$ with all waiting times 0 which can be found 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 assignment 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 assignment.

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 III-B suggests that the algorithm finds a solution on average when $P \geq 2n\tau$. 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.

An FPT algorithm

In this section we show how every assignment without waiting time can be put in a canonical form. We use that to provide an algorithm which always finds an assignment if there is some, in fixed parameter tractable (FPT) time with parameter n the number of routes (for more on parametrized complexity see [17]). This is justified since n is small in practice and the period P is large.

Let $(G, \mathcal{R})$ be a star routed network and $\mathcal{M}$ a (P, τ)-periodic assignment. A set of routes S is **coherent** if for all $r \in \mathcal{R}$, $r \in S$ if and only if $\rho(r) \in S$. We say that a coherent set $S \subseteq \mathcal{R}$ is **compact** for the assignment $\mathcal{M}$ if there is a route $r_0 \in S$ such that the following holds: for all coherent subsets $S' \subset S$ with $r_0 \notin S'$, if we remove 1 from all offsets of routes in S' then there is a collision with a route of $S \setminus S'$. We say that $\mathcal{M}$ is compact if $\mathcal{R}$ is compact for $\mathcal{M}$.

Proposition 3. *Let $(G, \mathcal{R})$ be a star routed network. If there is a (P, τ)-periodic assignment of $(G, \mathcal{R})$, then there is a compact (P, τ)-periodic assignment of $(G, \mathcal{R})$.*

Proof. Consider $\mathcal{M}$ a (P, τ)-periodic assignment of $(G, \mathcal{R})$. Let r_0 be an arbitrary route of $\mathcal{R}$, and let $COMP = \{r_0\}$. Now we apply the following algorithm to $\mathcal{M}$ and $COMP$ while $COMP$ is not equal to $\mathcal{R}$. While there is no collision, remove 1 (modulo P) from all offsets of routes in $\mathcal{R} \setminus COMP$. Then choose a route r in $\mathcal{R} \setminus COMP$ which would have a collision with a route r' of $COMP$ if one is subtracted from its offset. If r' is a forward route, let $COMP = COMP \cup \{r, \rho(r)\}$ otherwise $COMP = COMP \cup \{r, \rho^{-1}(r)\}$.

We prove by induction that $COMP$ is compact for $\mathcal{M}$ at every step of the algorithm. At the beginning $|COMP| = 1$

and the property is trivially satisfied. Then we assume that $COMP$ is compact and that we add to it $\{r, \rho(r)\}$ at some step of the algorithm. W.l.o.g we assume that it is the offset of r which cannot be decremented without collision. Consider $S \subseteq COMP$, if S contains an element different from r and $\rho(r)$ by induction hypothesis we cannot decrement the offsets of S without collision. If $S = \{r, \rho(r)\}$ by construction, we cannot decrement the offset of r.

Finally, there was no collisions between routes at the beginning and since we modify $\mathcal{M}$ only if it creates no collision, the assignment we obtain at the end has no collisions between routes. □

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

Theorem 1. $PALL \in \mathsf{FPT}$ *when parametrized by the number of routes.*

Proof. Let $(G, \mathcal{R})$ be a star routed network and let $\mathcal{M}$ be a (P, τ)-periodic assignment of $(G, \mathcal{R})$. First remark that for a given assignment and a route r_0 with offset m, by removing m to all offsets, we can always assume that its offset is zero. Therefore we need only to consider all *compact assignments* with an *offset* 0 for the route r_0. We now evaluate the number of compact assignments and prove that it only depends on n the number of routes which proves the theorem. We give a way to build a compact assignment $\mathcal{M}$ by determining its offsets one after the other. We fix an arbitrary total order on $\mathcal{R}$). First a route r_0 is chosen arbitrarily and its offset set to 0. Then at each step, if the offsets of $S \subseteq \mathcal{R}$ have been chosen, we select the smallest route r in S for the order. Then we select a route in $r' \in \mathcal{R} \setminus S$ and set its offset such that if we remove 1 then r' collides with r. Note that if r is a forward route (resp. a backward route) then r' is also a forward route (resp. a backward route). We can also decide to not use route r. At a given step of the algorithm, if $|S| = 2i$, we have $n - i$ choices of routes to select and the value of the offset is determined by the values of the offsets of routes in S. Therefore there at most $n!$ different compact assignments with offset r_0 fixed to 0.

