Scheduling Periodic Messages on a Shared Link

Maël Guiraud[1,2] and Yann Strozecki[1]

[1]David Laboratory, UVSQ, France, {mael.guiraud,yann.strozecki}@uvsq.fr
[2]Nokia Bell Labs, France

June 29, 2020

Abstract

Cloud-RAN is a recent architecture for mobile networks where the processing units are located in distant data-centers while, until now, they were attached to antennas. The main challenge, to fulfill protocol constraints, is to guarantee a low latency for the periodic messages sent from each antenna to its processing unit and back. The problem we address is to find a sending scheme of these periodic messages *without contention nor buffering*.

We study the problem PMA modeling this situation on a simple but common topology, where all contentions are on a single link shared by all antennas. The problem is reminiscent of classical scheduling and packing problems, but the periodicity introduces a new twist. We study how the problem behave with regard to the load of the shared link. The two main contributions are polynomial time algorithms which *always* find a solution for arbitrary size of messages and load less than 0.4 or for size one and load less than $\phi - 1$ (ϕ being the golden ratio $(\sqrt{5}+1)/2$). We also prove that, a randomized greedy algorithm almost always finds a solution on random instances, explaining why most greedy algorithms work so well in our experiments.

1 Introduction

Centralized radio network architecture, called C-RAN for Cloud Radio Access Network, has been proposed as a next generation architecture to reduce energy consumption costs [16] and more generally the total cost of ownership. The main challenge for C-RAN is to reach a latency compatible with transport protocols [18]. The latency is measured between the sending of a message by a Remote Radio Head (RRH) and the reception of the answer, computed by a BaseBand Unit (BBU) in the cloud. For example, LTE standards require processing functions like HARQ (Hybrid Automatic Repeat reQuest) in 3ms [7]. In 5G, some services need end-to-end latency as low as 1ms [2, 6]. 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: messages need to be emitted and received each millisecond [7].

Statistical multiplexing even with a large bandwidth does not satisfies the latency requirements of C-RAN [4, 3]. The current solution [20, 23] is to use dedicated circuits for the fronthaul. Each end-point, an RRH on one side and a BBU on the other side is connected through direct fiber or full optical switches. This eliminates all contentions since each message flow has its own link, but it is extremely expensive and do not scale in the case of a mobile network composed of about $10,000$ base stations.

Our aim is to operate a C-RAN on a low-cost shared switched network. The question we address is thus the following: *is it possible to schedule periodic messages on a shared link without using buffers*? 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. Our proposed solution is to compute beforehand a *periodic and deterministic* sending scheme, which completely avoids contention. This kind of deterministic approach has gained some traction recently: Deterministic Networking is under standardization in IEEE 802.1 TSN group [9], as well at IETF DetNet working group [1]. Several patents on concepts and mechanisms for DetNet have been already published, see for example [11, 13].

The algorithmic problem studied, called *Periodic Message Assignment* or PMA, is as follows: Given a period, a message size and a delay between the two contention points for each message, set a departure time in the period for each message, so that they go through both contention points without collision. It is similar to the two flow shop scheduling problem [24] with periodicity. The periodicity adds more constraints, since messages from consecutive periods can interact. In flow shop problems, the aim is usually to minimize the makespan, or schedule length, but in our periodic variant it is infinite. Hence, we choose to look for any periodic schedule without buffering, to minimize the trip time of each message.

To our knowledge, all studied periodic scheduling problems are quite different from the one we present. In some works [12, 10], the aim is to minimize the number of processors on which the periodic tasks are scheduled, while our problem corresponds to a single processor and a constraint similar to makespan minimization. In cyclic scheduling [14], the aim is to minimize the period of a scheduling to maximize the throughput, while our period is fixed. The train timetabling problem [15] and in particular the periodic event scheduling problem [21] are generalizations of our problem, since they take the period as input and can express the fact that two trains (like two messages) should not cross. However, they are much more general: the trains can vary in size, speed, the network can be more complex than a single track and there are precedence constraints. Hence, the numerous variants of train scheduling problems are very hard to solve (and always NP-hard). Thus, some delay is allowed to make the problems solvable and most of the research done [15] is devising practical algorithms using branch and bound, mixed integer programming, genetic algorithms ...

In previous articles of the authors, generalizations of PMA allowing buffering are studied on a single link [4] or on a cycle [5]. Heuristics (using classical scheduling algorithms as subroutines) and FPT algorithms are used to find a sending scheme with *minimal latency*, while here we consider only sending scheme with *no additional latency*. More complex scheduling problems for time sensitive networks have been practically solved, using mixed integer programming [17, 22] or an SMT solver [8], but without theoretical guarantees on the quality of the produced solutions. Typical applications cited in these works (out of C-RAN) are sensor networks communicating periodically inside cars or planes, or logistic problems in production lines. We think our approach can be brought to these settings.

Organization of the Paper In Sec. 2, we present the model and the problem PMA. In Sec. 3, we present several greedy algorithms and prove they always find a solution to PMA for moderate loads. These algorithms rely on schemes to build compact enough solutions, to bound measures of the size wasted when scheduling messages. Then, we illustrate their surprisingly good performance on random instances in Sec. 3.4. It turns out that PMA can be restricted to messages of size one

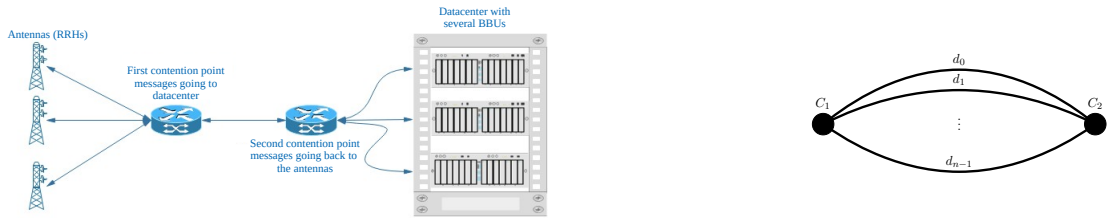

Figure 1: C-RAN network with a single shared link modeled by two contention points and delays

for the price of doubling the load or adding some latency, as explained in Sec. 4. Hence, in Sec. 5, we present a deterministic and a probabilistic algorithm for this special case. The deterministic algorithm is not greedy, contrarily to algorithms of Sec. 3, since it uses a swap mechanism which can move already scheduled messages. Experiments in Sec. 5.3 again shows that our algorithms work even better on random instances than what we have established for their worst case.

2 Modeling a C-RAN Network

In this article, we model a simple network, as represented in Fig.1, in which periodic messages flow through a single bidirectional link. The answer to each message is then sent back through the same link and it does not interact with the messages sent in the other direction, since the link we model is full-duplex. In the C-RAN context we model, all messages are of the same nature, hence they are all the same size denoted by τ. This **size** represents the time needed to send a message through some contention point of the network, here a link shared by all antennas. We denote by n the number of messages, which are numbered from 0 to $n-1$. A message i is characterized by its **delay** d_i: when the message number i arrives at the link at time t, then it returns to the other end of the link on its way back at time $t + d_i$. The model and problem can easily be generalized to any topology, that is any directed acyclic multigraph with any number of contention points, see [4]. We choose here to focus on the simplest, but realistic, non-trivial network, for which we can still obtain some theoretical results.

The time is discretized and the process we consider is periodic of fixed integer **period** P. We use the notation $[P]$ for the set $\{0, \ldots, P-1\}$. Since the process is periodic, we may consider any interval of P units of time and the times at which messages go through the two contention points during this interval to completely represent the state of our system. We call the representation of the interval of time in the first contention point the **first period** and the **second period** for the second contention point.

An **offset** of a message is a choice of time at which it arrives at the first contention point (i.e. in the first period). Let us consider a message i of offset o_i, it uses the interval of time $[i]_1 = \{(o_i + t) \mod P \mid 0 \leq t < \tau\}$ in the first period and $[i]_2 = \{(d_i + o_i + t) \mod P \mid 0 \leq t < \tau\}$ in the second period. Two messages i and j **collide** if either $[i]_1 \cap [j]_1 \neq \emptyset$ or $[i]_2 \cap [j]_2 \neq \emptyset$. If $t \in [i]_1$ (resp. $t \in [i]_2$), we say that message i uses time t in the first period (resp. in the second period).

