Design and Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions

Rashid Mijumbi*, Joan Serrat*, Juan-Luis Gorricho*, Niels Bouten‡, Filip De Turck‡ and Steven Davy†
*Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
‡Ghent University – iMinds, B-9050 Gent, Belgium
†Telecommunications Software and Systems Group, Waterford Institute of Technology, Ireland

***Abstract*—Network function virtualization has received attention from both academia and industry as an important shift in the deployment of telecommunication networks and services. It is being proposed as a path towards cost efficiency, reduced time-to-markets, and enhanced innovativeness in telecommunication service provisioning. However, efficiently running virtualized services is not trivial as, among other initialization steps, it requires first mapping virtual networks onto physical networks, and thereafter mapping and scheduling virtual functions onto the virtual networks. This paper formulates the online virtual function mapping and scheduling problem and proposes a set of algorithms for solving it. Our main objective is to propose simple algorithms that may be used as a basis for future work in this area. To this end, we propose three greedy algorithms and a tabu search-based heuristic. We carry out evaluations of these algorithms considering parameters such as successful service mappings, total service processing times, revenue, cost etc, under varying network conditions. Simulations show that the tabu search-based algorithm performs only slightly better than the best greedy algorithm.**

Keywords*—Network function virtualization, mapping, scheduling, placement, chaining, tabu search, resource allocation.

I. Introduction

Telecommunication services are currently based on network operators having physical proprietary devices and equipment for each service. As the requirements of users for more diverse and new (short-lived) services increase, it means that operators must correspondingly and continuously purchase, store and operate new physical equipment, which does not only require high and rapidly changing skills for technicians operating and managing these equipment, but also leads to increased capital expenditures for operators for purchasing, storing and maintaining such equipment. Therefore, telecommunications service providers must find ways of building dynamic virtualized networks with application and content awareness so they can deliver new and innovative services to subscribers, who are changing how they use connectivity services [1].

Network Function Virtualization (NFV) [2] has been proposed as a way to address these problems by leveraging virtualization technology to consolidate many network equipment types onto high volume servers, switches and storage, which could be located in datacentres, network nodes and in end user premises. It allows for the decoupling of physical network equipment from the services or functions that run on them, such that a given service can be decomposed into a set of virtual network functions (VNFs), which could then be implemented in software that can run on one or more industry standard physical nodes. The VNFs may then be relocated and instantiated in different network locations (for example aimed at introduction of a service targeting customers in a given geographical location) without necessarily requiring purchase and installation of new hardware.

In Figures 1 and 2, we show an example of the changes that may be achieved by NFV. Fig. 1 shows a typical (current) implementation of a customer premises equipment (CPE) which is made up of the functions: Dynamic Host Configuration Protocol (DHCP), Network Address Translation (NAT), routing, Universal Plug and Play (UPnP), Firewall, Modem, radio and switching. In this example, a single service (the CPE) is made up of eight functions. These functions may have precedence requirements, for example, it may be required to perform firewall functions before NAT. Currently, it is necessary to have these functions in a physical device located at the premises of each of the customers 1 and 2. The CPEs are then aggregated by the Broadband Remote Access Server (BRAS) that connects to the ISP. This way, if there is a need to make changes to the CPE, say, by adding, removing or updating a function, it may be necessary for a technician from the ISP to individually talk to or go to each of the customers. It may even require a complete change in the device in case of additions. This is not only expensive (operationally) for the ISPs, but also for the customers.

In Figure 2, we show a possible implementation based on NFV in which some of the functions of the CPE are transferred to a shared infrastructure at the ISP, which could also be a datacentre. This makes the changes described above easier since, for example, updating the DHCP for all customers would only involve changes at the ISP or in a datacentre. In the same way, adding another function such as parental controls for all or a subset of customers can be done at once. In addition to saving on operational costs for the ISP, this potentially leads to cheaper CPEs if considered on a large scale.

One of the objectives of NFV is to achieve fast, scalable, on-demand and dynamic composition of network functions to a service. However, since a network service requires a number of VNFs, achieving a NFV environment raises two questions; (1) how to define and implement network services, and (2) how to efficiently map and schedule the VNFs of a given service onto a physical network. The European Telecommunications Standards Institute (ETSI) through its NFV technologies group is partnering with network operators and equipment vendors to promote the NFV approach and are currently progressing with regard to the first question above. Specifically, they have already defined the NFV problem [3], some use cases [4] and a reference framework and architecture [5].

