Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Self-adaptive Online Virtual Network Migration in Network Virtualization Environments

Mahboobeh Zangiabady[1] | Alberto Garcia-Robledo[2] | Juan-Luis Gorricho[3] | Joan Serrat-Fernandez[3] | Javier Rubio-Loyola[1]

[1]Centre for Research and Advanced Studies of the National Polytechnic Institute, Mexico
[2]Consortium for the Study of Metropolitan Zones (CentroMet), Mexico
[3]Polytechnic University of Catalonia, Spain

Correspondence
Mahboobeh Zangiabady, Centre for Research and Advanced Studies of the National Polytechnic Institute. Email: mzangiabady@tamps.cinvestav.mx

Summary

In Network Virtualization Environments the capability of operators to allocate resources in the substrate network to support Virtual Networks (VNs) in an optimal manner is known as Virtual Network Embedding (VNE). In the same context, online VN migration is the process meant to re-allocate components of a VN, or even an entire VN between elements of the substrate network in real-time and seamlessly to the end-users. Online VNE without VN migration may lead to over- or under-utilization of the Substrate Network resources. However, VN migration is challenging due to its computational cost and the service disruption cost inherent to VN components re-allocation. Online VN migration can reduce migration costs insofar it is triggered proactively, not reactively, at critical times, avoiding the negative effects of both under- and over-triggering. This paper presents a novel online cost-efficient mechanism that self-adaptively learns the exact moments when triggering VN migration is likely to be profitable in the long term. We propose a novel self-adaptive mechanism based on Reinforcement Learning that determines the critical times when to trigger online VN migration, leading to the minimization of migration costs while simultaneously considering benefits on the online VNE acceptance ratio.

KEYWORDS:
Online VNE, Online VN Migration, Migration trigger ratio, Profit, Migration trigger requests, VNE acceptance ratio.

1 | INTRODUCTION

Online VNE is the process of allocating physical resources such as CPU and bandwidth (BW) to support VNs meeting both, the VN resource requirements and the physical network available capacity at the time of mapping[1]. It involves mapping each virtual node to one substrate node and each Virtual Link (VL) to one or more substrate links. The aim of the VNE problem is

to optimize the use of the Substrate Network (SN) resources while allocating resources for VNs[2]. Although embedding of VNs in a SN has been one of the resource allocation challenges in network virtualization, a mapping of VNs without VN migration may lead to inefficient resource utilization where physical network resources can be over-utilized whereas others may remain under-utilized[3,4]. VN migration is more challenging than VNE since VNs may have different constraints like CPU and memory in the physical nodes, as well as other QoS constraints like BW, delay and jitter. There may be multiple paths of substrate resources in a network that satisfy a given set of constraints. Moreover, VN migration algorithms have to ensure that the number of migrated virtualized network elements (virtual nodes and VLs) is minimal, to reduce the chances of service disruption, cost, and over-utilized physical resources. Nonetheless, efficient VN migration can improve the substrate's ability to accept future Virtual Network Requests (VNRs). However, excessive migrations may dramatically increase the operating costs[3,5,6] and may compromise the capacity of the physical network to handle VNRs that arrive at a high rate. Online VN migration can effectively enhance VNE acceptance ratio insofar it is triggered at critical times, avoiding the risks of both under- and over-triggering. In this paper we introduce the concept of online self-adaptive VN migration triggering in order to drive the triggering of VN migrations so that resource re-allocation can be profitable in the long term.

Our approach is based on Reinforcement Learning (RL)[7,8], a powerful paradigm to develop autonomic control components in dynamic online scenarios, such as virtual network environments. RL algorithms like Q-learning[9] enable the development of self-improving agents that learn through interactions with their environments[7]. To this end, in this paper an RL-based approach built upon Q-learning is utilized and configured to have as action the triggering of the VN migration in online VN environments. Q-learning learns a policy by trial and error, by interacting with the environment and by receiving feedback in the form of rewards, which may introduce additional costs associated to the learning process. Nonetheless, we show that the training process of the proposed algorithm stabilizes after only 60 episodes, highlighting the capacity of the self-adaptive trigger to learn the final policy quickly.

Finally, our approach introduces the concept of maximum substrate graph spanning tree to quantify how "strong" a substrate graph is in terms of its capacity to remain connected after link embedding link pruning stages, an aspect that severely impacts the VNE acceptance ratio. With the proposed method, the number of VN migrations can be significantly reduced while simultaneously considering the embedding acceptance ratios.

The rest of the paper is organized as follows. In Section 2 we state in detail the research problem. Section 3 describes the related state of the art. Section 4 presents the framework on which our self-adaptive VN migration trigger is based upon. Section 5 elaborates on a method for measuring the physical substrate strength, which is a pivotal element of our solution. Section 6 describes in detail the proposed reinforcement learning approach. Section 7 describes in detail our self-adaptive VN migration trigger algorithm. Section 8 presents the simulation setup and the assessment methodology. Sections 9 and 10 show the validation results. Finally, concluding remarks and our future work are presented in Section 11.

2 | PROBLEM STATEMENT

A migration trigger is a process that decides whether to trigger or to withhold (avoid triggering) the migration of VNs, with the expectation of improving the chances of future VNR acceptance. On the one side, triggering VN migration too often may increase the chances of VNR acceptance, but incurring into the cost of too many migrations at the same time (over-triggering). On the other side, withholding VN migration too often avoids the cost of migration but it may decrease the chances of VNR acceptance (under-triggering). The question is: At which times should on-line VN migration be triggered in order to avoid the negative effects of both over- and under-triggering?

We pose the VN migration triggering problem as follows. Assume that VNRs arrive sequentially and cyclically, i.e., they arrive one after another; and after the last one arrives, the first VNR arrives again, then the second one, and so on, in the same order. Assume that we only know in advance the number of VNRs that are going to arrive. Let this number be M. The VN migration trigger problem is the problem of finding the best trigger binary sequence of length M, i.e. the sequence that indicates the exact triggering times yielding the maximum profit. In the sequence a "1" at the i-th position denotes "trigger migrations just before the i-th VNR arrives", whereas a "0" denotes "withhold". Here by profit we mean a metric that measures the balance between the VNR acceptance capacity, or utility, and the incurred cost to achieve that utility. The search space of the VN migration trigger problem is large: 2^M.

A combinatorial optimization approach would try to efficiently explore the search space to produce a fixed trigger sequence with high profit in reasonable time. However, a trigger mechanism that is able to associate the "status" of the substrate network resource utilization with the right triggering times is needed to make the mechanism usable in realistic and dynamic networking settings. This is a challenging task since, due to the on-line nature of the embedding and migration problems, VNRs and their resource requests are not know in advance. Thus, a trigger mechanism is also needed that autonomously learns when to perform profitable triggerings as the VNRs arrive. Reinforcement Learning approaches specialize in solving this kind of decision learning problems.

3 | RELATED WORKS

The classical VNE problem [10] proposes the mapping of virtual nodes and VLs onto a SN, with the aim of achieving an efficient use of the SN resources while satisfying the VNRs. We can distinguish between static and dynamic VNE solutions. Static VNE solutions try to solve the two fold problem of node mapping and link mapping considering that all VNRs are known in advance [3] or that each new arrival of a VNR is a new embedding problem to be solved with the available resources at the time of the arrival [11]. Some works, like the ones reported in [12,13], solve the VNE problem in one-shot, assigning virtual nodes and links simultaneously, using linear programming or mixed integer programming techniques. In other works, like [14,15], node and link

mappings are solved independently; usually, first ranking the nodes to select the candidate ones and secondly finding the shortest paths among the available links. More recent works, like [16,17], have introduced the use of RL techniques while addressing the VNE problem. In [16] a Markov decision process and the Monte-Carlo tree search technique are used to solve the node and link mapping problem. In [17] a policy gradient technique is used to learn an appropriate policy, based on the previous history of VN requests. In any case, all the aforementioned works are static approaches; they do not alter the resources already allocated to any VN.

On the other hand, in dynamic VNE solutions like those reported in [18,19], the VNE problem is solved and if necessary, the actual mapping of existing VNs is modified in order to optimize the overall use of the SN while satisfying the arrival of new VN requests. The work reported in [18] considers the re-optimization of the mapping of existing VLs, selecting new underlying paths or adapting the splitting ratio of the existing ones. One relevant restriction of this approach is the periodicity on the execution of the re-optimization procedure. Our work presented in this paper however, avoids periodic re-configurations. Other proposals of dynamic VNE solutions, like the ones presented in [19,20] exploit RL techniques to learn the best policy to optimize the VNE performance. Authors in [19] use a tabular Q-learning technique to dynamically increase or decrease the resources allocated to the mapped VNs according to some input indicators of the online use of the assigned resources. The work presented in [19] addresses the mapping of new VN requests counting on unused but already assigned resources to previous VN requests. The subsequent works in [20,21] are refinements of the original ideas presented in [19], working with more complex RL techniques, as those combining RL with fuzzy logic [21] or RL with artificial neural networks [20]. The developments in [20,21] clearly improve the results from [19]. Nevertheless, the aforementioned works achieve a dynamic mapping of already embedded VNs without strictly remapping the virtual nodes or VLs of any VN. Namely, their approach is restricted to increasing or decreasing the resources assigned to already-mapped VNs. However, in the dynamic process of VN assignment, network conditions change over time due to the arrival and departure of VNs. For this reason, the VN assignment without migration may lead to inefficient resource utilization where some parts of the substrate network can become excessively loaded while others are under-utilized. Migrating existing VNs can help to improve performance under dynamic situations. But, periodical migration is not desirable for a number of reasons such as overhead, time, and resources consumption [3,5,22].

The authors in [23] propose a solution which aims at minimizing the number of congested substrate links by carrying out link migrations as well. But this is a reactive solution since it is carried out only when a new VN request cannot be satisfied according to the current mappings. The authors in [24] propose a RL-based approach to dynamic migration of resources, in this case when some urgent contingency occurs and a highest-priority VN needs to be mapped. Our approach differs from this last one as we aim to learn “when” it is more convenient to migrate some of the VNs. All in all, none of the above works have validated their approaches on networks with different sizes and link densities and their robustness is questioned.