We want to send all messages, so that there is no collision in the shared link. In other word, we look for a way to send the messages without using buffering and hence limiting the latency of messages to the physical length of the links. An **assignment** is a choice of an offset for each message such that *no pair of messages collide*, as shown in Fig. 2. Formally, an assignment is a

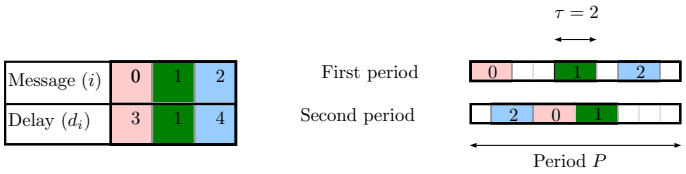

Figure 2: An instance of PMA with 3 messages, $P = 10$, $\tau = 2$, and one assignment

function from the messages in $[n]$ to their offsets in $[P]$. Let **Periodic Message Assignment** or PMA be the following problem: given an instance of n messages, a period P and a size τ, find an assignment or decide there is none. When an assignment is found, we say the problem is solved **positively**.

The complexity of PMA is not yet known. However, we have proven that, when parameterized by n the number of messages, the problem is FPT [3]. A slight generalization of PMA, with more contention points but each message only going through two of them, as in PMA, is NP-hard [3]. If the shared link is not full-duplex, that is, there is a single contention point and each message goes through it twice, it is also NP-hard [19]. Hence, we conjecture that PMA is NP-hard.

To overcome the supposed hardness of PMA, we study it when the load of the system is small enough. The **load** is defined as the number of units of time used in a period by all messages divided by the period that is $n\tau/P$. There cannot be an assignment when the load is larger than one; we prove in this article that, for moderate loads, there is *always* an assignment and that it can be found by a polynomial time algorithm.

3 Greedy Algorithms for Large Messages

In this section, we study the case of arbitrary values for τ. When modeling real problems, it is relevant to have $\tau > 1$ when the transmission time of a single message is large with regard to its delay, which is the case in C-RAN networks.

A **partial assignment** A is a function defined from a subset S of $[n]$ to $[P]$. The cardinal of S is the **size** of partial assignment A. A message in S is **scheduled** (by A), and a message not in S is **unscheduled**. We only consider partial assignments such that no pair of messages of S collide. If A has domain S, and $i \notin S$, we define the extension of A to the message i by the offset o, denoted by $A[i \to o]$, as A on S and $A[i \to o](i) = o$.

All presented algorithms build an assignment incrementally, by growing the size of a partial assignment. Moreover, algorithms of this section are *greedy*: Once an offset is chosen for a message, it is never changed. In the rest of the paper, we sometimes compare the relative position of messages, but one should remember that the time is periodic and these are relative positions on a circle. Moreover, when it is unimportant and can hinder comprehension, we may omit to write *mod* P in some definitions and computations.

3.1 First Fit

Consider some partial assignment A, the message i uses all times from $A(i)$ to $A(i) + \tau - 1$ in the first period. If a message j is scheduled by A, with $A(j) < A(i)$, then the last time it uses in the first period is $A(j) + \tau - 1$ and it should be less than $A(i)$, which implies that $A(j) \leq A(i) - \tau$.

Symmetrically, if $A(j) > A(i)$, to avoid collision between messages j and i, we have $A(j) \geq A(i)+\tau$. Hence, message i forbids the interval $]A(i) - \tau, A(i) + \tau[$ as offsets for messages still not scheduled because of its use of time in the first period. The same reasoning shows that $2\tau - 1$ offsets are also forbidden because of the times used in the second period. Hence, if $|S|$ messages are already scheduled, then $|S|(4\tau - 2)$ offsets are forbidden for any unscheduled message. It is an upper bound on the number of forbidden offsets, since the same offset can be forbidden twice because of a message on the first and on the second period.

Let $Fo(A)$ be the maximum number of forbidden offsets when extending A. Formally, assume A is defined over S and $i \notin S$, $Fo(A)$ is the maximum over all possible values of d_i of $|\{o \in [P] \mid A[i \to o] \text{ has no collision}\}|$. The previous paragraph shows that $Fo(A)$ is always bounded by $(4\tau - 2)|S|$.

Let `First Fit` be the following algorithm: for each unscheduled message (in the order they are given), it tests all offsets from 0 to $P - 1$ until one does not create a collision with the current assignment and use it to extend the assignment. If $Fo(A) < P$, then whatever the delay of the route we want to extend A with, there is an offset to do so. Since $Fo(A) \leq (4\tau-2)|S|$ and $|S| < n$, `First Fit` (or any greedy algorithm) will always succeed when $(4\tau - 2)n \leq P$, that is when the load $n\tau/P$ is less than $1/4$. It turns out that `First Fit` always creates compact assignments (as defined in [4]), that is a message is always next to another one in one of the two periods. Hence, we can prove a better bound on $Fo(A)$, when A is built by `First Fit`, as stated in the following theorem.

Theorem 1. *`First Fit` solves* PMA *positively on instances of load less than* $1/3$.

Proof. We show by induction on the size of S, that $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$. For $S = 1$, it is clear since a single message forbid at most $(3\tau - 1) + \tau - 1 = 4\tau - 2$ offsets, as explained before. Now, assume $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$ and consider a route $i \notin S$ such that `First Fit` builds $A[i \to o]$ from A. By definition of `First Fit`, choosing $o - 1$ as offset creates a collision. W.l.o.g. say it is a collision in the first period. It means that there is a scheduled message between $o-\tau$ and $o-1$, hence all these offsets are forbidden by A. The same offsets are also forbidden by the choice of o as offset for i, then only $3\tau-1$ new offsets are forbidden, that is $Fo(A[i \to o]) \leq Fo(A)+(3\tau-1)$, which proves the induction and the theorem. □

3.2 Meta-Offset

The method of this section is described in [4] and it achieves the same bound on the load using a different method. It is recalled here to help understand several algorithms in the article. The idea is to restrict the possible offsets at which messages can be scheduled. It seems counter-intuitive, since it decreases artificially the number of available offsets to schedule new messages. However, it allows reducing the number of forbidden offsets for unscheduled messages. A **meta-offset** is an offset of value $i\tau$, with i an integer from 0 to P/τ. We call `Meta Offset` the greedy algorithm which works as `First Fit`, but consider only meta-offsets when scheduling messages.

Let $Fmo(A)$ be the maximal number of meta-offsets forbidden by A. By definition, two messages with a different meta-offset cannot collide in the first period. Hence, $Fmo(A)$ can be bounded by $3|S|$ and we obtain the following theorem.

Theorem 2 (Proposition 3 of [4])**.** *`Meta Offset` solves* PMA *positively on instances of load less than* $1/3$.

A naive implementation of `Meta Offset` is in $O(nP/\tau)$, while `First Fit` is in $O(nP)$. However, it is not useful to consider every possible (meta-)offset at each step. By maintaining a list of positions of scheduled messages in first and second period, both algorithms can be implemented in $O(n^2)$.

3.3 Compact Tuples

We present in this section a family of greedy algorithms which solve PMA positively for larger loads. We try to combine the good properties of the two previous algorithms: the compactness of the assignments produced by `First Fit` and the absence collision in the first period of `Meta Offset`. The idea is to schedule several messages at once, using meta-offsets, to maximize the compactness of the obtained solution. We first describe the algorithm which schedules pairs of routes and then explain quickly how to extend it to any tuples of messages.

From now on, we use Lemma 3 to assume $P = m\tau$. This hypothesis makes the analysis of algorithms based on meta-offsets simpler and tighter. The load increases from $\lambda = n\tau/P$ to at most $\lambda(1 + 1/m)$: the difference is less than $1/m < 1/n$, thus very small for most instances. The transformation of Lemma 3 does not give a bijection between assignments of both instances but only an injection, which is enough for our purpose.