An algorithm to solve PALL build every possible compact assignment as described here, and test at each step whether there is a collision which can be done in time linear in the size of $(G, \mathcal{R})$. Therefore $PALL \in \mathsf{FPT}$. □

We call the algorithm described in Theorem 1 **Exhaustive Search of Compact Assignments (ESCA)**. To make it more efficient in practice, we make cuts in the search tree used to explore all compact assignments. Consider a set of k forward routes whose offsets has been fixed at some point in the search tree. We consider the times at which the messages of these routes cross the central arc. It divides the period into $[(a_0, b_0), \ldots, (a_{k-1}, b_{k-1})]$ such that the central arc is free only during the intervals (a_i, b_i). Therefore at most $\sum_{i=0}^{k} \lfloor (b_i - a_i)/\tau \rfloor$ forward routes can still use the central arc. If this value is more than $n - k$, it is not possible to create a compact assignment by extending the one on S and we backtrack in the search tree. We do the same cut for the backward routes.

B. Experimental evaluation

The following experimental results compare the three presented algorithms. Notice that both Greedy algorithm and Shortest-Longest are polynomial time algorithm but are not always able to find a solution, depending on P, τ, n and the size of the routes. On the other hand, the exhaustive search finds an optimal solution if it exists, but works in exponential time. We compare the performance of the algorithms in two different regimes: routes are either short with regard to τ, or unrestricted. From our C-RAN context we choose the following parameters: the number of routes is at most $n = 20$, τ is equal to 2500. Moreover the period is always larger than $n\tau$ otherwise no assignment is possible and smaller than $3n\tau$ otherwise the greedy algorithm always returns a solution.

We define the load of the network as $\frac{n\tau}{P}$ (here a value between one and three), it is the proportion of time slots used by messages on the central arc. In our experiments we will try to understand how our algorithms work with regards to the load. To change the load, we fix the parameters τ and n and modify the period P, which allows for a smoother control of the load and does not change the run time of our algorithms.

a) Short routes: First we consider routes which are shorter than τ, that is a message cannot be contained completely in a single arc which is very common for our application. We generate star routed networks in which the weights of the arcs (c_t, 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 under high load. To do that we try to evaluate the minimal period (that is the highest possible load) for which a (P, τ)-periodic assignment can be found by each algorithm.

In our experiment, we generate 1000 random instances of PAZL for 1 to 12 routes. We measure the average minimal period, for each algorithm. Remind that the exhaustive search always finds the best possible period for a given instance. The lower and upper bound $n\tau$ and $3n\tau$ are also represented.

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 the Shortest-Longest algorithm to find an optimal solution is a function of the difference between the longest and the shortest route and its average could be easily

