An efficient algorithm for virtual network function placement and chaining

Conference Paper · January 2017

DOI: 10.1109/CCNC.2017.7983207

4 authors:

Oussama Soualah
Orange, Paris, France
22 PUBLICATIONS **213** CITATIONS
SEE PROFILE

Marouen Mechtri
Orange Labs, Châtillon, France
22 PUBLICATIONS **459** CITATIONS
SEE PROFILE

Chaima Ghribi
Institut Mines-Télécom
17 PUBLICATIONS **521** CITATIONS
SEE PROFILE

Djamal Zeghlache
Institut Mines-Télécom
240 PUBLICATIONS **3,257** CITATIONS
SEE PROFILE

An Efficient Algorithm for Virtual Network Function Placement and Chaining

Oussama Soualah, Marouen Mechtri, Chaima Ghribi and Djamal Zeghlache
Institut Mines-Telecom, Telecom SudParis, CNRS UMR 5157: 9, rue Charles Fourier, 91000 Evry, France
Email: {oussama.soualah, marouen.mechtri, chaima.ghribi, djamal.zeghlache}@telecom-sudparis.eu

***Abstract*—The virtualized network functions placement and chaining problem is formulated as a decision tree to reduce significantly the complexity of service function chaining (SFC) in clouds. Each node in the tree corresponds to a virtual resource embedding and each tree branch to the mapping of a client request in some physical candidate. This transforms the placement problem to a decision tree search. We devise a new algorithm based on the Monte Carlo Tree Search (MCTS) to incrementally build and search within the decision tree. Thanks to the proposed SFC-MTCS strategy, an optimized solution is computed in a reasonable time. Extensive simulations assess the performance and show that SFC-MCTS outperforms state of the art strategies in terms of: i) acceptance rate, ii) providers revenue and iii) execution time.**

Keywords: Service Function Chaining, Network Function Virtualization, Monte-Carlo Tree Search.

I. Introduction

Network Functions Virtualization (NFV) has emerged as an innovative concept to simplify the deployment and management of networking services using virtualization and Cloud technologies. Deploying services on virtual machines brings time as well as cost savings and improves flexibility as opposed to using network hardware (i.e., routers, switches...). According to the 2016 Guide to SDN and NFV [1], the three primary factors that are driving companies interest in NVF are: reducing deployment time, reducing OPerating EXpense (OPEX) and having greater management flexibility. The Survey reports also that within four years, about 83% of IT organizations are likely to have made a significant deployment of NFV.

The NFV architecture, a European Telecommunications Standards Institute (ETSI) coordinated initiative, is based on the concept of Virtualized Network Functions (VNFs) representing the software implementation of network functions. VNFs (acting as network services building blocks) can be connected and chained to provide complex communication services. The problem of VNF placement and chaining is becoming one of the most important challenges associated with NFV. It consists in efficiently placing a sequence of virtual machines providing network services (e.g., load balancers, firewalls, IPS/IDS...) while steering traffic flows across them. This problem is known to be NP-Hard and has been explored using exact and approximation methods (e.g., [2], [3], [4]). However, exact solutions are only practical for very small problem sizes. The proposed heuristics are evaluated for small and moderate problem sizes.

The objective of this paper is to provide a scalable, fast and efficient solution while maximizing the profit of the provider and the acceptance rate of provisioning requests. It is clear that the ideal way to compute the optimal embedding is to enumerate all the candidates of each virtual resource (i.e., virtual node and/or link) within the physical network (i.e., hardware nodes and/or paths). But, this process is computationally intractable in a large scale scenario because of the exponentially increasing processing times. To reduce complexity, we formulate the Service Function Chaining (SFC) placement as a decision tree problem. Since building the entire decision tree is not feasible for a large input sizes, we propose a new Service Function Chaining algorithm using the Monte-Carlo Tree Search strategy [5] and name it SFC-MCTS. The key advantage of the Monte-Carlo Tree Search (MCTS) is the incremental construction of the decision tree while searching for an optimized solution. This characteristic of MTCS fits very well our problem since only partial building of the tree is needed to make decisions. This avoids the time consuming penalty of building the complete tree. In addition, MCTS is known to perform well in many combinatorial Computer Games (e.g., computer Go [6]) and complex real-world planning and optimization problems [7].