Lemma 3. *Let I be an instance of* PMA *with n messages of size τ, period P and $m = P/\tau$. There is an instance J with n messages of size τ' and period $P' = m\tau'$ such that any assignment of J can be transformed into an assignment of I in polynomial time.*

Proof. Fig. 3 illustrates the reductions we define in this proof on a small instance. Let $P = m\tau + r$ with $r \leq \tau$. We define the instance I' as follows: $P' = mP$, $d'_i = md_i$ and $\tau' = m\tau + r$. With this choice, we have $P' = m(m\tau + r) = m\tau'$. Consider an assignment A' of the instance I'. If we let $\tau'' = m\tau$, then A' is also an assignment for $I'' = (P', \tau'', (d'_0, \ldots, d'_{n-1}))$. Indeed, the size of each message, thus the intervals of time used in the first and second period begin at the same position but are shorter, which cannot create collisions. We then use a compactification procedure on A' seen as an assignment of I'', with size of messages multiple of m (see Th.4 of [4] for a similar compactification). W.l.o.g., the first message is positioned at offset zero. The first time it uses in the second period is a multiple of m since its delay is by construction a multiple of m. Then, all other messages are translated to the left by removing increasing values to their offsets, until there is a collision. It guarantees that some message j is in contact with the first one on the first or second period. It implies that either $A'(j)$ or $A'(j) + d_j \mod P'$ is a multiple of m and since d_j is a multiple of m, then both $A'(j)$ and $A'(j) + d_j \mod P'$ are multiples of m. This procedure can be repeated until we get an assignment A'' to I'', such that all positions of messages in the first and second period are multiples of m. Finally, we define A as $A(i) = A''(i)/m$ and we obtain an assignment of I. □

We are interested in the remainder modulo τ of the delays, let $d_i = d'_i\tau + r_i$ be the Euclidean division of d_i by τ. We assume, from now on, that *messages are sorted by increasing r_i*. A **Compact pair**, as shown in Fig. 4 is a pair of messages (i, j) with $i < j$ that can be scheduled using meta-offsets such that $A(i) + (d'_i + 1)\tau = A(j) + d'_j\tau$, i.e. j is positioned less than τ unit of times after i in the second period. The **gap** between i and j is defined as $g = d'_i + 1 - d'_j \mod m$, it is the distance in meta offsets between i and j in the first period. By definition, we can make a compact pair out of i and j, if and only if their gap is not zero.

Lemma 4. *Given three messages, two of them form a compact pair.*

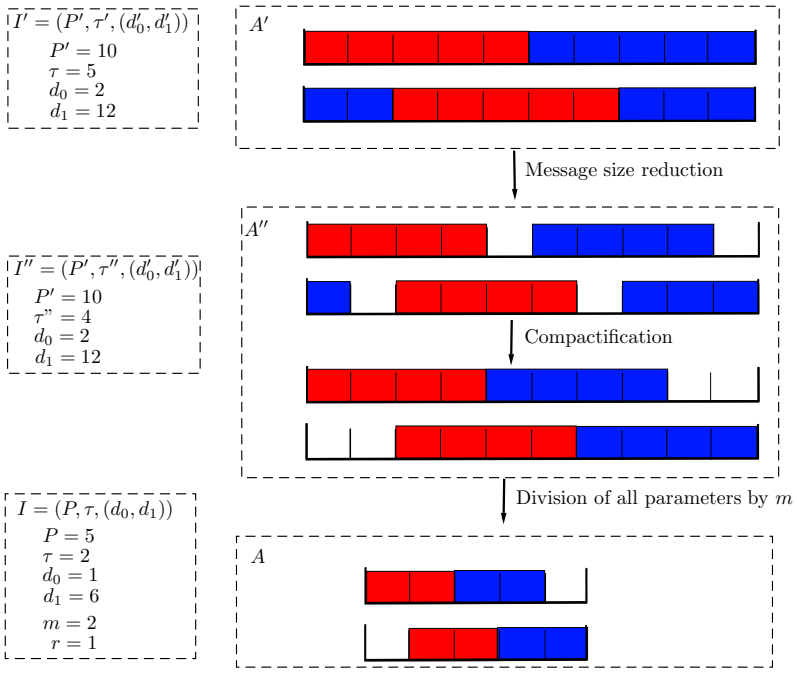

Figure 3: Transformation from A'' to A

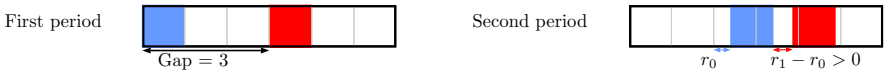

Figure 4: A compact pair scheduled using meta-offsets, with $d'_0 = 2$ and $d'_0 = 0$

Proof. If the first two messages or the first and the third message form a compact pair, then we are done. If not, then by definition $d'_1 = 1 + d'_2 = 1 + d'_3$. Hence, messages 2 and 3 have the same delay and form a compact pair of gap 1. □

Let `Compact Pairs` be the following greedy algorithm: From the messages in order of increasing r_i, a sequence of at least $n/3$ compact pairs is built using Lemma 4. Pairs are scheduled in the order they have been built using meta-offsets. If at some point all compact pairs are scheduled or the current one cannot be scheduled, the remaining messages are scheduled as in `Meta Offset`. The analysis of `Compact Pairs` relies on the evaluation of the number of forbidden meta-offsets. In the first phase of `Compact Pairs`, one should evaluate the number of forbidden offsets when scheduling a compact pair, that we denote by $Fmo_2(A)$. In the second phase, we need to evaluate $Fmo(A)$. When scheduling a message in the second phase, a scheduled compact pair only forbids *three* meta-offsets in the second period. If messages in a pair are scheduled independently, they forbid *four* meta-offsets, which explains the improvement from `Compact Pairs`. We first state a simple lemma, whose proof can be read from Fig. 5, which allows bounding $Fmo_2(A)$.

Lemma 5. *A compact pair already scheduled by `Compact Pairs` forbids at most four meta-offsets in the second period to another compact pair when scheduled by `Compact Pairs`.*

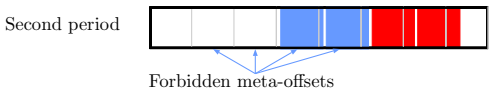

Figure 5: Meta offsets forbidden by a scheduled compact pair (in blue) when scheduling another compact pair (in red)

Theorem 6. *`Compact Pairs` solves* PMA *positively on instances of load less than* $3/8$.

Proof. Let n_2 be the number of compact pairs scheduled in the first phase. When scheduling a new pair, the position of the $2n_2$ messages on the first period forbid $4n_2$ offsets for a compact pair. Indeed, each scheduled message can collide with each of the two messages which form a compact pair. On the second period, we can use Lemma 5 to bound the number of forbidden offsets by $4n_2$. Hence, we have established that during the first phase, the partial solution A satisfies $Fmo_2(A) \leq 8n_2$. This first phase continues while there are available offsets for compact pairs, which is guaranteed when $Fmo_2(A) \leq m$, that is while $n_2 \leq m/8$. Hence, we assume that $n_2 = m/8$.

In the second phase, a compact pair forbids 3 meta offsets in the second period and 2 in the first. Hence, if we let n_1 be the number of messages scheduled in the second phase to build partial assignment A, we have $Fmo(A) \leq n_2 * 5 + n_1 * 3$. `Compact Pairs` can always schedule messages when $Fmo(A)$ is less than m, which is implied by $n_2 * 5 + n_1 * 3 \leq m$. Solving this equation, we obtain that $n_1 \geq \frac{m}{8}$ thus the number of routes scheduled is at least $2n_2 + n_1 \geq \frac{3}{8}m$. Assuming there are exactly $\frac{3}{8}m$ messages to schedule, then $\frac{2m}{8}$ messages are scheduled as compact pairs. It is two third of the $\frac{3}{8}m$ messages, hence Lemma 4 guarantees the existence of enough compact pairs. Therefore, an assignment is always produced when the load is less or equal to $\frac{3}{8}$. □

`Compact Pairs` can be improved by forming compact tuples instead of compact pairs. A compact k-tuple is a sequence of messages $i_1 < \cdots < i_k$ (with $r_{i_1}, \ldots, r_{i_k}$ increasing), for which meta-offsets can be chosen so that, there is no collision, the messages in the second period are in order $i_1, \ldots, i_k$ and for all l, $A(i_l) + (d'_{i_l} + 1)\tau = A(i_{l+1}) + d'_{i_{l+1}}\tau$. The algorithm `Compact k-tuples` works by scheduling compact k-tuples using meta offsets while possible, then scheduling compact $k-1$-tuples and so on until $k = 1$.