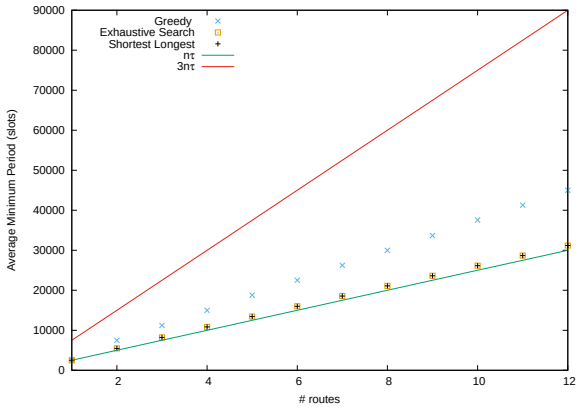


Fig. 1. Minimum period averaged over 1000 random instances

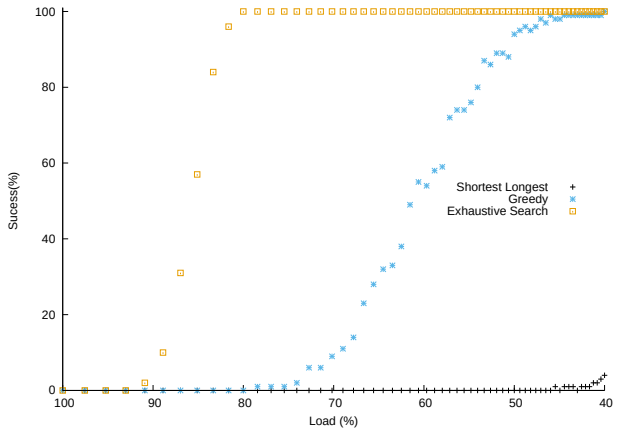


Fig. 2. Success rate for 8 routes over 1000 random instances

determined theoretically. The algorithm was expected to find a period close to the optimal one since the routes are short. It turns out it always finds the optimal period in our experiments. Therefore, we should use it in practical applications with short routes, instead of the exhaustive search which is much more computationally expensive.

Finally we remark that the greedy algorithm has 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 of $1.53n\tau$.

b) Long routes: We now want to understand the performance of these algorithms when the size of the routes is unbounded. In this experiment we fix the number of routes to 8 and the weights of the arcs (c_t, 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 to 50000 which corresponds to a load of 100% down to 40%.

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 so far from the case of short routes, which is expected since it does not directly depend on the size of the route. When repeating the experiments of the previous section on routes of arbitrary length, we find that on average the greedy algorithm finds a solution if $P > 1.89n\tau$.

When the load is larger than 50% (period less than 40000), the exhaustive search finds more solutions than the greedy algorithms which justifies its use. However, for load larger than 80% (period less than 25000) there are many instances for which there are no solutions to PAZL. It means that with long routes and high load, looking for an assignment without waiting time is far 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 load between 100% and 80% for which, as shown here, there are often no assignment without waiting times.

The computation time of the exhaustive search is in $O(n!)$, hence we need to study its scalability when n grows. In the following experiment, the weights of the arcs (c_t, t_i) are drawn following an uniform distribution between 0 and 30000 slots. We fixed a period of $n\tau + 1000$. The following table shows the time before the exhaustive search ends, for 6 to 21 routes, averaged on 100 random star routed networks. This shows that for less than 20 routes, which corresponds to all current topologies, the algorithm is efficient enough, but on more routes we should improve it further.

n	6	11	16	21
Time (s)	3.10^{-5}	6.10^{-4}	6.10^{-2}	15

IV. The Star Routed Network: Minimizing Latency

In this section, we still work on star routed networks, but we consider again the more general PALL problem. It means that the messages can wait in the target vertices (BBUs) in order to find solutions when there are none without waiting time. We allow the process times PT of the routes to be greater than twice the weights of the routes, but it should less than T_{max}.

A. Special cases

Often in real networks, the length of the routes are not arbitrary and we may exploit that to solve PALL easily. For instance if all the weights on the arcs (c_t, t_i) are the same, we can replace them all by 0 and substract this weight to T_{max}. It corresponds to a situation where all the BBUs are in the same datacenter. The assignment in that case is trivial, just

send all messages so that they follow each other without gaps in the central arc (c_s, c_t). Since the arcs (c_t, t_i) are of weigth 0, all messages will go through (c_t, c_s) on their way back in the same order and thus will not collide.

Another assumption would be that all weigths of the arcs (s_i, c_s) are the same.

Theorem 2. *Let $(G, \mathcal{R})$ be a symmetric routed network with n routes and let $P \geq n\tau$. If there is a constant c such that $\forall i < n$, $\omega(s_i, c_s) = c$, then there is a (P, τ)-peridoic assignment with process time $2 \times \max_{r \in \mathcal{R}} \lambda(r)$ and it can be built in time $O(n)$.*

Proof. We can simplify the weights on the arcs (s_i, c_s) by setting them all to 0. We find the route with maximal length, say that it is $\lambda(r_0)$. The idea is to set the waiting times of all routes so they behave exactly as the message in r_0.