The proposed SFC-MCTS algorithm uses five main steps: i) Tree initialization, ii) Selection of the node to explore during the tree navigation, iii) Generation of a new sub-branch, iv) Updating of nodes' relevance and finally v) Selection of the best solution. SCF-MCTS continuously and gradually builds the decision tree. At the initialization step, the root node, which is an abstract node matching the network state before handling the current client request, is created. Starting the navigation from the root node, SFC-MCTS checks at each visited node if all its children are already developed and added to the decision tree. If this condition is satisfied, the second step (i.e., selection) is performed to select the most suitable next node to continue the navigation process. Otherwise, the third step (i.e., sub-branch generation) is started and one of the non developed child(ren) is generated at each subsequent tree-level until arriving to a leaf node. The generation of a new child corresponds to the mapping of one virtual node and link of the VNF client request. A leaf node matches with: i) the mapping of the entire request or ii) a blocking status where it is not possible to further embed virtual resources due to a lack of physical resources. These steps are repeated until a computational budget limit used to stop the execution of SFC-MCTS. Finally, the branch with the best embedding is selected based on mapping quality. In fact, this information is directly found in the leaf node to avoid parsing of the tree branches. Results of extensive simulations show that of SFC-MCTS outperforms the related work in terms of i) acceptance rate, ii) provider's revenue and iii) execution time.

Section II of the paper presents related work. The system model is described in Section III. The proposed algorithm is introduced in Section IV. Section V reports the performance evaluation results.

II. Related work

The problem of VNF placement and chaining is gaining interest in the literature. Prior works typically cast the problem into a traditional virtual network embedding (VNE) problem. However, the two problems are distinct and have different characteristics [8], [3], [9]. In VNE, requests are modeled by simple undirected graphs, whereas VNF chains are more complex components that contain both the VNFs to place and the traffic flows to steer between the end points.

Typically proposed algorithms find exact solutions for VNF placement and chaining for small instances or problem size. Authors in [2] present Stratos, a network-aware orchestration layer for virtual middleboxes in Clouds and formulate middlebox provisioning as an Integer Linear Program (ILP). Similar works [3] and [4] propose a model for resource allocation in NFV networks and implement it as an ILP.

As VNF placement and chaining is NP-hard, optimal solutions can only be found for small instances. Heuristic algorithms are consequently proposed to scale with problem size. Authors in [4] propose Multi-Stage heuristic solution, for larger problem size that uses two steps to find solutions. The first one models the VNF orchestration problem as a multi-stage directed graph with associated costs. The second step runs the Viterbi algorithm [10] on the Multi-Stage graph. In [11], authors design heuristic algorithms that place network functions with a minimum overall network cost objective. In [12], authors proposed a Greedy algorithm for VNF placement and chaining to solve the problem iteratively. The Greedy solution is based on bipartite graph construction and matching techniques and solves the problem in two steps. The first one consists in mapping VNFs on physical hosts and steering traffic between them in the second step.

In this paper, we formalize the VNF placement as a tree decision problem to scale better with problem size. To the best of our knowledge, this is the first time this approach is used to address SFC optimization.

III. SFC problem representation and model

This section formalizes the service function chaining problem and presents the physical network and VNF chain representations or models.

A. Substrate and virtual networks models

The substrate or physical network, defined by the NFV Infrastructure (NFV-I) by ETSI [13], is modeled as an undirected weighted graph, denoted by $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}))$ where $E(\mathcal{G})$ is the set of the physical links and $V(\mathcal{G})$ is the set of the physical nodes. Each substrate node, $w \in V(\mathcal{G})$, is characterized by its i) available processing power (i.e., CPU) denoted by $C(w)$, and ii) type $T(w)$: switch, server or physical network function (PNF) [14], where PNFs are the traditional physical middleboxes offering network functions. In other terms, a PNF is a dedicated hardware that implements a network function. Regarding the physical links, each one (i.e., $e \in E(\mathcal{G})$) is characterized by its available bandwidth $B(e)$.

A client request (i.e., requested service function chain) is modeled as a directed graph, denoted by $\mathcal{D} = (V(\mathcal{D}), E(\mathcal{D}))$ where $V(\mathcal{D})$ is the set of the virtual nodes and $E(\mathcal{D})$ is the set the virtual links. Each virtual nodes, $v \in V(\mathcal{D})$, is characterized by its i) required processing power $C(v)$ and ii) its type $T(v)$: switch (i.e., ingress or egress) and VNF.

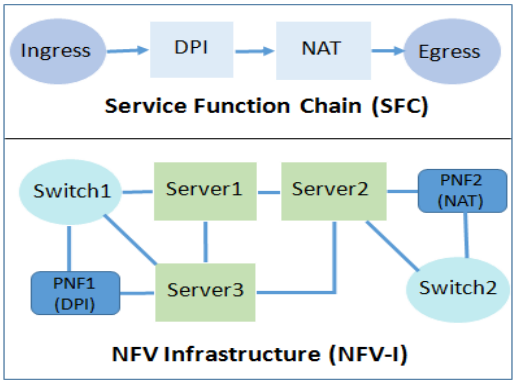


Fig. 1: The SFC and NFV-I topologies

Furthermore, each virtual link $d \in E(\mathcal{D})$ is described by its required bandwidth $B(d)$. Figure 1 depicts the SFC and the NFV-I topologies. The virtual topology is a chain of VNFs and switches. Examples of VNFs are: a firewall, a DPI, a SSL, a load balancer, a NAT, etc., as described in IETF [15] specifications. Regarding the NFV-I depicted in Figure 1, we note the existence of two PNFs and some interconnected servers and two switches serving the ingress and/or egress flows in the SFC topology.

