Deterministic Scheduling of Periodic Messages for Cloud RAN

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

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

***Abstract*—A recent trend in mobile networks is to centralize in distant data-centers the processing units which were attached to antennas until now. The main challenge is to guarantee that the latency of the periodic messages sent from the antennas to their processing units and back, fulfills protocol time constraints. We show that traditional statistical multiplexing does not allow such a low latency, due to collisions and buffering at nodes. Hence, we propose in this article to use a deterministic scheme for sending periodic messages without collisions in the network thus saving the latency incurred by buffering.**

We give several algorithms to compute such schemes for a common topology where one link is shared by all antennas. We show that there is always a solution when the routes are short or the load is small. When 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 show that even under full load, most of the time PMLS finds a deterministic sending scheme with no latency.

I. Introduction

Next generations of mobile network architectures evolve toward centralized radio network architectures called C-RAN for Cloud Radio Access Network, to reduce energy consumption costs [1] and more generally the total cost of ownership. The main challenge for this type of architecture is to reach a latency compatible with transport protocols [2]. The latency is measured between the sending of a message by a Remote Radio Head (RRH) and the receptions of the answer, computed by real-time virtualized network functions of a BaseBand Unit (BBU) in the cloud. For example, LTE standards require to process functions like HARQ (Hybrid Automatic Repeat reQuest) in 3ms [3]. In 5G, some services need end-to-end latency as low as 1ms [4], [5]. The specificity of the C-RAN context is not only the latency constraint, but also the periodicity of the data transfer in the *frontaul* network between RRHs and BBUs: frames need to be emitted and received each millisecond [3]. Our aim is to operate a C-RAN on a low-cost shared switched network. The question we address is the following: is it possible to schedule messages such that there are no collisions to avoid latency caused by queuing delays? Eliminating this source of latency leaves us with more time budget for latency due to the physical length of the routes in the network, and thus allows for wider deployment areas.

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 or slot corresponds to the time needed to transmit a minimal unit of data over the network. Since statistical multiplexing does not ensure a good latency we want to avoid any buffering in internal nodes of the graph. We take advantage of the deterministic nature of the messages we must manage i.e. the dates of arrival of messages are known beforehand. In fact, following LTE standard [3], we assume that the arrivals of all the packets are periodic with the same period. We propose to design a *periodic* process to send the messages through the network without collisions. By periodic process we mean that the network at times t and $t+P$, where P is the period, is in the exact same state.

We assume that the routes of the messages are already fixed, and there are no buffering allowed inside the network. Hence we only have two sets of values that we can set when building a periodic sending process, called a *periodic assignment*: the time at which each packet is sent by an 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 unusable. 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. The latency, that is the time between the emission of a message and the complete return of its answer must be minimized. This 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. Note that the model is technology agnostic, i.e. it is compatible with an optical network with a fixed packet size.

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 formalize the problem of finding a periodic assignment for sending messages without buffering (PAZL) or with buffering in the processing unit only (PALL). In Sec. III, we propose several polynomial time algorithms to solve PAZL when the length of the routes, the number of routes or the load is small and evaluate their efficiency experimentally. Finally in Sec. IV, we introduce two algorithms to solve PALL and our experimentations show that the deterministic communication schemes we design vastly outperform traditional statistical multiplexing with regard to latency.

Related works

Statistical multiplexing even with a large bandwidth does not comply with the latency requirements of C-RAN. Therefore, the current solution [6], [7] is to use dedicated circuits for the fronthaul. 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. The deterministic approach we propose has gained some traction recently: Deterministic Networking is under standardization in IEEE 802.1 TSN group [8], as well at IETF DetNet working group [9]. Several patents on concepts and mechanisms for DetNet have been already published, see for example [10], [11].

The algorithmic problems we study are similar to several optimization problems: such as the allocation of frequencies [12], designing train schedules [13] with small latency or the two flow shop scheduling problem [14]. However, the periodicity of our problems makes them significantly different and not reducible to these classical models.

II. Model and Problems

A long version of this paper [15] contains details on the models, hardness results, several additional algorithms, the proofs of all propositions and more experiments.

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 taken by a message to go from u to v using this arc. A **route** r in G is a directed path, that is, a sequence of adjacent vertices $u_0, \dots, u_l$, with $(u_i,u_{i+1}) \in A$. The **latency** of a vertex u_i in a path $r = (u_0, \dots, u_l)$ is defined by $\lambda(u_i,r) = \sum\limits_{0 \leq j < i} \Omega(u_j, u_{j+1})$. We also define $\lambda(u_0,r) = 0$. The length of the route r is defined by $\lambda(r) = \lambda(u_l,r)$. 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.