The offset of the route r_i is set to $i\tau$, which ensures that there are no collision on the way forward (on the arc (c_s, c_t)) as soon as $P \geq n\tau$ which is the minimal possible period. We set the waiting time for the route i to $w_i = 2(\lambda(r_0) - \lambda(r_i))$. We can compute the time at which the message of the route r_i arrives at the vertex c_t on its way back: $t(c_t, \rho_{r_i}) = w_i + i\tau + 2\lambda(r_i)$ by replacing w_i by its value we obtain $t(c_t, \rho_{r_i}) = i\tau + 2\lambda(r_0)$ As a conclusion there are no collision on the arc (c_t, c_s) as soon as the period is larger than $n\tau$ (there are no gaps between the messages).

Now let us compute the process time of the route r_i defined by $PT(r_i) = w_i + 2\lambda(r_i)$. We obtain $PT(r_i) = 2\lambda(r_0)$ and thus the maximum process time of the assignment is also equal to $2\lambda(r_0)$. Finally the complexity is $O(n)$ since we have to find a maximum between the size of n routes and the computation of each w_i is done by a constant number of artihmetic operations. □

Remark that this assignment always exists and has no collision when $P \geq n\tau$. If for any pair of indices (i, j) the difference between the weights of the arcs (s_i, c_s) and (s_j, c_s) is bounded by d and the length of the longest route is l, then the maximum process time is bounded by $2l + d$.

B. A two stage approach

We can decompose any algorithm solving the PALL problem in two parts: first set the offsets of the forward routes and then knowing this information set the offset of the backward routes or equivalently the waiting times. The offsets of the forward routes will be chosen so that all messages have no collision on the central arc and such that there are no free slots inbeetween the end of a message on the central arc and the beginning of the next one. It is done to minimize the period needed to send the messages of the forward route. We tried the five following orders.

- Longest-Shortest on Routes (LSR): Decreasing order on the length of the routes.
- Shortest-Longest on Routes (SLR): Increasing order on the length of the routes.
- Longest-Shortest on last Arc (LSA): Decreasing order on the length of the arcs (c_t, t_i).
- Shortest-Longest on last Arc (SLA): Increasing order on the length of the arcs (c_t, t_i). This sending order yields a (P, τ) periodic assignment in which all the $w_i = 0$, if the period is large enough (see proposition 1).
- Random: A random order of the routes.

In the rest of the section we will study different methods to compute the offsets of the backward routes, once the offsets of the forward routes are fixed by one of the previous orders.

C. Greedy scheduling of backward routes

Assume that the offsets of the forward routes are fixed by some order. Consider a forward route r_i and the corresponding backward route $r_{\rho(i)}$. We define the **deadline** of $r_{\rho(i)}$ as $m_i + T_{max} - \omega(s_i, c_s)$, that is the latest time at which the message can go out of c_t such that $PT(r_i) \leq T_{max}$. We say that a backward route $r_{\rho(i)}$ is **eligible** at time t if $m_i + \lambda(r_i) + \omega(c_t, t_i) \leq t$, that is the message of the route $r_{\rho(i)}$ arrives at c_t before time t when $w_i = 0$.

The first algorithm we propose is a greedy algorithm which sets the offset $m_{\rho(i)}$ of the backward routes. It prioritizes the routes with the earliest deadline to best satisfy the constraint on the process time. Set $t = 0$ and repeat the following: find $s \geq t$ the first time for which there is an eligible route with its offset not fixed. Then amongst all eligible routes at time s choose the one with the smallest deadline, fix its offset to $s - \omega(c_t, t_i)$ and set $t = s + \tau$.

