Dynamic resource allocations in virtual networks through a knapsack problem's dynamic programming solution

Vianney Kengne Tchendji*, Yannick Florian Yankam*

*Department of Mathematics and Computer Science
Faculty of Science
University of Dschang
PO Box 67, Dschang-Cameroon
vianneykengne@yahoo.fr, yyankam@yahoo.fr

RÉSUMÉ. La multitude des services à forte valeur ajoutée offert par Internet et améliorés considérablement avec l'intégration de la virtualisation réseau et de la technologie des réseaux définis par logiciels (Software Defined Networking), suscite de plus en plus l'attention des utilisateurs finaux et des grands acteurs des réseaux informatiques (Google, Amazon, Yahoo, Cisco, ...); ainsi, pour faire face à cette forte demande, les fournisseurs de ressources réseau (bande passante, espace de stockage, débit, ...) doivent mettre en place les bons modèles permettant de bien prendre en main les besoins des utilisateurs tout en maximisant les profits engrangés ou le nombre de requetes satisfaites dans les réseaux virtuels. Ce besoin est d'autant plus urgent que les requêtes des utilisateurs peuvent être interdépendantes, imposant de ce fait au FIP des contraintes de satisfaction mutuelle des requêtes, ce qui complexifie encore plus le problème. Dans cette optique, nous montrons que le problème d'allocation des ressources aux utilisateurs en fonction de leurs requetes, se ramène à un problème de sac à dos et peut par conséquent être résolu de façon efficiente en exploitant les meilleures solutions de programmation dynamique pour le problème de sac à dos. Notre contribution considère l'allocation dynamique des ressources comme une application de plusieurs instances du problème de sac à dos sur des requetes à valeurs variables.

ABSTRACT. The high-value Internet services that have been significantly enhanced with the integration of network virtualization and Software Defined Networking (SDN) technology are increasingly attracting the attention of end-users and major computer network companies (Google, Amazon, Yahoo, Cisco, ...). In order to cope with this high demand, network resource providers (bandwidth, storage space, throughput, etc.) must implement the right models to understand and hold the users' needs while maximizing profits reaped or the number of satisfied requests into the virtual networks. This need is even more urgent that users' requests can be linked, thereby imposing to the InP some constraints concerning the mutual satisfaction of requests, which further complicates the problem. From this perspective, we show that the problem of resource allocation to users based on their requests is a knapsack problem and can therefore be solved efficiently by using the best dynamic programming solutions for the knapsack problem. Our contribution takes the dynamic resources allocation as a multiple knapsack's problem instances on variable value requests.

MOTS-CLÉS : Réseau virtuel, allocation des ressources, sac à dos, programmation dynamique, fournisseur de services, fournisseur d'infrastructures

KEYWORDS : Virtual network, ressource allocation, knapsack, dynamic programming, service provider, infrastructure provider

1. Introduction

The limits of the Internet (security, architectural rigidity due to IP protocol, ...) like its resistance to the adoption of new services (such as VOD, telephony over IP, etc) generally known as the phenomenon of Internet ossification [3, 4], led to rethink its architecture. This is how network virtualization was proposed, the idea being the maximum exploitation of physical resources through their sharing and reusability in order to meet the dynamic needs of users ; the integration of the Software Defined Networking (SDN) [2] allowed to better face this resources allocation challenge (known as virtual network embedding problem [10]) through a central equipment called controller, which defines the management policies of the network. This resource allocation is a subproblem of a most global one, commonly known as the Virtual Network Embedding (VNE), which is NP-hard to solve [5] because of the number of constraints involved.

Nowadays, since the network virtualization involves the Internet operators to be divided into infrastructure providers (InP) who hold the physical resources and the service providers (SP) who exploit these resources to offer services, both parts must setup appropriate techniques to match their resources allocation with the varied requests of end-users [5]. Thus, techniques such as auctions or game theory [7] can be used to allocate these resources, although they do not always make it possible to decide in all cases. [6] proposes a resource allocation method also based on this auction approach, but this method focuses on the satisfaction of the interests of resource providers rather than customers. In order to take into account the multiple constraints related to the allocation, [13] proposes a technique of energy allocation based on the knapsack problem with restrictions on the power quantity. But, this approach only offers approximate solutions that are not close to optimal. In addition, [7, 6, 13] does not consider the dependency between the users' requests. In fact, the users' requests can be either totally independent (exclusive request for storage space or bandwidth or throughput), or dependent on each other (storage and bandwidth, computing capacity and storage, etc.). The main motivation of this paper is to improve the resources allocation process in the virtual network by providing some solutions to the drawbacks cited above concerning the works [7, 6, 13]. So, our contribution in this paper is to propose a 0-1 knapsack-based resource allocation approach both in presence of independent and dependent requests. We describe our contribution through four points :

– a modeling of the dynamic resources allocation problem as a 0-1 knapsack problem. We exploit a dynamic programming solution of the 0-1 knapsack problem to propose a solution for the dynamic resources allocation one in the case of independent requests ;

– the identification of the possible types of dependencies between the users' requests and the proposition of a method for building dependency graphs modeling the interdependence between the users'requests. We distinguish dependency graphs with one or more connected components ;

– the proposition of the solutions to the different dependency cases on the basis of the dynamic programming solution used above to face independent requests.

This work resumes our previous work [1] in which we were only interested in independent requests. Here, we removed this assumption by considering a number of constraints between the requests. This greatly increase the complexity of the problem considered. In fact, the previous work is only a special case of this one.

The rest of this paper is organized as follows : in section 2, we present network virtualization and SDN paradigms. Section 3 firstly presents a formulation of the resource allocation problem, showing the equivalence with the knapsack one. Secondly, this section describes the resolution method of an alleged version of the resource allocation problem, through a dynamic programming solution of to the 0-1 knapsack problem. Section 4 enhances our study to the dependent requests, section 5 proposes some resources allocation solutions for this case and section 6 shows the simulations results. A conclusion ends the paper.

2. Network virtualization and SDN paradigms

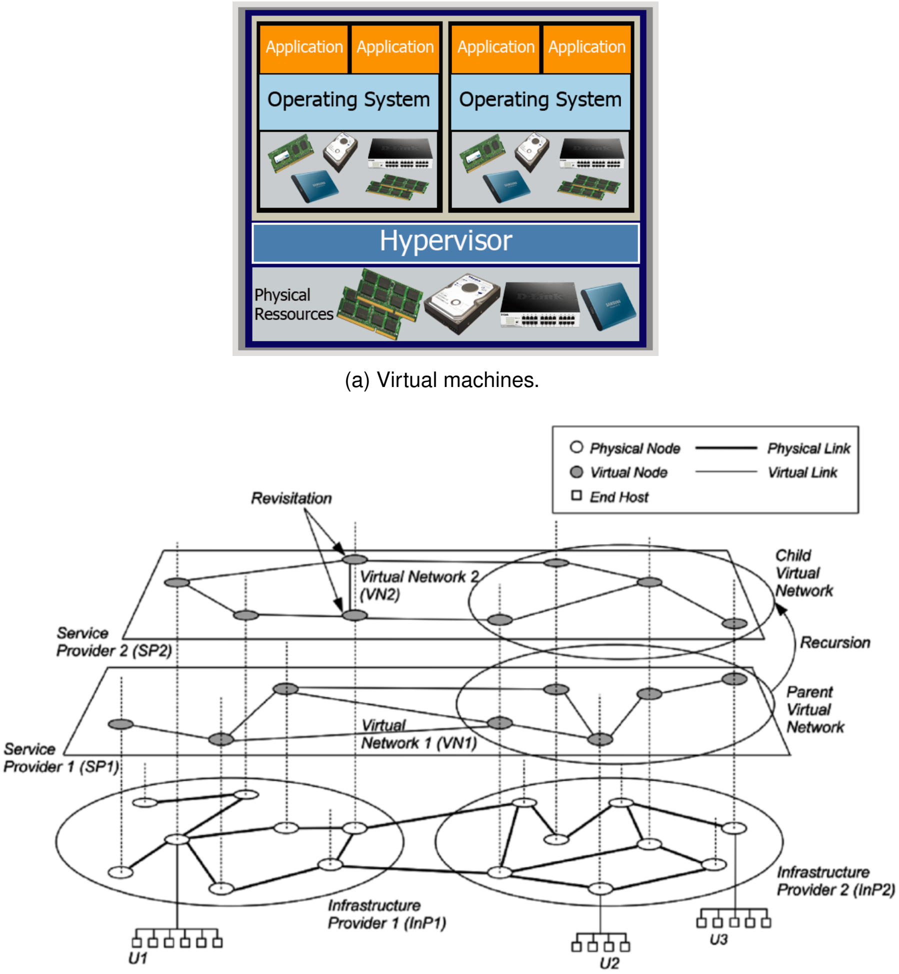
Our work environment is made up of several virtual networks under the supervision of a network controller. A network controller is a network equipment which defines and hosts all the network management policies (see figure 2).