978-1-4799-7899-1/15/$31.00 ©2015 IEEE

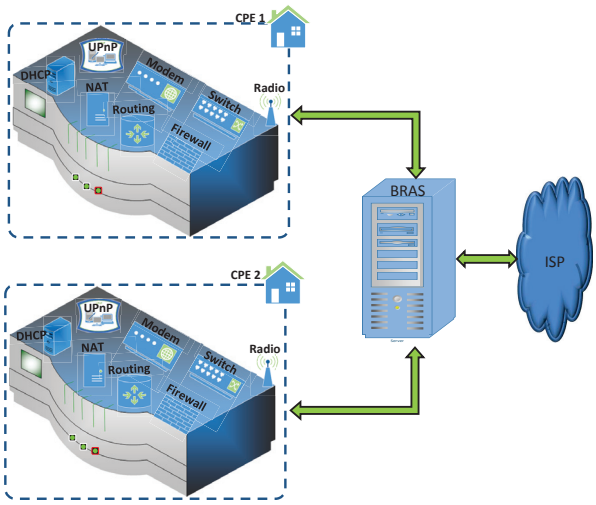

Fig. 1. Current CPE Implementations

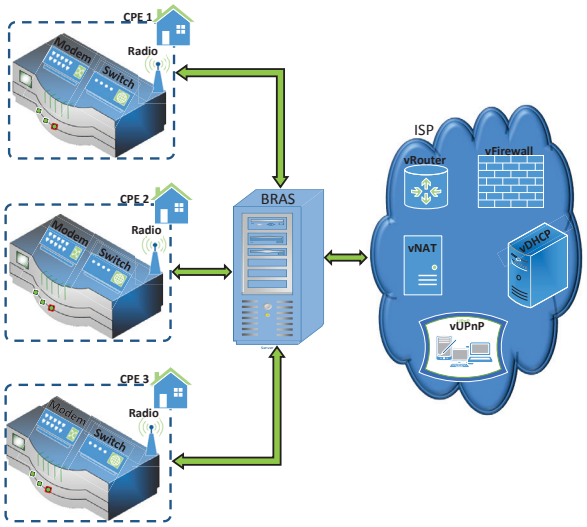

Fig. 2. Possible Implementation with NFV

However, the second question i.e. mapping and scheduling has received little attention. As networks become bigger and more dynamic, and user service requirements change more often, it will not be possible for network operators to manually map and schedule particular VNFs of a given service onto specific physical machines[1]. As noted in [2], automation is paramount to the success of NFV. This calls for, among other things, algorithms that are able to perform the mapping of VNFs onto the possible physical nodes. These algorithms should be able to accept an online and dynamic nature of network services, and must ensure that physical hardware resources are used efficiently. The success of NFV will depend, in part, on the existence and performance of algorithms that determine where, and how the VNFs are instantiated [6].

In this paper, we start by formulating the problem of online mapping and scheduling of VNFs, and then propose algorithms for its solution. In particular, we propose three algorithms that perform the mapping and scheduling of VNFs based on a greedy criterion such as available buffer capacity for the node or the processing time of a given VNF on the possible nodes. The algorithms perform both mapping and scheduling at the same time (one-shot), i.e. at the mapping of each VNF, it is also scheduled for processing. In addition, we propose a local search algorithm based on tabu search (TS) [7]. The TS algorithm starts by creating an initial solution randomly, which is iteratively improved by searching for better solutions in its neighborhood.

To the best of our knowledge, this is the first attempt to formulate the problem of online mapping and scheduling of VNFs, and to propose and evaluate algorithms for the same purpose. Given the important role that NFV might play in the future of the Telecommunications industry we hope that our proposals will be important as bench marks for other researchers or for implementations in this regard.

The rest of this paper is organized as follows: Section II presents a formal description of the mapping and scheduling problem. In Sections III and IV, we describe the proposed greedy and tabu search algorithms respectively. The algorithms are evaluated in Section V, the related work discussed in Section VI, and the paper concluded in Section VII.

[1]This paper uses the terms node and machine synonymously.

II. Problem Description

The problem of allocating physical resources in NFV is illustrated in Fig. 3, and can be split into two parts: (1) embedding/mapping virtual machines (VMs) onto physical machines which is known as VDCE [8], and is related to VNE [9]. Both these problems (VNE and VDCE) are well studied and are therefore out of the scope of this paper. The rest of this paper considers that we have virtual nodes already mapped onto physical nodes. (2) mapping and scheduling of VNFs onto the created virtual nodes. We refer to this problem as network function mapping and scheduling (NFMS), and is the focus of this paper. For the NFMS problem, there are many possibilities for resource sharing. One of them is that for each VNF, a dedicated VM is used. However, considering a service made up of multiple functions e.g. 8 functions for the CPE in Fig. 1 means that each customer (or CPE) would require 8 dedicated VMs. This would clearly not be feasible as physical resources would easily be depleted, and would be wasteful of resources since most functions are "light" and can therefore be processed by a single VM, say, by containers within the VM. What we consider in this paper is the resource sharing approach that allows for a given VM to process multiple VNFs, one after another (possibly) from a queue.

Therefore, NFMS consists of a need to process network services online (each service is created and embedded as its need arises) using a set $N = \{1, ..., n\}$ of n virtual network nodes. N represents all the virtual nodes created/mapped on all the physical nodes. Any given network service S is made up of a sequence $F = \{1, ..., m\}$ of m VNFs, where the function $1 \leq i \leq m$ must be processed on a set $N(i) \subseteq N$ of nodes. The functions $\{1, ..., m\}$ must be processed one after the other in the specified sequence, and each virtual node can process at most one function at a time. The processing time for function i on node $j \in N(i)$ is $\rho_{ij} > 0$, where $1 \leq j \leq n$, and while being processed or in the queue for processing, a function i utilizes a buffer δ_i from the node onto which it is mapped. The different function processing times on each node

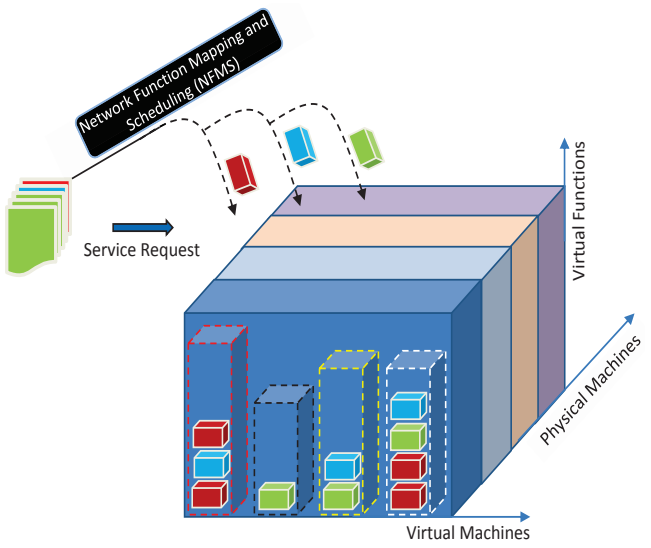

Fig. 3. Network Function Mapping and Scheduling Problem

may also capture function setup times, a given function may require different setup times on different nodes. At any point, a given node j has available buffer size B_j. For each node-function combination, we define a binary variable $\beta_{i,j}$ which takes a value of 1 if node j is able to process function i and 0 otherwise.

The problem then involves choosing for each VNF i a virtual node $j \in N(i)$ and a completion time t_i when its processing will be completed[2]. We also define a deadline t_l for processing a given service. The processing of the last function in the service must be completed by this time, or the service request is rejected. The deadline can be used to define processing priority for services, e.g. those service that require real time processing may have a deadline that only takes into consideration their processing and precedence requirements on the different virtual nodes, and zero waiting.