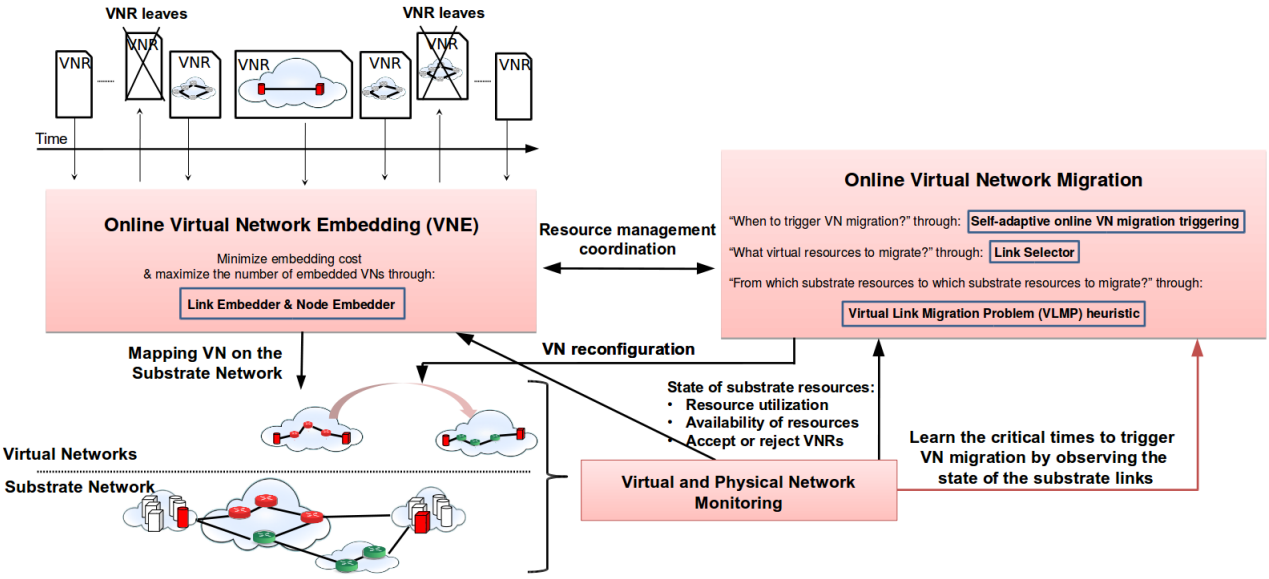


FIGURE 1 VN Migration Framework considering self-adaptive VN Migration Triggering.

There have been many advances on periodic VN migration in the state of the art[3,5,18,25–27], but there are still challenges to be explored. The work reported in[3] proposes an embedding algorithm with reconfiguration options that prioritize the migration of a selection of the VNs mapped onto stressed physical nodes or links. A VN reconfiguration mechanism proposed by[5] focuses on load balancing in the physical network. Authors in[18] developed a path migration algorithm to deal with the online VNE problem. They study the effect of periodic reconfiguration of VNs in order to enhance the embedding of new VNRs. The problem of minimizing over-utilized physical links and BW consumption has been approached by[25]. Other publications dealing with periodic VN reconfiguration are presented in[26,27]. They study the problem of VN reconfiguration considering the cost involved by the service disruption during VN reconfiguration.

The limitation of all these works is that periodic migration of VNs is extremely costly. Remapping a part or the whole VN topology periodically can interrupt services that are running due to unnecessary migrations.

The work presented in this paper differs from all previous works in that we present an approach that exploits a self-adaptive mechanism based on RL to identify the critical times to trigger online VN migration, leading to the minimization of migration costs while at the same time considering the VNE acceptance ratio. Our work contributes to the state of the art by defining a novel cost-efficient mechanism that allows a self-adaptive algorithm to identify intelligently the critical times when to trigger online VN migration, which is in turn validated through simulations on a variety of synthetic physical network topologies with different sizes and link densities.

To the best of our knowledge, this is the first work that addresses the VN migration trigger problem as formulated in Section 2, in VN environments through reinforcement learning.

4 | SELF-ADAPTIVE ONLINE VN MIGRATION FRAMEWORK

This section describes the framework of the self-adaptive online VN migration solution presented in this paper. The framework is graphically shown in Figure 1 for which a brief description of its main building blocks is given hereafter.

The two main components of the framework are the Online Virtual Network Embedding (Online-VNE hereafter) and the Online Virtual Network Migration (Online-VN migration hereafter). The allocation of resources in the SN to support the VNs, namely the VNE process (see upper left part of Figure 1) is achieved online in most real life scenarios [28]. When a VNR arrives, the Online-VNE tries to embed the VNR into the SN. The outcome can be either i) the acceptance of the VNR (successfully embedded) or ii) the rejection of the VNR (failed to be embedded). Our framework considers the online re-allocation of virtual elements between substrate resources during online-VNE. While online-VNE focuses on mapping VNs on the SN as they arrive, online VN migration is in charge of resource re-allocations in the SN in favor of more efficient resource management (see Section 4).

The framework shown in Figure 1 includes a Self-adaptive Online VN Migration Trigger Process (SA-MTrigger hereafter) whose objective is to drive dynamically the VN migration triggering times. Specifically, by associating the status of the physical network infrastructure in terms of link resource utilization, the SA-MTrigger approach learns the best (high reward) actions that reduce the migration costs, in order to avoid unprofitable migration triggerings.

The Online VN migration component of the framework (upper right part of Figure 1) addresses three critical questions; "when to trigger VN migration?", "what virtual resources to migrate?", and "from which substrate resources to which substrate resources to migrate?". A brief discussion of how this component addresses these questions is provided hereafter.

4.1 | What virtual resources to migrate?

When a trigger condition is evaluated to true, a selection process identifies the virtual resources with potential to be migrated. A Link Selector is in charge to detect over-utilized physical resources in mapped physical paths as well as to select the VLs that are candidate to be migrated. We opted for a VL-oriented migration approach because network link failures occur about 10 times more than node failures [29].

The monitoring element of the framework (see bottom part of Figure 1) provides information about the availability of physical resources in terms of BW and CPU. An important aspect in this regard is the Available Link Bandwidth (ALB), whose estimation in networking environments can be costly in terms of time consumption and increased overhead [30], making the estimation of the ALB a slow process. There are a number of BW estimation methods in the literature. Examples of these can be found in [31–37]. We consider the estimation of the ALB in our self-adaptive triggering approach, as further described in Section 8.

4.2 | From which substrate resources to which substrate resources to migrate?

The next step of the migration process is to find the appropriate resources where to re-configure VLs. This step relies on our Virtual Link Migration Problem (VLMP) approach[1], and carried out by a linear-time heuristic that we designed to find a set of substrate resources (nodes and links) that could host a VL. Our VLMP algorithm finds substrate resources that can meet the constraints of the VL prone to be migrated, and also the operator's constraints about the use of substrate resources. Our VLMP heuristic considers three stages, explained as follows:

1. The availability stage ensures the availability of physical resources by identifying bottlenecks in physical resources. This is accomplished by pruning the physical network graph to keep only physical nodes and links with available physical resources in terms of CPU and BW[1].

2. The feasibility stage is intended to find feasible paths that meet the delay and hop count constraints of the VL that is candidate to be migrated. For that, we use a reversed variation of the Dijkstra algorithm[38].

3. The optimality stage aims at finding an optimal path in the physical network that minimizes the BW and CPU in use in order to host the VL that is candidate to be migrated. For that, we use a Look-ahead variation of the Dijkstra algorithm[38].

The model and a detailed description of the VLMP algorithm are out of the scope of this paper, and can be found in[1]. The rest of the paper is devoted to describing the elements of our RL approach, as well as to show experimental evidence about the suitability of the proposed self-adaptive VN migration mechanism.

4.3 | When to trigger VN migration?

This critical question is central to our study in this work. The migration of resources is performed *only if it is triggered*. The trigger is controlled by means of a RL-based approach (described later). If triggered, the Online-VN migration proceeds to migrate a selection of virtual resources, which generally involves a cost. If not triggered, the Online-VN migration does not occur and there is no cost involved. As we are dealing with online VN migration, the RL-based approach evaluates whether to trigger VN migration for every VNR.

5 | MEASURING THE PHYSICAL SUBSTRATE "STRENGTH"

In order to design a self-adaptive triggering problem through RL, a critical decision is the selection of the substrate infrastructure resource usage variables that are going to be monitored to model the states of the environment. This is critical because some variables might be difficult to obtain in practical scenarios. Also, a wrong selection of such variables, due to the large number

[1] In this paper for this stage we ensured the availability of physical link's BW only, to enable us to reduce the potential effects of perceptual aliasing in our study and to better assess the suitability of the SN link BW measurement approach described in Section 5.

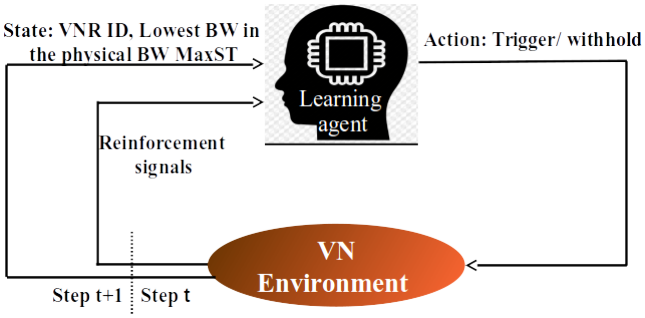


FIGURE 2 Reinforcement learning components of the self-adaptive VN migration trigger.

of possible elements to monitor in a physical infrastructure, might render an RL approach unable to distinguish among different states of the physical resource utilization. This is known as perceptual aliasing: the confounding of true states, making it difficult, if not impossible for RL algorithms, to learn stable decision policies[39].

The Online-VNE process in our Migration Framework (see Figure 1) considers node and link mapping in distinct steps, namely through a Node Embedder and a Link Embedder. The Link Embedder algorithm involves a pruning stage that removes links from the physical network graph that do not meet the requested BW requirements (see Section 4). This pruning stage enables the link mapping algorithm to avoid any physical link whose available BW is lower than BW requested by the VLs.