2.1. Virtual networks

A virtual network is a set of virtual devices interconnected by virtual links through a physical infrastructure [4]. In each virtual network, we find components created from a physical component by a special software called hypervisor : these are virtual machines [9] (see figure 1a). Thus, the resources used within a virtual network are provided by the substract network (see figure 1b). Basic physical network resources are provided by an Infrastructure Provider (InP) (see figure 1b). This InP allocates resources to service providers (SP) which create virtual networks to exploit them. There are three levels of resource allocation : virtual network, SP and InP; all of these levels are under the supervision of the controller which can initiate cooperation requests with other InPs when needed. Without this controller, it would not be easy to manage resources with a large virtual network instances.

2.2. The Software Defined Networking solution

Software Defined Networking (SDN) is a new network architecture paradigm where the control plane is completely decoupled from the data plane for each network equipment [11]. The control plane is a part of network which permits to calculate the network topology or to exchange routing information, while data plane or forwarding plane is a part of network where the packets are commutated. A network controller who have the control plane, defines the network management policies (routing, bandwith allocation, topology discovery,...) and assign it to the equipments (see figure 2). This decoupling allows to deploy a monitoring plane on standard servers with flexible computing capabilities [12], compared to conventional switches. Thus it opens the opportunity to design an efficient centralized control plane. In addition, the creation of a standardized API (Application Programming Interface) between the control plane and the data plane allows developing network services. The control plane is capable of injecting states in the network elements.



(a) Virtual machines.

(b) A network virtualization environment.

Figure 1 – Virtualization principles.

3. The resource allocation problem

3.1. Problem description

Intuitively, resource allocation is a problem of finding the best way to satisfy the most important parts of possible requests from a given set, taking into consideration several constraints involved [6]. It can also consist in satisfying a less important range of requests submitted with the same constraints. There are several problem formulations for virtual network provision [6, 8]. However, these different formulations focus on the allocation of virtual links and bandwidth [8] in a restricted virtual network ; these formulations would be more general if the storage space, computing capacity and a set of virtual networks were also considered. Another work [13] proposes in the context of the Internet of Things, a power allocation knapsack-based model which approaches the optimal solution, whe-

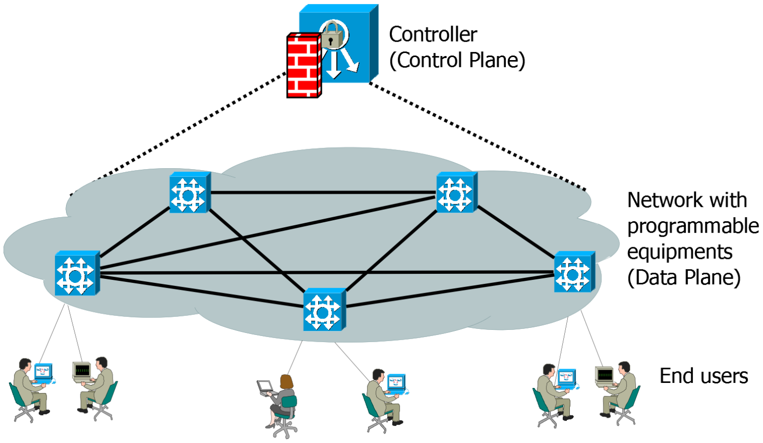


Figure 2 – The SDN paradigm.

reas ours allows to reach it using the dynamic programming solution for our resource allocation problem. In this work we look at this allocation problem as a sharing problem, that is, a problem from which we have resources to share among multiple users. The SDN controller ensure the monitoring and the provision of that resources to the end-users ; this controller can also initiate and manage some cooperation between Infrastructure Providers (InP) to get the resources matching the users'constraints. It is therefore an optimization or decision problem that takes as input :

– a set of n applicants. In our context we associate it to the term of user ;

– limited common resource (s) ;

– a common language for expressing preferences and preferences of n users on the resource (s) ;

– a set of constraints on the possible resources to be allocated ;

– an optimization or decision criterion.

As output, we have a resource allocation model, matching the constraints and optimize the criterion. Note that shared resources can be continuous (split), indivisible, discrete or mixed, though in this paper, we consider divisible and shareable resources. This means that a supplier can divide the resources in its basket before sharing them. In this light, resource allocations can be defined and characterized in the following ways :

Definition 1 : Let be a population $P = p_1, p_2, ..., p_n$ of n requests and a set of m resources $R = r_1, r_2, ..., r_m$ owned by a resource provider. A resource allocation between these n applicants is a list of n baskets containing the resources $r_i \subseteq R$ obtained by each applicant, matching the following properties : $\cup_{i \in \{1,2,..m\}} r_i = R$ and $\cap_{i \in \{1,2,..m\}} r_i = \phi$.

We define the physical infrastructure provider network as an undirected graph $G = (N, L)$ where N is a set of nodes and L is a set of links. Similarly, the virtual network of a service provider is defined as a graph $G^{'} = (N^{'}, L^{'})$ in which $N^{'}$ and $L^{'}$ are the nodes and virtual links built on the substrate network of an InP. Since each resource is associated with a constraint, at each node $n \in N$ we also associate a constraint $C^N(n)$ and with each link $l \in L$ a constraint $C^L(l)$. These constraints can represent at the level of nodes, constraints on the portion of resources available for packets process and delay constraints at the link level.

At the request of a user (see figure 3), the SP submits a request composed of a set of resources that it wants to get from the InP_k. This request consists of a matrix in which the SP specifies its needs.

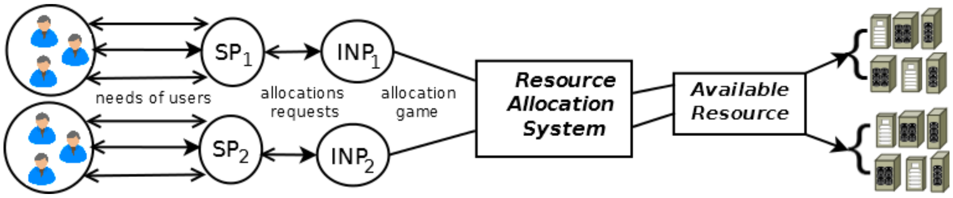


Figure 3 – Ressource allocation process.

This matrix defines the SP's needs (resource and quantity) to satisfy the end users. The physical InP ensures that requested resource quantities do not exceed the total capacity available at the physical network level. In all cases, for a set of requests to satisfy according to given criteria, a set $D = d_1, d_2, ..., d_n$ of n allocation requests to be satisfied, a quantity of available resources $W \in N$ at time t, a quantity $p_i \in N \setminus \{0\}$ of the resource i wanted through the application $d_i \in D$ and criteria $v_i \in N \setminus \{0\}$ to optimize when selecting grant requests to satisfy, the problem can be summarized as :

$$min \sum_{i=1}^{n} x_i p_i \tag{1}$$

or

$$max \sum_{i=1}^{n} x_i p_i \tag{2}$$

under the constraint :

$$\sum_{i=1}^{n} x_i p_i \leq W \tag{3}$$

where W is the total of available resources.