Finally, for each virtual node j, we define the expected completion time π_j of the last function queued for processing on the node, and for each service, we define an arrival time t_a, which is the time when the request for mapping and scheduling the service is received by the physical network.

The NFMS problem can be considered to consist of two parts: deciding onto which virtual nodes each VNF should be mapped (the mapping problem), and for each node, deciding the order in which the mapped VNFs should be processed (the scheduling problem.). It is in general possible to solve these two problem in two separate steps, such that VNFs are first mapped to nodes and then scheduled.

A. Mapping and Scheduling Example

1) Function Mapping: As an example, in Figs. 4 - 7, we show two services S_1 and S_2, being mapped and scheduled on a virtual network. In Fig. 4, the network is represented showing the different VNF processing capabilities of each node. Consider that at a time T_1 a service S_1 arrives with the request for mapping with functions $\{f_8 \rightarrow f_2 \rightarrow f_3 \rightarrow f_6 \rightarrow f_5\}$. At this point, it is possible that S_1 would be mapped as shown in Fig. 5. It can be observed that in such a mapping the node n_1 is hosting the functions f_8 and f_3. Similarly, assuming that after some time another service request S_2 with functions $\{f_6 \rightarrow f_8 \rightarrow f_4\}$ arrives, the mapping shown in Fig. 6 may be achieved. At this point, two services are simultaneously being processed by the virtual nodes. It can be noted that while the function f_8 could be processed at node n_1 as well, it may be better to use node n_7 for S_2 to avoid a long queue at n_1.

2) Function Scheduling: Finally, using hypothetical processing times, in Fig. 7, we represent the possible scheduling of the function processing at each node. The processing of S_1 begins immediately on its arrival at T_1 at node n_1. There after, each of the successive functions has to wait until its preceding function has been processed before its processing can commence. The processing of S_1 ends when its last function has been processed at T_8. Therefore, the total processing time (flow time) of S_1 would be given by $\left(T_8 - T_1\right)$, and is equivalent to the summation of the processing times of the functions at the various nodes.

However, for illustration purposes, consider that S_2 arrives at T_4. As can be observed, since its first service is mapped onto node n_5 which already has a scheduled function from the first service, the commencement of the processing of S_2 has to wait until the completion of function f_6 from S_1 at time T_6. This delay $(T_6 - T_4)$ in commencing the processing of S_2 increases its total processing time, and may ultimately lead to inefficient resource utilization since S_2 will occupy the virtual nodes for a longer time than it would have without this kind of delay. We should note that this is a direct consequence of the mapping step, since for example, if we had mapped the function f_6 of S_2 onto n_3 (which also has the capacity to process it), then we would have avoided this extra delay on scheduling. Therefore for efficient resource utilization, the mapping and scheduling steps should have some form of coordination.

B. Function Timing Restrictions

As illustrated in Section II-A2, while the processing of a service transits from one function to the next, there may be some delay. There are three mutually exclusive possibilities: (1) immediately after processing a function, the proceeding function is started (for example, in Fig. 7 for S_1, the processing of function f_2 commences immediately after that of f_8), (2) a function (either initial or proceeding) has to wait for the start of its schedule (In Fig. 7, for S_2, assuming that f_8 had been mapped onto n_6 instead of n_7, then after the processing of f_6 at time T_7, f_8 would have had to wait until the node is available.), and (3) the node to which a proceeding function is scheduled has to starve while waiting for the preceding function to be completed (for example, in Fig. 7, node n_2 starves for the time $T_9 - T_4$ waiting to start processing f_4. An efficient NFMS algorithm needs to be able to use such timing gaps and delays to ensure both efficient resource utilization and short service processing times.

C. Mapping and Scheduling Objectives

In NFMS, there may be many objectives such as minimizing flow time, cost and revenue.

[2]The time t_s at which the processing of the function i on node j starts can then be derived from $t_s = t_i - \rho_{ij}$.