B. SFC optimization problem

We now summarize the main constraints related to the service function chaining optimization problem. Each virtual node $v \in V(\mathcal{D})$ may be embedded in a substrate node, $w \in V(\mathcal{G})$, that has enough physical resources. Formally, $C(w) \geq C(v)$. Note that a virtual switch is mapped only onto a physical switch, and a VNF embedded in a PNF or a physical server based on the ETSI recommendations [13]. Doing so makes it easy to identify for each virtual node, $v \in V(\mathcal{D})$, a set of physical candidate hosts $Cand(v) \subset V(\mathcal{G})$. Each virtual link, $d \in E(\mathcal{D})$, should be mapped to one physical path P that meets the required bandwidth. In other terms, each physical link $e \in E(\mathcal{G})$ forming the path P should have enough remaining bandwidth to serve the required bandwidth. Formally, $\forall e \in P, B(e) \geq B(d)$. Note that splittable link mapping is not considered in this paper.

Obviously, the ideal way to compute the optimal mapping is to check all the candidates of each virtual resource (i.e., virtual node and/or link) within the substrate network. However, this is computationally intractable in a large scale scenario because of the exponential processing time penalty. In order to reduce this complexity, we model the SFC mapping problem as a decision tree denoted by $\mathcal{T}$. Each tree-node represents the embedding of one virtual node, $v \in V(\mathcal{D})$, in one of its physical candidate hosting node $w \in Cand(v)$. The connection between two tree-nodes $n1$ and $n2$ (i.e., the father and its child) describes a valid embedding of the virtual node $v1 \in V(\mathcal{D})$ and its successor $v2 \in V(\mathcal{D})$ as well as the mapping of the virtual link between them in a physical path. Accordingly, generating a new child $n2$ in the decision tree $\mathcal{T}$ from the father tree-node $n1$, that matches with the embedding of a virtual node $v1$, requires a valid mapping of i) the virtual node $v2$ (i.e., the successor virtual node of $v1$ in the SFC), and ii) the connection link. The root tree-node is defined as an abstract node that matches with the state of the NFV-I before handling the current request. The leaf tree-node corresponds to a final status that can be: i) the successful mapping of the entire SFC request or ii) a blocking state where it is impossible

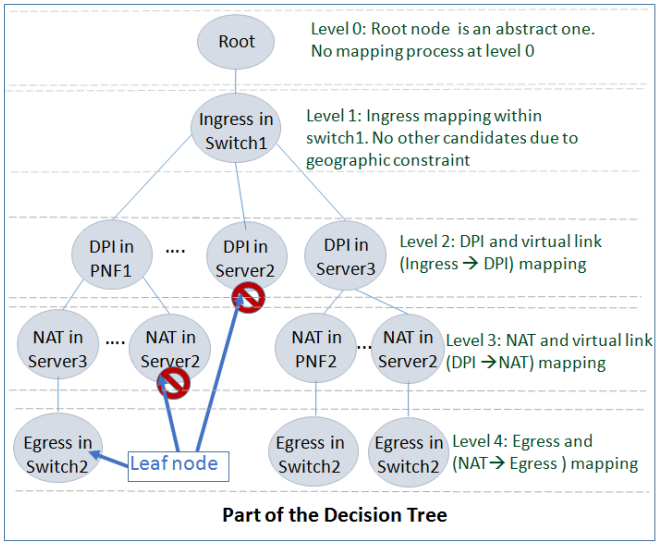


Fig. 2: The decision Tree of the SFC optimization problem

to embed further virtual resources due to a lack of physical resources. A branch in the decision tree $\mathcal{T}$ corresponds to the mapping of the SFC virtual resources. For the sake of clarity, an example of decision tree $\mathcal{T}$ is illustrated in Figure2. We present just a part of $\mathcal{T}$ for readability reason. We can see that each tree level matches with the mapping of one virtual node except the level 0 where the "abstract" root node is defined. For instance, in level 1 the ingress node must be mapped only in the $switch1$ due to geographic constraints. Then, the DPI may be mapped to one of the three servers or the $PNF1$ with regard to the available computing resources. Next, the NAT should be mapped into one server or $PNF2$ machine while respecting the required computing demand. At the last level, the egress should be embedded only on $switch2$ to respect the geographic constraints. Consequently, the SFC problem is transformed to a decision tree search. Based on this description, it is straightforward to deduce the huge number of all possible combinations if the NFV-I and/or SFC size is high. Accordingly, a judicious strategy should be devised to incrementally build the decision tree $\mathcal{T}$ and to search for an optimized mapping. In Section IV, we will describe the proposed strategy relying on the Monte-Carlo Tree Search.