Lemma 7. *Given $k + k(k-1)(2k-1)/6$ messages, k of them always form a compact k-tuple and we can find them in polynomial time.*

Proof. We prove the property by induction on k. We have already proved it for $k = 2$ in Lemma 4. Now assume that we have found C a compact $(k-1)$-tuple in the first $(k-1)^3/3$ messages. Consider the next $(k-1)^2 + 1$ messages. If k of them have the same delay modulo τ, then they form a compact k-tuple and we are done. Otherwise, there are at least k different values modulo τ in those $(k-1)^2 + 1$ messages. Each element of the compact $(k-1)$-tuple forbids one value for the delay modulo τ of a new kth element in the tuple. By pigeonhole principle, one of the k messages with distinct delays modulo τ can be used to extend C. We have built a compact k-tuple from at most $(k-1) + (k-1)(k-2)(2k-3)/6 + (k-1)^2 + 1$ messages. It is equal to $k + k(k-1)(2k-1)/6$ which proves the induction. □

Theorem 8. *`Compact 8-tuples` always solves* PMA *positively on instances of load less than* $4/10$*, for instances with* $n \geq 220$*.*

Proof. We need the following fact, which generalizes Lemma 5: A k-tuples forbids $k+j+1$ offsets in the second period when scheduling a j-tuple. It enables us to compute a lower bound on the number of scheduled i-tuples for i equal k down to 1 by bounding $Fmo_i(A)$, the number of forbidden meta-offsets when placing i-tuple in the algorithm. If we denote by n_i the number of compact i-tuples scheduled by the algorithm, we have the following equation:

$$Fmo_i(A) \leq \sum_{j=i}^{k} n_j(j * i + j + i + 1).$$

The equation for n_1 is slightly better:

$$Fmo(A) \leq \sum_{j=1}^{k} n_j(2j + 1).$$

A bound on n_i can be computed, using the fact that A can be extended while $Fmo_i(A) < m$. Lemma 7 ensures there are enough compact k-tuples, when $n - \sum_{j \leq i \leq 8} n_j$ is larger than $i + i(i-1)(2i-1)/6$. A numerical computation of the n_i's shows that `Compact 8-tuples` always finds an assignment when the load is less than $4/10$ and for $n \geq 220$. □

Th. 8 is obtained for $k = 8$. Taking arbitrary large k and using refined bounds on $Fmo_i(A)$ is not enough to get an algorithm working for a load of $41/100$ (and it only works from larger n).

The code computing the n_i can be found on one author's website[1]. To make `Compact 8-tuples` work, there must be at least 220 messages to produce enough compact 8-tuples in the first phase. It is not a strong restriction for two reasons. First, the bound of Lemma 7 can be improved, using a smarter polynomial time algorithm to find compact tuples, which better takes into account repetitions of values and compute the compact tuples in both directions. Second, on random instances, the probability that k messages do not form a compact k-tuples is low, and we can just build the tuples greedily. Therefore, for most instances, forming compact k-uples is not a problem and in practice `Compact 8-tuples` works even for small n.

3.4 Experimental Results

In this section, the performance on random instances of the algorithms presented in Sec. 3 is experimentally characterized. The implementation in C of these algorithms can be found on one author's website[2]. We experiment with several periods and message sizes. For each set of parameters, we try every possible load by changing the number of messages and give the success rate of each algorithm. The success rate is measured on 10000 instances of PMA generated by drawing uniformly and independently the delays of each message in $[P]$.

We consider the following algorithms:

- `First Fit`

[1] `https://yann-strozecki.github.io/`
[2] `https://yann-strozecki.github.io/`

- `Meta Offset`
- `Compact Pairs`
- `Compact Fit`
- `Greedy Uniform`, the algorithm introduced and analyzed in Sec. 5, used for arbitrary τ
- `Exact Resolution` using an algorithm from [4]

Message (i)	0	1	2	3
Delay (d_i)	2	0	3	1
$d_i \bmod \tau$	0	0	1	1

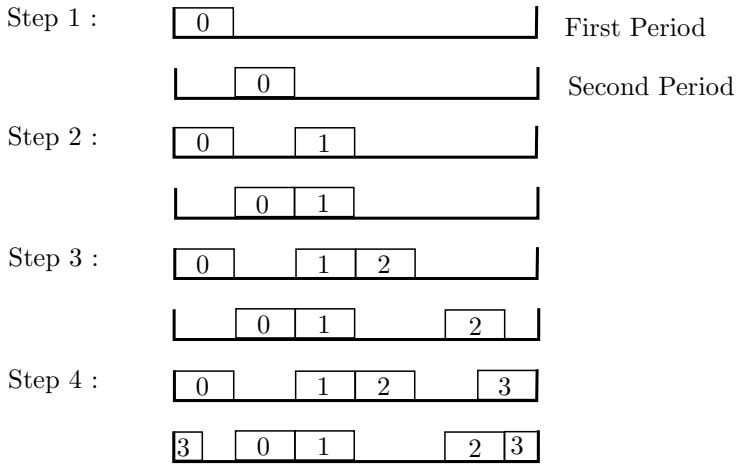

Figure 6: Execution of `Compact Fit` creating two compact pairs with $P = 12$ and $\tau = 2$

The only algorithm we have yet to describe is `Compact Fit`. The idea is, as for `Compact Pairs`, to combine the absence of collision on the first period of `Meta Offset` and the compactness of assignments given by `First Fit`. The messages are ordered by increasing remainder of delay modulo τ, and each message is scheduled so that it extends an already scheduled compact tuples. In other words, it is scheduled using meta offsets, so that using one less for meta offset creates a collision on *the second period*. If it is not possible to schedule the message in that way, the first possible meta-offset is chosen. This algorithm is designed to work well on random instances. Indeed, it is easy to evaluate the average size of the created compact tuples, and from that, to prove Compact Fit works with high probability when the load is strictly less than $1/2$. Fig. 6 shows how `Compact Fit` builds an assignment from a given instance. The messages are ordered by increasing remainder of delay modulo τ. A compact pair is built with messages 0 and 1. Message 2 cannot increase the size of the compact pair, it so create a new uple, completed by message 3

On a regular laptop, all algorithms terminates in less than a second when solving 10000 instances with 100 messages except the exact resolution, whose complexity is exponential in the number of routes (but polynomial in the rest of the parameters). Hence, the exact value of the success rate

given by the exact resolution is only available in the experiment with at most 10 messages (the algorithm cannot compute a solution in less than an hour for twenty messages and high load). Note that while `First Fit`, `Compact Pairs`, `Meta Offset`, `Compact Fit` all run in almost the same time, `Greedy Uniform` seems to be three times longer than the other algorithms to run on instances with 100 messages. It is expected since it must find all available offsets at each step instead of one.

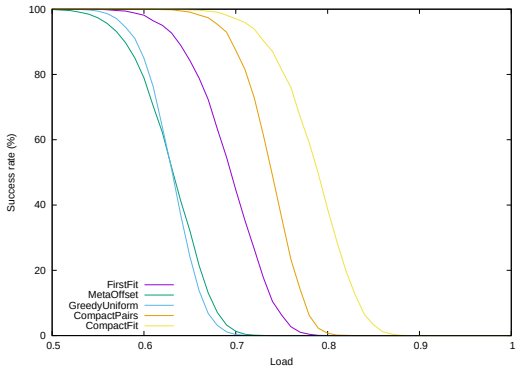

Figure 7: Success rates of all algorithms for increasing loads, $\tau = 1000$, $P = 100,000$

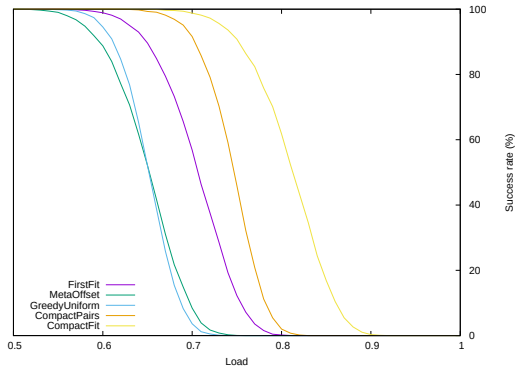

Figure 8: Success rates of all algorithms for increasing loads, $\tau = 10$, $P = 1,000$