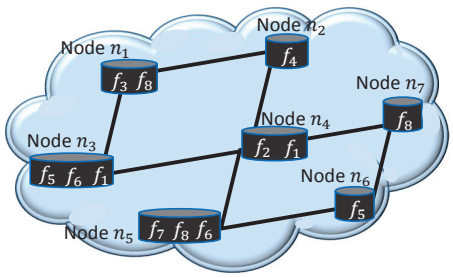

Fig. 4. Network with node capabilities

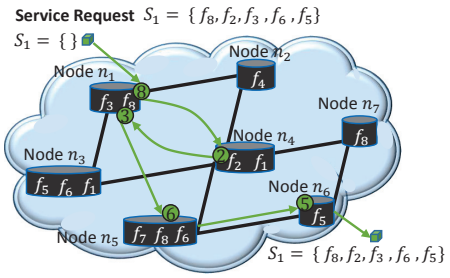

Fig. 5. After mapping of service 1

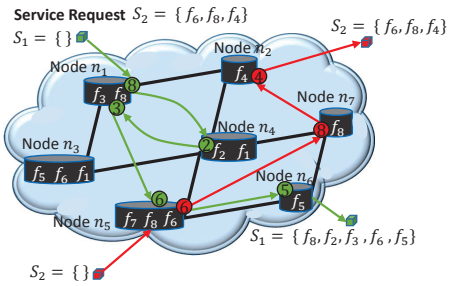

Fig. 6. After mapping of service 2

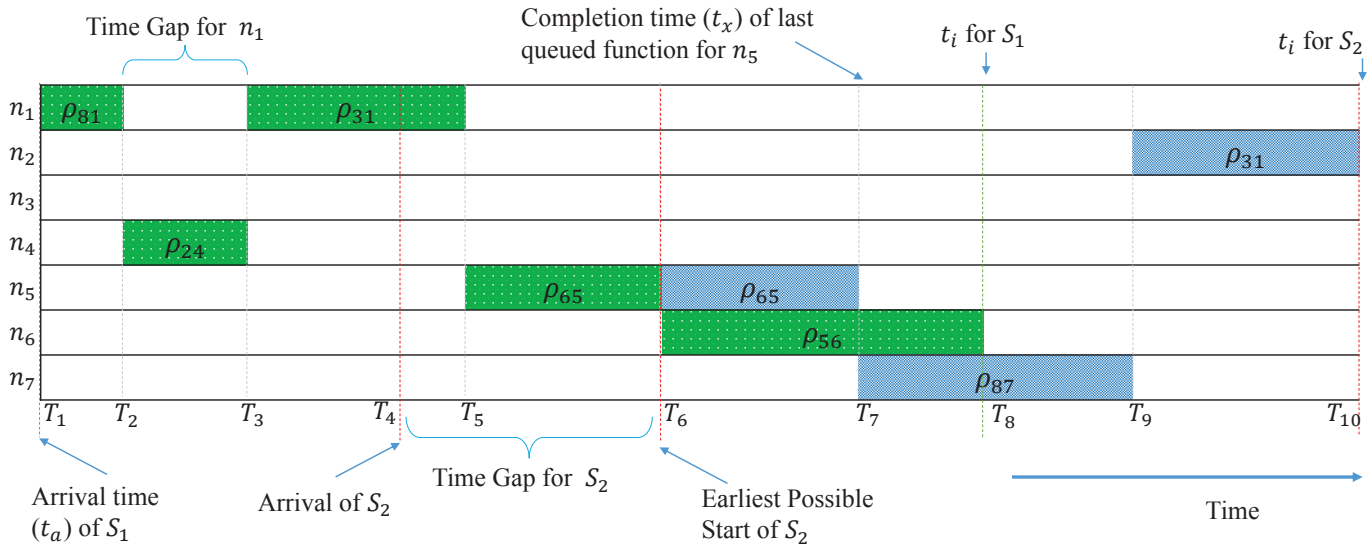

Fig. 7. Function Scheduling

1) Flowtime: We define the flow time of a service as the difference between when the processing of the last function of a service is completed and when the service arrived. The flow time is a measure of two parameters. On one hand it is a measure of how efficiently resources are being utilized (since a high flow time would mean that a given service occupies the network for extended periods leading to among other things high network loading and hence high power consumption.) while on the other hand it could be used a measure of quality of service if it is associated to the delay of processing a given service. Minimizing flow time means that an average service is processed quickly, at the expense of the largest service taking a long time.

2) Revenue: The revenue R can be defined as the income from the total amount of physical network resources that are utilized by a given mapping and scheduling. It includes the buffer requirements for each function of the service on the node where it is mapped, as well as their processing times. We represent it mathematically in (1).

$$R = \sum_{i=1}^{m} \delta_i + \sum_{i=1}^{m} \sum_{j=1}^{n} \Upsilon_{ij} \times \rho_{ij} \tag{1}$$

where Υ_{ij} is a binary variable which is 1 if a function i is mapped on node j and zero otherwise.

3) Cost: In addition, we can define the cost C as the total amount of physical network resources (both time and buffer) that are utilized by a given mapping and scheduling. In particular, the cost is defined as in equation (2).

$$C = \theta \sum_{i=1}^{m} \delta_i + \varrho\left(t_i - t_a\right) \tag{2}$$

where θ and ϱ are constants aimed at scaling the costs of buffer and time resources in relation to the revenue. These constants are set as $\theta = \varrho = 0.2$ in this paper. The difference between revenue and cost is in the fact that the revenue only consists of the actual processing times of the functions, while the cost also involves those time gaps that are left unused due to functions waiting for their assigned nodes to become available.

III. Greedy Function Mapping and Scheduling

In this Section, we propose three greedy function mapping and scheduling algorithms. The first algorithm, Greedy Fast Processing (GFP), is based on functions being mapped to those nodes that offer the best processing times. The second, Greedy Best Availability (GBA) is based on functions being mapped to those nodes whose current function queue has the earliest completion time. Finally, the Greedy Least Loaded (GLL)

algorithm is based on functions being mapped to the node with the highest available buffer capacity. While all these three variations have the potential to minimize the flow time of the mapped service, anyone of them may be used with a specific objective. For example, if services are billed based on the time they spend while being processed, then it may be cheaper to map functions to nodes that process them faster. On the other hand, if the billing is directly proportional the total time the function or service will spend queued for processing or if it is required to perform the processing as early as possible or to balance the actual loading of the network, then it might be better to use GLL. Balancing the load of the substrate network could lead to a better acceptance ratio for service as it was proved for the embedding of virtual networks in network virtualization environments [10]. A combination of these two parameters e.g linear may also be used. Finally, the GBA may be used in case it is required that the service spends the least amount of time in a queue.

The algorithms perform as follows: On arrival of a service request, the functions of the service are mapped and scheduled sequentially i.e. the first is mapped and scheduled, then the next one etc. For each function i, all the nodes $N(i) \subseteq N$ that have the capacity to process it are determined. These nodes are then ranked based on the greedy criterion, i.e. least loading for GLL, lowest processing times for GFP and shortest virtual node queues for GBA. Then the node with the best rank is chosen for the mapping, and the function is scheduled for processing at the end of the node queue. The actual processing start time is based on both the node being available (completing the processing of previously queued functions), and the the processing of the preceding function (if applicable) being complete. It should also follow all the other constraints as stated in Section II. For example, assuming that the node with the best processing time does not have enough buffer resources for the function being considered, then the second best node is chosen as the one with the best rank. In addition, at each scheduling step, the completion of the processing is determined to ensure that it is within the deadline for the service. The failure of a mapping or scheduling can happen anytime during algorithm run time. For example, it is possible that while mapping the last function of a service, the completion time exceeds the deadline, or that the function has no candidate node (due to all candidate nodes being fully loaded). In this case, the service request is rejected, and all resources allocated to other functions in the service are rolled back. In algorithm 1, we show the psuedocode for these algorithms.

