Design and Evaluation of Learning Algorithms for Dynamic Resource Management in Virtual Networks

Rashid Mijumbi*, Juan-Luis Gorricho*, Joan Serrat*, Maxim Claeys†, Filip De Turck† and Steven Latré‡
*Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
†Ghent University − iMinds, B-9050 Gent, Belgium
‡University of Antwerp − iMinds, B-2020 Antwerp, Belgium

***Abstract*—Network virtualisation is considerably gaining attention as a solution to ossification of the Internet. However, the success of network virtualisation will depend in part on how efficiently the virtual networks utilise substrate network resources. In this paper, we propose a machine learning-based approach to virtual network resource management. We propose to model the substrate network as a decentralised system and introduce a learning algorithm in each substrate node and substrate link, providing self-organization capabilities. We propose a multiagent learning algorithm that carries out the substrate network resource management in a coordinated and decentralised way. The task of these agents is to use evaluative feedback to learn an optimal policy so as to dynamically allocate network resources to virtual nodes and links. The agents ensure that while the virtual networks have the resources they need at any given time, only the required resources are reserved for this purpose. Simulations show that our dynamic approach significantly improves the virtual network acceptance ratio and the maximum number of accepted virtual network requests at any time while ensuring that virtual network quality of service requirements such as packet drop rate and virtual link delay are not affected.**

Keywords*—Network virtualization, Dynamic Resource Allocation, Virtual Network Embedding, Artificial Intelligence, Machine Learning, Reinforcement Learning, Multiagent Systems.

I. Introduction

Network virtualisation [1] has gained attention in the research community as a means of allowing for flexibility and innovation in the future Internet. It provides a mechanism for allowing multiple virtual networks (VNs) to share resources from one or more substrate networks (SNs). These resources - for any given VN - are completely isolated from the others, and appear as though they belong to different physical networks. VN operators can then lease these resources to other VNs, or use them to provide services to end users, allowing them to establish multiple specialised and flexible networks that are driven by end user requirements.

One key aspect in network virtualisation is the allocation of physical resources to VNs. This involves embedding VNs onto SNs, and managing of the allocated resources throughout the lifecycle of the virtual network. The virtual network embedding (VNE) problem involves embedding virtual nodes and links to substrate nodes and links respectively. The efficiency, optimality and flexibility of this resource allocation are fundamental factors for network virtualisation to be successful.

VNE is a well studied problem [1]. However, most current solutions perform static embeddings in that they do not consider the possibility of remapping or adjusting resource allocation to one of more virtual networks. Even approaches that propose dynamic virtual network embedding solutions still allocate a fixed amount of resources to the virtual nodes and links for their entire lifetime. As Internet traffic is not static, this could lead to an inefficient utilisation of overall network resources, especially if a substrate network rejects requests to embed new VNs while reserving the resources for VNs that are lightly loaded.

In this paper, instead of allocating a fixed amount of resources to a given VN throughout its lifetime, we dynamically and opportunistically allocate resources to virtual nodes and links depending on the perceived needs. The opportunistic use of resources involves carefully taking advantage of unused virtual node and link resources to ensure that VN requests are not rejected when resources reserved to already embedded requests are idle. To this end, we use a *demand-driven* dynamic approach that allocates resources to virtual nodes and links using reinforcement learning (RL) [2].

The contribution of this paper is two-fold: A distributed learning algorithm that allocates resources to virtual nodes and links dynamically and an initialisation scheme that biases the learning policy to improve the rate of convergence.

The rest of the paper is organised as follows: We present related work in Section II. Section III defines the dynamic resource allocation problem in the context of network virtualisation. Section IV briefly introduces reinforcement learning while our proposed RL approach is described in Section V. Section VI presents the evaluation of the proposed solution and a discussion of the results. The paper is concluded in Section VII.

II. Related Work

Many variants to the VNE problem have been proposed by different authors. Some of them such as [3] perform the node and link embedding in two uncoordinated steps, while [4] proposes a coordination between the two stages. A one-shot embedding solution based on a multiagent system (MAS) [5] is proposed in [6], while [7] and [8] propose mathematical programming based solutions to VNE. All these approaches propose a static allocation scheme in that once a given virtual network is mapped, the allocations are not altered for its entire lifetime.