We have observed that, under some substrate BW congestion settings, the main reason of a high VNR rejection rate is the embedding algorithm breaking BW congested physical network[2] graphs into different connected components during the pruning stage (see Section 4). When this happens, source and destination nodes can be located at two different connected components. Under these conditions, VNRs are usually rejected due to the unavailability of a physical path between the sources and targets. A solution is to balance the BW utilization of congested physical links by means of timely path migrations in such a way that the physical network becomes "strong enough" to remain connected after future Link Embedder pruning stages. These timely migrations should happen before the physical infrastructure becomes too congested. Learning physical substrate BW congestion patterns introduces the need of assessing the connectivity or "substrate strength" of the physical infrastructure graph at any moment.

Next section introduces the concept of maximum BW substrte spanning treee, which would enable a RL agent to learn a policy that tells the Online-VNE process whether triggering migration at a given moment is going to raise the VNR acceptance rate in the long-run based on the current "BW strength" of the physical network.

[2] A physical network with low available BW in most of its physical links.

5.1 | The maximum BW substrate spanning tree

In the mathematical field of graph theory, a Spanning Tree (ST) T of an undirected graph G is a subgraph of G that includes all of the vertices of G, with the minimum possible number of edges. In other words, the ST of a graph contains the minimum set of links that are required to keep the whole graph connected. In the context of our work, an ST of the substrate infrastructure graph contains a minimum set of physical links that enable all physical nodes to reach each other.

Note that a graph may have several STs. A maximum ST of a weighted graph is a ST where the link weight sum is maximum. If we weight the physical substrate graph with the link BW, the maximum BW ST of the substrate graph (SST^{max} hereafter) will contain the set of links that: i) concentrate most of the BW available resources; and ii) keep the substrate infrastructure graph connected. These are the links that are most likely to "survive" the link pruning. Therefore, the SST^{max} constitutes a sort of "skeleton" of the physical network graph. Let $G^{phys} = (V^{phys}, E^{phys})$ be the weighted substrate graph with physical node set V^{phys} and physical link set E^{phys}, where each link in E^{phys} is weighted with its available link BW. The SST^{max} can be calculated efficiently by negating all the BW link weights in the weighted substrate graph and then using a conventional greedy minimum spanning tree method, like the Kruskal algorithm ($O(|E^{phys}| \log |V^{phys}|)$ time).

By quantifying the "strength" of the whole SST^{max} "skeleton" we can determine its "weakest bone", i.e. the minimum link BW in the ST (minBW-SST^{max} hereafter). The lower the minBW-SST^{max}, the lower the chances of the substrate graph to remain connected after the Link Embedder pruning stage (i.e. the higher the chances the substrate graph will break into different connected components). On the other hand, the higher the minBW-SST^{max}, the larger the chances of the substrate graph to remain connected. We propose to exploit the properties of the minBW-SST^{max} to model the state of the physical network utilization in our RL-based self-adaptive triggering approach, which is developed in the next section.

6 | RL MODEL FOR A SELF-ADAPTIVE ONLINE VN MIGRATION TRIGGER

This section elaborates on the RL-based self-adaptive trigger approach for online VN migration. Figure 2 depicts the self-adaptive triggering approach, which involves the definition of the states, actions and the definition of a reward scheme to evaluate the effectiveness of the agents' learning. Reinforcement Learning (RL) is a technique for solving problems in which an agent interacts with an environment (VN environment in Figure 2) and receives a reward signal at the successful completion of every step. RL finds a policy, which means a mapping from state to action (trigger/withhold in our case) to maximize the expected cumulative reward (value function) under that policy [7,40]. The following sections describe the model for the proposed RL-based self-adaptive VN migration trigger.

6.1 | RL environment state space

Our approach exploits the properties of the minimum link BW in the ST minBW-SST^{max} to define a RL agent that associates the current "strength" of the substrate network to the best action (trigger migration or withhold migration) to take at any given moment t to both: i) increase the chances that the arriving VNR would be successfully embedded; and ii) decrease the chances that unprofitable migration triggering may take place.

In our model, each step consists of a new VNR arrival. The state of the environment at step t is represented as a 2-tuple with two variables: (1) the minimum link BW in the SST^{max} (minBW-SST^{max}) and (2) the ID of the t-th VNR which is about to arrive at step t, and it is formalized with the following expression:

$$state_t = (\text{VNR}_t \text{ ID}, \ \text{minBW-SST}_t^{max}) \tag{1}$$

Let minBW-SST^{max} $\in \{0, 1, 2, 3...100\}$ be an integer number between 0 and 100 representing a percentage of available BW in a physical link. Hence, minBW-SST^{max} can take one of 101 different values. If M is the total number of arriving VNRs, then the total number of possible states is $101M$. Thus, the state space is finite but large. However, in practice, we have experimentally observed that we do not have to explore all the $101M$ states in order to learn a useful Online-VN migration trigger policy for a given substrate network and a given set of VNRs, but a fraction of this space (this is demonstrated through simulations in Section 9).

Please note that deriving exact expressions analytically for the number of states is not trivial given the online nature of the VNR embedding and migration problems. This hinders us to know neither the link BW requirements of the VNRs nor the embedding and migration paths in advance, being these two important factors that bound the number of different minBW-SST$_t^{max}$ values that the agent will observe. Moreover, the actual number of states that the agent will see is also affected by the topology of the physical networks (size and density), as we will develop later. As for the terminal states, these can be any state where the VNR$_t$ ID is the ID of the M-th VNR. Therefore, in our approach an episode ends when the last VNR arrives.

6.2 | RL actions

RL systems learn by executing actions on a trial-and-error basis. In our model, at any given step t, the agent's action, $action_t$, can take one of the following two values:

- Trigger Online-VN migration
- Withhold Online-VN migration

TABLE 1 Used reward values for each reward type r_1 to r_4.

$action_t$	$outcome_t$	$reward_t$	$\mathbf{VNR}_t$
Trigger Online-VNM	VNR_t is rejected	r_1	-500
Withhold Online-VNM	VNR_t is rejected	r_2	-50
Trigger Online-VNM	VNR_t is accepted	r_3	50
Withhold Online-VNM	VNR_t is accepted	r_4	500

At any given step t, if the agent chooses to trigger, then Online-VN migration process is triggered, namely the online VN migration process described in Section 4 is executed. On the contrary, if the agent chooses to withhold, then the Online-VN migration process is not triggered.

6.3 | RL reward function

At any step t, the immediate reward score $reward_t$ is defined as follows:

$$reward_t = \begin{cases} r_1 & \text{if } action_t = \text{trigger and } outcome_t = \text{rejected} \\ r_2 & \text{if } action_t = \text{withhold and } outcome_t = \text{ rejected} \\ r_3 & \text{if } action_t = \text{trigger and } outcome_t = \text{accepted} \\ r_4 & \text{if } action_t = \text{withhold and } outcome_t = \text{accepted} \end{cases} \tag{2}$$

The $reward_t$ is calculated by observing whether the agent decided to trigger at the current step t, and whether that decision led to either a positive (accepted VNR) or negative (rejected VNR) outcome for the embedding of the corresponding VNR. The rationale behind the reward values listed in Table 1 is as follows:

r_1: If Online-VN migration was triggered but the t-th VNR is rejected, then the agent migrated fruitlessly and it could have avoided triggering migrations to reduce costs, thus the agent receives a large punishment (-500).

r_2: If Online-VN migration was not triggered and the t-th VNR is rejected, then the outcome was not positive, but still the agent avoided the costs of a potential fruitless migration, thus the agent receives a regular punishment (-50).

r_3: If Online-VN migration was triggered and the t-th VNR is accepted, then the outcome was positive although it involved the costs of migration, thus the agent receives a regular reward (50).

r_4: If Online-VN migration was not triggered and the t-th VNR is accepted, then the outcome was positive and the agent avoided the costs of migration, thus the agent receives a large reward (500).

Algorithm 1 Self-adaptive Migration Trigger Algorithm

Input: γ, α, ϵ, $decay$
Output: Q-table Q
  Initialize Q
  for each simulation episode **do**
    Observe the initial state $state_t = (\text{VNR}_t \text{ ID}, \text{ minBW-SST}_t^{\max})$
    while state $state_t$ is not terminal **do**
      Choose an action $action_t \in \{\text{trigger}, \text{withhold}\}$, either randomly or by using a policy derived from Q (ϵ-greedy)
      Apply $action_t$ and await for VNR_t to arrive
      Await for the embedder to embed VNR_t and observe $outcome_t$
      Choose a reward $reward_t \in \{r_1 ... r_4\}$ according to $action_t$ and $outcome_t$
      Observe the new state $state_{t+1} = (\text{VNR}_{t+1} \text{ ID}, \text{ minBW-SST}_{t+1}^{\max})$
      $Q(state_t, action_t) = (1-\alpha) \cdot Q(state_t, action_t) + \alpha \left(reward_t + \gamma \cdot \max_{action} Q(state_{t+1}, action)\right)$
      $state_t = state_{t+1}$
    end while
    $\epsilon = \epsilon \cdot decay$
  end for
  return Q

Please note that the magnitudes of the values in Table 1 are not as important as the relative differences among them. We sought to provide scores that: 1) allowed the agent to differentiate between punishments and rewards by means of shifted signs, being the former negative and the later positive; and 2) allowed the agent to differentiate between regular and large reward/punishments, being the later one order of magnitude larger than the former.

7 | SELF-ADAPTIVE VN MIGRATION TRIGGER ALGORITHM

Recall that the state space of our migration triggering problem is finite, as well as the number of actions. Also, recall that, at any given state, each action brings us to another state while producing a reward in $\{r_1,...r_4\}$. This allows to model our approach of learning the best policy with a Finite Markov Decision Process (FMDP). As such, the SA-MTrigger can learn such policy by means of Q-learning [9]. For any finite FMDP, it has been proved that the Q-learning algorithm can eventually find an optimal policy, i.e. a policy that maximizes the total reward returned over all steps.