Algorithm 1 Greedy Function Mapping(S, N, T)

```
Start
Backup Substrate Network State
for Function i ∈ S do
    Initialise: Capable Node Set N' = ∅
    if (i = 1) then
        t_{i-1} = t_a
    end if
    for Node j ∈ N do
        t_e = ρ_ij + max(π_j, t_{i-1})
        if ((β_ij == 1) ∧ (B_j ≥ δ_i) ∧ (t_e ≤ t_l)) then
            N' = N' ⊎ n
        end if
    end for
    if N' ≡ ∅ then
        Mapping and Scheduling Failed
        Reset Substrate Network Status
        return
    end if
    Sort N' according to T
    Select the top node j* from N'
    Map the function i onto j*
    Set t_i = max(π_j, t_{i-1})
    Update B_j, π_j, and t_{i-1}
end for
Mapping and Scheduling Completed
End
```

IV. Tabu Search-based NFMS

A. Tabu Search (TS)

TS is a metaheuristic search method based on local (neighborhood) search methods used for mathematical optimization [11]. Local search [12] takes a potential solution Z to a problem and checks its immediate neighbors $N(Z)$ in the hope of finding an improved solution Z'. The solutions in $N(Z)$ are expected to be similar to Z except for one or two minor details. However, local search methods have a tendency to become stuck in sub-optimal regions or on plateaus where many solutions are equally fit. TS enhances the performance of local search by relaxing its basic rule. First, at each step, worsening moves can be accepted if no improving move is available (like when the search is stuck at a strict local mimimum). In addition, TS uses memory structures that describe the visited solutions or user-provided sets of rules. If a potential solution has been previously visited within a certain short-term period or if it has violated a rule, it is marked as "tabu" (forbidden) so that the algorithm does not consider moving to that solution repeatedly. A move like this is called a tabu move. However, when a tabu move has a sufficiently attractive evaluation where it would result in a solution better than any visited so far, then its tabu classification may be overridden. A condition that allows such an override to occur is called an aspiration criterion [7].

B. Proposed TS Algorithm

In order to design a TS algorithm, five major components must be determined: the initial solution, the neighborhood solutions, tabu list, aspiration criterion and stopping condition. In what follows, we discuss these aspects with respect the proposed algorithm.

1) Initial solution: We start by determining an initial solution Z_0. This is determined randomly. It is achieved in two steps: First, for each function i in the service to be mapped, a candidate virtual machine j which meets the requirements described in Section II is randomly chosen from the set $N(i) \subseteq N$. Then, starting with the first one, the functions are scheduled onto the virtual machines where they have been mapped, taking into consideration all the function and machine timing restrictions as described in Section II-B. The current solution Z is then set as Z_0.

2) Neighborhood Solutions: In order to find another solution Z' which is better than the current solution Z, we need

to evaluate solutions $N(Z)$ in the neighborhood of Z. To this end, we should first define $N(Z)$. Ideally, all solutions that involve moving each function from one virtual machine to another could produce a different solution. However, this would lead to a big search space. Therefore, we restrict the neighborhood to be based on changes in the mapping of the function with the biggest preceeding time gap. In other words, we evaluate the time gaps between each function. This is the time between the completion of a preceeding function and the start of processing of the current function. The function f' with the biggest time gap is chosen as a candidate for migrating. This way, $N(Z)$ involves all possible solutions which would result from migrating f' from its current virtual node to another virtual node which has the required capabilities to process it. If there is no candidate virtual machine for f', the function with the next biggest time gap is chosen for migration. After the migration, then the scheduling of all the proceeding functions is evaluated to ensure that the flow time is minimum and its according to the restrictions in Section II-B.

3) Tabu List: If a function i has been moved from virtual machine j_1 to virtual machine j_2, we declare it as tabu to move this function back to j_1 during the next $m-1$ iterations. The reason for using $m-1$ is to give a chance for the remaining $m-1$ functions in a service to be moved before the function under consideration can be returned back to its original virtual machine. It is also tabu to choose a solution with a higher flow time than the best known solution. Therefore, the tabu in this paper is recorded in short-term memory as a 2-tupple $T(i, j_1)$.

4) Aspiration criterion: We allow for aspiration in which the criterion is to allow a tabu move if it results in a solution with a lower flow time than that of the best known solution Z^*, since this would imply that the solution that results from moving f back to j_1 has not been previously visited. In addition, if all available moves are classified as tabu due to having a higher flow time than the best known solution, then we determine and select a "least tabu" move, which is the move with lowest flow time of all the tabu moves.

5) Stopping condition: Finally, we have defined two criteria which determine when the algorithm stops: (1) if after m consecutive iterations without an improvement in the flow time, (2) no feasible solution in the neighborhood of solution Z for all functions.

The psuedocode for the proposed solution is shown in algorithm 2. An initial solution is determined in line 1, and if we are successful in getting an initial solution, the initialization step in line 6 sets this as the current solution Z, and as the best known solution Z^*. We also initialize the tabu list T_l as empty. The while loop starting at line 8 will continue searching for a solution until the *stopping condition* is met. In lines 10 to 14, the neighborhood solutions are checked to eliminate those which are tabu. The solution with the lowest flow time is then chosen in Line 15, and the counters in the tabu list are updated in line 17. If the candidate solution has a lower flow time than the current best (line 18), and its features are added to the tabu list (line 19) and it is set as the new best (line 20).