In the context of cloud-RAN applications, we need to send a message from an 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**. 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.

B. Messages dynamic

Time is discretized, hence the unit of all time values is a slot, the time needed to transmit a minimal unit of data over the network. The weight of an arc is also expressed in slots, it is the time needed by a message to go through this arc. In the process we study, a message is sent on each route at each period, denoted by P. Let $r = (u_0, \dots, u_l)$ be a route, if a message is sent at time m from u_0 the first vertex of r then it will arrive at vertex u_i in r at time $m + \lambda(u_i,r)$. Since the process is periodic, if the message from r goes through an arc at time $t \in [0,P-1]$, then it goes through the same arc at time $t + kP$ for all positive integers k. Therefore, every time value can be computed modulo P and we say that the first time slot at which a message sent at time m on r reaches a vertex u_i in r is $t(u_i,r) = m + \lambda(u_i,r) \mod P$.

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. In this paper, we assume that τ is the same for all routes. Indeed, the data flow sent by an RRH to its BBU is the same, regardless of the route. Let us call $[t(u,r)]_{P,\tau}$ the set of time slots used by route r at vertex u in a period P, that is $[t(u,r)]_{P,\tau} = \{t(u,r) + i \mod P \mid 0 \leq i < \tau\}$. 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 (u,v) and $[t(u,r_1)]_{P,\tau} \cap [t(u,r_2)]_{P,\tau} \neq \emptyset$.

A (P,τ)**-periodic assignment** of a routed network $(G,\mathcal{R})$ is a function that associates to each route $r \in \mathcal{R}$ its **offset** m_r that is the time at which a message is emitted at the first vertex of the route r. In a (P,τ)-periodic assignment, *no pair of routes has a collision.*

Let us give an interpretation of a (P,τ)-periodic assignment of $(G,\mathcal{R})$ a symmetric routed network, so that it represents the sending of a message and of its answer. First a message is sent at u, through the route $r \in F$, at time m_r. This message is received by v, i.e., the last vertex of r at time $t(v,r)$. The answer 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. Fig. 1 illustrates this process in an RRH and its corresponding BBU.

The process time for a message sent on the route r is equal to $PT(r) = \lambda(r) + w_r + \lambda(r)$. Each route must respect some time limit that we call a *deadline*. To represent these deadlines, we use a deadline function d, which maps to each route r an integer such that $PT(r)$ must be less than $d(r)$. We can now define the problem we solve in this article.

Periodic Assignment for Low Latency (PALL)

Input: A symmetric routed network $(G,\mathcal{R})$, the integers P, τ and a deadline function d.

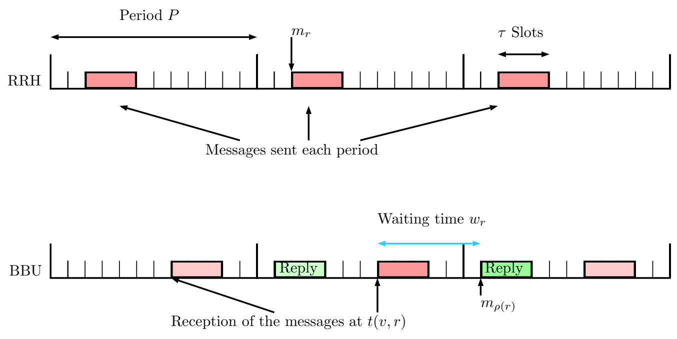

Fig. 1. The process defined by a (P,τ)-periodic assignment

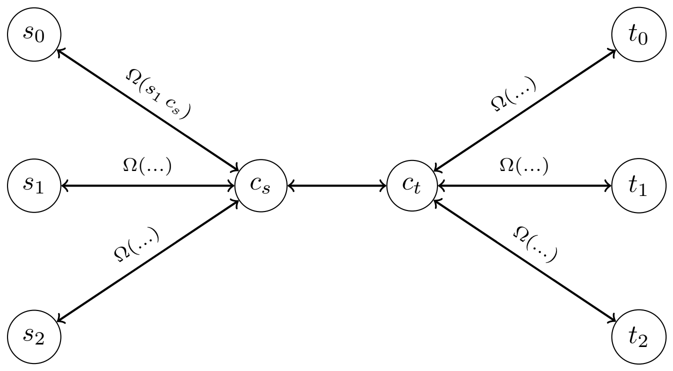

Fig. 2. A star routed network

Question: does there exist a (P,τ)-periodic assignment m of $(G,\mathcal{R})$ such that for all $r \in \mathcal{R}$, $PT(r) \leq d(r)$?