IV. Proposal: SFC-MCTS strategy

In this section we present our **Monte-Carlo Tree Search** based on **Service Function Chaining** algorithm (SFC-MCTS). The SFC embedding problem is formulated as a decision tree model. Accordingly, we present the state-of-the-art main decision tree approaches and justify the adoption of the MCTS strategy. Lateron, we describe the steps involved in our proposed new algorithm SFC-MCTS.

A. Tree search optimization algorithms

One of the most challenging issues in decision tree search is scalability since the problem becomes computationally intractable for large trees. Some approaches were introduced to reduce the search complexity: Best-First-Search [16], Branch&Bound [17], A* [18], etc. These algorithms present some disadvantages in terms of : i) relevance of the solution and ii) the convergence or execution time. These strategies assume that the decision tree is already entirely built and the remaining task consists in finding the best solution. In our case the decision tree is not entirely generated. Building the complete decision tree is time consuming in large scale scenarios and this makes the proposed algorithms in the literature not suitable for solving the problem for large sizes.

Algorithm 1: SFC-MCTS psuedo-code

```
Inputs: NFV-I, SFC
Output: B_b
Initialization (T)
while Computational Budget do
    n ← Selection-Node-Explore(T)
    B ← SubBranch-Generation(T, n, NFV-I, SFC)
    T ← T ∪ B
    Update-Relevance(T)
B_b ← Selection-Best-Solution(T)
return B_b
```

SFC-MCTS mainly relies on the Monte-Carlo Tree Search [5] (MCTS) strategy. In fact, MCTS has the key advantage of coupling i) incremental tree building with ii) an efficient search mechanism. This fits well the SFC problem because the decision tree is constructed gradually and only partially. MCTS has in addition good performance in the Computer Game area such as computer Go [6] as well as some other optimization problems [5]. Recently, MCTS was successfully applied to solve the batch virtual network embedding problem [19]. Taking all these elements into consideration, we adopt the MCTS strategy to efficiently guide the decision policy. The suitable game category that best fits our problem description is the **Single Player** one [20] where there is no opponent. In fact, the objective is to maximize the single player's payoff that matches well the objective of enhancing the provider revenue for our problem. We present next, the different steps of our proposal.

B. SFC-MCTS description

SFC-MCTS incrementally builds the decision tree $\mathcal{T}$ and in parallel searches for the best mapping. SFC-MCTS is devised around five main steps: i) Tree initialization, ii) Selection of the node to explore during the tree navigation, iii) Generation of a new sub-branch, iv) Update of nodes' relevance (i.e., back propagation) and finally v) Selection of the best solution. We summarize our proposal in Algorithm 1.

1) Tree initialization: At this step, the root of the decision tree $\mathcal{T}$ is initialized and created. The root node is an abstract node that represents the system status before embedding any resource of **the current SFC** request.

2) Selection of the node to explore: This stage is mandatory for the exploration process. Once arrived to a tree-node (i.e., starting from the abstract root node) one child should be selected, among its children, to continue the navigation. In order to select one child, SFC-MCTS ensures balance between i) exploitation of promoted branches and ii) exploration dealing with less promising ones to avoid local optima. For this aim, we define for each tree-node $n \in \mathcal{T}$ a relevance tree-function denoted by χ_n. It describes the importance of the tree-node n to be selected. Formally,

$$\chi_n = \bar{\mathcal{R}}_n + \alpha \cdot \sqrt{\frac{\ln(\hat{\mathcal{V}}_n)}{\mathcal{V}_n}} + \sqrt{\frac{\sum_l {\mathcal{R}_n^l}^2 - \hat{\mathcal{V}}_n \cdot (\bar{\mathcal{R}}_n)^2 + \beta}{\mathcal{V}_n}} \quad \text{(IV.1)}$$