Algorithm 2 Tabu Search-based NFMS(S, N)

```
1: Determine Initial Solution Z_0
2: if Z_0 notPossible then
3:    Mapping and Scheduling Failed
4:    return
5: end if
6: Initialize: Z = Z_0, Z* = Z_0, T_l = ∅
7: Determine function f to move and neighborhood N(Z)
8: while StopConditionNotMet() do
9:    solutionSet = ∅
10:   for sCandidate ∈ N(Z) do
11:      if sCandidate doesNotViolate T_l then
12:         solutionSet = solutionSet ⊎ sCandidate
13:      end if
14:   end for
15:   sCandidate = BestFlowTime(solutionSet)
16:   Z = sCandidate
17:   updateTabuList(T_l)
18:   if FlowTime(sCandidate) < FlowTime(S*) then
19:      addNewTabu(T_l, f, getNode(f), getSize(S))
20:      Z* = sCandidate
21:   end if
22: end while
23: return Z*
```

V. Evaluations

A. Simulation Environment

To evaluate our algorithms, we have implemented a discrete event simulator in Java. In these evaluations, the services to be mapped and scheduled arrive one at a time following a Poisson distribution. We defined 10 different network functions with unique labels $1-10$. Any given service is created as a combination of one or more of these functions. The service arrival rate is 1 service in every 5 time units, and any service utilizes a given node resources until it has been processed by the corresponding node. Unless stated otherwise, the main parameters used in these simulations for creating the virtual nodes and services are chosen randomly following a uniform distribution with minimum and maximum values shown in Table I. Each simulation involves services arriving, being mapped and scheduled (or rejected in case the constraints cannot be met), and departing after being processed. Simulations are carried out for 1,500 service arrivals. The choice of these simulation parameters as well as their distribution are motivated by simulations and evaluations of a well studied and related virtual network embedding problem [13].

B. Compared Algorithms

Since we are not aware of any online mapping and scheduling proposals for network functions, we have only compared the performance of our proposed algorithms. The codes in Table II are used to represent each of the algorithms.

C. Simulation Results

The results of the simulations are shown in Figures 8 - 11. In what follows we discuss these results.

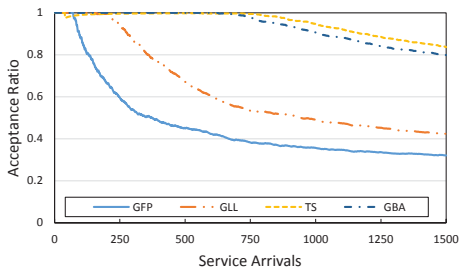

Fig. 8. Service Acceptance Ratio

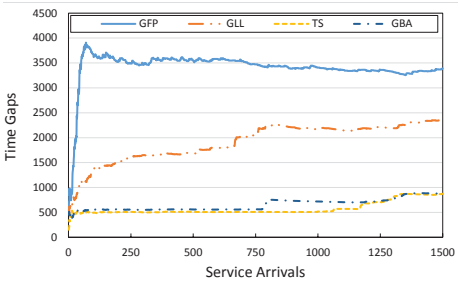

Fig. 9. Average Time Gaps

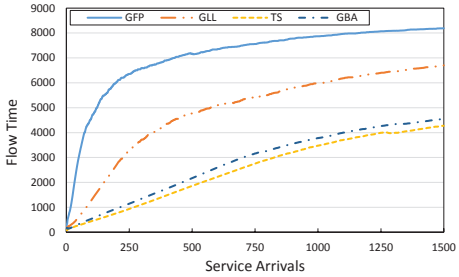

Fig. 10. Average Flow Time

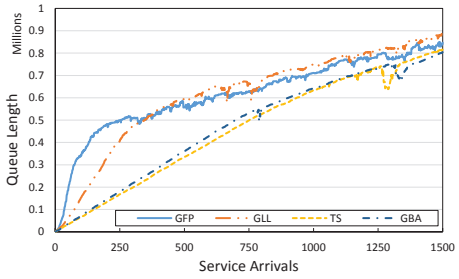

Fig. 11. Queue Length

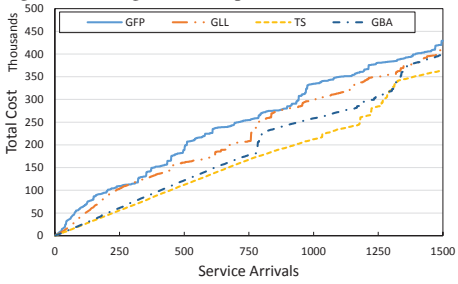

Fig. 12. Total Cost

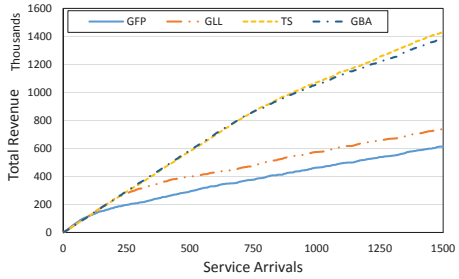

Fig. 13. Total Revenue

TABLE I. SIMULATION PARAMETER RANGES

Parameter	Minimum	Maximum
Number of nodes	50	50
Node buffer capacity	75	100
Function processed by each node	1	7
Function processing times	15	30
Function buffer demand	7.5	10
Number of functions per service	5	10
Service processing deadline	5000	10000

TABLE II. EVALUATED ALGORITHMS

Code	Function Mapping and Scheduling Algorithm
GFP	Greedy mapping with bias towards Fast Processing
GLL	Greedy mapping with bias towards Least Loaded
GBA	Greedy mapping with bias towards Best Availability
TS	Tabu Search-based Network Function Mapping and Scheduling

1) Acceptance ratio: Fig. 8 shows the variation of service acceptance ratio with service arrivals. The acceptance ratio is defined as the proportion of the total service requests that are accepted by the network. It is a measure of how efficiently the algorithm uses network resources for accepting service requests. As can be noted from the figure, GBA performs significantly better than the other greedy algorithms GFP and GLL, while TS performs only slightly better than GBA. The reason why GBA performs well could be attributed to ensuring that services are mapped to nodes which have earlier availability. This ensures that the total time that the service (or any of its functions) takes waiting in a queue for processing is minimized, which does not only avoid holding up resources which could be used by other services, but also ensures that the flow time is within the service requirements. The fact that TS performs better than GBA could be due to the fact that unlike GBA, TS has a chance to iteratively improve the solution. But since the evaluation of new solutions for TS is based on shortest flow time (just like the greedy criterion in GBA), we can observe that the difference in performance for these two algorithms is small. Finally, we observe that the greedy approach, which is biased towards favoring using nodes that are least loaded (GLL), performs better than GFP, which is based on favoring nodes with the best processing capacities. This can be attributed to the fact that least loaded nodes are likely to have shorter queues, which implies that services mapped onto such nodes get processed earlier, and hence they do not occupy the node resources for longer periods which could possibly lead to rejecting service requests.