It turns out that the problem PALL is NP-hard to solve and even to approximate for *general* routed networks (see the long version [15]). We also consider a simpler version of PALL, that we call **Periodic Assignment for Zero Latency** or PAZL: we ask for a (P,τ)-periodic assignment **with all waiting times equal to** 0. We introduce PAZL because it is simpler to study and we are able to prove theoretical results and find better algorithms for this variant than for PALL. As we experimentally show in Sec. III-B, this problem can often be solved positively albeit less often than PALL. Finally, a solution to PAZL is simpler to implement in real telecommunication networks, since we do not need buffering at all.

C. The star routed network

Let us define a family of simple routed networks modeling a Point-to-Multipoint fronthaul (PtMP), which has been designed for C-RAN [7]. The graph G has two sets of vertices, $S = \{s_0,\ldots,s_{n-1}\}$ and $T = \{t_0,\ldots,t_{n-1}\}$ 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 which modelizes a *full-duplex* network. The forward routes are the directed paths $r_i = (s_i,c_s,c_t,t_i)$ and the backward routes are $\rho(r_i) = (t_i,c_t,c_s,s_i)$. The symmetric routed networks $(G,\{r_i,\rho(r_i)\}_{i<n})$ is called a **star routed network** and is represented in Fig. II-C. While it seems very simple, every network in which all routes share an arc can be reduced to it. This topology is realistic, since often all the BBUs are located in the same data-center. The network interface of the data-center is modeled by the vertex c_t and the BBUs inside it are represented by the vertices $t_0,\ldots,t_{n-1}$. In the following we will consider the problems PALL and PAZL on star routed networks only. In this context, we can assume w.l.o.g. that the weights are zero on (c_s,c_t) and for all i on (s_i,c_s) (see [15]).

III. The Star Routed Network: No Waiting Time

In this section, we deal with the problem PAZL on star routed networks. We want an assignment with waiting times zero, that is, $d(r) = 2\lambda(r)$. For a route r, choosing the offset m_r also sets the offset of the route $\rho(r)$ to $m_{\rho(r)} = m_r + \lambda(r) \mod P$.

A. Three algorithms to solve PAZL

We first present a simple policy, which works when the period is large with regard to the lengths of the routes. The messages are sent in order from the shortest route to the longest route, without any gap between two messages of the forward routes. In other words, we assume that the route r_i are sorted by increasing $\lambda(r_i)$ and we set m_{r_i} the offset of r_i to $i\tau$. We call this algorithm **Shortest-Longest**. By construction, there are no collisions in the forward period and if the period is long enough, in the backward period the order of the messages are the same as in the forward period thus no collision can occur.

Proposition 1. *Let $(G,\mathcal{R})$ be a star routed network, and let $n\tau + 2(\lambda(r_{n-1}) - \lambda(r_0)) \leq P$. There is a (P,τ)-periodic assignment of $(G,\mathcal{R})$ with waiting times 0 given by Shortest-Longest in time $O(n\log(n))$.*

We define the **load** of a star routed network as $\frac{n\tau}{P}$, it is the proportion of time slots used by messages on the central arc in a period. Therefore if the load is larger than 1 there cannot be an assignment. We propose a **Greedy algorithm** to build a (P,τ)-periodic assignment. We consider a period and cut it into consecutive intervals of size τ that we call macro-slots. The algorithm works by choosing an offset for each forward route in the following way: try all offsets which put the message in a yet not used macro-slot in the period. Since this choice also sets the offset of the corresponding backward route, choose the first one which does not create a collision. This algorithm always finds an assignment when the load is sufficiently large, since it guarantees that enough macro-slots are free.

Proposition 2. *There is a (P,τ)-periodic assignment of a star routed network with waiting times* 0 *if the load is less than* $1/3$ *and it can be found in time* $O(n^2)$.

Finally, there is an algorithm which finds an assignment when it exists, in fixed parameter tractable time (FPT) with parameter n the number of routes, that is the algorithm is exponential in n only and linear in the other parameters. This is justified since n is small in practice (from 10 to 20) and the other parameters such as P, τ or the weights are large. We call this algorithm **Exhaustive Search of Compact Assignments**. The algorithm is based on a canonical form for assignments with waiting time zero. We call such an assignment compact which roughly means that no message can be sent a slot before in a forward route without collisions. The number of compact assignments is bounded by $n!$, hence we enumerate them in reasonable time when n is small. To make it more efficient in practice, we make cuts in the search tree used to explore all compact assignments, see [15] for more details.