Algorithm 2: SubBranch-Generation

Inputs: $\mathcal{T}, n$, NFV-I, SFC
Output: New sub-branch $\mathcal{B}$
Initialization ($\mathcal{B}$)
while *isLeafNode(n) = False* **do**
    $n \leftarrow$ Compute-Child-Node(n)
    $\mathcal{B} \leftarrow \mathcal{B} + n$
return $\mathcal{B}$

where i) $\bar{\mathcal{R}}_n$ is the average revenue generated by all the branches crossing the tree-node n, ii) $\mathcal{V}_n$ is the number of visits to n, iii) $\hat{\mathcal{V}}_n$ is the number of visits of n's parent (i.e., predecessor in the tree), iv) $\mathcal{R}_n^l$ is the revenue generated by the branch $\mathcal{B}_l$ transiting over n. Regarding α and β, they are calibration constants. Note that $\mathcal{R}_n^l$ is computed once the leaf node is generated (i.e., an entire branch is built). In fact, $\mathcal{R}_n^l$ quantifies the quality of the mapping corresponding to the generated branch. It can be defined based on load balancing or consolidation objectives or any other evaluation function dealing with embedding quality.

The third term in this equation quantifies a possible search deviation of node n [20]. Indeed, it includes the sum of the squared revenues $\mathcal{R}_n^l$ corrected by the expected revenues $\hat{\mathcal{V}}_n \cdot (\bar{\mathcal{R}}_n)^2$. The β constant is useful to push for the exploration of rarely visited nodes. The above equation IV.1 allows SFC-MCTS to control the trade off between the exploitation of promising nodes and the exploration of rarely visited nodes. By doing so, SFC-MCTS tries to avoid local optima. Note that the selection step is applied only when **all the children** of the visited tree-node are already generated. Otherwise, the next step is processed.

3) Generation of a new sub-branch: Arriving to one tree-node that has at least one child not already developed in $\mathcal{T}$, SFC-MCTS generates **one** new tree-node, **at each subsequent level**, until reaching a leaf node. Algorithm 2 summarizes the sub-branch generation step. The generation of a new tree-node requires the embedding of the next virtual node (i.e., the following node in the SFC chain) as well as the mapping of the virtual link connecting both of them. The process of new node generation at each level will end by creating a leaf node based on two cases:

- Mapping all the virtual nodes and links of the SFC, or
- Blocking state due to unavailability of physical resource.

At the end of this stage, a new sub-branch $\mathcal{B}$ is generated. Accordingly, the decision tree $\mathcal{T}$ is enhanced by adding the new sub-branch $\mathcal{B}$. Formally, $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{B}$.

4) Update of nodes' relevance: Once the decision tree $\mathcal{T}$ is enhanced by the new sub-branch $\mathcal{B}$, SFC-MCTS computes the mapping quality at the leaf node n (i.e., $\mathcal{R}_n^l$) then it updates the parameters of the selection function (i.e., χ_{n^i}), defined in equation IV.1, for any tree-node n^i belonging to $\mathcal{B}$. More precisely for all nodes $n \in \mathcal{B}$, i) $\bar{\mathcal{R}}_n$ and $\mathcal{R}_n^l$ are updated with respect to the cumulative revenue induced by $\mathcal{B}$ and ii) the visit frequency register $\mathcal{V}_n$ is incremented. It is straightforward to notice the importance of this stage for the selection process (i.e., stage 2).

5) Selection of final solution: SFC-MCTS repeats the steps 2, 3 and 4 until the expiration of a computational budget denoted by δ. Note that δ can be expressed based on i) a time period or ii) the maximum number of loops or iii) any other metrics fitting the problem definition and the execution environment. It is worth pointing out that in a large scale scenario, SFC-MCTS will end without building the entire decision tree. Note also that the generated branches include an optimized solution thanks to the previous steps. Once the δ period is expired, our proposal searches for the best branch $\mathcal{B}_b$ in the decision tree $\mathcal{T}$. This search process is optimized by SFC-MCTS because during the tree $\mathcal{T}$ building process we save the information related to the relevance of the branch (i.e., mapping quality) at the leaf nodes. Looking for the best solution is, thus, equivalent to finding the leaf node with the best mapping quality. Formally, the selected leaf tree-node denoted by n_b has to satisfy this equation:

$$\chi_{n_b} = \max_{n \in LeafNodes(\mathcal{T})} (\bar{\mathcal{R}}_n) \tag{IV.2}$$

Then, $\mathcal{B}_b$ is the best sequence in $\mathcal{T}$ containing n_b. Note that $\mathcal{B}_b$ is unique since each node in the tree is attached to one and only one parent thanks to the tree topology. Finally, a bottom-up navigation computes the best branch $\mathcal{B}_b$. Note that the complexity of SFC-MCTS algorithm is $O(I\times \mid Cand(v) \mid \times \mid V(\mathcal{D}) \mid \times \mid V(\mathcal{G}) \mid^2)$. Note that $\mid Cand(v) \mid$ is the size of the candidates set of a virtual node v. The computational budget, $\mathcal{B}_b$, is defined as the maximum number of loops denoted by I. The complexity of our proposal, SFC-MCTS, is hence polynomial thanks to the use of the MCTS algorithm.