2) Average Time Gaps and Flow Time: Figs 9 and 10 show average time gaps and flow time respectively. The time gaps for a given service are determined by summing all those times from when the service arrives when none of its constituent functions is being processed. The average time gaps are determined by averaging the cumulative time gaps of all successfully mapped and scheduled services. Similarly, the flow time is determined as the total time that a given service spends in the network while any of its function is being processed or waiting to be processed. In the same way, values shown in 10 are average values taking into consideration the total successfully mapped and scheduled services. It can be observed that GFP has a considerably higher value of time gaps compared to the other algorithms. This could be attributed to always trying to map a given function to the node that processes it faster. This means that a node that has the least processing time for a given function is likely to always be over loaded, causing a longer queue at such a node, which implies that other functions for the same service should wait for such a function. This is contrary to GLL, for example, which specifically ensures that functions are mapped to those nodes with the lowest loading, which ultimately reduces the waiting times of their executions and hence the time gaps. The reason that TS and GBA perform better than GLL can be attributed to the fact the former algorithms are specifically formulated with the objective of minimizing the flow time. This gives these algorithms an edge in reducing the time gaps,

since we could have a scenario where a given node is lightly loaded in terms of buffer/storage utilization, but only because the mapped functions do not require a lot of buffer resources, but take a long time to process. In this way, a function may actually be mapped onto a node where it may have to wait longer, even though this node is not highly loaded. For these reasons, we observe that the actual flow times shown in Fig. 10 have a similar profile as the time gaps in Fig. 9.

3) Node Queue Length: Figure 11 shows the cumulative queue length for all the nodes in the network. We define the queue length of a given network as the total completion time of processing all the functions queued at all its nodes. Subsequent values involve accumulating these values for each accepted service. As expected, we observe that the queue sizes increase as the network accepts more services. In addition, at the beginning, we have a profile similar to that in figures 9 and 10 where GBA and TS perform better than GFP and GLL. Once more this can be explained by the differences in the time gaps from all the algorithms. However, we also observe that as the number of arriving requests increases, the queue of GFP starts increasing at a lower rate, and so does that of GLL. Towards the end of the simulation (1500 service requests), all the four queue are almost comparable, and increase at the same rate. The reason for this could be that at that point, the queues have grown to the maximum sizes that could be permitted by the processing deadlines of each arriving service, hence all services that would make the queues grow higher than these levels are rejected.

4) Cost and Revenue: Figures 12 and 13 show the mapping and scheduling costs as defined in Section II-C. The values shown are cumulative, implying that after every successful mapping and scheduling, both the cost and revenues are determined using equations (2) and (1) respectively, and then added to values from the previous accepted service. It can be observed from 12 that TS has the lowest cost while GFP the highest, and that these costs increase at almost the same rate for all algorithms. Once more, the reason for the differences in these costs is due to the differences in waiting times. What is worth noting is that even with the lowest cumulative cost, TS has the highest acceptance ratio, implying that its average cost per accepted service is much lower. Looking at the cumulative revenue profiles in Fig. 13 shows a profile similar to that of the acceptance ratio in Fig. 8, where the more services that are accepted, the higher the revenue. An algorithm that has a higher acceptance ratio is likely to have a high revenue in the long run, which would lead to better profitability for infrastructure providers.

VI. Related Work

A. Virtual Network Embedding

The mapping of VNFs is related to both virtual network embedding (VNE) [9] and virtual data center embedding (VDCE) [8]. In both of these problems, it is required to map virtual resource requests to physical infrastructures in such a way that a given physical node may only map a single virtual node from the same virtual resource request (one-to-one mapping). However, as shown in Fig. 3, the NFMS problem has a third additional dimension of network functions on top of the virtual machines. Moreover, in the mapping of VNFs, a given virtual node may map more than one function from the same service if it is able to process them (possibility of many-to-one mapping). Therefore, the problem involves not only the mapping of virtual machines onto physical machines, but also VNFs onto the virtual machines. It also involves the scheduling of these functions so as to ensure that their timing and precedence requirements are met. Virtual network embedding does not have this requirement.

B. Real-time Scheduling

The function scheduling part of our proposal is related to real-time scheduling [14]. Real-time scheduling algorithms may be classified as either being offline [15], [16] or online [14], [17]. Static algorithms allocate jobs to processors assuming that all jobs are available to start processing at the same time, while online scheduling is intended for applications in which the jobs or tasks which may unexpectedly arrive [14]. In this context, we can compare the mapping and scheduling of VNFs to the classical flexible job shop scheduling problem (FJSP) [18]. The FJSP deals with need to determine a schedule of jobs that have pre-specified operation sequences in a multi-machine environment. While the FJSP is well studied with approaches ranging from meta-heuristics, and artificial intelligence-based approaches [19] almost all approaches in this regard concentrate on the offline problem, in which it is assumed that all the jobs to be scheduled are available at the same time [20]. However, to benefit from the full potential of NFV, it is necessary to allow for an environment where the need to create services appears when there is need. The online scheduling problem is more difficult since we need to consider the arrival times of the requests and there are more possibilities of inefficient resource utilization due to time gaps created by earlier mappings and schedules.

C. NFV Frameworks and Architectures

There have already been a number of frameworks and architectures proposed for NFV. These include [2] which is a non-proprietary white paper authored by network operators to outline the benefits, enablers and challenges for NFV, [3] an Internet draft that discusses the problem space of NFV and [5] a reference architecture and a framework to enforce service function chaining with minimum requirements on the physical topology of the network. In addition, Clayman et al. [21] describe an architecture that uses an orchestrator to monitor the utilization of infrastructure resources so as to place virtual routers according to policies set by the infrastructure provider, while DROP [22] proposes a distributed paradigm for NFV through the integration of software defined network [23] and information technology (IT) platforms. T-NOVA [24] outlines an architecture, which allows operators to deploy VNFs and offer them to their customers, as value-added services. All these approaches do not address the mapping and scheduling aspects of NFV.