This algorithm does not take into account the periodicity. Say that $t_0 = t(c_t, r_{\rho_i} = m_{\rho_i} + t(c_t, r(\rho_i)$, where m_{ρ_i} is the smallest offset that the algorithm has fixed for a backward route. Then for all routes with $t(c_t, r_{\rho_i}$ smaller than $t_0 + P - \tau$, by construction, there are no collisions on the central arc. However, for routes with larger $t(c_t, r_{\rho_i}$, since we should consider everything modulo P, they may collide with any other route. Therefore we must adapt the greedy algorithm of the previous paragraph by finding $s \geq t$ the first time for which there is an eligible route with its offset not fixed and *such that there is no collision if a message go through the central arc at time s*.

Algorithm 1 Greedy deadline (**GD**)

Input: A routed network $(G, \mathcal{R})$, a period P, packet size τ, T_{max}, the offsets m_i
Output: (P, τ)-periodic assignment of $(G, \mathcal{R})$, or failure

```
H ← empty set //set of eligible routes
free_ intervals ← [0,P] //list of intervals of free slots
for all route r_i do
    deadline[r_i] ← m_i + T_max − ω(s_i, c_s)
    eligible_time[r_i] ← m_i + λ(r_i) + ω(c_t, t_i)
end for
while There is some non-assigned routes do
    if H is empty then
        t ← min_non_assigned(eligible_time)
        take r_i the route with eligible_time[r_i] = t

    end if
    r ← extract_min(H)
    t ← next_free_interval(free_intervals, t) //if there is no
    more free intervals of size τ, the algorithm fails
    w_i ← t - eligible_time[r_i]
    update(t,free_ intervals)
    t ← t + τ
    for all routes r_i with eligible_time[r_i] ≤ t do
        insert ( H,r_i).
    end for
end while
```

The function min_non_assigned(eligible_time) returns the lowest time in eligible_time, for which the corresponding route is not assigned yet. The function update(t,free_intervals) removes an interval of size τ beginning at t, which correspond to the message, from free_intervals.

The greedy algorithm can be made to work in time $O(n \log(n))$. To do that the set of eligible routes must be maintained in a binary heap to be able to find the one with smallest deadline in time $O(\log(n))$. To deal with the possible collisions, one can maintain a list of the intervals of time at which a message can be send on the arc (c_t, c_s). Each time the offset of a route is fixed an interval is split into at most two intervals in constant time. Since the algorithm goes over the elements of the list at most twice when doing an insertion or looking for the next free interval, the time needed to maintain it is $O(n)$.

D. Earliest deadline scheduling

The problem to solve in the second stage of our approach is very similar to the following scheduling problem: we are given a set of jobs and each job has a *release time* and a *deadline*. The problem is to schedule all jobs on a single processor, that is choosing the time at which they are computed, so that no two jobs are scheduled at the same time. A job is always scheduled after its release time and it must be finished before its deadline. Let us call n the number of jobs, the problem can be solved in time $O(n^2 \log(n))$ [18] when all jobs have the same running time and it gives a solution with the earliest possible deadline. On the other hand if the running times are different the problem is NP-complete [19]. The polynomial time algorithm which solves this scheduling problem is similar to the Greedy algorithm presented in the previous section. However, when it fails because a job finishes after its deadline, it changes the schedule of the last messages to find a possible schedule for the problematic job. The change in the scheduling is so that the algorithm cannot fail on the same job a second time except if there is no solution, hence the polynomiality of the algorithm.

We reduce our problem of setting the offsets of the backwards routes once the order of the forward routes is fixed to this scheduling problem. The backward routes are the jobs, the size of a message is the running time of a job, the deadline of a route is the deadline of the corresponding job and the smallest time at which it is eligible is the release time. Let us call **Minimal Latency Scheduling (MLS)** this algorithm. Remark that we do not deal with the periodicity. When MLS finds a solution $\mathcal{M}$ its always satisfies $MT(\mathcal{M}) < T_{max}$. On the other hand, it is a (P, τ) periodic assignment only if $m_{n-1} - m_0 \leq P - \tau$ where m_{n-1} is the largest offset and m_0 the smallest one. The algorithm we use minimize $m_{n-1} - m_0$, so it may work quite well in practice (as shown in Section IV-E).