The proposed Self-adaptive Migration Trigger algorithm is listed in Algorithm 1. The algorithm is based on the well known episodic Q-Learning algorithm. Our approach exploits the online nature of the VNRs to link their timing to each step of the RL-based concept. The inputs of the algorithm are: α, the learning rate ($0 < \alpha \leq 1$); γ, the discount factor ($0 \leq \gamma \leq 1$); ϵ, the probability of choosing random actions; and $decay$, the decay rate for ϵ. These parameters will be further explained. The output of the algorithm is the Q-table Q, which will store the learned policy indicating the migration trigger agent the actions to take under different VN environment states.

The learning process is divided in episodes. In turn, each episode is divided in steps. Each step involves a single VNR arrival[3]. At each step t, the self-adaptive trigger agent observes the current state $state_t$ from the network environment. The current state is composed of the ID of the arriving VNR and the current minBW-SST$_t$, measured before the arrival. Then, it takes an action $action_t$ (either trigger or withhold), by choosing the action either randomly or by using a policy derived from Q, obeying the ϵ-greedy strategy, explained later in this section. Next, when VNR$_t$ finally arrives, the agent observes the outcome of the previously taken action: the acceptance or rejection of VNR$_t$. The agent measures minBW-SST$_{t+1}$ once again, and associates the measurement to the ID of the upcoming VNR to calculate the next state $state_{t+1}$. Finally, the Q-table value for the current state and the taken action is updated by using the well known Q-learning value iteration update formula, which weighs the old and the new acquired information. In the update formula, the learning rate α determines the precedence of the new memories when updating Q while the discount factor γ values the importance of future feedback. An episode ends when state $state_t$ is a terminal state, i.e. when the M-th VNR arrives.

We followed the ϵ-greedy strategy to control the duration of the learning exploratory stage in order to find a good trade-off between exploration and exploitation. ϵ-greedy has been used in several areas of machine learning, and it is often stated to be the method of first choice[41]. As described by[41], ϵ-greedy: 1) does not need to retain any exploration specific data; and 2) it can find a near optimal outcome by hand-tuning a single parameter only (ϵ)[42].

When the agent has to choose an action, with probability ϵ it takes a random action, and with probability $1-\epsilon$ it takes the action with the maximum Q-value, as prescribed by the ϵ-greedy approach. In this paper, we extend the ϵ-greedy approach as follows. If a random action is going to be taken, with probability ϵ the agent takes an action completely at random; and with probability $1-\epsilon$ it randomly adds scaled values[4] to the Q-values for the current state, to later take the action with the maximum randomized Q-value. As the RL episodes progress, if a random action is taken, the chances that the agent takes into account the learned policies increase, helping the algorithm to converge. We smoothly reduce the ϵ value by updating it at the end of each episode with the well-known ϵ decay formula $\epsilon = \epsilon \times decay$. In this way, as the episodes go, the agent will tend to reduce the chances of taking random actions as it reaches the exploitation stage (last episodes) of the learning process.

In order to assess the performance of our self-adaptive time triggering approach, we propose the following performance metric. The *profit* of the trigger evaluates the achieved balance between the ability of the VN migration process to embed VNRs (utility) and the costs involved to achieve such an ability. More formally, we measure the difference between the acceptance ratio of the Online-VNE process (utility) and the percentage of migration triggerings (cost). In this respect, the profit is given by the following expression:

$$Profit = AcceptanceRatio - MigrationTriggerRatio \tag{3}$$

[3] An exception is the first step which involves two VNR arrivals. The first arrival is needed to have an initial embedding to migrate. The second arrival corresponds to the actual VNR of the first step.

[4] The random values are scaled by the maximum Q-value of the current state.

, where $AcceptanceRatio$ is defined as the fraction of successfully embedded VNRs over the total number of VNR arrivals:

$$AcceptanceRatio = \frac{\text{Number of successfully embedded VNRs by Online-VNE}}{\text{Total number VNR arrivals}}, \tag{4}$$

and $MigrationTriggerRatio$ denotes the fraction of the total number of times in which the Online-VN migration process was triggered:

$$MigrationTriggerRatio = \frac{\text{Number of times that Online-VNM was triggered}}{\text{Total number VNR arrivals}}. \tag{5}$$

8 | SIMULATION SETUP AND ASSESSMENT METHODOLOGY

8.1 | Simulation Setup

In order to demonstrate the effectiveness of our approach we have produced a discrete-event simulator that emulates the performance of the framework shown in Figure 1. Our simulation platform includes an instance generator that produces SNs and VNRs. The network topologies (substrate and virtual) are produced with the GT-ITM tool[4]. Attributes of the GT-ITM graphs were added to meet the requirements of online VNE environments like CPU processing capacity of the substrate and virtual nodes, BW and delay of the substrate and VLs, and time of life of the VNRs. The values of CPU and BW capacities as well as link delays have been randomly generated using a minimum and maximum range following a uniform distribution. The time of life of VNRs follows an exponential distribution with the desired mean. VNRs are generated following a Poisson distribution that determines the order and arrival time for each VNR. A summary of online VNE parameters of our simulation setup is shown in Figure 3, which has been elaborated taking into account setups from the literature for assessment purposes[18,43–45].

The settings shown in Figure 3 allowed us to experiment in scenarios where migration triggerings are likely to be unprofitable in the long term (by the time of the last VNR arrival), hence helping us to assess the capacity of the RL agent to self-adaptively learn to avoid unprofitable VN migrations. Finally, our simulation setup where performed on substrate networks of sizes 100, 150, 200 and 250 and densities of 0.06, 0.12, 0.25 and 0.6, to demonstrate the robustness of our self-adaptive approach on different topologies.

In order to produce statistically significant results, for each network we executed 31 experiments, where each experiment consisted of running 150 RL episodes of $M = 100$ VNRs simulated arrivals (steps). The presented results show averages of the 31 RL experiments +/- one standard deviation. We used a high initial ϵ value of 1.0 to motivate the exploration during the first stage of the learning process and an ϵ decay of 0.96 to avoid the agent to become greedy too quickly. Likewise, we used a mild

Substrate Network					Virtual Network Requests					
# Nodes	Connection probability	CPU (%) Min, Max	Delay (msec) Min, Max	BW (%) Min, Max	# Nodes Min, Max	Connection probability	CPU (%) Min, Max	BW (%) Min, Max	Life time	# VNRs
100	0.06	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
100	0.12	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
100	0.25	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
100	0.6	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
150	0.06	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
150	0.12	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
150	0.25	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
150	0.6	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
200	0.06	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
200	0.12	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
200	0.25	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
200	0.6	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
250	0.06	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
250	0.12	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
250	0.25	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100
250	0.6	50, 100	0, 150	50, 100	2, 10	0.5	1, 20	1, 50	500	100

FIGURE 3 Summary of online VNE parameters for our simulation setups. The "# VNRs" column does not count the first VNR needed to have an initial embedding.

learning rate $\alpha = 0.8$ to give precedence to new memories, as well as a discount factor $\gamma = 0.9986$ to motivate the agent to strive for long-term high rewards.

We used the VLMP algorithm to re-allocate the substrate resources that host the VLs to-be-migrated via path resolution of substrate resources (CPU and BW). To this end, the VLMP algorithm requires the constraints (dly_{constr} and hop_{constr}) for the feasibility stage, as well as thresholds of resources in use (cpu^{use}_{thresh} and bw^{use}_{thresh}) for the optimality stage, as described in Section 4. While the feasibility constraints gives us control on the delay and the hop count along the path of substrate resources allowing us to set upper bounds of QoS parameters, the optimality thresholds makes sure that the cost of the paths to-be-migrated are minimized.

An exhaustive analysis of the optimal values of the VLMP thresholds and constraints is out of the scope of this paper as they are mostly dependant on specific scenarios, most probably with values linked to high-level objectives. Nonetheless, for validation purposes, we tried the following thresholds: $cpu^{use}_{thresh} = 60\%$ and $bw^{use}_{thresh} = 50\%$, whose impact to online VNE has been previously evaluated in [1]. We used relaxed constraints values $dly_{constr} = 500$ and $hop_{constr} = 6$ that, according to empiric observations, ensure good VLMP success ratios regardless of the physical network size and density, allowing the agent to observe the effect of link migrations during the learning process. The interested readers can find a detailed explanation of the VLMP configuration values in [1,2].

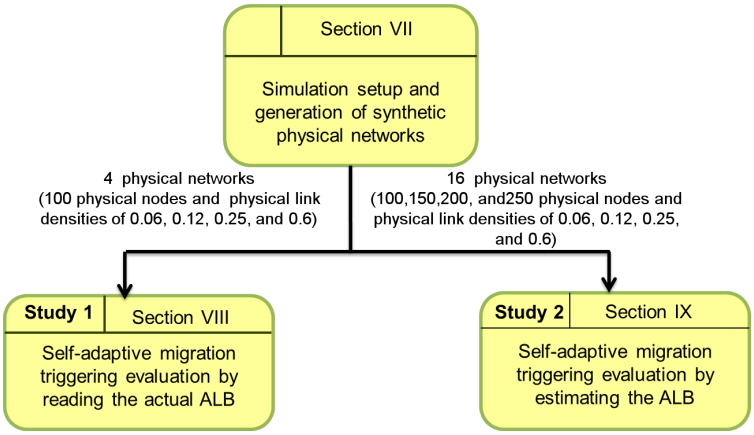


FIGURE 4 Evaluation methodology of the proposed approach

8.2 | Assessment Methodology for Validation of Self-adaptive VN migration triggering

We define an assessment methodology towards two objectives: i) to demonstrate that our approach enhances the VNE acceptance ratio in the long run; ii) to demonstrate that our approach reduces the migration costs. Moreover, pivotal in our assessment is the estimation of the Available Link Bandwidth (ALB), which is an aspect that has several technical implications in practical approaches as it can be costly in terms of time consumption and increased overhead[30] (see Figure 4). In summary, the estimation of the ALB can become slow.

Please note that comparing our work with previous work is difficult since, to the best of our knowledge, this paper represents the first attempt towards self-adaptive online VN migration in network virtualization environments. Nonetheless, we compare the effects of our self-adaptive VN migration trigger to those of periodic VN migration in Sections 9 and 10 over different physical network topologies.