B. Experimental evaluation

In this section we compare the three presented algorithms experimentally. The C code used in this experimentations can be found on the author's web page [16]. From the C-RAN context we choose the following parameters: the number of routes is at most $n = 20$, τ is equal to 2,500. It corresponds to slots of 64 bytes, messages of approximately 1 Mbit and links of bandwidth 10 Gbit/s when P is one millisecond (19531 slots). First we consider routes which are shorter than τ: a message cannot be contained completely in a single arc which is common in our applications. We generate star routed networks with weights of the arcs (c_t,t_i) drawn uniformly between 0 and 700 which corresponds to links of less than 5km between a BBU and an RRH.

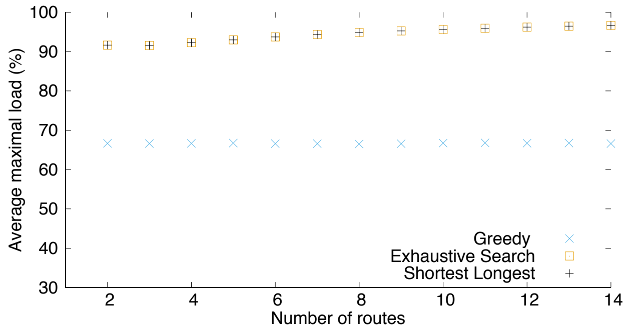

Fig. 3. Maximal possible load (averaged over 1,000 random instances) for which an assignment is found

As shown in Fig. 3, for short routes, both Shortest Longest or the exhaustive search have the same performance and succeed very often but *not always*. It is surprising, since exhaustive search is guaranteed to find a solution when it exists, while Shortest Longest is only a heuristic. The Greedy algorithm is less effective but it works when the load is less than $2/3$, twice better than the theoretical lower bound of $1/3$. We now look at the performance of the algorithms on longer routes. The weights of the arcs are drawn between 0 and 20,000.

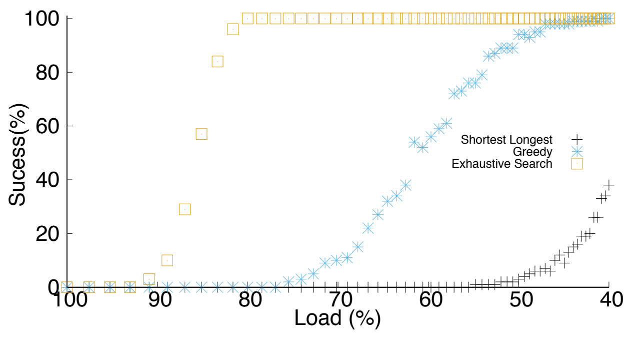

Fig. 4. Average success rate of the three algorithms computed from 1,000 random instances with 8 routes

As shown in Fig. 4, on long routes, the performances of Shortest-Longest are abysmal since the difference between the length of the routes is large. When the load is larger than 50%, the exhaustive search finds more solutions than the greedy algorithms which justifies its use. However, for load larger than 80% 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 propose algorithms for the general PALL problem in our next section.

IV. Solving PALL on Star Routed Networks

In this section, we consider the more general PALL problem on star routed networks. The messages are allowed to wait in the target vertices (BBUs) to yield more possible assignments. Hence, we allow the process time of a route to be greater than twice the weights of the route, but it is bounded by its deadline.

A. Two algorithms to solve PALL

As for PAZL, it is possible to design an FPT algorithm for PALL [15]. However, it is complicated and its running time is too large even for small n, thus we do not describe it here or test it against the two other algorithms we present.

In the two following algorithms, we assume that the offsets of the forward routes are already fixed by some mechanism, hence they fix only the offsets of the backward routes.

The first algorithm we propose to solve PALL is a greedy algorithm which sets the offset $m_{\rho(r_i)}$ of the backward routes. It prioritizes the routes with the earliest deadline to best satisfy the constraint on the process time. We call it **Greedy Deadline (GD)**. We say that a backward route $\rho(r_i)$ is **eligible** at time t if the message of the route $\rho(r_i)$ arrives at c_t before time t with $w_i = 0$. For each time t, amongst all eligible routes at t, we choose r with the smallest deadline and fix its offset to $m_r = t - \lambda(r)$.

