1 Graphs of conflic depth two

The star topology has a conflict depth one. We now look at the topologies with a conflict depth two.

In order to generate some random digraphs $G = (V,A)$ of conflict depth two, we first generate some random bipartite graphs $G' = (V',E')$, which represent the core of the network. Indeed, it seems to be a good idea to model the internet topology as mentioned in [1]. This papers also gives us some background to generate some good random bipartite graphs. V' is composed of two sets: The first set S_1, of size a, models the last switch before the data-centers, and the second set S_2, of size b, models some distant switches, reparted on the filed. We fix a and b and we uniformly draw the edges of E' with a given probability. Note that $|E'| = a \times b$. If the generated graph G' is not connected, we generate another bipartite graph. Figure **??** is an example of a random bipartite graph generated, with $a = 2$, $b = 3$ and $|E'| = 4$.

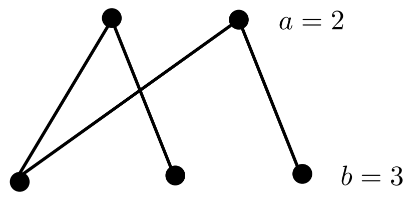

Figure 1: A random bipartite graph G'.

We now use G' to build our routed network G In order to create some arcs with conflict, the vertices of G' are extended to become some arcs in G. Let us call **conflict arcs** those arcs. The egdes of E' becomes some arcs of A, that we will call **core arcs**.

Figure **??** and figure **??** show how the first part of G is build from G'.

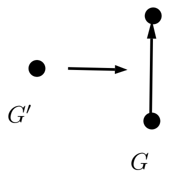

Figure 2: A node of G' represent an arc in G.

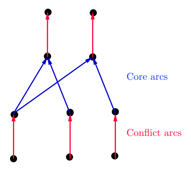

Figure 3: A node of G' represent an arc in G.

We now want to create some **final arcs**, that will model the last link before the antennas. We generate $1 \leq k \leq K$ final arcs before each contention arc (u,v). We add k new vertices $\{w_1, \ldots, w_k\}$ of indegree 0 and outdegree 1 to the set of vertices V. The new arcs $\{(w_1,u), \ldots, (w_k,u)\}$ are then added to A.

Each route of the set of routes $\mathcal{R}$ of the routed network (G,$\mathcal{R}$) is composed either of one final arc and one contention arc (obtained from the set S_1 of arcs in G'), or one final arc, two contention arcs (one from S_1, one frome S_2)) and the core arc between those two contention arcs. Note that one final arc belongs to one and only one route, while the core and contention arcs can be shared by several routes.

Figure **??** and figure **??** shows how the final arcs are generated and define the routes of the routed network (G,$\mathcal{R}$). The routes of the same color share the same arcs after the final arc.

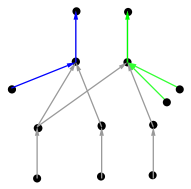

Figure 4: Final arcs before contention arcs obtained from S_1.

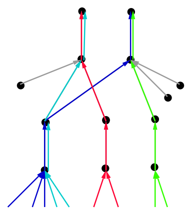

Figure 5: Final arcs before contention arcs obtained from S_2.

The obtained routed network (G,$\mathcal{R}$) models a meshed network. The vertices of indegree 0 of final arcs represent the antennas, the contention arcs from S_1 represent the last link before the datacenters, and the other nodes and arcs of the graph represent some links and switch of the network. It appears that this topology is realitic in core networks. Figure **??** shows the shape of the network modelized.

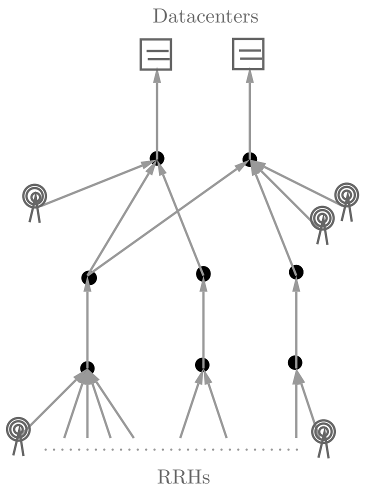

Figure 6: A network represented by a graph G.

2 Implemented algorithms

2.1 Simplification of the model

The steams are composed of one datagram. For simplicity in the notations, the datagram of the stream on the route r is then denoted d_r. All links have the same speed, so $l(d,u,r) = 0$, for all vertex u and route r of the network. All the datagrams have the same size $|d_r| = \tau$. Also, we do not allow the messages to be buffered in the nodes of the network so $l(d,u,r) = 0$, exept the last node of the routes, which correspond to the datacenter. The **waiting time** of a datagram d_r on a route r is defined by $w_r = \{\theta_{\rho_r}(d_r) - t(d_r,v,r)$. This is the time it waits in v, the last node of the route before beeing sent back.

The presented algorithm are designed to solve the problem PAZL and PALL. A **partial assignment**, is a set of departure times and waiting times $\{(m_0,w_0),\ldots,(m_k,w_k)\}|k < n$, where n is the number of routes in the network.

In most of the presented algorithm, the following subroutine is used. Given a route r, an offset m_r, a partial assignment of the routed network, and a way of the message, it returns 1 if the message sent at time m_r (if the way is forward) or w_r (if the way is backward) does not collide with the other messages.

Algorithm 1 MessageNoCollisions

```
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 and without collisions, 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
```

2.2 Algorithm to solve PAZL

2.2.1 Greedy Prime

The basic idea is to try to schedule the routes one by one. Here, given a route, we try all the departure time for the datagram at the begining of the route, and we assign to the route the first departure date that allows the datagram to pass the arcs without collisions with the routes already scheduled.

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
    date = 0
    while !MessageCollisions($i$,date,FORWARD) || !MessageCollisions($i$,date+routeLength($i$),BACKWARD) do
      date++;
      if date> $P$ then
        return FAILURE
      end if
    end while
    DepartureDate($i$) = date;
  end for
  return departureDate
```

The complexity of this algorithm is $\mathcal{O}(n \times P)$, with n the number of routes and P the period. It is very bad, since it depends of the period.

2.2.2 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 $\leftarrow 0$
  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 $\leftarrow$ INT MAX
    minId $= -1$
    for all meta interval $j$ do
      tmp = sizeLost($r$,$j \times \tau$)
      if (tmp != $-1$) AND (tmp ¡ minTicLost) then
        minTicLost $\leftarrow$ tmp
        minId $\leftarrow j$
      end if
    end for
    if minId $= -1$ then
      return FAILURE
    end if
    departureTime($i$) $= j$
  end for
  return departureTime
```

2.3 Algorithms to solve PALL

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

2.3.1 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.