We now present a variant of the previous algorithm that we call **Periodic Minimal Latency Scheduling (PMLS)** which takes into account the periodicity. We fix arbitrarily a backward route so that its message is the first to go through the central arc at time t. Then we modify the deadline of each forward route to be the minimum of their previous deadline and $t + P$. We execute this algorithm for every possible first backward route and we return the solution that minimize $MT(\mathcal{M})$. Since we run the previous algorithm at most n times, the complexity is in $O(n^3 \log(n))$. Remark that it will always find more solution than MLS, because if MLS finds a solution with first route r and such that $m_{n-1} - m_0 \leq P - \tau$, then this solution will be found by PMLS when it selects r as the first route. Moreover when PMLS finds a solution, it is always a (P, τ) periodic assignment since we guarantee that all messages are scheduled in a period of time of size P and it satisfies $MT(\mathcal{M}) < T_{max}$ by construction of the deadlines. The scheduling algorithm we use always finds a solution to the original problem it solves. However, in our periodic context, we do not allow to choose an offset for a route after $t + P - \tau$ which modulo P may not collide with another route. Therefore PMLS may fail while there is a solution.

E. Experimental evaluation

In this section, we first try to understand what is the best choice of order for the first stage of the algorithm. We fix the number of routes to 8 to make comparisons with the results of Section III-B easier. We draw uniformly the weights of the arcs between 0 and 20000 slots. For a given routed network, we

define the *margin* as the difference between T_{max} and twice the longest route. Hence, for a routed network choosing the margin or T_{max} is the same, but since we will consider many networks whith different longest route's size, it will be more meaningful to set the margin rather than T_{max}. In fact the margin represents the latency imposed by the communication process without taking into account the physical length of the network which cannot be changed. In our experiments the margin range from 0 to 3000 slots. We look at two different regimes: we take a period equal to $1,05n\tau$ to represent a very loaded network and $1,5n\tau$ for a mildly loaded network. Choosing a larger period is not relevant since then we know how to solve the problem even without waiting times as shown in Section III-B. We represent the success rate with regards to the margin, for the five kind of orders using the Greedy algorithm for the second stage, computed over 10000 random star routed networks. Note that for the random order, we compute 1000 random orders and count it as a success as soon as there is a solution for one order.

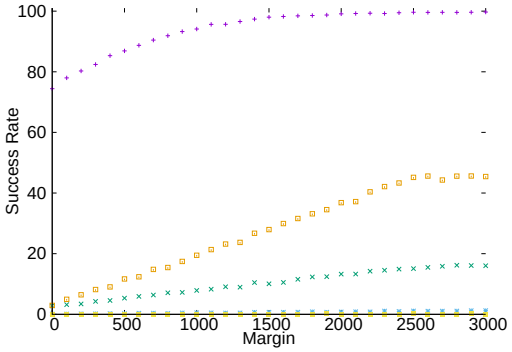


Fig. 3. Success rate of different orders, 95% load.

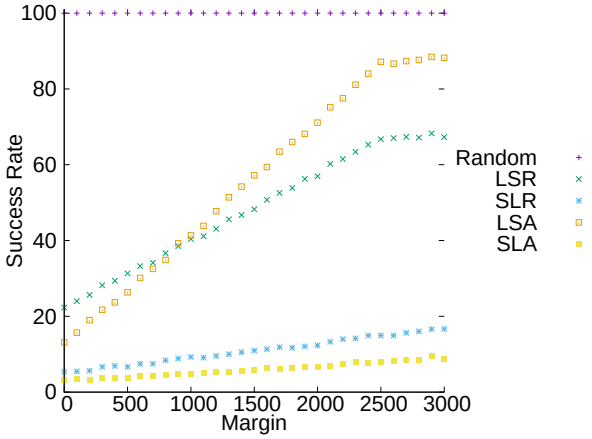


Fig. 4. Success rate of different orders, 80% load.