We must improve this algorithm since it does not take into account the periodicity. Say that $t_0 = t(c_t,r)$ such that r is the first backward route selected by the algorithm. Then if all backward routes r are such that $t(c_t,r)$ is smaller than $t_0 + P - \tau$, by construction, there are no collisions on the central arc. However, if a route r has a larger $t(c_t,r)$, since we must consider everything modulo P, it may collide with another backward 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 are no collisions if a message go through the central arc at time s.*

The problem PALL (with offsets of forward routes already fixed) is very similar to the following scheduling problem: Given a set of jobs with *release times* and *deadlines*, schedule all jobs on a single processor, that is choose the time at which they are computed, so that no two jobs are scheduled at the same time. When the running time is the same on all jobs, this problem can be solved in polynomial time [17]. We use the polynomial time scheduling algorithm to solve PALL and we make some adjustments to take the periodicity into account: we run it once for each forward route r and we fix m_r so that the waiting time for this route is zero and we change the deadline of each route r' to be less than $m_r + P$. To improve the chance of finding a solution, we change the value of $m_{r'}$ and $d(r')$ by adding the same multiple of P to both of them so that $m_{r'}$ is in $[m_r, m_r + P]$. This transformation does not change the original problem but avoid the instance to be rejected by the scheduling algorithm for trivial reason. For the same reasons, when $m_{r'}$ is in $[m_r + P - \tau, m_r + P]$, we set $m_{r'}$ to m_r and $d(r')$ to $d(r') - P$. The algorithm we obtain, when it succeeds always finds a correct periodic assignment and we call it **Periodic Minimal Latency Scheduling (PMLS)**. In all our experiments, its misses a solution to the problem of finding offsets for backward routes when there is one on 0.1% of the instances, which makes it almost optimal in practice.

B. Experimental evaluation

We set 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 20,000. We consider load of 95%, since for small load solving PAZL is enough as shown in Section III-B. To simplify the experiments, we use *the same deadline* for all routes. We define the **margin** as the difference between the deadline and twice the longest route. The margin represents the latency imposed by the communication process without taking into account the physical length of the network which cannot be changed.

To set the offsets of the forward routes, we draw random orders on these routes, so that their messages goes through the central arc in this order. If we have n routes and draw the order given by the permutation σ then we set $m_i = \sigma(i)\tau$. In the experiments, we draw 1000 random orders and consider that the algorithm fixing the offsets of the backward routes succeeds when it succeeds for one of the random orders. We also looked at the behavior of several other orders for the routes: ordered from the shortest to the longest route, from the longest to the shortest route, from the shortest to the longest arc (c_t,t_i),or from the longest to the shortest (c_t,t_i). After some simulations, it clearly appeared that drawing some random orders is gives by far some better results on the success rate of our algorithms.

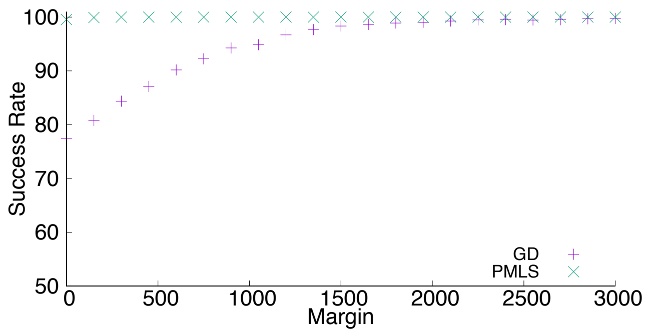

Fig. 5. Success rate of GD and PMLS, over 10,000 instances, 95% load

Fig. 5 shows that PMLS is better than GD since it finds a solution in more than 99% 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. With a margin of 600, which corresponds to about 0.03ms of additional delay with the chosen parameters, we always find a solution. The success rate of these algorithms depends on the number of random orders drawn to fix the offsets of forward routes. It turns out that PMLS needs much less random orders than GD to work well, which makes it even better than GD.

C. Comparison to statistical multiplexing

We now compare the performances of our algorithms to find periodic assignments against the actual way to manage the messages in a network: statistical multiplexing with a FIFO buffer in each node of the network to resolve collisions. 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 1,000 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.