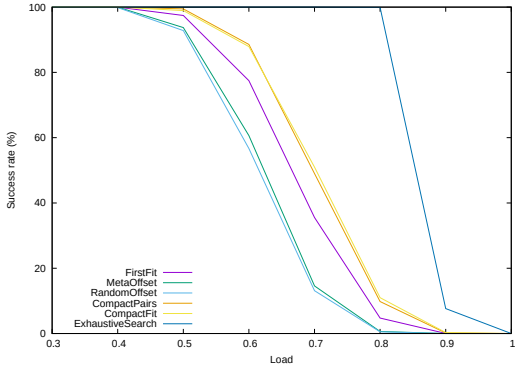

Figure 9: Success rates of all algorithms for increasing loads, $\tau = 1000$, $P = 10,000$

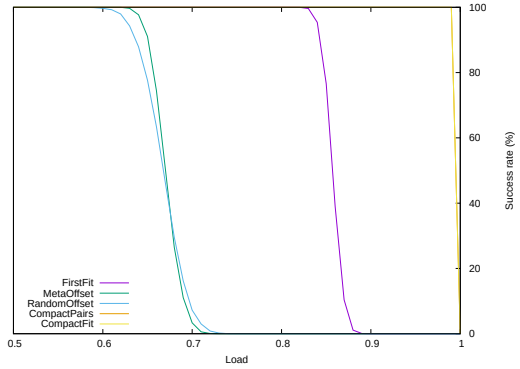

Figure 10: Same parameters as in Fig. 7, delays uniformly drawn in $[\tau]$

For all sets of parameters, the algorithms have the same relative performances. `Meta Offset` and `Greedy Uniform` perform the worst and have almost equal success rate. Remark that they have a 100% success rate for load less than 1/2, while it is easy to build an instance of PMA of load $1/3+\epsilon$ which makes them fail. The difference between the worst case analysis and the average case analysis is explained for `Greedy Uniform`, when $\tau = 1$ in Sec. 5.

`First Fit` performs better than `Meta Offset` while they have the same worst case. `Compact Pairs`, which is the best theoretically also performs well in the experiments, always finding assignments for load of 0.6. `Compact Fit`, which is similar in spirit to `Compact Pairs` but is designed to have a good success rate on random instances is indeed better than `Compact Pairs`, when there

are enough messages.

As demonstrated by Fig. 7 and Fig. 8, the size of the messages have little impact on the success rate of the algorithms, when the number of messages is constant. Comparing Fig. 9 and Fig. 7 shows that for more messages, the transition between success rate of 100% to success rate of 0% is faster. Finally, the results of Exact Resolution in Fig. 9 show that the greedy algorithm are far from always finding a solution when it exists. Moreover, we have found an instance with load 0.8 with no assignment, which gives an upper bound on the highest load for which PMA can always be solved positively.

We also investigate the behavior of the algorithms when the delay of the messages are drawn in $[\tau]$ in Fig. 10. The difference from the case of large delay is that `Compact Pairs` and `Compact Fit` are extremely efficent: they always find a solution for 99 messages. It is expected, since all d'_i are equal in these settings and they will both build a 99-compact tuples and thus can only fail for load 1.

4 From Large to Small Messages

In this section, we explain how we can trade load or buffering in the network to reduce the size of messages up to $\tau = 1$. This further justifies the interest of Sec. 5, where specific algorithms for $\tau = 1$ are given.

4.1 Message of Size One by Increasing the Load

We describe here a reduction from an instance of PMA to another one with the same period and number of messages but the size of the messages is doubled. This instance is equivalent to an instance with $\tau = 1$, by dividing everything by the message size. Thus we can always assume that $\tau = 1$, if we are willing to double the load.

Theorem 9. *Let I be an instance of* PMA *with n messages and load λ. There is an instance J with n messages of size 1 and load 2λ such that an assignment of J can be transformed into an assignment of I in polynomial time.*

Proof. From $I = (P, \tau, (d_0, \ldots, d_{n-1}))$, we build $I' = (P, 2\tau, (d'_0, \ldots, d'_{n-1}))$, where $d'_i = d_i - (d_i \mod 2\tau)$. The instance I' has a load twice as large as I. On the other hand, all its delays are multiples of 2τ hence solving PMA on I' is equivalent to solving it on $I'' = (P/2\tau, 1, (d_0/2\tau, \ldots, d_{n-1}/2\tau))$, as already explained in the proof of Lemma 3.

Let us prove that an assignment A' of I' can be transformed into an assignment A of I. Consider the message i with offset $A'(i)$, it uses all times between $A'(i)$ and $A'(i) + 2\tau - 1$ in the first period and all times between $A'(i) + d_i - (d_i \mod 2\tau)$ to $A'(i) + 2\tau - 1 + d_i - (d_i \mod 2\tau)$ in the second period. If $d_i \mod 2\tau < \tau$, we set $A(i) = A'(i)$, and the message i of I is scheduled "inside" the message i of I', see Fig. 11. If $\tau \leq d_i \mod 2\tau < 2\tau$, then we set $A(i) = A'(i) - \tau$. There is no collision in the assignment A, since all messages in the second period use times which are used by the same message in A'. In the first period, the messages scheduled by A use either the first half of the same message in A' or the position τ before, which is either free in A' or the second half of the times used by another message in A' and thus not used in A. □

Remark that combining Greedy Random and Th. 9 allows to solve PMA on random instances, with probability one when the number of routes goes to infinity and the load is strictly less than

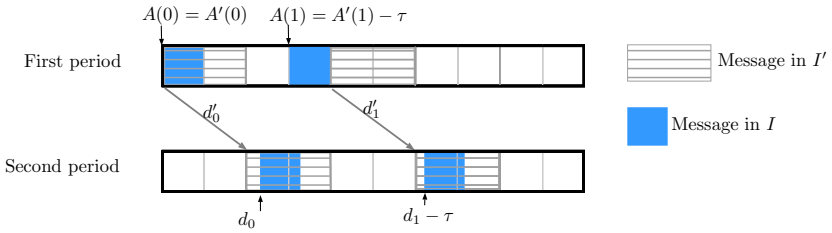

Figure 11: Building I from I'

1/2. This explains why we have not presented nor analyzed in details an algorithm designed for arbitrary τ on random instances, since any greedy algorithm, relying on optimizing $Fo(A)$, cannot guarantee anything for load larger than 1/2. However, in Sec. 3.4, we present Compact Fit, a simple greedy algorithm which exhibits good performance on random instances.

4.2 Trade-off between Latency and Message Size

The problem PMA is simplified version of the practical problem we adress, allowing a single degree of freedom for each message: its offset. If we relax it slightly to be more similar to what is studied in [3], we allow buffering a message i during a time b between the two contention points, which translates here into changing d_i to $d_i + b$. The quality of the solutions obtained for such a modified instance of PMA are worst since the buffering adds latency to the messages. We now describe how we can make a trade-off between the added latency and the size of the messages, knowing that having smaller messages helps to schedule instances with higher load.

The idea is to buffer all messages so that their d_i have the same remainder modulo τ. It costs at most $\tau - 1$ of buffering, which is not so good, since algorithms optimizing the latency do better on random instances, see [3]. However, it is much better than buffering for a time P, the only value for which we are guaranted to find an assignment, whatever the instance. When all delays are changed so that d_i is a multiple of τ, we have an easy reduction to the case of $\tau = 1$, by dividing all values by τ, as explained in the proof of Lemma. 3.

We can do the same kind of transformation by buffering all messages, so that d_i is a multiple of τ/k. The cost in terms of latency is then at most $\tau/k - 1$ but the reduction yields messages of size k. For small size of messages, it is easy to get better algorithm for PMA, in particular for $\tau = 1$ as we have shown in Sec. 5. Here we show how to adapt `Compact Pairs` to the case of $\tau = 2$, to get an algorithm working with higher load.

Theorem 10. *`Compact Pairs` on instances with $\tau = 2$ always solves* PMA *positively on instances of load less than* 4/9.