3.2. Correspondence between knapsack problem and that of resources allocation

The knapsack problem consists of determining among a set of objects, a selection with a maximum total value and not exceeding the total permissible weight in the knapsack. This principle is similar to the resource allocation ones, which consists in finding the resource price combination that maximizes the supplier's profits within the limits of available resources for a set of expressed demands. That is to say for each resource allocation problem, there is a knapsack formulation that matches.

Formally, for a set of n demands in resource allocation, we consider a set S of n objects with weight $p_i > 0$ and values $v_i > 0$. We have to find binary variables $x_1, x_2, ..., x_n \in \{0, 1\}$ such as : $\sum_{i=1}^{n} x_i.p_i \leq W$, and $\sum_{i=1}^{n} x_i.v_i$ is maximum. For a variable x_i, value 1 means the element will be put in the knapsack (ie the resource demand i will be supplied) and 0 means that it will not be selected.

Generally, some constraints are added to avoid singular cases :

– $\sum_{i=1}^{n} p_i > W$: we cannot take all the objects (the SP cannot supply all the needs at the same time) ; that is because in virtual networks, a spare resource must be always available in the substract network for the network recovery ;

– $p_i \leq W, \forall i \in \{1, 2, ..., n\}$: no object weight could exceed the knapsack capacity (each resource demand is less than the total capacity of the knapsack) ;

– $v_i > 0, \forall i \in \{1, 2, ..., n\}$: each object has a value and brings a gain (the profit collected by the supplier for the allocated resources) ;

– $p_i > 0, \forall i \in \{1, 2, ..., n\}$: any object has a weight (in ressource allocation, there is not null request).

So, to sort out an allocation resource problem, we can use some solutions of the knapsack problem like the dynamic programming solution.

3.3. Solving the resource allocation problem using a dynamic programming solution of the 0-1 knapsack's problem

The dynamic programming resolution method aims at obtaining the optimal solution to a problem by combining optimal solutions with similar, smaller and overlapping subproblems. Using it involves a recurrent formulation of the problem that will be used to find the optimal solutions. We proceed as follow :

Decomposition of the problem into sub-problems : Let be $M(k, w), 0 \leq k \leq n$ and $0 \leq w \leq W$ the maximum cost that can be obtained with objects $1, ..., k$ of S, and a maximum load knapsack W (we assume that the p_i and w are integers). If we can compute all the entries of this array, then the array entry $M(n, W)$ will contain the maximum cost of objects that can fit into the knapsack, that is, the solution to our problem. The cost could be the number of requests or the profit collected.

The recursive equation : Now, we recursively define the value of an optimal solution in terms of solutions to sub-problems. We have two cases :

– **we don't select the object** k : in this case, $M(k, w)$ is the maximum benefit by selecting among the $k - 1$ first objects with the limit w ($M(k - 1, w)$) ;

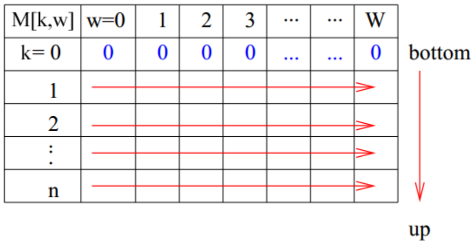
– **we select the object** k : $M(k, w)$ is the value of the object k plus the maximum benefit by selecting among the $k - 1$ first objects with the limit $w - p_k$.

The recursive equation is then :

$$M(k, w) = \begin{cases} 0 & \text{if } i = 0 \\ M(k-1, w) & \text{if } p_i > w \\ max\{M(k-1, w), v_k + M(k-1, w - p_k)\} & \text{else} \end{cases} \qquad (4)$$

This recursive equation result in the dynamic programming algorithm 1 with a space complexity $O(nW)$. We choose this algorithm to perform a bottom-up computation (see figure 4), looking for the optimal solution. This bottom-up computation means that the resource evaluation values will increase gradually during computations. The horizontal red arrows show that calculations are made from left to right ; the vertical red arrow shows that calculations are also done vertically taking into consideration dependency relationships.

Algorithm 1 provides the optimal solution on a set of objects for the knapsack problem, and also indicates which subset gives this optimal solution. From line 1 to 15, the



M[k,w]	w=0	1	2	3	···	···	W
k= 0	0	0	0	0	...	...	0
1							
2							
⋮							
n							

Figure 4 – Bottom-up computation principle.

algorithm computes the maximum requests to satisfy. From line 16 to 21, the algorithm selects the applicants to provide with resources.

```
Algorithm 1: knapsack
    Data: p,v,n,M
    Result: A maximum benefit on objects $p$
1   Let $M[0..n, 0..W]$ be a new table;
2   Let $x[1..n]$ be a new table;
3   begin
4       for $w = 1$ to $W$ do
5           $M[0, w]$=0;
6       for $k = 1$ to $n$ do
7           $M[k, 0]$=0;
8       for $k = 1$ to $n$ do
9           for $w = 1$ to $W$ do
10              if $p[k] > w$ then
11                  $M[k, w] = M[k-1, w]$;
12              else if $M[k-1, w] > v[k] + M[k-1, w-p[k]]$ then
13                  $M[k, w] = M[k-1, w]$;
14              else
15                  $M[k, w] = v[k] + M[k-1, w-p[k]]$;
16
17      w=W;
18      for $k = n$ to $1$ do
19          if $M[k, w] == M[k-1, w]$ then
20              $x[k] = 0$;
21          else $x[k] = 1$; $w = w - p[k]$;
22      return $x$;
```

Application to resource allocation : Let us consider a total available resources $W = 11$ in the network. This resource could be the bandwidth, the storage space or throughput. We also consider a set of k applicants with values v_k as the number of requests sent, and weight p_k as the resource quantity corresponding, as given in table 1. Let us assume that all the requests are about the same resource type and they arrive at the same time.

k	weight(p_k)	cost(v_i)
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28

Table 1 – Request sets to an InP for 5 simultaneous arrivals.

Looking for the optimal solution (the maximum requests satisfied by the InP which have resources) with the bottom-up computation, we obtain table 2. M is the different amounts of available resources. Each n-uplet $\{a_{i1}, a_{i2}, ..., a_{in}\}$ represents the fact that the element a_{in} have dependencies with the previous elements $a_{i1}, a_{i2}, ..., a_{in-1}$; this means that according to the recursive equation 4, the resource computation for a_{in} is linked to those of $a_{i1}, a_{i2}, ..., a_{in-1}$. For example, to obtain the cost for $M[4, 11]$ which is also written {1,2,3,4}, the computations made are :
$M[4, 11] = max\{M[4-1, 11], v_4+M[4-1, 11-p_4]\} = max\{M[3, 11], 22+M[3, 11-6]\} = max\{25, 22 + 18\} = max\{25, 40\} = 40$.

M	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	1	6	7	7	7	7	7	7	7	7	7
{1,2,3}	0	1	6	7	7	**18**	19	24	25	25	25	25
{1,2,3,4}	0	1	6	7	7	18	22	24	28	29	29	**40**
{1,2,3,4,5}	0	1	6	7	7	18	22	28	29	34	35	**40**

Table 2 – Bottom-up costs evaluation.

Table 2 shows that the maximum request numbers could be up to 40 UoC (Unit of Cost) with this example. Then, the optimal solution is {4,3} based on algoritm 1 and the applicants number 3 and 4 would be satisfied by the InP firstly ; the provided resources will be used during a time before they are allowed to other applicant. Within this period of time, other applicant requests are saved in a waiting mode. When the previously allocated resources are totally or partially released, other applicant requests could be satisfied. For each allocation game, the dynamic programming solution is used with various data at different times. This allocation process is presented in figure 5.

Depending on the objectives targeted by the InP (maximizing the number of requests fulfilled, maximizing the economic benefit derived from the allocation of resources), the previous example can be adapted. More detailed examples can be found in the appendix.

4. Interdependent requests and their constraints

Two interdependent requests refer to requests whose satisfaction of one directly or indirectly influences that of the other. As a result, we can distinguish several types of dependencies : strict bijections, partial bijections and non-bijections (injections).