D. Network Function Mapping and Scheduling

The authors in [6] present and evaluate a formal model for resource allocation of VNFs within NFV environments, called VNF placement. However, they do not consider the fact that network functions have precedence constraints, hence they consider the problem as a mapping one, similar to VNE.

Similarly, [25] formulates the problem assuming that nodes have unlimited buffer/storage space to store network functions as they wait for forwarding it to the next node in the sequence. The authors only formulate the problem without solving it. Finally, [26] proposes an algorithm for finding the placement of the VNFs and chaining them together considering the limited network resources and requirements of the functions. All these proposals consider the offline problem in which all the service requirements are known at the same time.

VII. Conclusion

In this paper, we have formally defined the problem of mapping and scheduling functions in a NFV environment. We have proposed a set of greedy algorithms and tabu search-based heuristic. We have evaluated different aspects of our algorithms including acceptance ratio, cost and revenue and discussed the advantages and disadvantages of each of them. We have noted that while the tabu search-based algorithm performs better than the greedy ones, its superiority with respect to the best greedy algorithm is not significant. It is our opinion that these algorithms can be used as a starting point for future algorithms.

However, the algorithms proposed in this paper do not consider the links between physical/virtual nodes, and consequently, the link delays for transferring a given function from one node to another (for processing of the proceeding function) are considered to be negligible. It would be interesting to evaluate the effect of link delays. The algorithms considered in this paper are also static in that after the initial mapping and scheduling, the functions are not migrated with changing network conditions. Finally, while these algorithms are simple and therefore represent a good starting point, it is necessary to formulate more efficient algorithms, say based on mathematical optimization, so as to act as benchmarks for future performance evaluations.

Acknowledgment

This work is partly funded by FLAMINGO, a Network of Excellence project (318488) supported by the European Commission under its Seventh Framework Programme, and project TEC2012-38574-C02-02 from Ministerio de Economia y Competitividad.

References

[1] F. Yue, "Network Functions Virtualization - Everything Old Is New Again. Techinical White Paper," F5 Networks, Inc., Tech. Rep., February 2014.

[2] R. Guerzoni, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action. Introductory white paper," in *SDN and OpenFlow World Congress*, June 2012.

[3] W. Xu, Y. Jiang, and C. Zhou, "Problem Statement of Network Functions Virtualization Model. Internet-Draft, draft-xjz-nfv-model-problem-statement-00," Active Internet-Draft, IETF Secretariat, Tech. Rep., September 2013.

[4] W. Liu, H. Li, O. Huang, M. Boucadair, N. Leymann, Z. Cao, Q. Sun, and C. Pham, "Service Function Chaining (SFC) Use Cases. Internet-Draft draft-liu-sfc-use-cases-05," Active Internet-Draft, IETF Secretariat, Tech. Rep., April 2014.

[5] M. Boucadair, C. Jacquenet, R. Parker, D. Lopez, J. Guichard, and C. Pignataro, "Service Function Chaining: Framework and Architecture. Internet-Draft draft-boucadairsfc-framework-02," Active Internet-Draft, IETF Secretariat, Tech. Rep., February 2014.

[6] H. Moens and F. De Turck, "VNF-P: A Model for Efficient Placement of Virtualized Network Functions," in *1st IEEE International Workshop on Management of SDN and NFV Systems.*, November 2014.

[7] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[8] M. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, May 2013, pp. 177–184.

[9] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[10] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 206 –219, feb. 2012.

[11] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533 – 549, 1986, applications of Integer Programming.

[12] W. Michiels, E. Aarts, and J. Korst, *Theoretical Aspects of Local Search (Monographs in Theoretical Computer Science. An EATCS Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[13] R. Mijumbi, J. Serrat, and J. L. Gorricho, "Self-managed resources in network virtualisation environments," Ph.D. dissertation, Universitat Politecnica de Catalunya, 2014, uRL: http://www.maps.upc.edu/rashid/files/Rashid_PhD_Dissertation.pdf.

[14] M. Hu and B. Veeravalli, "Dynamic scheduling of hybrid real-time tasks on clusters," *Computers, IEEE Transactions on*, vol. 63, no. 12, pp. 2988–2997, Dec 2014.

[15] J. Xu and D. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *Software Engineering, IEEE Transactions on*, vol. 16, no. 3, pp. 360–369, Mar 1990.

[16] K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 6, no. 4, pp. 412–420, Apr 1995.

[17] K. Ramamritham, J. Stankovic, and P.-F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 1, no. 2, pp. 184–194, Apr 1990.

[18] U. K. Chakraborty, Ed., *Computational Intelligence in Flow Shop and Job Shop Scheduling*, ser. Studies in Computational Intelligence. Berlin: Springer, 2009, vol. 230.

[19] V. Roshanaei, A. Azab, and H. ElMaraghy, "Mathematical modelling and a meta-heuristic for flexible job shop scheduling," *International Journal of Production Research*, vol. 51, no. 20, pp. 6247–6274, 2013.

[20] G.-J. Chang, "On-line job shop scheduling with transfer time in supply chain," in *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on*, Sept 2008, pp. 284–289.

[21] S. Clayman, E. Mainiy, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, ser. NOMS2014, 2014, pp. 1–9.

[22] R. Bolla, C. Lombardo, R. Bruschi, and S. Mangialardi, "Dropv2: energy efficiency through network function virtualization," *Network, IEEE*, vol. 28, no. 2, pp. 26–32, March 2014.

[23] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, S. Latre, and F. D. Turck, "Dynamic resource management in sdn-based virtualized networks," in *IEEE International Workshop on Management of SDN and NFV Systems*, November 2014.

[24] G. Xilouris, E. Trouva, F. Lobillo, J. Soares, J. Carapinha, M. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis, "T-nova: A marketplace for virtualized network functions," in *Networks and Communications (EuCNC), 2014 European Conference on*, June 2014, pp. 1–5.

[25] J. Ferrer Riera, E. Escalona, J. Batalle, E. Grasa, and J. Garcia-Espin, "Virtual network function scheduling: Concept and challenges," in *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*, June 2014, pp. 1–5.

[26] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 7–13.