Proof. We assume w.l.o.g that there are less message with even d_i than odd d_i. We schedule compact pairs of messages with even d_i, then we schedule single message with odd d_i. The worst case is when there is the same number of the two types of messages. In the first phase, if we schedule $n/2$ messages, the number of forbidden offsets is $(2 + 3/2)n/2 = 7n/4$. In the second phase, if we schedule $n/2$ additional offsets, the number of forbidden offsets is bounded by $(1 + 3/2)n/2 + (1 + 1)n/2 = 9n/4$. Hence, both conditions are satisfied and we can always schedule messages when $n \leq (4/9)m$. □

We may want to add less latency to the message using the longest route. A natural idea is choose the message with the longest route as the reference remainder by subtracting its remainder to every delay. As a consequence, this message needs zero buffering. However, the message with the second longest route may have a remainder of $\tau-1$, thus the worst case increase of total latency is $\tau-1$.

Another aim would be to minimize the average latency rather than the worst latency. We prove that we can do the transformation yielding $\tau = 1$ while optimizing the average latency. The only degree of freedom in the presented reduction is the choice of the reference remainder since all other delays are then modified to have the same remainder. Let us define the total latency for a choice t of reference time, denoted by $L(t)$, as the sum of buffering times used for the messages, when t has been removed from their delay. If we sum $L(t)$, from $t = 0$ to $\tau - 1$, the contribution of each message is $\sum_{i=0}^{\tau-1} i$. Since there are n messages, the sum of $L(t)$ for all t is $n\tau(\tau-1)/2$. There is at least one term of the sum less than its average, hence there is a t_0 such that $L(t_0) \leq n(\tau-1)/2$. Hence, the average delay for a message, with t_0 as reference is less than $(\tau-1)/2$.

5 Messages of Size One

When $\tau = 1$ and the load is less than $1/2$, *any greedy algorithm* solves PMA positively since $Fo(A) \leq (4\tau-2)|S| = 2|S|$ where S is the number of scheduled messages. We give, in this section, a method which always finds an assignment for a load larger than $1/2$.

5.1 Deterministic Algorithm

To go above $1/2$ of load, we optimize a potential measuring how many offsets are available for all messages, scheduled or not. Messages are scheduled while possible using any greedy algorithm. Then, when all unscheduled messages have no available offset, we use a Swap operation defined later, which improves the potential. When the potential is high enough, it ensures that there are two messages whose offset can be changed so that a new message can be scheduled.

The algorithm is not greedy, since we allow to exchanging a scheduled message with an unscheduled one. It cannot work online, since it requires to know all delays of the messages in advance.

Definition 11. The potential of a message of delay d, for a partial assignment A is the number of integers $i \in [P]$ such that i is used in the first period and $i+d \mod P$ is used in the second period.

The computation of the potential of a message of delay 3, is illustrated in Fig. 12. The potential of a message counts the configurations which reduce the number of forbidden offsets. Indeed, when i is used in the first period and $i + d \mod P$ is used in the second period, then the same offset is forbidden *twice* for a message of delay d. Hence, the potential of a message is related to the number of possible offsets as stated in the following lemma.

Lemma 12. *Given a partial assignment A of size s, and i an unscheduled message of potential v, then the set $\{o \mid A(i \to o)$ has no collision$\}$ is of size $P - 2s + v$.*

For our algorithm, we need a global measure of quality of a partial assignment, that we try to increase when the algorithm fail to schedule new messages. We call our measure **the potential of an assignment** and we denote it by $Pot(A)$, it is the sum of potentials of all messages in the instance.

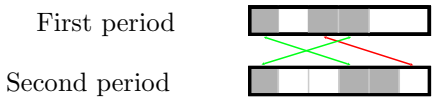

Figure 12: A message of delay 3 has potential 2 in the represented assignment

Delays (d_i)	2	3	6

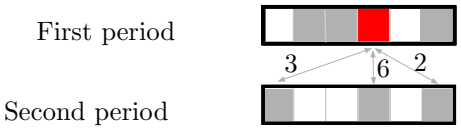

Figure 13: Position colored in red has potential 3, in this assignment

Definition 13. The potential of a position i, for a partial assignment A, is the number of messages of delay d such that $i + d \mod P$ is used by a route scheduled by A.

The potential of a position is illustrated in Fig. 13. Instead of decomposing the global potential as a sum over messages, it can be understood as a sum over positions, as stated in the next lemma.

Lemma 14. *The sum of potentials of all positions used in the first period by messages scheduled by A is equal to $Pot(A)$.*

By definition of the potential of a position, we obtain the following simple invariant.

Lemma 15. *The sum of potentials of all positions for a partial assignment with k scheduled messages is nk.*

As a consequence of this lemma, $Pot(A) \leq nk$. Let us define a **Swap operation**, which guarantees to obtain at last half the maximal value of the potential. Let A be some partial assignment of size s and let i be an unscheduled message of delay d. Assume that i cannot be used to extend A. The Swap operation is the following: select a free position o in the first period, remove the message which uses the position $o + d$ in the second period from A and extend A by i with offset o. We denote this operation by $Swap(i, o, A)$.

Lemma 16. *Let A be some partial assignment of size k and let i be an unscheduled message. If i cannot be used to extend A, then either $Pot(A) \geq kn/2$ or there is an o such that $Pot(Swap(i, o, A)) > Pot(A)$.*

Proof. The positions in the first period can be partitioned into P_u the positions used by some scheduled message and P_f the positions unused. Let V_f be the sum of the potentials of the positions in P_f and let V_u be the sum of the potentials of the positions in P_u. By Lemma 15, since P_f and P_u partition the positions, we have $V_f + V_u = kn$. Moreover, by Lemma 14, $Pot(A) = V_u$, then $V_f + Pot(A) = kn$.

By hypothesis, i cannot be scheduled, then, for all $p \in P_f$, $p + d_i$ is used in the second period. Let us define the function F which associates to $p \in P_f$ the position $A(j)$ such that there is a scheduled route j which uses $p + d$ in the second period, that is $A(j) + d_j = p + d \mod P$. The

function F is an injection from P_f to P_u. Remark now that, if we compare $Swap(i, p, A)$ to A, on the second period the same positions are used. Hence, the potential of each position stay the same after the swap. As a consequence, doing the operation $Swap(i, p, A)$ adds to $Pot(A)$ the potential of the position p and removes the potential of the position $F(p)$.

Assume now, to prove our lemma, that for all p, $Pot(Swap(i, p, A)) \leq Pot(A)$. It implies that for all p, the potential of p is smaller than the potential of $F(p)$. Since F is an injection from P_f to P_u, we have that $V_f \leq V_u = Pot(A)$. Since $V_f + Pot(A) = kn$, we have that $Pot(A) \geq kn/2$. □

Let us precisely describe the algorithm `Swap and Move`: messages are scheduled while possible by `First Fit` and then the Swap operation is applied while it increases the potential. When the potential is maximal, `Swap and Move` schedule a new message by moving at most two scheduled messages to other available offsets. If it fails to do so, `Swap and Move` stops, otherwise the whole procedure is repeated. We analyze `Swap and Move` in the following theorem.

Theorem 17. *`Swap and Move` solves positively* PMA*, in polynomial time, for instances with* $\tau = 1$ *and load less than* $1/2 + (\sqrt{5}/2 - 1) \approx 0,618$.

Proof. We determine for which value of the load `Swap and Move` always works. We let $n = (1/2 + \epsilon)P$ be the number of messages, the load is $1/2 + \epsilon$. We need only to study the case when $n - 1$ messages are scheduled by A and `Swap and Move` tries to schedule the last one, since the previous steps are similar but easier.

Let d be the delay of the unscheduled message. We consider the pairs of times $(o, o + d)$ for $o \in [P]$. Since the message cannot be scheduled, there are three cases. First, o is unused in the first period but $o + d$ is used in the second period. Since there are $n - 1$ scheduled messages, there are $P - n + 1$ such value of o. If a message using the time $o + d$ in the second period can be scheduled elsewhere, so that the unscheduled message can use offset o, then `Swap and Move` succeeds. Otherwise, the message has no possible offsets, which means its potential is equal to $2(\epsilon P - 1)$. The second case is symmetric: o is used in the first period but $o + d$ is unused in the second period. Finally, we have the case o is used in the first period and $o + d$ is used in the second period. There are $2(\epsilon P - 1)$ such values of o. If the two messages using times o and $o + d$ can be rescheduled so that offset o can be used for the unscheduled message, then `Swap and Move` succeeds. This is always possible when one message is of potential at least $2\epsilon P - 1$ and the other of potential at least $2\epsilon P + 1$. Since the messages must be of potential more than $2(\epsilon P - 1)$ and at most $n - 1$, it is satisfied when the sum of the two potentials is at least $2(\epsilon P - 1) + n$.

