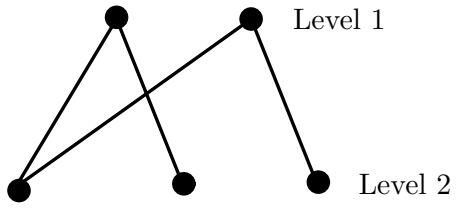
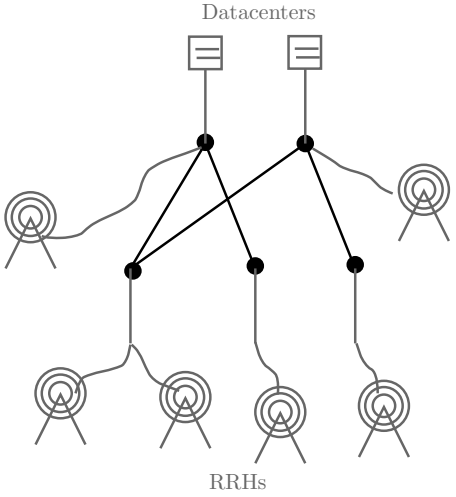
1 Implementation details

1.1 Model of the generated graphs

The graphs we study here are some two levels trees. First of all, two parameters set up the number of each node (contention point) at each level of the tree. The first level of the tree models the datacenters, and the second level of the trees models the switches in the core of the network. Here is an example of a tree generated with 2 nodes of level 1 and 3 node of level 2.



This architecture models a real topology which can correspond to the following figure, in which we add some links directly connected to somme RRHs.



Thus, the first challenge is to generate some bipartite graphs which seems to be a good model for the internet topology as mentioned in [1]. This papers also gives us some background to generate some good random bipartite graphs. In a first time, we chose a simple model to generate our random bipartite graphs. Thus, we first randomly draw a link between each couple of node (u,v), where u is a node of level 1 and v a node of level 2, with a given probability (the same for each couple). If the generated graph is not connexe, we generate another

bipartite graph. A **flow** is a group of antennas, which communicate with a same datacenter. There is a flow corresponding to each arc previously drawn. This means that there is a group of antennas all connected to a same switch (node of level 2). Also, we generate a flow directly connected to each datacenter. For each flow, we randomly draw between 1 and N antennas. We say there is a **route** for each antenna generated. The routing of the routes in a same flow is the same. Note that we chose to generate several antenna per flows. TODO: Je ne me souviens plus de la justification

1.2 Different implemented algorithms

Consider that the size τ of the messages is the same for all routes. Also, the messages are sent in one packet, which can not be fragmented in the network.

We first introduce a basic routine that helps us to test if the messages sent on a route at a given departure time can pass through the entire route without collisions.

Algorithm 1 MessageCollisions

Input: A route r, a departure time, and a way of the message (FORWARD/BACKWARD).
Output: 1 if the messages can use the route with the given departure time, 0 otherwise.

```
for all Arcs in the route do
    if There is a collision with the previous scheduled messages then
        return 0
    end if
end for
return 1
```

1.3 Algorithms without waiting times

1.3.1 Greedy Prime

The idea here is to send the messages as soon as possible on each route. This is a greedy algorithm that does not try to optimize anything.

Algorithm 2 Greedy Prime

Input: A graph, a set of routes, a period P
Output: A P-periodic assignment in p $\leq P$, or FAILURE

```
for all routes i do
  tmp = 0
  while !MessageCollisions(i,tmp,FORWARD) ||
  !MessageCollisions(i,tmp+routeLength(i),BACKWARD) do
    tmp++;
    if tmp> P then
      return FAILURE
    end if
  end while
  DepartureTime(i) = tmp;
end for
return departureTime
```

This algorithm treats the routes one by one by id (arbitrarily chosen), and set the departure time of the messages on the route as soon as possible.

1.3.2 Greedy Min

We now try a smarter greedy algorithm. We start for the greedy algorithm proposed in [2]. This algorithm works on one forward and one backward period. The idea is to cut the forward period in meta intervals of size τ. Then, for each routes, we try each free meta interval in the forward period until the message can pass in the backward period without collisions. Here, the principle is the same, but instead of looking at only one backward period, we take into consideration all the contention points of the route in the same time. Since we did not study this topology enough for now, we do not have the theoretical result that ensure us to find a solution under a given load, but we use the idea of the algorithm and try to adapt it in order to optimize the chances of success. Indeed, instead of trying the meta intervals one by one and scheduling the route on the first meta interval that gives not collisions, we try to minimize the size lost in all the collisions points of the route. The size lost is the number of tics between the end of the previous message and the beginning of the message scheduled. The goal is then to choose, for a route i the meta interval that allow the message to pass though all the arc of its route without collisions, but also that minimise the sum of the size lost in every contention point of the route.

Algorithm 3 sizeLost

Input: A route r, a departure time t
Output: -1 if the message can not pass without collisions, the size lost otherwise

```
tmp ← 0
for all Arcs j in the route (forward AND backward) do
  if There is a collision with the previous scheduled messages then
    return −1
  else
    tmp += numberOfTicsLost(t,j)
  end if
end for
return tmp
```

Algorithm 4 Greedy Min

Input: A graph, a set of routes, a period P
Output: A P-periodic assignment in p $\leq P$, or FAILURE

```
for all routes i do
  minTicLost ← INT MAX
  minId = −1
  for all meta interval j do
    tmp = sizeLost(r,j × τ)
    if (tmp != −1) AND (tmp ¡ minTicLost) then
      minTicLost ← tmp
      minId ← j
    end if
  end for
  if minId = −1 then
    return FAILURE
  end if
  departureTime(i) = j
end for
return departureTime
```

1.4 Algorithms with waiting times

We now allow the messages to be buffered in the BBUs. Thus we can manage the messages in both the RRH and the BBU.

1.4.1 Greedy Loaded

In this algorithm, we choose to first take care of the more critical contention points first. Thus, we sort the arcs of the graph for the one in which there is

the most of routes, to the one in which there is the less of routes. Then we take first the more loaded link and we schedule the routes on it in two steps. First, we search the lowest departure time such that there is no collisions in the way forward, then we do it again on the way backward. Then, we can have some waiting times but it gives us a greater degrees of freedom to find an assignment.

Algorithm 5 Greedy Loaded

```
Input: A graph, a set of routes, a period P
Output: A P-periodic assignment
  for all arcs i sorted by decreasing number of routes using i do
    for all route j on i do
      if j has not been scheduled yet then
        tmp = 0
        while !MessageCollisions(j,tmp,FORWARD) do
          tmp++;
          if tmp> P then
            return FAILURE
          end if
        end while
        DepartureTime(j) = tmp;
        tmp2 = tmp + routeLength(j);
        tmp = tmp2;
        while !MessageCollisions(j,tmp,BACKWARD) do
          tmp++;
          if tmp> P + tmp2 then
            return FAILURE
          end if
        end while
        waitingTime(j) = tmp-tmp2;
      end if
    end for
  end for
  return departureTime, waitingTime
```

References

[1] F. Tarissan, B. Quoitin, P. Mérindol, B. Donnet, J.-J. Pansiot, and M. Latapy, "Towards a bipartite graph modeling of the internet topology," *Computer Networks*, vol. 57, pp. 2331–2347, Aug. 2013.

[2] D. Barth, M. Guiraud, B. Leclerc, O. Marce, and Y. Strozecki, "Deterministic scheduling of periodic messages for cloud RAN," in *2018 25th International Conference on Telecommunications (ICT) (ICT 2018)*, (Saint Malo, France), June 2018.