8.2.1 | Self-adaptive migration triggering evaluation without ALB estimation (Study 1)

Recall that the Link Selector in our framework is in charge of i) detecting over-utilized physical resources in mapped physical paths and ii) selecting VLs that are candidate to be migrated. Physical paths are mapped/updated after each VN embedding/VN migration according to the requirements of VNRs. In Study 1 we assume a fully dynamic environment with a Link Selector that can read the actual ALBs in the SN, which are updated after the execution of either the VNE or the VN migration algorithms. In this way, Study 1 assumes a setting where actual ALBs can be easily gathered and that in turn, do not need to be estimated. In Study 1, we tested the RL approach on an array of 100 physical nodes with four different densities (see Figure 3). We ran experiments 31 times to obtain statistically significant results.

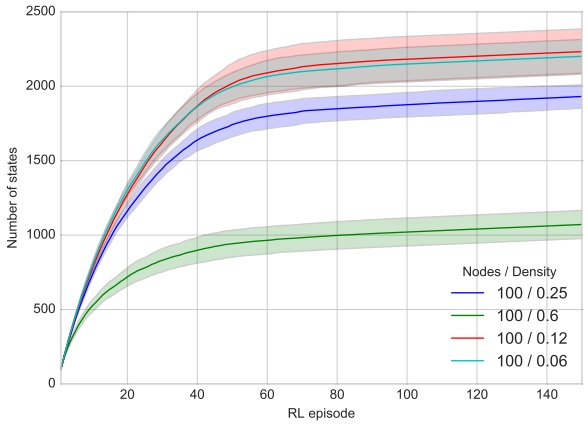


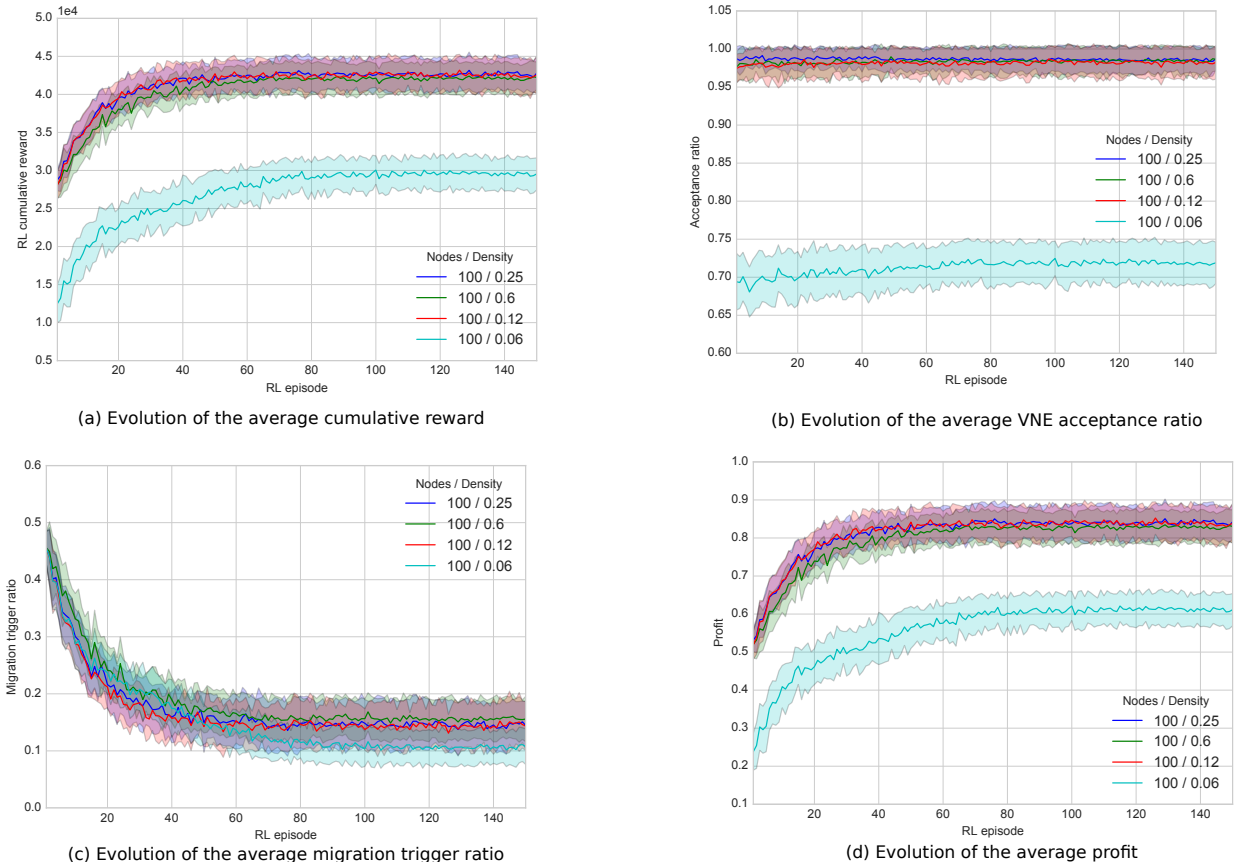
FIGURE 5 Average number of visited states (+/- one standard deviation) during the SA-MTrigger learning process for 100 physical nodes and 4 different physical link densities.

8.2.2 | Self-adaptive migration triggering evaluation with ALB estimation (Study 2)

Since the estimation of ALB in real word scenarios is not trivial, in Study 2 the Link Selector reads and remembers the ALBs in a mapped/remapped path as available at the moment that the path is computed by either the VNE or the VN migration algorithm. Once the Link Selector registers the ALBs of a newly mapped/remapped path, the ALBs of the path remain fixed in the Link Selector memory. The idea is that the ALBs are gathered only just before the execution of the VNE and VN migration algorithms. Therefore, the Link Selector has to remember and to rely on that information to detect BW bottlenecks in future. In this study we tested our RL approach on a full array of the 16 physical networks with different densities and sizes (see Figure 3). As in Study 1, we ran experiments 31 times to obtain statistically significant results.

9 | VALIDATION RESULTS FOR THE SELF-ADAPTIVE MIGRATION TRIGGER (STUDY 1)

In this section and in Section 10, we present the validation of our RL approach in terms of: i) the assessment of the RL process as the RL episodes progressed and ii) the comparison between self-adaptive migration triggering and periodic migration triggering[5,22,26]. We adopted values for periodic triggering from the state of the art[5,22,26]. Results report the average of the 31 simulation runs.



(a) Evolution of the average cumulative reward

(b) Evolution of the average VNE acceptance ratio

(c) Evolution of the average migration trigger ratio

(d) Evolution of the average profit

FIGURE 6 Evolution of the average cumulative reward, the average profit, the average migration trigger ratio, the average embedding acceptance ratio (+/- one standard deviation) during the SA-MTrigger learning process for 100 physical nodes and 4 different physical link densities.

We provide a detailed analysis of the impact of the proposed self-adaptive VN migration trigger to those of periodic VN migration for various physical network sizes and densities, in order to show that the conclusions are valid across different topologies.

9.1 | Assessment of the RL process

Figure 5 shows the average number of observed states for all the experimented physical networks as the RL process progressed. The agent observed around 1,000 different states (around 10% of the total number of states) in the densest network (density 0.6). Figure 5 demonstrates that the denser the physical network, the less the number of different states the agent is likely to see. Namely, the actual space of states that the agent is going to observe can be smaller than $101M$. Nonetheless, exploring only a fraction of the total number of states does not prevent the SA-MTrigger to learn useful policies that maximize the profit, as it is demonstrated with the following results.

Figure 6.a shows the evolution of the cumulative reward during the RL process progressed. It can be observed that the self-adaptive trigger learned to increase the RL cumulative reward from around 3×10^4 to around 4×10^4 (for densities 0.12, 0.25 and

