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

Vianney Kengne Tchendji*, Kerol Roussin Donteu Djoumessi*, Yannick Florian Yankam*

*Department of Mathematics and Computer Science
Faculty of Science
University of Dschang
PO Box 67, Dschang-Cameroon
vianneykengne@yahoo.fr, djoumessikerol@gmail.com, 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. 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. From this perspective, we show that the problem of resource allocation to users based on their queries 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 queries.

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 [2, 3], 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)[1] allowed to better face this resources allocation challenge (known as virtual network embedding problem[9]) 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[4] 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[4]. Thus, techniques such as auctions [6] or game theory can be used to allocate these resources, although they do not always make it possible to decide in all cases.

Our contribution in this paper is the proposition of a 0-1 knapsack-based model for resources allocation in a virtual network environment integrating the SDN architecture. We exploit the dynamic programming solutions of the 0-1 knapsack problem to build an efficient allocation solution within the limits of the available substrate network resources. The goal is to find the best solution that satisfies the majority sides in competition. We also model a dynamic resource allocation as a multiple resource allocation instances with various requests at different times.

The rest of this paper is organized as follow : In section 2, we present network virtualization and SDN paradigms. Section 3 presents a formulation of the resource allocation problem, showing the equivalence with the 0-1 knapsack one, followed by the problem resolution through a dynamic programming solution related to the knapsack problem. A conclusion ends this paper.

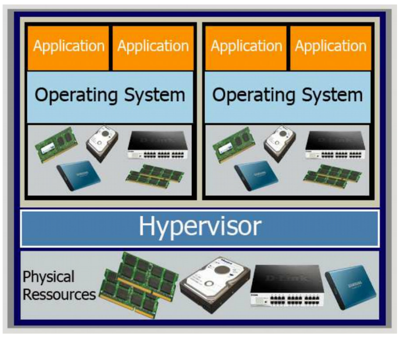
2. Network virtualization and SDN paradigms

Our work environment is made up of several virtual networks under the supervision of a network controller.

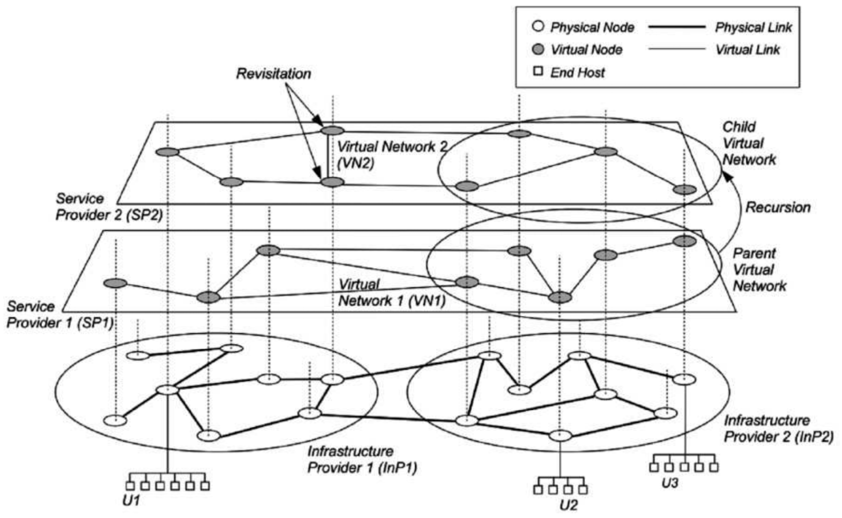
2.1. Virtual networks

A virtual network is a set of virtual devices interconnected by virtual links through a physical infrastructure [3]. In each virtual network, we find components created from a physical component by a special software called hypervisor : these are virtual machines [8] (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 Infrastructure Provider hires resources from service providers (SP) which creates 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.



(a) Virtual machines.



(b) A network virtualization environment.

Figure 1 – Virtualization principles.

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 [10]. 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. This decoupling allows to deploy a monitoring plane on standard servers with flexible computing capabilities [11], 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.

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 queries from a given set, taking into consideration several constraints involved[5]. 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 [5, 7]. However, these different formulations focus on the allocation of virtual links and bandwidth [7] in a restricted virtual network, while it shall be more general. Another work [12] proposes in the context of the Internet of Things, a power allocation knapsack-based model which approaches the optimal solution, whereas 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 (IP) 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 $R = r_1, r_2, ...$ 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 : $\cup_{i \in m} R_i = R$ and $\cap_{i \in m} R_i = \phi$

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 substract 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 2), 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.

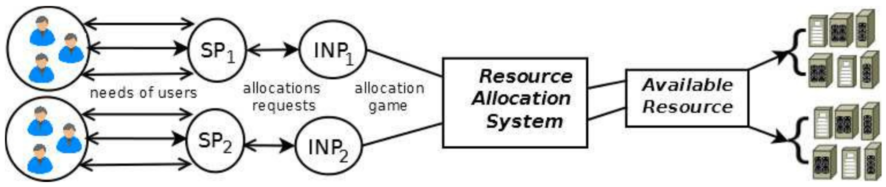


Figure 2 – Ressource allocation process.

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.

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

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

– $p_i > 0, \forall i \in 1, ..., n$: 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 sub-problems. 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 computing time of files that can fit into the storage, 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 :

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

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 3), 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.

Application to resource allocation :

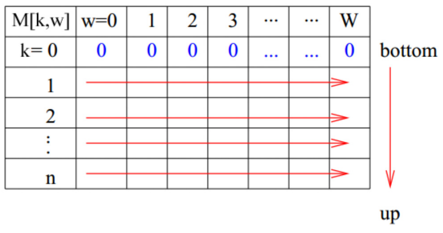


Figure 3 – Bottom-Up Computation principle.