There is a limited number of decentralised and dynamic solutions to VNE [1]. The authors in [9] and [10] study the VN embedding problem when the substrate network is dynamically changing. The approach in this paper differs from these works in that our consideration is on the changes in actual loading of the virtual networks, rather than on a

978-1-4799-0913-1/14/$31.00

changing substrate network. In [11], a solution that considers dynamic requests for embedding/removing virtual networks is presented. The authors map the constraints of the virtual network to the substrate network by splitting the requirements of one virtual link in more than one substrate link. On the other hand, the proposal in [12] is aimed at network survivability, performing re-embeddings in case of failures in the substrate network. Both approaches differ from the work in this paper in that our approach does not require changing virtual network embeddings. The authors in [13] propose a solution which aims at minimising the number of congested substrate links by carrying out link migrations. But this is a reactive solution since it is carried out only when an embedding strategy cannot assign a VN request in the SN. [14] proposes algorithms for the problem of efficiently re-configuring and embedding VN requests submitted to a cloud-based data center. The authors require that the ISPs submit new requests to modify existing ones, and that only one such request can be handled at a given time. In a related approach, [15] proposes a migration-aware dynamic virtual data center (VDC) embedding framework that also includes VDC scaling as well as dynamic VDC consolidation, while Butt et. al. [16] propose a topology-aware embedding that performs re-embeddings aimed at improving performance of previously embedded VNs. Our work differs from previous ones in that our resource re-allocations are proactive (not triggered by failed embeddings), autonomous (not triggered by either users or network providers) and do not involve any re-embeddings of already mapped requests.

Most existing works on dynamic resource management are based on three approaches: control theory, performance dynamics modeling and workload prediction. [17] and [18] are control theoretic approaches while [19] and [20] are based on perfomance dynamics. The authors in [21] and [22] use workload prediction. There are two major differences between these works and the work in this paper. The first is the use of multi-agent reinforcement learning while the other is based on the application domain. Dynamic resource management in virtual networks presents additional challenges as we have to deal with different resource types (such as bandwidth and queue size) which are not only segmented into many links and nodes, but also require different quality of service guarantees.

To summarise, the difference between our approach and the ones mentioned above is that, in our proposal, the resources reserved for use by the virtual nodes and links is not left unchanged throughout the entire lifetime of the virtual network. The virtual nodes and links are monitored, and based on their actual resource utilisation, resources are re-allocated, in which case un-used resources are returned to the substrate network for use by other virtual networks. We also note that unlike all the dynamic approaches in the state of the art, our approach does not involve the migration of virtual nodes and/or links.

III. Problem Description

The virtual network resource allocation problem is made up of two stages; VNE and dynamic resource management. As shown in Fig. 1, VNE involves embedding of VNs onto a SN and is initiated by a virtual network provider specifying resource requirements for both nodes and links to the substrate network provider. The specification of virtual network resource requirements can be represented by a weighted undirected graph denoted by $\hat{G}_v = (\hat{N}_v, \hat{L}_v)$, where $\hat{N}_v$ and $\hat{L}_v$ represent

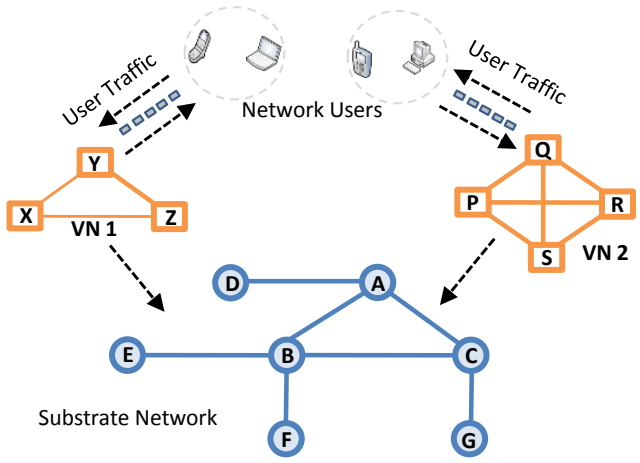

Fig. 1. Virtual Network Resource Allocation

the sets of virtual nodes and links respectively. Each virtual node $i \in \hat{N}_v$ has a queue size[1] $\hat{Q}_i$ and a location $\hat{P}_i(x, y)$ as well as a constraint on its location $\Delta\hat{P}_i(\Delta x, \Delta y)$ which specifies the maximum allowed deviation for each of its x and y coordinates, while each virtual link $\hat{l}_{ij} \in \hat{L}_v$ connecting the virtual nodes i and j has a maximum delay $\hat{D}_{ij}$ and bandwidth (data rate) $\hat{B}_{uv}$, In the same way, a substrate network can be modeled as an undirected graph denoted by $G_s = (N_s, L_s)$, where N_s and L_s represent the sets of substrate nodes and links, respectively. Each substrate link $l_{uv} \in L_s$ connecting the substrate nodes u and v has a delay D_{uv} and a bandwidth B_{uv}, while each substrate node $u \in N_s$ has queue size Q_u and a location $P_u(x, y)$.