Strict bijections : Given two requests $p_1 \in P$ and $p_2 \in P$. We say that there is a strict bijection between p_1 and p_2 if the request p_1 depends on p_2 and p_2 depends on p_1. In this

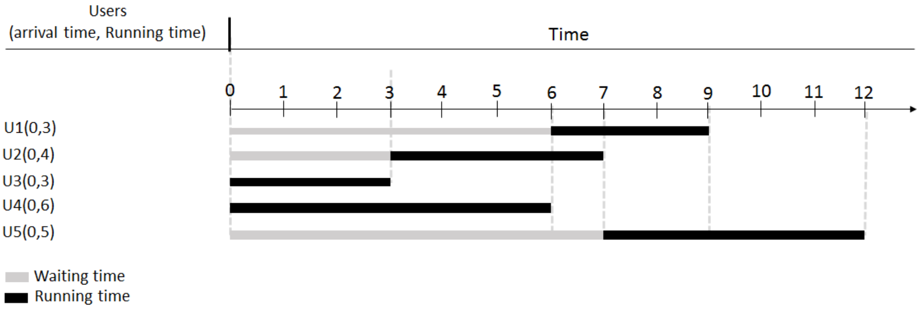


Figure 5 – Gannt chart for a set of five applicants for resources.

bijection case, if the resource required by p_1 is assigned to the user (SP), then it will be the same for the request p_2. This suggests that :

– any allocation is total and not partial;

– a dependence concerns resources of the same nature as those of different natures.

For example, you may need the bandwidth to store. In this case, if p_1 is the bandwidth request and p_2 that of the storage capacity, we will not be able to provide bandwidth and not satisfy the need for storage space as well as providing storage space and does not provide the bandwidth needed to store.

Notation 1 : The strict bijection between two requests p_1 and p_2 will be noted $p_1 \leftrightarrow p_2$.

Non-bijections (injections) : We talk about non-bijection between two requests p_1 and p_2 when p_1 depends on p_2, but the reverse is not true. As a result, if the resource requested by p_1 is assigned to it, then the one requested by p_2 must be allocated as well; on the other hand, the satisfaction of p_2 does not induce that of p_1.

Notation 2 : The non-bijection between two requests p_1 and p_2 will be noted $p_1 \rightarrow p_2$.

Partial bijections : Consider three requests $p_1, p_2, p_3 \in P$. We say that there is a partial bijection between these three requests if there exists at least one strict bijection between two of these requests or cyclic injections between these three requests. These different cases can result in one of the following situations :

1) $p_1 \leftrightarrow p_2 \rightarrow p_3$: p_1 depends on p_2, p_2 depends on p_1 and p_3;

2) $p_1 \rightarrow p_2 \leftrightarrow p_3$: p_1 depends on p_2, p_2 depends on p_3 and p_3 depends on p_2;

3) $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_1$: p_1 depends on p_2, p_2 depends on p_3 and p_3 depends on p_1. This is a *cyclic injection* which can be also qualified as a *group bijection*, that is to say a set of requests mutually dependent.

5. Solution Modeling

In this section, we present our solution for the dynamic resource allocation problem in the case of interdependent requests. To do this, let us consider the following assumptions :

– all requests sent contain dependent requests;

– requests are related to a maximum of two resources.

5.1. General principle of our approach

Our approach is initially to set up a request dependency graph from the various requests received by the FIP. Next, we use this dependency graph to extract all the connected components that will be used to build new requests. These new requests will be provide later to the resource allocation process described in [1]. These new requests highlight the different dependency relationships that exist between users' original requests, in order to better manage them during the resource allocation process. Figure 6 summarizes our approach to solve the problem of dynamic resource allocation for the interdependent requests.

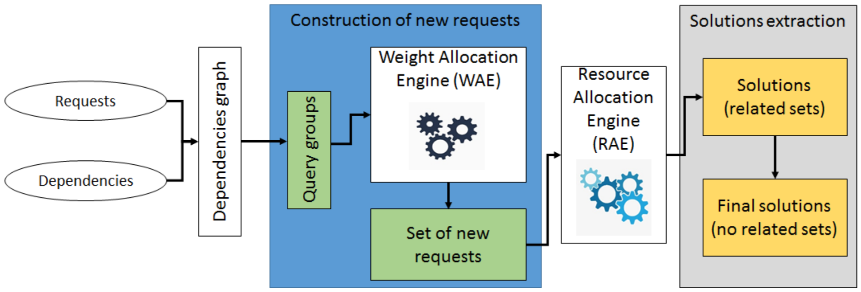

Requests
Dependencies
Dependencies graph
Construction of new requests
Query groups
Weight Allocation Engine (WAE)
Set of new requests
Resource Allocation Engine (RAE)
Solutions extraction
Solutions (related sets)
Final solutions (no related sets)

Figure 6 – Our resource allocation approach for interdependent requests.

5.2. Building the dependency graph

5.2.1. Building principle of the dependency graph

Let P be the set of the requests received from the supplier F. Let D be the set of dependencies between the requests $p_i, i \in \{1, 2, .., n\}$. Let R be the set of resources considered and $C_i, i \in \{1, 2, .., n\}$ the available quantity of the resource r_i. The dependency graph is created by building an arc between any pair of requests that have a dependency. Algorithm 2 describes in more detail the construction process of this graph.

```
Algorithm 2: Building the dependency graph
  Data: A set of requests $P$
  Result: A dependency graph $G$

  Let $b$ be a boolean;
  foreach request $p_i \in P$ do
      $b \leftarrow 0$;
      foreach $p_k \in P \setminus \{p_i\}$ do
          if $(p_i p_k \in D)$ then
              $b \leftarrow 1$;
              add $p_i p_k$ into $G$;
      if $(b = 0)$ then
          add $b$ into $G$;
```

5.2.2. Different graph possibilities

The algorithm 2 for building the dependency graph can give rise to several types of dependency graphs, the main ones being :

1) *A graph composed of several connected components* : in this case, the received requests do not make it possible to obtain a single global connected component grouping all the requests at the end of the construction process. To illustrate it, consider the 12 requests $p_i, i = \{1, 2, .., 12\}$ and the set of dependencies $D = \{p_1 \to p_2, p_3 \leftrightarrow p_7, p_4 \to p_5, p_2 \leftrightarrow p_6, p_8 \leftrightarrow p_9, p_{10}, p_{11}, p_{12}\}$. The exploitation of algorithm 2 provide the dependencies graph of the figure 7 whose construction process is illustrated in figure 8.

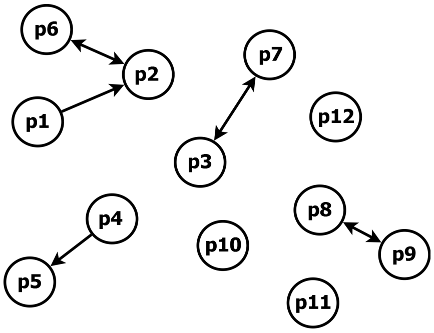


Figure 7 – Disconnected dependency graph.