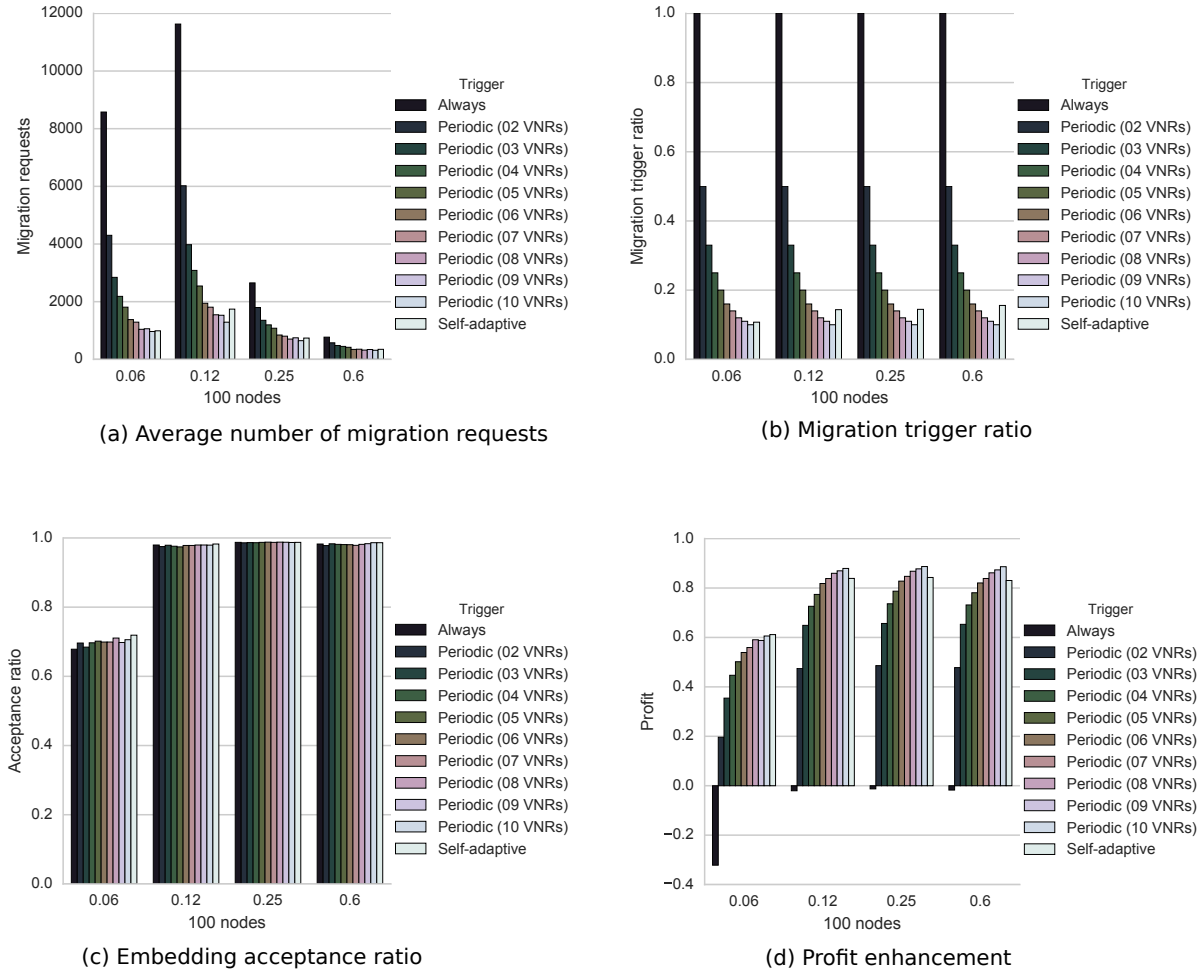


FIGURE 7 Triggers comparison for self-adaptive VN migration triggering vs. always VN migration triggering and periodic VN migration triggering. Periodic triggers are set to trigger every 2, 3, 4, 5, 6, 7, 8, 9 and 10 VNRs. Values for the self-adaptive trigger are from the last RL episode.

0.6), stabilizing between Episodes 60 and 80 regardless of the network size or density. The SA-MTrigger behaved differently only on the sparsest network (0.06 density), going from cumulative rewards of around 1.5×10^4 to rewards of around 3×10^4. The Figure 6.b shows the rate of embedding acceptance ratio evolution. As the learning process progressed, the observed acceptance ratios were of around 70% for the smallest and sparsest network (100 nodes, 0.06 density), and of above 95% for the rest of the networks. Note that the SA-MTrigger learned how to slightly improve the acceptance ratio from below 70% to above 70% for the sparsest physical network. Figure 6.c shows the migration trigger ratio evolution. During the exploratory stage (first episodes), in which the agent tried actions mostly at random, the SA-MTrigger triggered 40% to 50% of the time, as expected. However, as the RL process progressed, the SA-MTrigger learned that it had to trigger only around 10% to 15% of the time, depending on the physical network density. Figure 6.d shows the profit evolution with the RL episodes. As our agent learned to minimize the triggering ratio, the observed profits increased from 20% to around 60% for the sparsest network, and increased from 50% to around 80% for the rest of the physical networks.

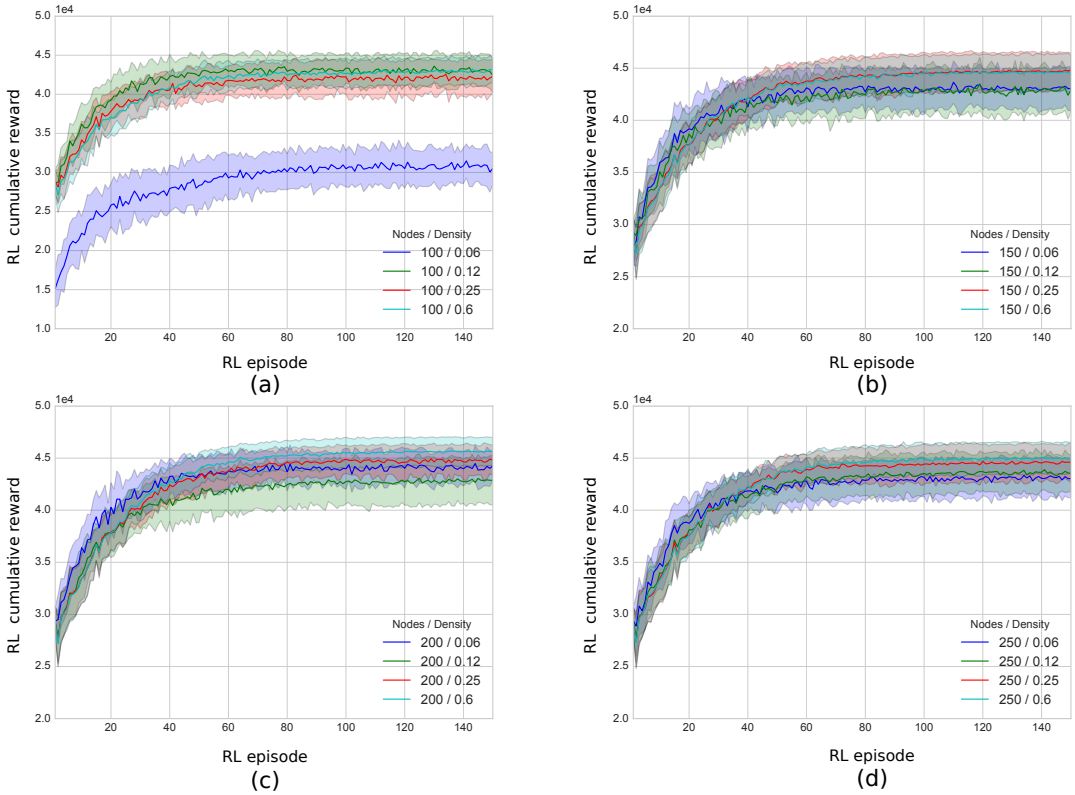


FIGURE 8 Figures (a) to (d) show the evolution of the average cumulative reward (+/- one standard deviation) during the SA-MTrigger learning process for 4 different network sizes and 4 different physical link densities.

9.2 | Comparison between periodic and self-adaptive triggering

In this section, we compare the performance of the SA-MTrigger with always and periodic VN migration triggering in terms of number of migration requests (Figure 7.a), migration trigger ratio (Figure 7.b), embedding acceptance ratio (Figure 7.c) and profit (Figure 7.d). The number of VN migration requests helped us to further assess the actual cost involved in migrations, beyond the migration trigger ratio.

Figure 7.a shows the average number of migration requests for the SA-MTrigger. Note that the number of VN migration requests for the SA-MTrigger was similar to the number of VN migration requests for the periodic triggers (triggering every 7, 8, 9 and 10 VNRs) even when the self adaptive trigger ratio was comparatively higher (Figure 7.b). On the other hand, Figure 7.d shows that with our SA-MTrigger the profit reached a value above 60% for the network with density of 0.06, being this value the highest profit observed among all triggers for that density.

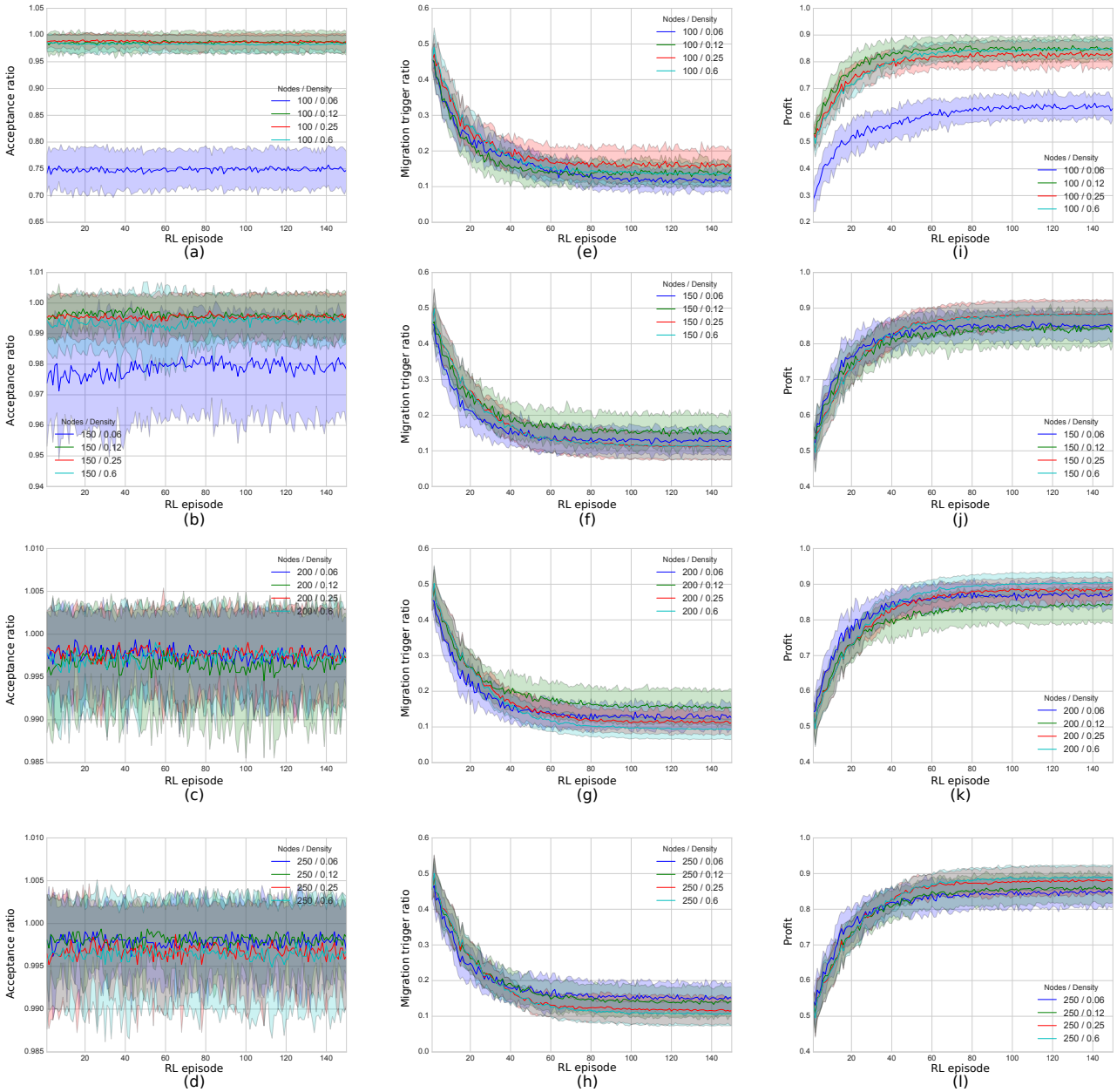


FIGURE 9 Figures (a) to (d) show the evolution of the average embedding acceptance ratio (+/- one standard deviation) during the SA-MTrigger learning. Figure (e) to (h) show the evolution of the average migration trigger ratio (+/- one standard deviation) during the SA-MTrigger learning process. Figures (i) to (l) the evolution of the average profit (+/- one standard deviation) during the SA-MTrigger learning process for 4 different network sizes and 4 different physical link densities.