If we assume that `Swap and Move` was unable to schedule the last message by moving two scheduled messages, the previous analysis gives us a bound on twice $Pot(A)$:

$$2Pot(A) \leq 2(P - n + 1)2(\epsilon P - 1) + 2(\epsilon P - 1)(2(\epsilon P - 1) + n)$$

$$Pot(A) \leq (\epsilon P - 1)(P + n)$$

By Lemma 16, we know that $Pot(A) \geq n(n - 1)/2$, hence `Swap and Move` must succeed when

$$n(n - 1)/2 \geq (\epsilon P - 1)(P + n).$$

By expanding and simplifying, we obtain a second degree inequation in ϵ, $1/4 - 2\epsilon - \epsilon^2 \geq 0$. Solving this inequation yields $\epsilon \leq \sqrt{5}/2 - 1$.

Let us prove that `Swap and Move` is in polynomial time. All Swap operations strictly increase the potential. Moreover, when one or two messages are moved, the potential may decrease but

a message is added to the partial assignment. The potential is bounded by $O(n^2)$ and the move operations all together can only remove $O(n^2)$ to the potential, hence there are at most $O(n^2)$ Swap operations during `Swap and Move`. A Swap operation can be performed in time $O(n)$, since for a given message, all free offsets must be tested and the potential is evaluated in time $O(1)$ (by maintaining the potential of each position). This proves that Swap and Move is in $O(n^3)$. □

Consider a partial assignment of size $n' = (1/2 + \epsilon)P$, and a message of delay d. If we consider all n' used offsets o and all times time $o + d$ in the second period, then o and $o + d$ are both used for at least $n' - (P - n') = 2\epsilon P$ values of o. The potential of any message is thus larger or equal to $2\epsilon P$. When a message cannot be scheduled, its potential is less or equal to $2\epsilon P$, hence it is equal to $2\epsilon P$.

Hence, the potential of any assignment of size n' is at least $2\epsilon Pn$. As a consequence, the method of Lemma 16 will guarantee a non-trivial potential for $2\epsilon Pn < nn'/2$, that is $\epsilon < 1/6$. Any algorithm relying on the potential and the Swap operation cannot be guaranteed to work for load larger than $2/3 = 1/2 + 1/6$. However, we may hope to improve on the analysis of Lemma 16, since it is not optimal: $2\epsilon P$ positions in P_u are not taken into account in the proof.

We conjecture that `Swap and Move` works for load up to $2/3$. On random instances, we expect the potential to be higher than the stated bound and to be better spread on the messages, which would make `Swap and Move` works for larger loads, as it is indeed observed in experiments (see Appendix 5.3).

5.2 Randomized Algorithm for Random Instances

We would like to understand better the behavior of our algorithms on instances drawn uniformly at random. To this aim, we analyze the algorithm `Greedy Uniform`, defined as follows: for each message in the order of the input, choose one of the offsets, which does not create a collision with the current partial assignment, uniformly at random.

We analyze `Greedy Uniform` over random instances: all messages have their delays drawn independently and uniformly in $[m]$. We compute the probability of success of `Greedy Uniform` over all random choices by the algorithm *and all possible instances*. It turns out that this probability, for a fixed load strictly less than one, goes to one when m grows. For a given partial assignment, we are only interested in its trace: the set of times which are used in the first and second period. Hence, if n messages are scheduled in a period of size m, the trace of an assignment is a pair of subsets of $[m]$ of size n. We now show that these traces are produced uniformly by `Greedy Uniform`.

Theorem 18. *The distribution of traces of assignments produced by `Greedy Uniform` when it succeeds, from instances drawn uniformly at random, is also uniform.*

Proof. The proof is by induction on n, the number of messages. It is clear for $n = 1$, since the delay of the first message is uniformly drawn and all offsets can be used. Assume now the theorem true for some $n > 1$. `Greedy Uniform`, by induction hypothesis has produced uniform traces from the first n messages. Hence, we should prove that, if we draw the delay of the $n + 1^{th}$ message randomly, extending the trace by a random possible offset produces a random distribution on the traces of size $n + 1$.

If we draw an offset uniformly at random (among all m offsets) and then extend the trace by scheduling the last message at this offset or fail, the distribution over the traces of size $n + 1$ is the same as what produces `Greedy Uniform`. Indeed, all offsets which can be used to extend the trace

have the same probability to be drawn. Since all delays are drawn independently, we can assume that, given a trace, we first draw an offset uniformly, then draw uniformly the delay of the added message and add it to the trace if it is possible. This proves that all extensions of a given trace are equiprobable. Thus, all traces of size $n+1$ are equiprobable, since they each can be formed from $(n+1)^2$ traces of size n by removing one used time from the first and second period. This proves the induction and the theorem. □

Since `Greedy Uniform` can be seen as a simple random process on traces by Th. 19, it is easy to analyze its probability of success.

Theorem 19. *The probability over all instances with n messages and period m that* ***`Greedy Uniform`*** *solves* PMA *positively is*

$$\prod_{i=m/2}^{n-1} (1 - \frac{\binom{n}{2i-m}}{\binom{m}{i}}).$$

Proof. We evaluate $\Pr(m,n)$ the probability that `Greedy Uniform` fails at the n^{th} step assuming it has not failed before. It is independent of the delay of the n^{th} message. Indeed, the operation which adds one to all times used in the second period is a bijection on the set of traces of size $n-1$. It is equivalent to remove one to the delay of the n^{th} message. We can thus assume that the delay is zero.

Let S_1 be the set of times used in the first period by the $n-1$ first messages and S_2 the set of times used in the second period. We can assume that S_1 is fixed, since all subsets of the first period are equiprobable and because S_2 is independent of S_1 (Th. 19). There is no possible offset for the n^{th} message, if and only if $S_1 \cup S_2 = [m]$. It means that S_2 has been drawn such that it contains $[m] \setminus S_1$. By Th.19, S_2 is uniformly distributed over all sets of size $n-1$. Hence, the probability that $[m] \setminus S_1 \subseteq S_2$ is the probability to draw a set of size $n-1$ which contains $m-n+1$ fixed elements. This proves $\Pr(m,n) = \frac{\binom{n}{2(n-1)-m}}{\binom{m}{n-1}}$.

From the previous expression, we can derive the probability of success of `Greedy Uniform` by a simple product of the probabilities of success $(1 - \Pr(m,i))$ at step i, for all $i \leq n$, which proves the theorem. □

If we fix the load $\lambda = n/m$, we can bound $P(m,n)$ using Stirling formula. We obtain for some constant C, that $P(m,n) \leq C\left(\frac{\lambda^{2\lambda}}{(2\lambda-1)^{2\lambda-1}}\right)^m$. We let $f(\lambda) = \frac{\lambda^{2\lambda}}{(2\lambda-1)^{2\lambda-1}}$. The derivative of f is strictly positive for $1/2 < \lambda < 1$ and $f(1) = 1$, hence $f(\lambda) < 1$ when $\lambda < 1$. By a simple union bound, the probability that `Greedy Uniform` fails is bounded by $C\lambda m f(\lambda)^m$, whose limit is zero when m goes to infinity. It explains why `Greedy Uniform` is good in practice for large m.

5.3 Experimental Results

In this section, the performance on random instances of the algorithms presented in Sec. 5 is experimentally characterized. The settings are as in Sec. 3.4, with $\tau = 1$. The evaluated algorithms are:

- `First Fit`
- `Greedy Uniform`

- `Greedy Potential`, a greedy algorithm which leverages the notion of potential introduced for Swap. It schedules the messages in arbitrary order, choosing the possible offset which maximizes the potential of the unscheduled messages
- `Swap and Move`
- `Exact Resolution`

As in Sec. 3.4, the success rate on random instances is much better than the bound given by worst case analysis. In the experiment presented in Fig. 14, all algorithms succeed on all instances when the load is less than 0.64. `Greedy Uniform` behaves exactly as proved in Th. 19, with a very small variance. The performance of `Swap and Move` and of its simpler variant `Greedy Potential`, which optimizes the potential in a greedy way, are much better than `First Fit` or `Greedy Uniform`. Amazingly, `Swap and Move` always finds an assignment when the load is less than 0.95. `Swap and Move` is extremely close to Exact Resolution, but for $P = 10$ and load 0.9 or 1, it fails to find some assignments, as shown in Fig. 15.