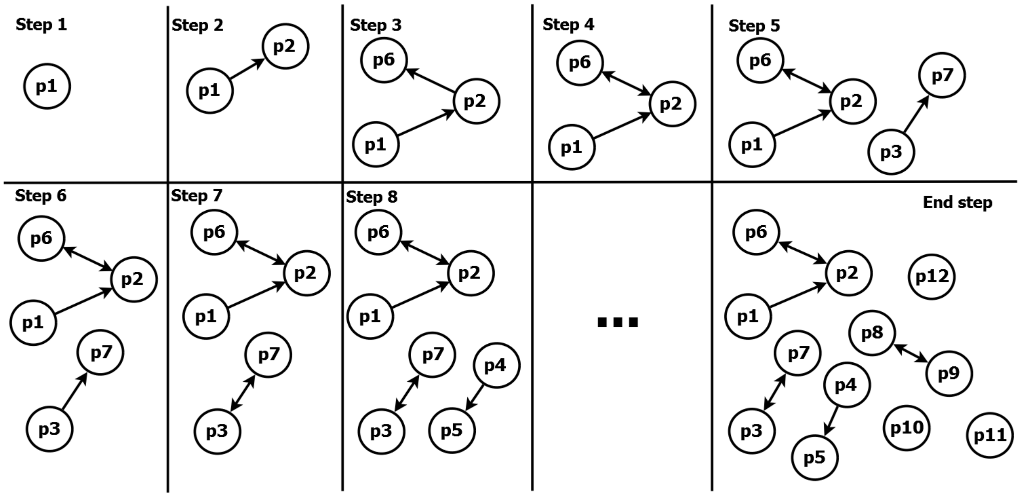
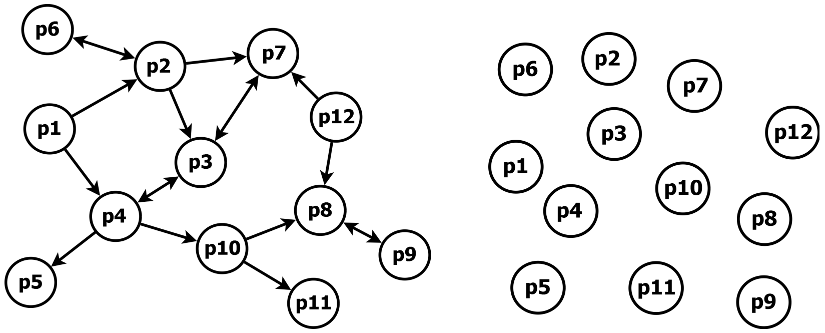


Figure 8 – Disconnected dependency graph construction process.

2) *A graph consisting of a single connected component grouping all requests* : Figure 9a gives an illustration of this case. This figure comes from the application of the algorithm 2 on the requests $p_i, i = 1, 2, .., 12$ with the set of dependencies $D = \{p_1 \to p_2, p_3 \leftrightarrow p_7, p_4 \to p_5, p_2 \leftrightarrow p_6, p_8 \leftrightarrow p_9, p_{10}, p_{11}, p_{12}, p_1 \to p_4, p_4 \to p_{10}, p_{10} \to p_{11}, p_{10} \to p_8, p_8 \leftarrow p_{12}, p_{12} \leftarrow p_7, p_2 \to p_7, p_2 \to p_3, p_3 \leftrightarrow p_4\}$. This type of graph can be obtained in the presence of interdependent requests from one and the same user, because the requests from different users are not interdependent.

3) *A graph of singletons* : this type of graph correspond to [1]. An example is presented in figure 9b.



(a) Connected dependency graph.

(b) A graph of singletons.

Figure 9 – Connected dependency graph and graph of singletons.

5.3. Generating new requests from the dependency graph

The new requests are obtained from the dependency graph built on the basis of the users' requests.

Given a dependency graph G, the construction of new requests consist to extract from the dependency graph all connected components as G_i groups (or tuplets) of dependency D_i, such as $\bigcup_{i=1}^{n} G_i = G$. A dependency group G_i is composed of dependencies D_i (which are either strictly bijective, injective or partially bijective) and original requests that have no dependency (such as requests p_{11} and p_{12} of the figure 7).

Case of graphs composed of several connected components : we consider the case of figure 7. In this case, there is no cyclic dependence (i.e. if a depends on b and b depends on c, then we do not have c depends on a). The generation of the new requests gives the groups $G_1 = \{p_1, p_2, p_6\}$, $G_2 = \{p_3, p_7\}$, $G_3 = \{p_4, p_5\}$, $G_4 = \{p_8, p_9\}$, $G_5 = \{p_{10}\}$, $G_6 = \{p_{11}\}$ and $G_7 = \{p_{12}\}$ of the figure 10. In a group that contains dependencies, constituent requests can be related to different resources. For example, in the group G_1, p_1 can be related to the computing power while p_2 is related to the storage capacity.

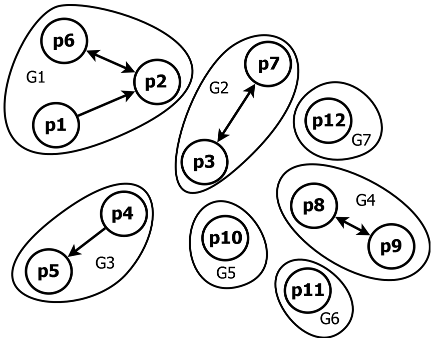


Figure 10 – New requests from the connected components of the graph.

Case of graphs composed of a single connected component grouping all original requests : this is the case of the figure 9a. The set of dependencies D produces here a single connected graph from which we can extract directly only one dependency group (or tuplet), in opposition to the case of the figure 7. This dependency group is the entire graph.

Case of the singletons graphs : each group G_i is composed of a singleton request.

5.4. Assigning weights to new requests

5.4.1. Our approach

Each request sent by the user has a certain weight w which is associated with a cost v and relates to a certain resource. In [1], this weight is used to determine the best solutions through the calculation of the maximum cost by the bottom-up evaluation method (see figure 4) applied to the equation 4.

Since new requests $G_i, i = 1, 2, ..., n$ are generated here using the user's original requests, new weights must be assigned to these requests as well as an appropriate cost. We assume that all the resources are comparable independently of their respective unit, that is to say that their quantity, in their respective units, can be translated to a fictional unit named UoC. In other words, 1 MB of storage is supposed be equivalent to 1 UoC and 1 kbit/sec of bandwidth is also supposed be equivalent to 1 UoC. Thus, in a G_i group of dependencies, we have the value of weight $w_i = \sum_{k=1}^{m} w_k$. It is the same for the cost $v_i = \sum_{k=1}^{m} v_k$.

However, this allocation can not remain at this level, since the type of dependence is not taken into account. For example, we should distinguish a bijection (case of the dependence between the p_6 and p_2 requests of G_1) from an injection (case of the dependency between the requests p_1 and p_2 of G_1). Indeed, the main difference between these two types of dependencies is the fact that, there is no possibility of mutual exclusion between the requests during the allocation in a bijection, which is the case with the injection. For example, in the group G_1, the set consisting of p_6 and p_2 can constitute a single request (since if we allocate the resource to p_6 we are obliged to do the same with p_2 and vice versa); this is not necessarily the case with the injection between p_1 and p_2. Either we provide the resources to p_1 and p_2, either we provide it to p_2 and we reject p_1. A choice must be made between these two possibilities, hence exclusion.

Since it is quite complex to express the constraints of mutual exclusion of variables in combinatorial analysis, we can distinguish here two approaches to assign weights to new requests :

1) *We consider an injection as a bijection* : in this case, the weight of the new request is $w_i = \sum_{k=1}^{m} w_k, k = 1, 2, ..., m$; this approach groups the situations of strict bijections (groups G_4 and G_2 of figure 10), simple injections (group G_3) and injections by transitivity (as p_1 which depends on p_6 by transitivity in group G_1). Thus, each connected component of the graph will induce only one request;

2) *We distinguish the bijection from the injection* : this situation requires taking into account the mutual exclusion of resources. For some injection cases, we propose the creation of new requests to isolate singletons of requests that do not have dependencies with others, following the model of the figure 11. Indeed, suppose the existence of a request p_n such that we have the injection $p_6 \rightarrow p_n$. The weight assignment engine creates a new request $G_1^{'}$ which is a singleton containing the request p_n independent of all other requests of G_1 with respect to the injection, as well as a request G_1 that includes all the requests of the connected graph. In general, the singletons are requests that do not have an outgoing link in an injection.