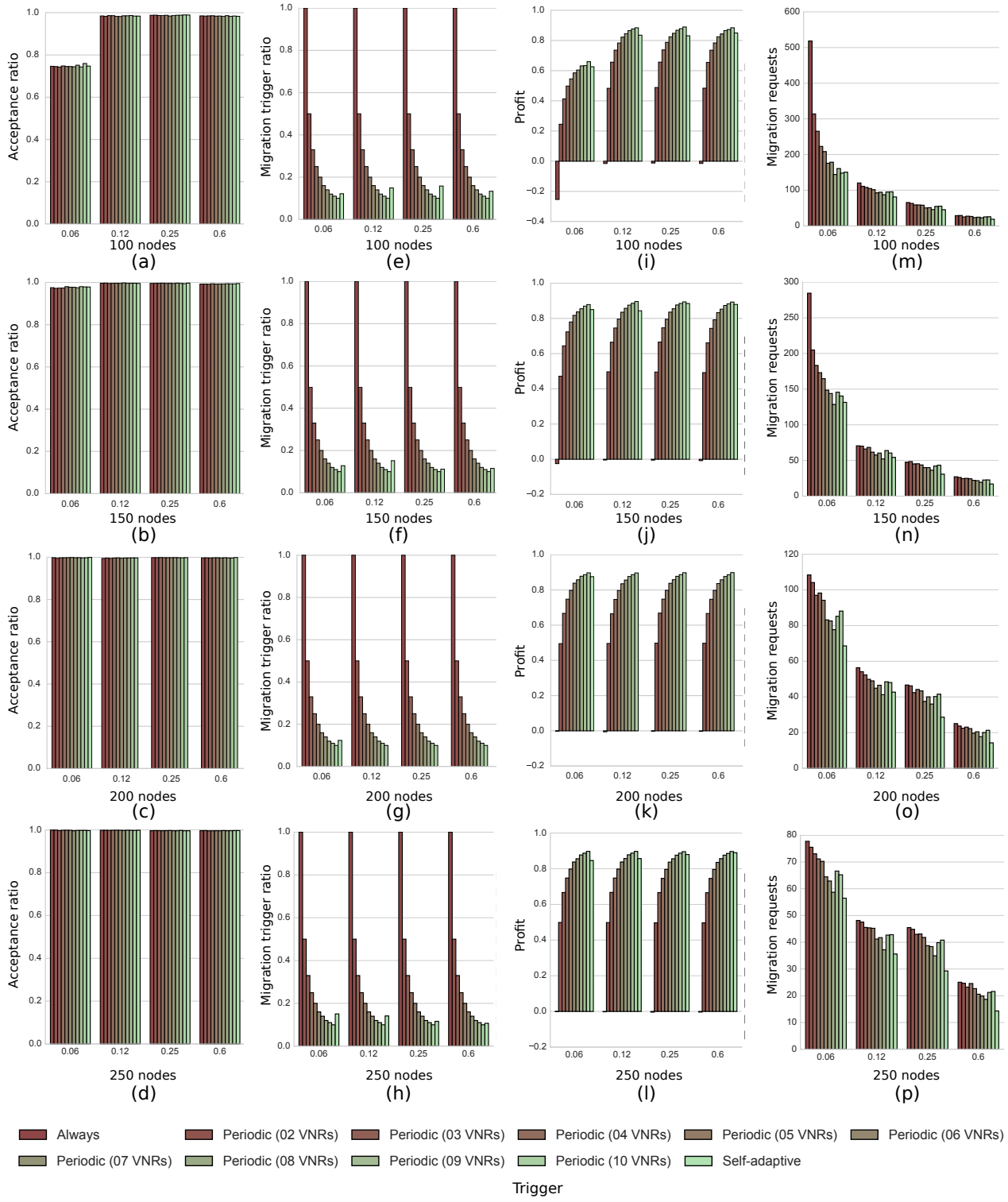


FIGURE 10 Figures (a) to (d) show the VNE acceptance ratio for self-adaptive VN vs. always VN migration triggering and periodic VN migration triggering. Figures (e) to (h) show migration trigger ratio for self-adaptive VN vs. always VN migration triggering and periodic VN migration triggering. Figures (i) to (l) show profit for self-adaptive VN vs. always VN migration triggering and periodic VN migration triggering. Figures (m) to (p) show the average number of VN migration requests for self-adaptive VN vs. always VN migration triggering and periodic VN migration triggering. Periodic triggers are set to trigger every 2, 3, 4, 5, 6, 7, 8, 9 and 10 VNRs. Values for the self-adaptive trigger are from the last RL episode.

10 | VALIDATION RESULTS FOR THE SELF-ADAPTIVE MIGRATION TRIGGERING (STUDY 2)

10.1 | Assessment of the RL process

Similarly to the previous study, for the sparsest physical networks (density of 0.06), the agent observed up to around 2,000 different states (around 19% of the total number of states), regardless of the network size. In contrast, for the densest networks (density 0.6), the agent observed less than 500 states (less than 4% of the total number of states), again regardless of the network size.

Figures 8.a to 8.d show the evolution of the cumulative reward as the RL process progressed for the different network sizes. It can be observed that the SA-MTrigger learned to increase the RL cumulative reward from around 3×10^4 to around 4×10^4, stabilizing between Episodes 60 and 80 regardless of the network size or density. The SA-MTrigger behaved differently only on the smallest and sparsest network (100 nodes, 0.06 density), going from cumulative rewards of 1.5×10^4 to rewards of around 3×10^4, as in Study 1.

Figures 9.a to 9.d show the rate of embedding acceptance ratio evolution. As the learning process progressed, the observed acceptance ratios were of around 75% for the smallest and sparsest network (100 nodes, 0.06 density), and of above 95% for the rest of the networks. A high link sparsity might have caused the smaller and sparsest network to break more often during experiments, negatively impacting the embedding acceptance ratio.

On the other hand, please note in Figure 9.b that the SA-MTrigger learned how to slightly improve the acceptance ratio for the sparsest 150-node physical network. The benefits of the RL approach are more evident in Figures 9.e to 9.h, which show the evolution of the migration trigger ratio. As the RL process progressed, the SA-MTrigger learned that it had to trigger only around 15% of the time for physical networks, regardless of their size. Particularly, the SA-MTrigger learned that it had to trigger only around 10% of the time for the synthetic physical networks with 200 and 250 nodes (higher densities). Figures 9.i to 9.l further show the benefits of our RL approach, showing the profit evolution with the RL episodes. As our agent learned to minimize the trigger ratio, the observed profits increased from 30% to around 60% for the smallest and sparsest network (100 nodes, density 0.06), and increased from 50% to around 85% for the rest of the physical networks.

10.2 | Comparison between periodic and self-adaptive triggering

Figures 10.e to 10.h show comparisons of the migration trigger ratio for the SA-MTrigger with the always VN migration and periodic VN migration triggering. With our SA-MTrigger the migration trigger ratio was reduced to about 10% to get an acceptance ratio of about 98% (see Figures 10.a to 10.d). This means that the SA-MTrigger learned that it had only to trigger between around 9% and 15% of the time. This VN migration reduction pattern was observed for all experimented physical networks.

Figures 10.m to 10.p show the comparison of the average number of VN migration requests for the SA-MTrigger with the always and periodic VN migration triggering.

Note that there were networks for which SA-MTrigger profits (Figures 10.i to 10.l) were not the best among all triggers, but for which the number of migration requests were the lowest. This is particularly noticeable on the 250-node networks (Figure 10.p). It can also be observed that the denser the physical network, the lower the number of VN migration requests (e.g. the number of VN migration requests for the denser physical network is less than 20).

11 | CONCLUDING REMARKS

VN migration is costly in terms of migration time, service disruption and resources utilization. Most of the current works on VN migration rely on the periodic migration of virtual resources. In this work we have presented the first attempt towards a cost-efficient self-adaptive online VN migration trigger, tested on different virtual network environment conditions, to address the critical question: "When to trigger VN migration?". One novel aspect of our contribution to the state of the art is that we formulated the problem as a Markov Decision Process problem and approached it by proposing a novel RL solution. Moreover, we introduced the concept of minimum BW of the maximum substrate ST in order to quantify the "strengh" of an SN and use it as a component of the state model of an RL approach. By increasing the physical network sizes and densities, the number of service requests and operational constraints are likely to increase exponentially[46]. For this reason, we have validated our methodology with a variety of physical network sizes and densities, demonstrating that the proposed self-adaptive online VN migration approach can serve as a key mechanism to reduce over- or under-utilization of substrate resources.

In the dynamic process of VN assignment, network conditions change over time due to the arrival and departure of VNs. As a result, VN assignment without migration may lead to inefficient resource utilization where some part of the substrate network can become excessively loaded while others are under-utilized. Migrating existing VNs can help to improve performance under dynamic situations. However, periodically migrating all existing VNs is not desirable for a number of reasons[3].

In this paper we described a novel cost-efficient self-adaptive VN migration trigger based on RL in order to learn the critical times when VN migration should be triggered to obtain profits in the long-term. Our SA-MTrigger self-adaptively learned to reduce the trigger ratios and the number of migration requests for the studied substrate networks. Moreover, it self-adaptively learned to improve the acceptance ratio for two of the experimented substrate networks.