As expected, the success rate is much better for large periods. According to our experiments, sending the messages from the shortest to the longest route or arc does not work well. It corresponds to the policy of Proposition 1 which does not work well when the routes are long as it is the case in these experiments. Sending from the longest to the shortest route or arc works better and it seems that sorting the routes according to the length of the last arc rather than the route is better, at least in a loaded network.

The random order is the best by far. When the load is not too high ($P = 30000$) there is always a solution with margin 0. Note that, when disallowing waiting times, there were instances without solutions for these parameters (see Section III-B), which justifies the interest of studying the PALL problem. We now want to compare the performances of the three different algorithms used in the second stage. Since GD already showed excellent results on mild loads, it is more interesting to focus on the behavior of the algorithms on high load. Moreover, we will use 1000 random orders for the first stage as it gives the best results. The following experiment is realized on routed networks with 8 routes, weights of the arcs drawn between 0 and 20000 and with a period of 21000. We represent the success rate with regards to the margin, for the three algorithms computed over 10000 random star routed networks.

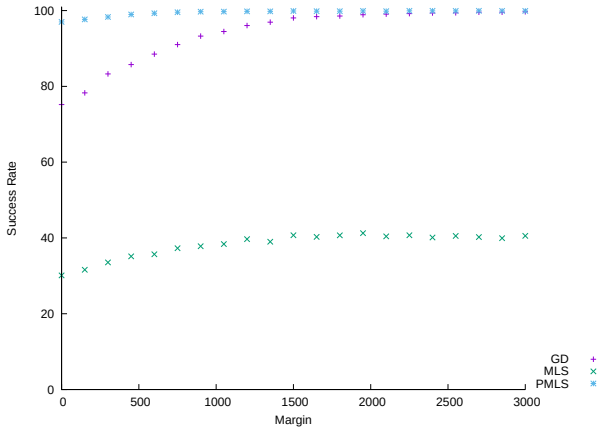


Fig. 5. Success rate of GD, MLS and PMLS for $P = 21000$

The MLS algorithm performs poorly, worst than GD and PMLS, which shows that taking into account the periodicity is fundamental. The GD algorithms is close to 100% success rate for margins larger than 1500 while the PMLS algorithm finds a solution in more than 95% of the experiments, even with a margin 0. Therefore, for the worst possible constraints on load and margin, there are a few instances for which we do not find a solution. However, with a margin of 1000, which corresponds to about 0.05ms of additional delay with

the chosen parameters, we always find a solution.

Remark that the random order corresponds to the best of 1000 orders drawn uniformly which means that a computation can be up to 1000 time slower than for a fixed order. The choice of the number of random orders drawn yields a trade-off between the computation time and the success rate. Up to now, we chose 1000 random orders arbitrarily. We investigate the success rate of our algorithm with regards to the number of random orders drawn, a period of 21000 and a margin 0. The following tables presents the average success rate for each number of sending orders, on 1000 instances, for GD and PMLS.

# orders	1	10	100	1000	10000	100000
GD	0.6	7.1	33.4	76.4	90.0	90.0
PMLS	$41,1$	$82,5$	$94,5$	$96,2$	$97,2$	$97,2$

Fig. 6. Impact of the number of random sending orders

First remark that we can improve our previous results by taking 10000 random orders, which yields the best results for the two algorithms. The number of different orders is $7! = 5040$ since we have 8 routes and the solutions are invariant up to a circular permutation of the order. Therefore instead of doing 10000 random draws we could test every possible order in less time. However this method would not be practical with regard to the computation time for $n = 15$. On the other hand, remark that the success rate of PMLS is already high for 100 random orders, which means it will work even better than the other methods when n is larger and that the number of random orders drawn is critical.