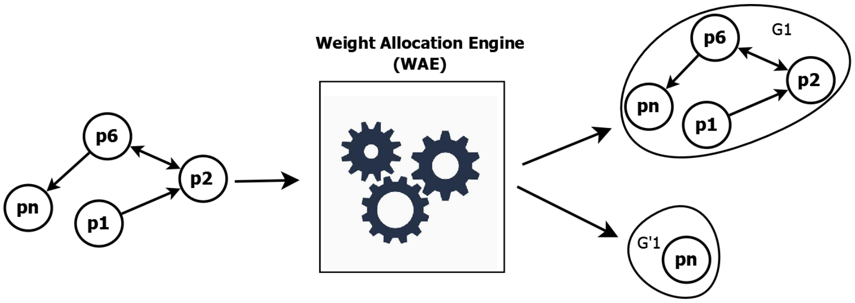


Figure 11 – Injection treatment.

5.4.2. The overflow capacity risk of a group weight or the global dependency graph weight

The methods of constructing the dependency graph and assigning weight to new requests can give rise to several situations that deserve special attention when allocating the resource :

1) *The total weight in a group exceeds the total weight of all available resources :* For example, suppose we have 10 MB of storage and 12 kbits/sec of bandwidth. A user requests $w_1 = 12$ MB and $w_2 = 13$ kbits/sec, provide a weight $w_k = 25, k = 1, 2$ which exceeds the total weight $W = 10 + 12 = 22$ of available resources. In such a case, the requests must not be processed, since they have individual weights that exceed those of the available resources.

2) *The total weight in a group does not exceed the total weight of all available resources, but the weight of one or more requests individually exceeds the amount of a resource type* : Considering 10 MB of storage and 12 kbits/sec of bandwidth as before, suppose a user requests $w_1 = 8$ MB and $w_2 = 13$ kbits/sec. We have a weight $w_k = 21, k = 1, 2$ that does not exceed the total weight of available resources $W = 10 + 12 = 22$. Since requests are dependent here, such groups must be excluded before starting the resource allocation process. The requests of these excluded groups are eventually considered in the next allocation round when the quantity of available resources (released by the users and added to the actual available resources) is enough to handle it.

Apart from the above cases, we can be sure because of the resources type heterogeneity within each group that, by summing the weights of the different requests of the group, we do not obtain a weight that exceeds the total sum of available resources ; if necessary, this overflow could be explained by the fact that a request weight already exceeds the total quantity available for the type of the requested resource.

5.5. Resource allocation for new requests

Once the weights and costs are assigned to each group of requests, we obtain a formulation of the resource allocation problem similar to that of [1]. The algorithm used in this work can be launched on this set of new requests in order to choose the best solutions to satisfy within the limits of available resources.

For the case of the first approach where we do not distinguish the bijection from the injection, only the groups obtained at the end of the process of building new requests (see figure 10 for example) are moved to allocation engine.

With regard to the second approach, if during the allocation process a group G_i and its singleton $G_i^{'}$ are part of the optimal solution, then only $G_i^{'}$ will be selected because of the optimal solution studied. The other will be returned to the list of candidate requests for the next resource allocation instance.

In both weight allocation approaches, the resource allocation engine produces solutions that are each composed of one or more connected components. The resource allocated to a group is distributed to all of the requests included in this group, as described in the figure 12.

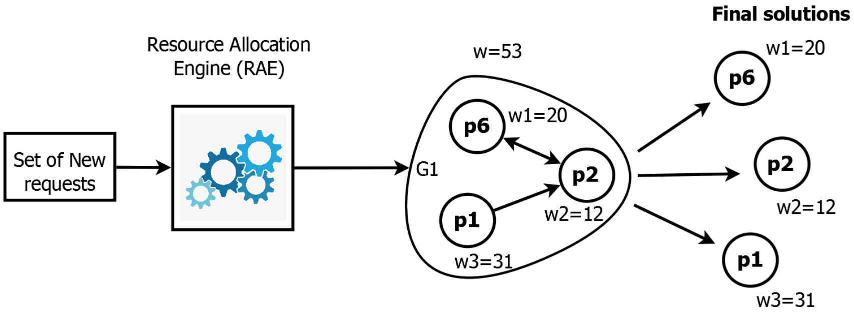


Figure 12 – Derivation process of the final solution.

Our resource allocation problem studied in this paper bears some resemblances to the knapsack problem with neighbor constraints described in [14] and [15]. As our approach, [14] mentions the different types of dependencies (injections, bijections, cycles, ...), suggests some formulations and proposes approaches to solve them. In addition, [14] and [15] use a dependencies graph to analyze the dependencies between the items, even if [15] limits the study on dependency graphs that are in-arborescences. Moreover, the allocation and reallocation process is not discussed in [14], while it is the case in our approach (we recall that we are studying the dynamic resource allocation problem, which involves multiple instances of a 0-1 knapsack problem). No algorithm about this reallocation process is discussed in [14]. The exclusions between the potential solutions in the particular case of the injections are not discussed in [14] while it is the case in our approach.

6. Simulations results

Our resource allocation approaches for the interdependent requests have been simulated in the discrete event network simulator OMNET++. This network simulator is well known for its high flexibility in the network topology customization even when the simulation is running, which meets the needs of our study. The simulations have been done in a computer with the following configuration : Core i5 2.40GHz, 4.00 GB RAM, and 12 MB cache. We ran our simulations on two networks : network1 (5 nodes and 7 links) and network2 (60 nodes and 90 links). The objectives of our simulations was to compare our both approaches of weight allocations to the new requests (the approach which do not distinguish a bijection from an injection, and the second one which distinguish bijection from injection), in order to select the best one. This comparison aimed to focus on the number of allocations done during the simulations and not the transmission delays of packets. During the simulation, the bandwidth and storage have been considered as the resources to allocate to the users. The requests were collected permanently from the controller and the allocations done during each period of time $t = 1$sec. In order to visualize the allocations made, we were interested in the sudden variations of packets routing delays related to bandwidth variations.

The analysis of the variation of packet routing delays following the allocation of the bandwidth in the network1 allows us to obtain the data of the figure 13. The peaks observed in this figure represents the drop in packet routing delay due to bandwidth allocation to certain nodes. The approach considered is the first which does not distinguish the bijection from the injection.

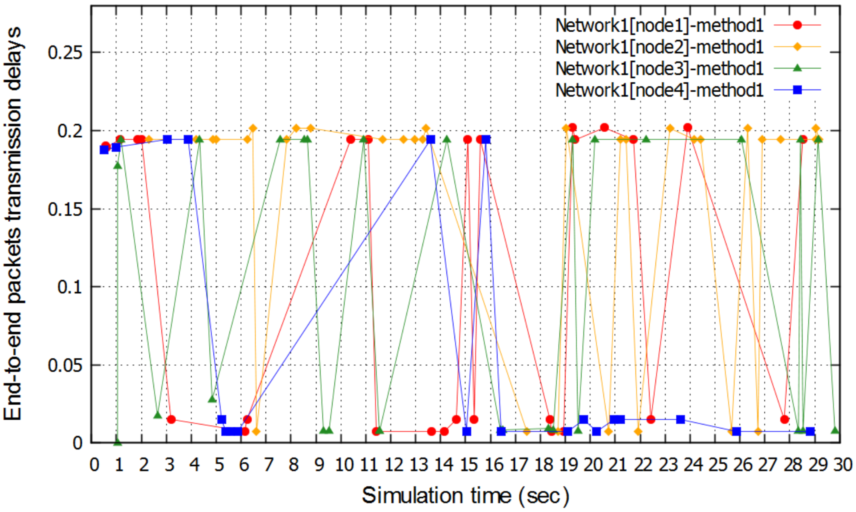


Figure 13 – End-to-end delays variations after resource allocation in network1 with method1.

For the different allocation peaks, the requests of the different nodes selected tend to form groups around some allocation zones (at times $t = 0$ to $t = 1$, $t = 5$ to $t = 7$, $t = 13.5$ to $t = 15$, ...) as part of a single request. This can be explained by the principle of method 1 which constructs groups of requests that do not take into account the particularity of each of them. On the other hand, groupings of satisfied requests are less frequent with method2 (see figure 14), which is much more likely to show isolated peaks. This certainly translates to the presence of singleton nodes in the allocation process. Moreover, there is a better distribution of resources in method2, which involves the majority of nodes in each allocation. Indeed, we observe that the allocations are more regular in method2 between the set of four nodes considered than in method1. This prove that the allocations trough the dependencies types, try to involve each time, the maximum number of users in the game. On the other hand, there are less latency times without allocation compared to method1. For example, with method1, the latency times of allocation found are between $t = 1$ and $t = 2$, $t = 7$ and $t = 9$, $t = 10$ and $t = 11$, while with method2, there is almost no latency. This means that method2 manage allocations better than method1.