V. Performance Evaluation

In this section, we assess the performance of our proposed algorithm SFC-MCTS based on extensive simulations. We describe our simulator and its parameters. Besides, we define the performance metrics to evaluate our proposal and the related strategies. Finally, we discuss the effectiveness of our proposal by comparing with a Greedy algorithm proposed in [12] and a Multi-Stage algorithm proposed in [4].

A. Simulation Environment

To evaluate our approach, we run simulations using realistic topologies and parameters. We used the same conditions and simulation settings as in the literature to obtain meaningful comparisons [4], [12]. Requests arrive according to a Poisson process with an average rate of 5 requests per 100 time units and each resource request has an exponential lifetime with a mean of 500 time units. These settings load gradually the NFV-I that becomes fully loaded with the simulation runs. Performance results are averaged over all simulation experiments and each reported value is an average of 100 instances ensuring a confidence interval of 95%.

The considered NFV-I sizes vary between 100 and 3000 nodes with connectivity of 50%. The available CPU capacity per physical node and available bandwidth capacity per link are randomly drawn in the [150, 160] units and [100, 110] units intervals, respectively. The SFC chains are randomly generated with a number of virtual nodes per request in the range [10, 100]. The requested CPU capacity of each VNF was generated based on the data provided in [21], [15]. The CPU is drawn randomly in [2, 8] with random requested bandwidths in the [10, 20] interval.

B. Performance Metrics

We define the metrics used for the performance assessment and comparison purposes.

1) $\mathbb{Q}$*:* is the rejection rate of the incoming requests during the simulation. In other terms, the rate of client requests that have not been accepted due to the physical resource shortage.

2) $\mathbb{R}$*:* We define $\mathbb{R}(t)$ that measures the total revenue generated by the embedded VNF requests at time t :

$$\mathbb{R}(t) = \sum_{\mathcal{D} \in \mathcal{AR}_t} \mathbb{R}\left(\mathcal{D}\right) \tag{V.3}$$

where $\mathcal{AR}_t$ is the set of accepted requests until t time and $\mathbb{R}(\mathcal{D})$ is the request revenue defined as in [22]:

$$\mathbb{R}(\mathcal{D}) = \left(\sum_{v \in V(\mathcal{D})} C\left(v\right)\right) * U_C + \left(\sum_{d \in E(\mathcal{D})} B\left(d\right)\right) * U_B \tag{V.4}$$

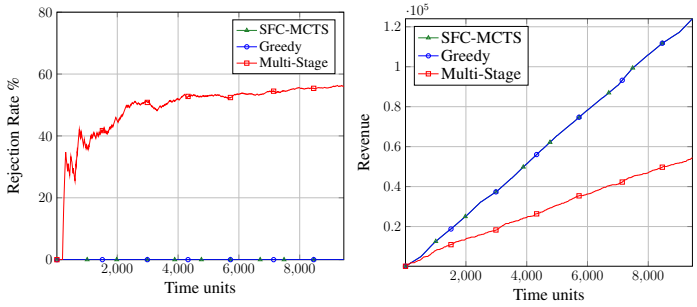
where U_C and U_B are the revenue gained by allocating a unit of computing resource and bandwidth, respectively. Note that $\mathbb{R}$ is the final revenue at the end of the simulation.

3) $\mathbb{F}$*:* The execution time is the time needed to embed the virtual resources requested by a client.

C. Simulation results

1) ***Rejection rate and provider revenue****:* The first evaluation compares our approach with the Greedy algorithm and the Multi-Stage algorithm in terms of rejection rate. The SFC requests sizes are in the range [10, 15].

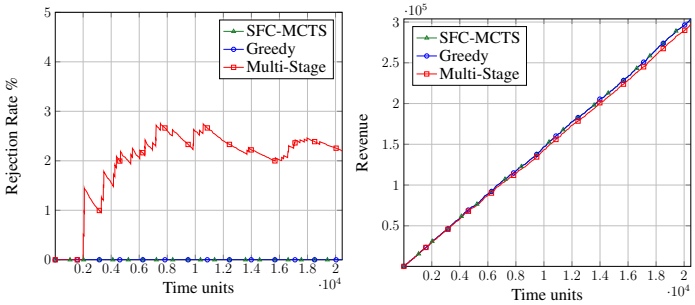
For this scenario (Figure 3), we generate a low load condition where the inter-arrival times is equal to 5 time units and sojourn time is equal to 500 time units. The results show that the proposed SFC-MCTS algorithm performs better than the Multi-Stage algorithm since it accepts all requests arriving to the system (NFV-I). The Multi-Stage algorithm rejects more than 50% of the requests when the system reaches the steady state and 56% at the end of the simulation. The performance degradation is due to the limitation of the Multi-Stage algorithm in finding the link mapping. This result is confirmed in the next evaluation scenario (Figure 4). Note that the Greedy algorithm has the same performance as our proposed algorithm since it accepts all requests but experiences difficulty in finding solutions in reasonable time (see execution time results in figures 6 and 7).