Now that we have found the best amongst the algorithms solving PALL, we need to compare its performances against the actual way to manage the messages in a network. The classical way to manage messages in the network is statistical multiplexing, where there is a FIFO buffer in each node of the network to deal with the collisions. The time at which the messages are sent in the network is not computed as in our approach, thus we fix the offsets of each route to some random value. Even if this policy seems to work in practice when the network is not too loaded, it does not give any guarantee on the latency. Remark that the process is not periodic, therefore we must measure the process time of each route over several periods if we want to compute its maximum. We choose to simulate it for 1000 periods and we have observed that the process time usually stabilizes after 10 periods. The margin is defined as the maximum process time, computed as explained, minus twice the size of the longest route.

We draw 1000 star routed networks with 8 routes, with weights on the arcs drawn between 0 and 20000. Note that the results are almost the same when the arcs are small. We measure, for periods from 20000 (network fully loaded), to 50000 (40% of load), the median, the first, and the third quartile of the margins computed in 1000 simulations using FIFO buffer to resolve collisions.

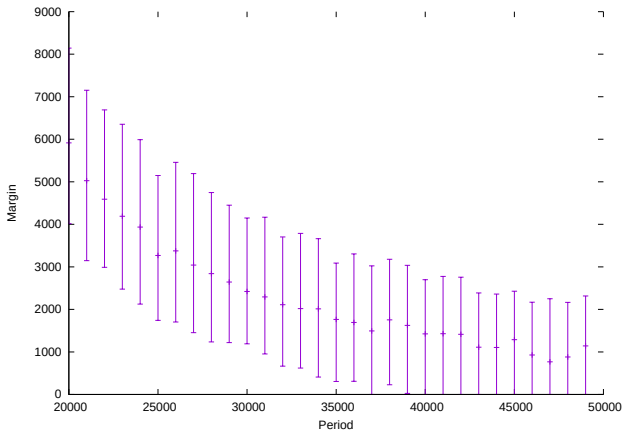


Fig. 7. Needed margin with respect to the period for a stochastic policy

We have also computed the minimum margin needed by PMLS to find a solution but we have not represented it in Figure 7 since even the third quartile was always at 0 for a period of 20000. There is around 20% of the instances for which the margin is larger than 0 for $P = 20000$ and 2% for $P = 21000$. For the instances with a margin larger than 0, the margin is on average 400.

The simulations show clearly that the stochastic policy does not ensure a minimal latency. As we can see in Fig 7), the more the network is loaded, the higher the latency is. When the network is highly loaded, we have instances with a margin of about 10000 which corresponds to half the period, that is 0.5ms. Even when the network is lightly loaded, a quarter of the instances have a margin of more than 2000 which is more than what PMLS finds for the worst instance and a network fully loaded ! We feel that it strongly justifies the use of a deterministic sending scheme for latency critical applications such as our C-RAN motivating problem.

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] A. Pizzinat, P. Chanclou, F. Saliou, and T. Diallo, "Things you should know about fronthaul," *Journal of Lightwave Technology*, vol. 33, no. 5, pp. 1077–1083, 2015.

[4] Z. Tayq, L. A. Neto, B. Le Guyader, A. De Lannoy, M. Chouaref, C. Aupetit-Berthelemot, M. N. Anjanappa, S. Nguyen, K. Chowdhury, and P. Chanclou, "Real time demonstration of the transport of ethernet fronthaul based on vran in optical access networks," in *Optical Fiber Communications Conference and Exhibition (OFC), 2017*, pp. 1–3, IEEE, 2017.

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

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

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

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

[9] 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.

[10] 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.

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

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

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

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

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

[16] A. J. Orman and C. N. Potts, "On the complexity of coupled-task scheduling," *Discrete Applied Mathematics*, vol. 72, no. 1-2, pp. 141–154, 1997.

[17] R. G. Downey and M. R. Fellows, *Parameterized complexity*. Springer Science & Business Media, 2012.

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

[19] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of discrete mathematics*, vol. 1, pp. 343–362, 1977.