Large-scale simulations (network2 of 60 nodes) provide us the result of the figure 15. It can be seen that method2 makes it possible to achieve on average more allocations than method1. This data was obtained by estimating the average variations of the packets transmission delays over all the 60 nodes.

The results obtained in the small and large networks, show that the method2 of assigning the weights to the requests in order to manage the injections, is better than that of

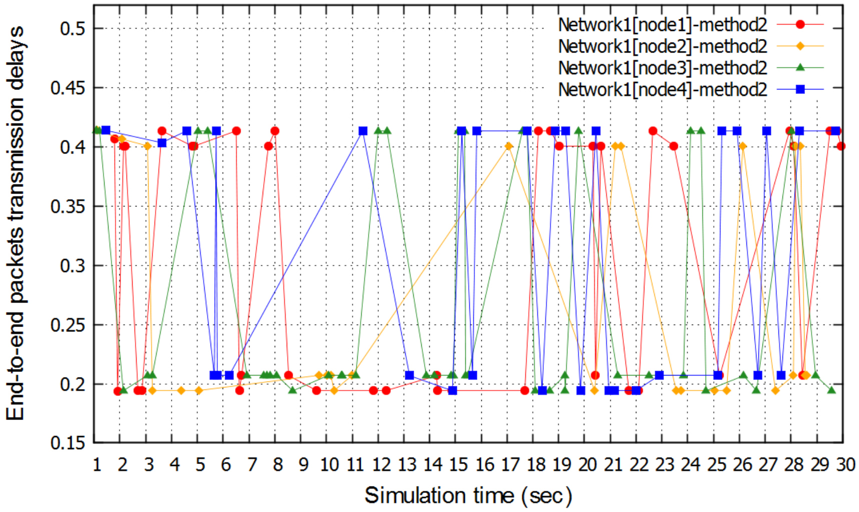


Figure 14 – End-to-end delays variations after resource allocation in network1 with method2.

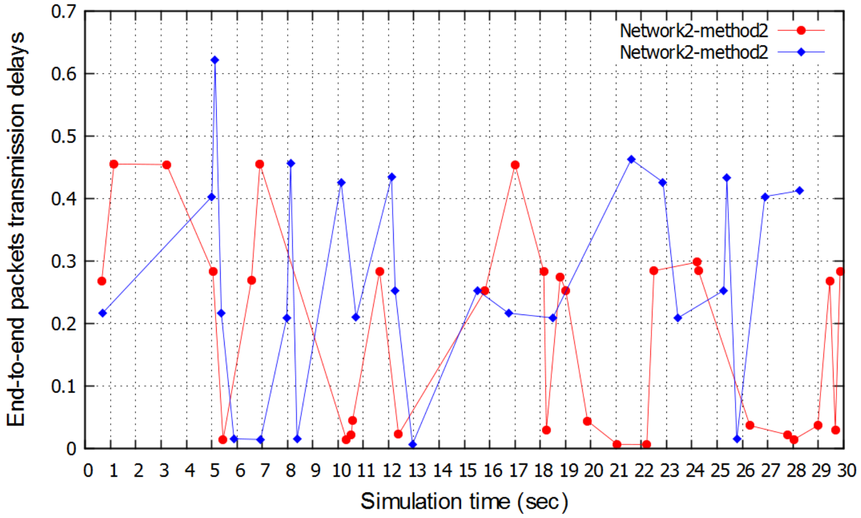


Figure 15 – End-to-end delays variations after resource allocation in network1 with method2.

method1 which does not distinguish the bijection from the injection. But, the gap between both method is not very significative.

7. Conclusion

In this paper, we have presented a knapsack-based dynamic resource allocation model that allows Infrastructure Providers (InP) in a network virtualization environment to select the most suitable users' requests meeting the aims of this InP. Our aim was to provide an efficient decision mechanism to face challenging difficulties encountered by the InP with the multiple requests of end-users or Service Providers. We propose a solution based on a knapsack dynamic programing solution to choose the most suitable users to satisfy both in case of interdependent and non-interdependent requests. We managed dynamic allocations as multiple resource allocation instances occurring at different times. To manage the dependencies between the requests, an approach has been proposed. This approach consisting of generating a dependency graph based on the users' requests, extracting from that dependency graph a set of new requests whose weights and costs depends on the initial dependencies. The simulations showed that our method improve significantly the quality of services in the virtual networks.

In an upcoming future, we intend to work on a decision mechanism taking into consideration important constraints as the fidelity of the user to an InP. It would not be suitable that a new customer, even providing a good profit to an InP, is chosen in replacement of an older and regular customer.

8. Bibliographie

[1] Vianney Kengne Tchendji, Kerol Roussin Donteu Djoumessi, Yannick Florian Yankam, « Dynamic resource allocations in virtual networks through a knapsack problem's dynamic programming solution », *Proceedings of CARI 2018*, Vol. 31, p. 120-131, Stellenbosch, South Africa, October 2018.

[2] Jain Raj, Paul Sudipta, « Network virtualization and software defined networking for cloud computing : a survey », *Mobile Networks and Applications*, Vol. 51, N° 11, p. 24-31, 2013.

[3] Niebert Norbert, El Khayat, Baucke Stephan, Keller Ralf, Rembarz René, Sachs Joachim, « Network virtualization : A viable path towards the future internet », *Wireless Personal Communications*, Vol. 45, N° 4, p. 511–520, 2008.

[4] N.M. Mosharaf Kabir Chowdhury, RRaouf Boutaba, « A survey of network virtualization », *Elsevier, IEEE*, Vol. 54, p. 862-876, 2010.

[5] Haider Aun, Potter Richard, Nakao Akihiro, « Challenges in resource allocation in network virtualization », *20th ITC Specialist Seminar*, Vol. 18, N° 2009, 2009.

[6] Mohamed Said Seddiki, « Allocation dynamique des ressources et gestion de la qualité de service dans la virtualisation des réseaux », *PhD thesis, Université de Lorraine*, 2015.

[7] Amraoui Asma, Benmammar Badr, Krief Francine, Bendimerad Fethi Tarik, « Négociations à base d'Enchères dans les Réseaux Radio Cognitive », *Nouvelles Technologies de la répartition-Ingénierie des protocoles NOTERE/CFIP 2012*, 2012.

[8] Zhu Yong, Ammar Mostafa H, « Algorithms for Assigning Substrate Network Resources to Virtual Network Components », *INFOCOM*, Vol. 1200, N° 2006, p. 1–12, 2006.

[9] Popek G. J., Goldberg R. P, « Formal requirements for virtualizable third generation architectures », *Communications of the ACM*, Vol. 17, July, 1974.

[10] Fischer Andreas, Botero Juan Felipe, Beck Michael Till, De Meer Hermann, , Hesselbach Xavier, « Virtual network embedding : A survey », *IEEE Communications Surveys & Tutorials*, Vol. 15, N° 4, p. 1888–1906, 2013.

[11] Kreutz Diego, Ramos Fernando MV, Verissimo Paulo Esteves, Rothenberg Christian Esteve, Azodolmolky Siamak, Uhlig Steve, « Software-defined networking : A comprehensive survey », *Proceedings of the IEEE*, Vol. 103, N° 1, p. 14–76, 2015.

[12] Kim Hyojoon , Feamster Nick, « Improving network management with software defined networking », *IEEE Communications Magazine*, Vol. 51, N° 2, p. 114–119, 2013.

[13] Morimoto Naoyuki, « Power allocation optimization as the multiple knapsack problem with assignment restrictions », *2017 8th International Conference on the Network of the Future (NOF)*, *IEEE*, p. 40–45, 2017.