(a) (b)

Fig. 3: Rejection Rate and Provider Revenue for a recommended scenario [12]

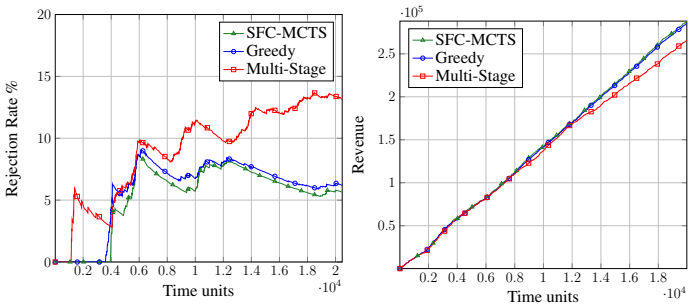
In order to provide a **favorable condition for the Multi-Stage algorithm**, we increased bandwidths on NFV-I by multiplying them by 5 (from 100 to 500) and we reported results in Figure 4. As depicted in this Figure, our proposed algorithm still accepts all the requests (Figure 4a) and the Multi-Stage algorithm rejects approximately 2.2% which is better than the last scenario where rejection rate is equal to 56%. Figure 4b shows the associated revenue computed using equation V.3. This figure shows that the SFC-MCTS algorithm increases the provider revenue by 2.15% compared to the Multi-Stage algorithm and confirms the results obtained by the rejection rate figure.



(a) (b)

Fig. 4: Rejection Rate and Provider Revenue for low load

To evaluate the performance of our proposal when system is heavily loaded, we have increased the sojourn time to 3500 time units and kept the same condition of the low load scenario. Figure 5 shows that the SFC-MCTS and Greedy algorithms accept all requests when simulation time is less than 3575 time units while the Multi-Stage algorithm rejects some requests and reaches more than 5% of rejection. When the simulation time reaches the interval [3780, 4512], the Multi-Stage rejects requests less than the Greedy approach but our proposed algorithm still performs better and accepts more than other algorithms. After that the Multi-Stage strategy continues rejecting requests and reaches approximately 13.59% while the SFC-MCTS rejects 5.65%. In addition, our proposal outperforms the Greedy algorithm which rejects 6.2% of the requests. Figure 5b shows that the SFC-MCTS increases the provider revenue by 7.6% compared to the Multi-Stage and by 0.82% compared to Greedy. These results confirm that our proposed algorithm accepts more requests and increases provider revenue compared with the Greedy and Multi-Stage strategies. In addition our proposal provides a good performance under low and high load.



(a) (b)

Fig. 5: Rejection Rate and Provider Revenue for high load

2) ***Execution time evaluation*** $\mathbb{F}$*:* We evaluate the execution time needed to handle one SFC request for the three algorithms. We fix the NFV-I size to 1000 nodes and we vary the SFC requests size. Then, we fix the SFC request size to 20 nodes and we vary the NFV-I size.

Figure 6 describes the obtained results for the first evaluation scenario. Our proposal outperforms the two other

strategies based on $\mathbb{F}$ metric. Accordingly, SFC-MCTS ensures the best provider revenue $\mathbb{R}$ in the shortest time. The Multi-Stage algorithm has an execution time comparable to SFC-MCTS but it has poor performance with respect to the revenue metric (i.e., see Figure 3). As already depicted in the Figure 4 and Figure 5, sometimes the Greedy algorithm has roughly the same performance as SFC-MCTS based on $\mathbb{Q}$ and $\mathbb{R}$ metrics. However, the former degrades in execution time, as shown in Figure 6. For instance, the Greedy needs 3466.15 seconds (i.e., $57,76$ minutes ≈ 1 hour) to deal with 60-nodes requests size compared with only 23.66 required seconds by SFC-MCTS. Hence, our new algorithm is the most suitable approach since it maximizes the provider revenue while minimizing the execution time.