Our algorithm also showed promising results in that the training process stabilized after only 60 episodes. In this regard, our algorithm learned a good policy quickly, over the first episodes. We believe that conducting longer trainings might lead to further reductions of trigger ratios in the experimented scenarios. However, a full separated study would be needed in order to derive the exact number of episodes that leads to the minimization of trigger ratios.

Our future work will be directed to implement our RL approach in a Software Defined Networking (SDN) architecture where we envision the SDN controller to plays the role of a global resource control[47] with three main components: 1) Online Virtual Network Embedding; 2) Online Virtual Network migration; and 3) Virtual and physical network monitoring. The SDN controller can use the embedding algorithms described in[48] to map VNs onto the physical infrastructure. As for online VN migration, the SDN controller can use the VLMP algorithm briefly described in Section 4 and in[2] in which the virtual path reconfigurations are driven by the VLMP migration process. Since resource allocation/reallocation is very challenging in the context of SDN, the proposed RL approach can be used to automate the resource reallocation process by proactive learning and interactions with the environment[46]. Our RL approach can self-adaptively decide which actions should the control plane take in order to minimize the costs of migrations while considering the VNE acceptance ratio. Based on the action received from the self-adaptive RL trigger, if VN migration is required, the SDN controller will trigger the VLMP algorithm described in Section 4 resulting in appropriate resource reallocations, keeping acceptable VNE acceptance rates, and minimizing the migration trigger ratio.

ACKNOWLEDGEMENTS

This paper has been supported by the National Council of Research and Technology (CONACYT) through grant FONCICYT/272278 and the ERANetLAC (Network of the European Union, Latin America, and the Caribbean Countries) project ELAC2015/T100761. This paper is partially supported also by project TEC2015-71329-C2-2-R (MINECO/FEDER) and the European Union's Horizon 2020 research and innovation programme under grant agreement No 777067 (NECOS project).

References

1. Zangiabady M, Aguilar-Fuster C, Rubio-Loyola J. A virtual network migration approach and analysis for enhanced online virtual network embedding. In: IEEE. Network and Service Management, 12th International Conference on; 2016: 324-329.

2. Zangiabady M. *A systematic approach to QoS-driven network migration in virtual network environments*. PhD dissertation. Centre for Research and Advanced Studies of the National Polytechnic Institute, 2018.

3. Zhu Y, Ammar MH. Algorithms for assigning substrate network resources to virtual network components. In: INFOCOM; 2006: 1–12.

4. Lo S, Ammar M, Zegura E, Fayed M. Virtual network migration on real infrastructure: A PlanetLab case study. In: IEEE. Networking Conference, 2014 IFIP; 2014: 1–9.

5. Masti SB, Raghavan SV. Simulated annealing algorithm for virtual network reconfiguration. In: IEEE. Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on; 2012: 95–102.

6. Liu L, Zheng S, Yu H, Anand V, Xu D. Correlation-based virtual machine migration in dynamic cloud environments. *Photonic Network Communications* 2016; 31(2): 206–216.

7. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: A survey. *Journal of artificial intelligence research* 1996; 4: 237–285.

8. Busoniu L, Babuska R, De Schutter B. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008* 2008.

9. Watkins CJ, Dayan P. Q-learning. *Machine learning* 1992; 8(3-4): 279–292.

10. Fischer A, Botero JF, Beck MT, De Meer H, Hesselbach X. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials* 2013; 15(4): 1888–1906.

11. Chowdhury M, Rahman MR, Boutaba R. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking (TON)* 2012; 20(1): 206–219.

12. Chowdhury NMK, Rahman MR, Boutaba R. Virtual network embedding with coordinated node and link mapping. In: IEEE. INFOCOM 2009, IEEE; 2009: 783–791.

13. Shanbhag S, Kandoor AR, Wang C, Mettu R, Wolf T. Vhub: Single-stage virtual network mapping through hub location. *Computer Networks* 2015; 77: 169–180. doi: 10.1016/j.comnet.2014.12.006

14. Cheng X, Su S, Zhang Z, et al. Virtual Network Embedding Through Topology-aware Node Ranking. *SIGCOMM Comput. Commun. Rev.* 2011; 41(2): 38–47. doi: 10.1145/1971162.1971168

15. Razzaq A, Rathore M. An Approach towards Resource Efficient Virtual Network Embedding. 2010; 0: 68-73.

16. Haeri S, Trajkovix0107 L. Virtual Network Embedding via Monte Carlo Tree Search. *IEEE Transactions on Cybernetics* 2018; 48: 510-521.

17. Yao H, Chen X, Li M, Zhang P, Wang L. A novel reinforcement learning algorithm for virtual network embedding. 2018; 284: 1-9.

18. Yu M, Yi Y, Rexford J, Chiang M. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review* 2008; 38(2): 17–29.

19. Mijumbi R, Gorricho JL, Serrat J, Claeys M, Turck FD, Latr S. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. *2014 IEEE Network Operations and Management Symposium (NOMS)* 2014: 1-9.

20. Mijumbi R, Gorricho JL, Serrat J, Claeys M, Famaey J, De Turck F. Neural network-based autonomous allocation of resources in virtual networks. In: European Conference on Networks and Communications (EUCNC); 2014: 1–6.

21. Mijumbi R, Gorricho J, Serrat J, Shen M, Xu K, Yang K. A neuro fuzzy approach to self-management of virtual network resources. 2015; 42: 1376–1390.

22. Yuan Y, Wang C, Wang C, Zhu S, Zhao S. Discrete particle swarm optimization algorithm for virtual network reconfiguration. In: Springer. International Conference in Swarm Intelligence; 2013: 250–257.

23. Fajjari I, Aitsaadi N, Pujolle G, Zimmermann H. VNR Algorithm: A Greedy Approach for Virtual Networks Reconfigurations. *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011* 2011: 1-6.

24. Miyazawa T, Kafle VP, Harai H. Reinforcement learning based dynamic resource migration for virtual networks. In: IEEE. Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on; 2017: 428–434.

25. Chowdhury SR, Ahmed R, Shahriar N, et al. Revine: Reallocation of virtual network embedding to eliminate substrate bottlenecks. In: IEEE. Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on; 2017: 116–124.

26. Tran PN, Timm-Giel A. Reconfiguration of virtual network mapping considering service disruption. In: IEEE. Communications (ICC), 2013 IEEE International Conference on; 2013: 3487–3492.

27. Tran PN, Casucci L, Timm-Giel A. Optimal mapping of virtual networks considering reactive reconfiguration. In: IEEE. ; 2012: 35–40.

28. Fajjari I, Saadi NA, Pujolle G, Zimmermann H. VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic. In: IEEE. Communications (ICC), 2011 IEEE International Conference on; 2011: 1–6.

29. Herker S, Khan A, An X. Survey on survivable virtual network embedding problem and solutions. In: International Conference on Networking and Services, ICNS; 2013.

30. Ahuja K, Singh B, Khanna R. Network selection based on available link bandwidth in multi-access networks. *Digital Communications and Networks* 2016; 2(1): 15–23.

31. Xu D, Qian D. A bandwidth adaptive method for estimating end-to-end available bandwidth. In: IEEE. Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on; 2008: 543–548.

32. Goldoni E, Rossi G, Assolo AT. A New Method for Available Bandwidth Estimation. In: Proceedings of Fourth International Conference on Internet Monitoring and Protection, ICIMP; 2009.

33. Oshiba T, Nakajima K. Quick end-to-end available bandwidth estimation for QoS of real-time multimedia communication. In: IEEE. Computers and Communications (ISCC), 2010 IEEE Symposium on; 2010: 162–167.

34. Liebeherr J, Fidler M, Valaee S. A system-theoretic approach to bandwidth estimation. *IEEE/ACM Transactions on networking* 2010; 18(4): 1040–1053.

35. Zhou A, Liu M, Song Y, Li Z, Deng H, Ma Y. A new method for end-to-end available bandwidth estimation. In: IEEE. Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE; 2008: 1–5.

36. Huang YC, Lu CS, Wu HK. Available bandwidth estimation via one-way delay jitter and queuing delay propagation model. In: IEEE. Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE; 2006: 112–121.

37. Demircin MU, Van Beek P. Bandwidth estimation and robust video streaming over 802.11 E wireless lans. In: IEEE. Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on; 2005: 1250–1253.

38. Korkmaz T, Krunz M. Multi-constrained optimal path selection. In: IEEE. INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings; 2001: 834–843.

39. Shani G, Brafman RI. Resolving perceptual aliasing in the presence of noisy sensors. In: Advances in Neural Information Processing Systems; 2005: 1249–1256.

40. Sutton RS. Reinforcement learning architectures. *Proceedings ISKIT* 1992; 92.

41. Heidrich-Meisner V. : Interview with Richard S. Sutton. In: . 3. Kunstliche Intelligenz; 2009: 41–43.

42. Tokic M. Adaptive ϵ-greedy exploration in reinforcement learning based on value differences. In: Springer. ; 2010: 203–210.

43. Chang XL, Mi XM, Muppala JK. Performance evaluation of artificial intelligence algorithms for virtual network embedding. *Engineering Applications of Artificial Intelligence* 2013; 26(10): 2540–2550.

44. Zhang Z, Cheng X, Su S, Wang Y, Shuang K, Luo Y. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *International Journal of Communication Systems* 2013; 26(8): 1054–1073.

45. Jiang Y, Lan J, Wang Z, Deng Y. Embedding and reconfiguration algorithms for service aggregation in network virtualization. *International Journal of Communication Systems* 2016; 29(1): 33–46.

46. Ma L, Zhang Z, Ko B, Srivatsa M, Leung KK. Resource management in distributed SDN using reinforcement learning. In: International Society for Optics and Photonics. Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX; 2018: 106350M.

47. Han P, Guo L, Liu Y. Virtual network embedding in SDN/NFV based fiber-wireless access network. In: IEEE. Software Networking (ICSN), 2016 International Conference on; 2016: 1–5.

48. Rubio-Loyola J, Aguilar-Fuster C, Toscano-Pulido G, Mijumbi R, Serrat-Fernández J. Enhancing Metaheuristic-Based Online Embedding in Network Virtualization Environments. *IEEE Transactions on Network and Service Management* 2018; 15(1): 200–216.