[14] Glencora Borradaile, Brent Heeringa , Gordon Wilfong, « The knapsack problem with neighbour constraints », *Journal of Discrete Algorithms*, *Elsevier*, Vol. 16, p. 224–235, 2012.

[15] David S. Johnson , K.A. Niemi, « On knapsacks, partitions, and a new dynamic programming technique for trees », *Mathematics of Operations Research*, *INFORMS*, Vol. 8, N° 1, p. 1–14, 1983.

A. A practical example of resource allocation with succeeding request arrivals of 8 applicants to the InP

In this example, we suppose that the applicant requests reach the InP at different times. So, those requests are satisfied successively. When new requests occur from another applicant, preceding allocated resources can be divided to provide the other ones.

let us assume a total available resources $W = 10$ in the InP network. We also consider a set of k applicants with values v_k as in the previous example, as given in table 3. Let us assume that all the requests are concerned with the same resource type and they arrive successively according to time.

k	weight(p_k)	cost(v_i)	Arrival time
1	5	10	0
2	4	40	
3	6	30	
4	5	50	6
5	4	60	
6	3	80	13
7	5	20	16
8	7	30	

Table 3 – Request sets to an InP for 8 applicants.

We suppose that requests from the applicants number 1, 2 and 3 come first. The computation of the maximum satisfied requests will be 70 UoC (see table 4). This means that the optimal solution is {3,4}. In case of competition, applicants 3 and 4 would be selected before the others.

M	0	1	2	3	4	5	6	7	8	9	10
$\emptyset$	0	0	0	0	0	0	0	0	0	0	0
$\{1\}$	0	0	0	0	0	10	10	10	10	10	10
$\{2\}$	0	0	0	0	**40**	40	40	40	40	50	50
$\{3\}$	0	0	0	0	40	40	40	40	40	50	**70**

Table 4 – Bottom-up costs evaluation with applicants coming at the time 0.

When other applicant requests will reach the InP, another computations will be made to choose the most suitable user to provide with resources. Table 5 illustrates the computations done for the requests coming at the time 6, and result in a maximum of 110 requests that could be satisfied by the InP. The applicant numbers 1 and 2 correspond respectively to numbers 3 and 4 in table 3.

M	0	1	2	3	4	5	6	7	8
$\emptyset$	0	0	0	0	0	0	0	0	0
$\{1\}$	0	0	0	50	50	50	50	50	50
$\{2\}$	0	0	60	60	60	110	110	110	**110**

Table 5 – Bottom-up costs evaluation with applicants coming at the time 6.

with regards to what is stated above, the results of table 6 are obtained for applicants number 7 and 8 coming at the time 16.

M	0	1	2	3	4	5	6	7	8	9
$\emptyset$	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	20	20	20	20	20
2	0	0	0	0	0	0	0	30	30	**30**

Table 6 – Bottom-up costs evaluation with the applicants coming at the time 16.

Gant chart of the figure 16 presents the resource allocation order of all different applicants, mapping with their requests. It considers that the running time of each applicant is proportional to its weight p_k.

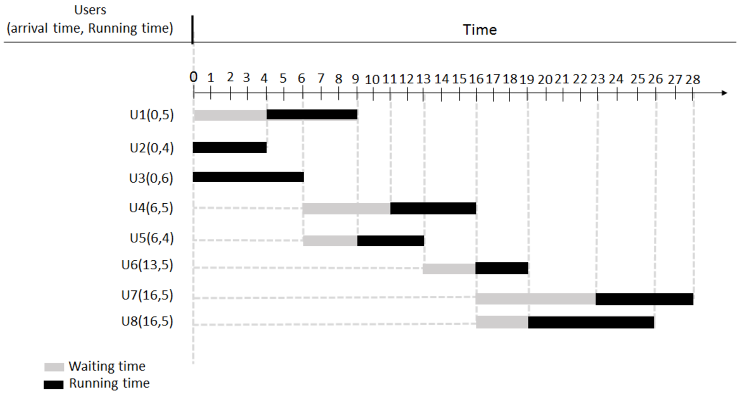


Figure 16 – Gannt chart for 8 sequential arrivals.

B. An enhanced example of resource allocation with 24 applicants and 150 UoC of resources to the InP

In this example, we enhance the resource allocation scenario presented in appendix A. let us assume a total available resources $W = 150$ in the InP network. We also consider a set of $k = 24$ applicants with values v_k as given in table 7. The column A.t. (t) is the arrival time represented as t. Let us assume that all the requests are concerned with the same resource type and they arrive successively according to time t. Such configuration provide a maximum of 420 satisfied requests with the following provision scheme for the users at $t = 0$: users' requests 2, 3 and 5 will be satisfied firstly, then users 4 and 1. In the same way, at time $t = 20$, users' requests 11 and 13 will be satisfied before 12, resulting a maximum requests number of 808. At $t = 30$, the maximum satisfied requests is 543 and the resource allocation process will consider the users 18 and 19 before user 20. These maximum satisfied requests are computed using the algorithm 1. In each period of the allocation process, this maximum request number can be increased with the running requests of the preceding period. The Gannt chart is provided by the figure 17.

k	weight(p_k)	cost(v_i)	A.t. (t)	k	p_k	v_i	A.t. (t)	k	p_k	v_i	A.t. (t)
1	103	200	0	9	62	120	18	17	90	210	27
2	30	101		10	45	138		18	16	187	30
3	54	174		11	35	350	20	19	107	356	
4	101	250		12	92	670		20	88	231	
5	46	145		13	110	750		21	42	199	33
6	22	80	13	14	63	680	21	22	61	225	
7	6	20	16	15	102	110	23	23	115	165	38
8	14	30		16	87	220	25	24	84	194	

Table 7 – Request sets to an InP for 24 applicants.

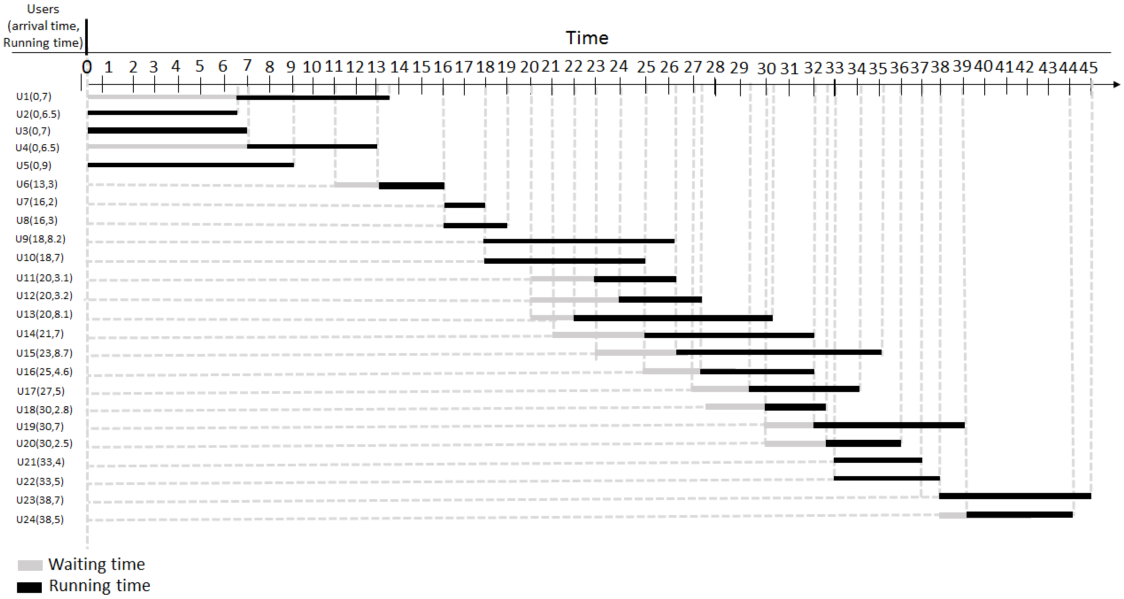


Figure 17 – Gannt chart for 24 sequential arrivals.