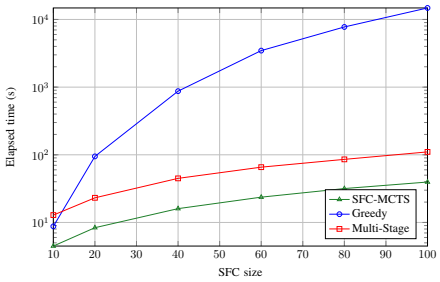


Fig. 6: Execution Time w.r.t SFC size

Figure 7 depicts the execution time when we set the SFC requests size to 20 and we vary the NFV-I size [100,3000]. It is clear that this scenario confirms the previous one since SFC-MCTS ensures the shortest time to serve one client request. For example, for 1000 NFV-I size, SFC-MCTS needs only 8.43 seconds to compute the solution compared with the 62.35 and 22.96 seconds required by the Greedy and the Multi-Stage algorithms. Starting from 2000 NFV-I size, the Multi-Stage is unable to compute any solution and needs unacceptably long times to reject the requests. This can be explained by the construction of the augmented graph that the Multi-Stage algorithm uses. Hence, the Multi-Stage performs poorly in terms of scalability. Our proposal needs reasonable times to handle large scale networks, which is the case of a real Data-Center. For instance, SFC-MCTS needs only 32.87 seconds to handle NFV-Is with 2000 nodes and 20 SFC request nodes. Accordingly, SFC-MCTS is the best algorithm for large scale networks and SFC requests.

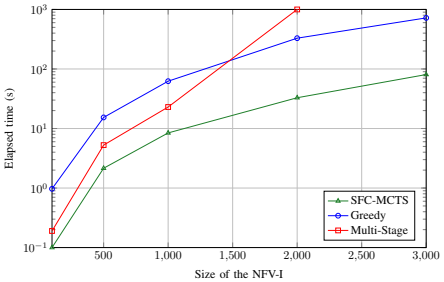


Fig. 7: Execution Time w.r.t NFV-I size variation

VI. Conclusion

In this work, we proposed a new algorithm for Service Function Chaining across VNFs. To the best of our knowledge, our solution is the first to deal with the problem using the Monte Carlo Tree Search method that has been successfully applied to several types of problems such as puzzles and complex games. Experimental results show that our proposal outperforms other methods in the literature in terms of execution time, acceptance rate and cloud provider profit. In addition, our algorithm is the most suitable for large scale infrastructures/networks which is the case of Cloud Data-Centers.

References

[1] J. Metzler, 2016, "The 2016 Guide to SDN and NFV".

[2] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. [Online]. Available: http://arxiv.org/abs/1305.0209

[3] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Network and Service Management (CNSM)*, Nov 2014, pp. 418–423.

[4] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," *CoRR*, vol. abs/1503.06377, 2015. [Online]. Available: http://arxiv.org/abs/1503.06377

[5] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

[6] K.-H. Chen, D. Du, and P. Zhang, "Monte-Carlo Tree Search and Computer Go," *Springer - Advances in Information and Intelligent Systems*, vol. 251, 2009.

[7] C. Browne and E. Powley, "A survey of monte carlo tree search methods," *Intelligence and AI*, vol. 4, no. 1, pp. 1–49, 2012.

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

[9] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "A novel approach to virtual networks embedding for SDN management and orchestration," in *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, 2014, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/NOMS.2014.6838244

[10] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

[11] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354.

[12] M. Mechtri, C. Ghribi, and D. Zeghlache, "Vnf placement and chaining in distributed cloud," in *the 9th IEEE International Conference on Cloud Computing, June 27 - July 2, 2016, San Francisco, USA*.

[13] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases".

[14] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV". [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.01.01_60/gs_nfv003v010101p.pdf

[15] S. Kumar and et al., "Service function chaining use cases in data centers," Internet-Draft, January 2016. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-sfc-dc-use-cases-04.txt

[16] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2003.

[17] J. Clausen, "Branch and Bound Algorithms - Principles and Examples," *Depart. of Computer Science, Univ. of Copenhagen*, pp. 1–30, 1999.

[18] N. Huyn, R. Dechter, and J. Pearl, "Probabilistic analysis of the complexity of A*," *Artificial Intell*, vol. 15, pp. 241 –254, 1980.

[19] O. Soualah, I. Fajjari, N. Aitsaadi, and A. Mellouk, "A batch approach for a survivable virtual network embedding based on Monte-Carlo Tree Search," *IFIP/IEEE Integrated Network Management (IM)*, 2015.

[20] M. P. Schadd, M. H. Winands, M. J. Tak, and J. W. Uiterwijk, "Single-player Monte-Carlo tree search for SameGame," *A Special Issue on Artificial Intelligence in Computer Games: AICG*, vol. 34, pp. 3–11.

[21] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," ser. SIGCOMM '13. ACM, 2013, pp. 27–38.

[22] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," *IEEE INFOCOM*, pp. 1–12, 2006.