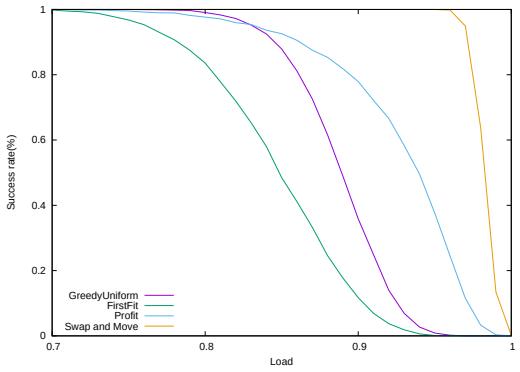

Figure 14: Success rates of all algorithms for increasing loads, $\tau = 1$ and $P = 100$

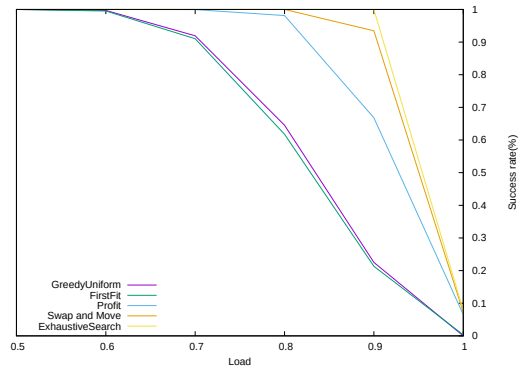

Figure 15: Success rates of all algorithms for increasing loads, $\tau = 1$ and $P = 10$

Finally, we evaluate the computation times of the algorithms to understand whether they scale to large instances. We present the computation times in Fig. 16 and we choose to consider instances of load 1, since they require the most computation time for a given size. The empirical complexity of an algorithm is evaluated by a linear regression on the function which associates to $\log(n)$, the log of the computation time of the algorithm on n messages. `First Fit`, `Greedy Uniform` and `Swap and Move` scale almost in the same way, with an empirical complexity slightly below $O(n^2)$, while `Greedy Potential` has an empirical complexity of $O(n^3)$. The empirical complexity corresponds to the worst case complexity we have proved, except for `Swap and Move` which is in $O(n^3)$ worst case. There are two explanations: most of the messages are scheduled by the fast `First Fit` subroutine and most Swap operations improve the potential by more than 1, as we assume in the worst case analysis.

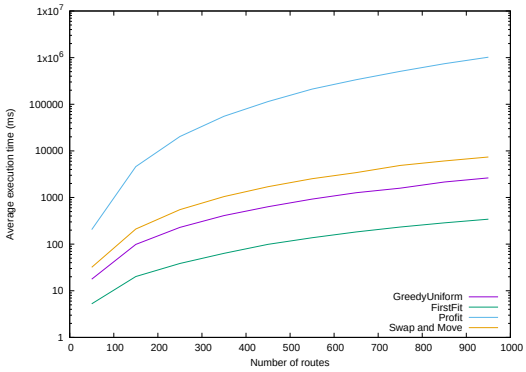

Figure 16: Computation time (logarithmic scale) function of the number of messages of all algorithms on 10000 instances of load 1

6 Conclusion

In this article, we have proved that there is always a solution to PMA and that it can be found in polynomial time for large τ and load 0.4 or for $\tau = 1$ and load 0.61. Moreover, the performance of the presented algorithms over average instances are shown to be excellent empirically but also theoretically for `Greedy Uniform`. Hence, we can use the simple algorithms presented here to schedule C-RAN messages *without buffer nor additional latency*, if we are willing to use only half the bandwidth of the shared link.

Several questions on PMA are still unresolved, in particular its NP-hardness and the problem of doing better than load 0.5 for arbitrary τ and random instances. We could also consider more complex network topologies with several shared links. `First Fit` or `Meta Offset` can easily be transfered to this context, and we could also try to adapt `Compact Pairs` or `Swap and Move`. Finally, to model networks carrying several types of messages, different message sizes must be allowed, which would require to design an algorithm which does not use meta-offsets.

References

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

[2] 3GPP. *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service requirements for the 5G system;*. Stage 1 (Release 16).

[3] Dominique Barth, Maël Guiraud, Brice Leclerc, Olivier Marcé, and Yann Strozecki. Deterministic scheduling of periodic messages for cloud ran. In *2018 25th International Conference on Telecommunications (ICT)*, pages 405–410. IEEE, 2018.

[4] Dominique Barth, Maël Guiraud, and Yann Strozecki. Deterministic scheduling of periodic messages for cloud ran. *arXiv preprint arXiv:1801.07029*, 2018.

[5] Dominique Barth, Maël Guiraud, and Yann Strozecki. Deterministic contention management for low latency cloud RAN over an optical ring. In *ONDM 2019 - 23rd International Conference on Optical Network Design and Modeling (ONDM 2019)*, 2019.

[6] Federico Boccardi, Robert W Heath, Angel Lozano, Thomas L Marzetta, and Petar Popovski. Five disruptive technology directions for 5G. *IEEE Communications Magazine*, 52(2):74–80, 2014.

[7] Yannick Bouguen, Eric Hardouin, Alain Maloberti, and François-Xavier Wolff. *LTE et les réseaux 4G*. Editions Eyrolles, 2012.

[8] Aellison Cassimiro T dos Santos, Ben Schneider, and Vivek Nigam. Tsnsched: Automated schedule generation for time sensitive networking. In *2019 Formal Methods in Computer Aided Design (FMCAD)*, pages 69–77. IEEE, 2019.

[9] Norman Finn and Pascal Thubert. Deterministic Networking Architecture. Internet-Draft draft-finn-detnet-architecture-08, Internet Engineering Task Force, 2016. Work in Progress. URL: `https://tools.ietf.org/html/draft-finn-detnet-architecture-08`.

[10] Claire Hanen and Alix Munier. *Cyclic scheduling on parallel processors: an overview*. Université de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique, 1993.

[11] W. Howe. Time-scheduled and time-reservation packet switching, March 17 2005. US Patent App. 10/947,487. URL: `https://www.google.com.gt/patents/US20050058149`.

[12] Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic multiprocessor scheduling. In *PARLE'91 Parallel Architectures and Languages Europe*, pages 166–178. Springer, 1991.

[13] B. Leclerc and O. Marcé. Transmission of coherent data flow within packet-switched network, June 15 2016. EP Patent App. EP20,140,307,006. URL: `https://www.google.com.gt/patents/EP3032781A1?cl=en`.

[14] Eugene Levner, Vladimir Kats, David Alcaide López de Pablo, and TC Edwin Cheng. Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering*, 59(2):352–361, 2010.

[15] Richard M Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR spectrum*, 33(4):843–883, 2011.

[16] China Mobile. C-RAN: the road towards green RAN. *White Paper, ver*, 2, 2011.

[17] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Incremental flow scheduling and routing in time-sensitive software-defined networks. *IEEE Transactions on Industrial Informatics*, 14(5):2066–2075, 2017.

[18] Time-Sensitive Networking Task Group of IEEE 802.1. Time-sensitive networks for fronthaul. July 2016. IEEE P802.1/D0.4.

[19] Alex J Orman and Chris N Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.

[20] Anna Pizzinat, Philippe Chanclou, Fabienne Saliou, and Thierno Diallo. Things you should know about fronthaul. *Journal of Lightwave Technology*, 33(5):1077–1083, 2015.

[21] Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

[22] Wilfried Steiner, Silviu S Craciunas, and Ramon Serna Oliver. Traffic planning for time-sensitive communication. *IEEE Communications Standards Magazine*, 2(2):42–47, 2018.

[23] Z Tayq, L Anet Neto, B Le Guyader, A De Lannoy, M Chouaref, C Aupetit-Berthelemot, M Nelamangala 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*, pages 1–3, 2017.

[24] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. *Journal of Scheduling*, 7(5):333–348, 2004.