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 $a_{i1}, a_{i2}, ...$ and 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 4.

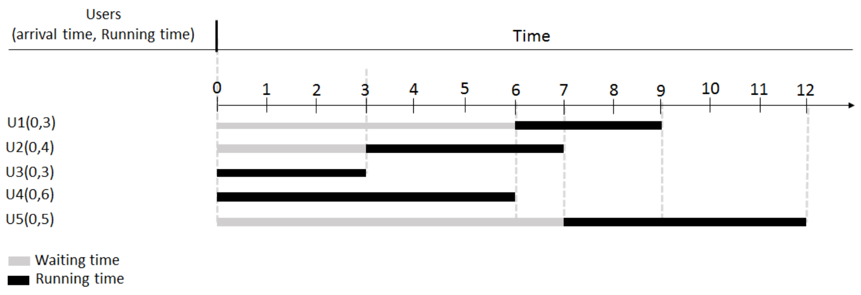


Figure 4 – Gannt chart for a set of five applicants for resources.

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.

4. 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. We managed dynamic allocations as multiple simple resource allocation instances occuring at different times.

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.

5. Bibliographie

[1] 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.

[2] 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.

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

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

[5] 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.

[6] Amraoui, Asma and Benmammar, Badr and Krief, Francine and 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.

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

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

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

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

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

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

A. Knapsack dynamic programing algorithm

Algorithm 1 provide 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, we compute the maximum requests to satisfy. From line 16 to 21, the algorithm select the applicants to provide with resources.

B. 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.

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.

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.

Algorithm 1: knapsack

```
Data: p,v,n,M
Result: A maximum benefit on objects $p$
Let $M[0..n, 0..W]$ be a new table ;
Let $x[1..n]$ be a new table ;
begin
    for $w = 1$ to $W$ do
        $M[0, w]$=0 ;
    for $k = 1$ to $n$ do
        $M[k, 0]$=0 ;
    for $k = 1$ to $n$ do
        for $w = 1$ to $W$ do
            if $p[k] > w$ then
                $M[k, w] = M[k-1, w]$ ;
            else if $M[k-1, w] > v[k] + M[k-1, w-p[k]]$ then
                $M[k, w] = M[k-1, w]$ ;
            else
                $M[k, w] = v[k] + M[k-1, w-p[k]]$ ;

    w=W ;
    for $k = n$ to $1$ do
        if $M[k, w] == M[k-1, w]$ then
            $x[k] = 0$ ;
        else $x[k] = 1$ ; $w = w - p[k]$ ;
    return $x$ ;
```

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.

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.

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 5 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.

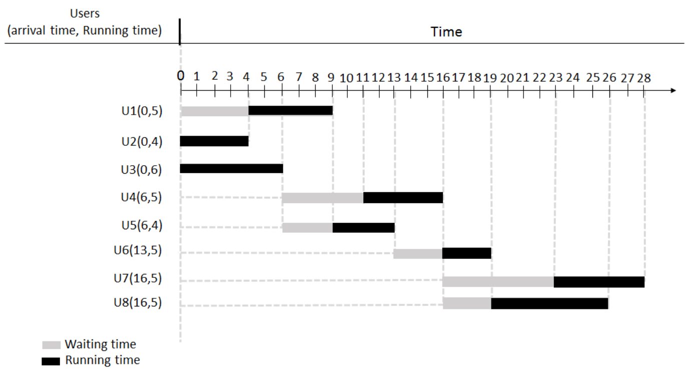


Figure 5 – Gannt chart for 8 sequential arrivals.

C. 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 B.

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 6.

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.

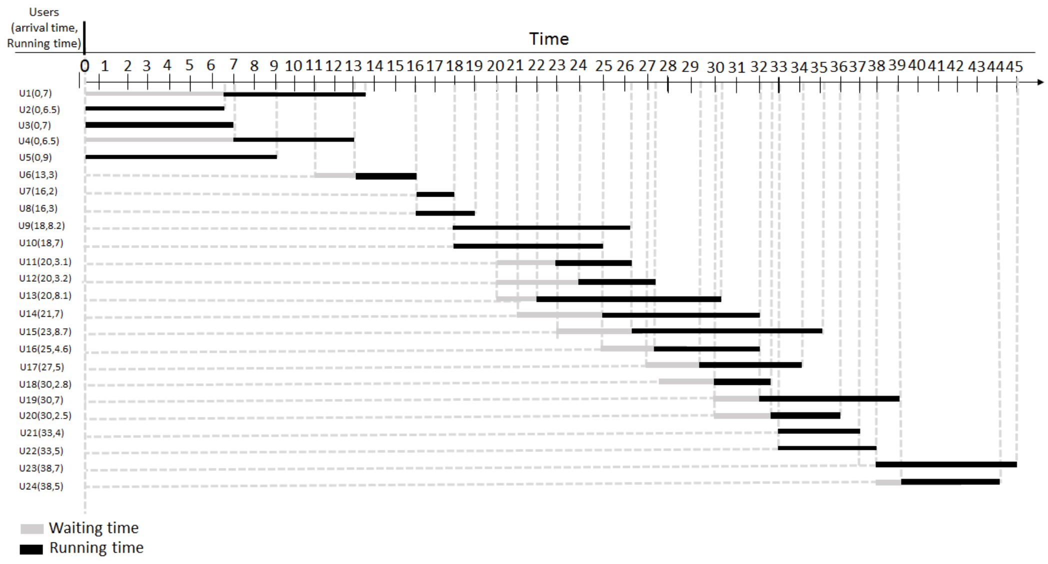


Figure 6 – Gannt chart for 24 sequential arrivals.