In Fig. 6, we represent the probability of success for statistical multiplexing and PMLS for different margin. The success rates are computed from 10,000 star routed networks

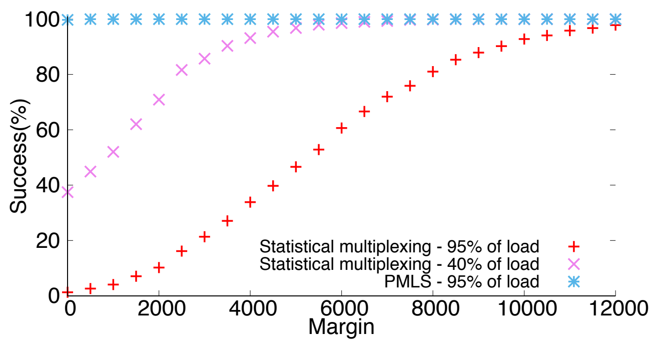

Fig. 6. Probability of success of statistical multiplexing and PMLS

for each margin, drawn with the same parameters as before. We represent the distribution under high and light load for statistical multiplexing and under high load only for PMLS since under light load the algorithm always succeed.

The experiment clearly shows that statistical multiplexing does not ensure a minimal latency. The latency is extremely high when the load is high, with a margin of about $10{,}000$ for the worst 10% which corresponds to half the period, that is 0.5ms. Even when the network is lightly loaded, 20% of the instances have a margin of more than $2{,}000$. On the other hand, PMLS finds an assignment with margin 0 in a highly loaded network for 99% of the instances!

For each $1{,}000$ slots of latency we save from the periodic process, we are able to lengthen the routes of 10km, which has a huge economical impact. We feel that it strongly justifies the use of a deterministic sending scheme for latency critical applications such as our C-RAN motivating problem.

V. Conclusion

In this paper, we proposed two deterministic methods to establish a low latency periodic communication between BBUs and RRHs in a star shaped fronthaul network. The first method uses no buffering and has no latency overhead. It works when the routes are short (Longest-Shortest policy) or when the load is less than 80% (Exhaustive search of compact assignments). When the load is higher, buffering is allowed in the BBUs and we propose the algorithm PMLS which finds a deterministic communication scheme with almost no additional latency. Our deterministic approach is vastly superior to the classical statistical multiplexing. This emphasizes that deterministic sources of traffic are always best dealt with in a deterministic manner.

We plan to generalize our study of the PALL problem to other common fronthaul topologies, such as caterpillar, trees or cycles. The cycles in particular are different since their forward and backward routes are not symmetric. We would like to design an FPT algorithm for PALL which is as efficient as the one for PAZL and prove that both problems are NP-hard.

Acknowledgments: The authors thank Christian Cadéré and David Auger for the friendly discussions they had on the subject and their insightful remarks. This work is partially supported by the french ANR project N-GREEN.

References

[1] C. Mobile, "C-RAN: the road towards green RAN," *White Paper, ver*, vol. 2, 2011.
[2] T.-S. N. T. G. of IEEE 802.1, "Time-sensitive networks for fronthaul," July 2016. IEEE P802.1/D0.4.
[3] Y. Bouguen, E. Hardouin, A. Maloberti, and F.-X. Wolff, *LTE et les réseaux 4G*. Editions Eyrolles, 2012.
[4] 3GPP, *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service requirements for the 5G system;*. Stage 1 (Release 16).
[5] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5G," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, 2014.
[6] 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.
[7] 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.
[8] N. Finn and P. Thubert, "Deterministic Networking Architecture," Internet-Draft draft-finn-detnet-architecture-08, Internet Engineering Task Force, 2016. Work in Progress.
[9] "Time-sensitive networking task group." http://www.ieee802.org/1/pages/tsn.html. Accessed: 2016-09-22.
[10] W. Howe, "Time-scheduled and time-reservation packet switching," Mar. 17 2005. US Patent App. 10/947,487.
[11] B. Leclerc and O. Marcé, "Transmission of coherent data flow within packet-switched network," June 15 2016. EP Patent App. EP20,140,307,006.
[12] 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.
[13] C. Strotmann, *Railway scheduling problems and their decomposition*. PhD thesis, PhD thesis, Universität Osnabrück, 2007.
[14] W. Yu, H. Hoogeveen, and J. K. Lenstra, "Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard," *Journal of Scheduling*, vol. 7, no. 5, pp. 333–348, 2004.
[15] D. Barth, M. Guiraud, and Y. Strozecki, "Deterministic scheduling of periodic messages for cloud ran," *arXiv*, 2017.
[16] "Yann strozecki's web page." http://www.prism.uvsq.fr/~ystr/textesmaths.html. Accessed: 2018-01-19.
[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.