The VNE problem involves the mapping of each virtual node $i \in \hat{N}_v$ to one of the possible substrate nodes with in the set $\Theta(i)$. $\Theta(i)$ is defined as a set of all substrate nodes $u \in N_s$ that have enough *available* queue size and are located within the maximum allowed deviation $\Delta\hat{P}_i(\Delta x, \Delta y)$ of the virtual node i. For a successful mapping, each virtual node must be mapped and any given substrate node can map at most one virtual node from the same request. Similarly, all the virtual links have to be mapped to one or more substrate links connecting the nodes to which the virtual nodes at its ends have been mapped. Each of the substrate links must have a sufficient data rate to support the virtual link. In addition, the total delay of all the substrate links used to map a given virtual link must not exceed the maximum delay specified by the virtual link. VNE is out of the scope of this work. Any of the static approaches [3] - [8] can be used for this stage[2].

The second stage – which is the focus of the work in this paper – follows a successful embedding of each VN, in which case the resources allocated/reserved for the embedded VN should be managed to ensure optimal utilisation of overall SN resources. For this work, we simulate the use of VN resources by transmitting user traffic in the form of packets over the virtual network. The characteristics of the user traffic used for this purpose are discussed in Section VI (A). By monitoring

[1]The queue size is a measure of the maximum number of packets (or Bytes) a given node can have in its buffer before dropping packets.

[2]In this paper, a mathematical programming formulation that performs both node and link embedding in one step, and solved using ILOG CPLEX 12.5 [24] is used to represent a static solution for the evaluations.

the actual use, the resources allocated to the VN are then dynamically managed. This is however performed carefully to ensure that quality of service parameters such as packet drop rate and delay for the VNs are not affected. We present our proposal for this purpose in Section V.

IV. Reinforcement Learning

RL is a technique from artificial intelligence [5] in which an agent placed in an environment performs actions from which it gets numerical rewards. For each learning episode [2], the agent perceives the current *state* of the environment and takes an *action*. The action leads to a change in the state of the environment, and the desirability of this change is communicated to the agent through a scalar *reward*. The agent's task is to maximise the overall reward it achieves throughout the learning period [2]. It can learn to do this over time by systematic trial and error, guided by a wide variety of *learning algorithms* [23]. One such learning algorithm is Q-learning. This is a temporal difference [2] learning algorithm that gradually builds information about the best actions to take in each possible state. This is achieved by finding a *policy* that maximises some long-term measure of reinforcement. A policy defines the learning agent's way of behaving at a given time. It is a mapping from perceived environment states to actions to be taken when in those states [2]. The action to be taken in a given state depends on the *Q-values* $Q(s,a)$ that are representative of the desirability of each action, a in that state, s. The learning process therefore involves continuously updating these values until they guide the agent to taking the best action while in any of the possible states [2]. Therefore, after every learning episode, an agent updates its Q-values using the Q-learning rule in (1).

$$Q(s_p,a_p) \leftarrow (1-\alpha)Q(s_p,a_p) + \alpha\left\{r_p + \lambda \max_{a\in\mathcal{A}} Q(s_n,a)\right\} \tag{1}$$

where $Q(s_p,a_p)$ is the new value of state s_p corresponding to action a_p, r_p is the reward obtained from taking the action a_p while in state s_p and s_n is the next state resulting from taking the action a_p while in state s_p, implying that $Q(s_n,a)$ is the value associated with the action a of the state s_n. The parameters $0 \leq \alpha \leq 1$ and $0 \leq \lambda \leq 1$ are referred to as learning rate and discount factor respectively. The value of α determines how fast learning occurs, while λ models the importance that is attached to future rewards in comparison to immediate rewards.

In general, there are many possible ways to *select actions* in RL. Two common action selection methods are ϵ-greedy and softmax. In ϵ-greedy, a greedy action is selected most of the time, and – using a small probability – a random action is chosen once in a while. This ensures that after many learning episodes, all the possible actions will be tried a high number of times, leading to an optimal policy. Softmax differs from ϵ-greedy in the way the random action is selected. A weight is assigned to each of the actions depending on their estimated values. A random action is selected based on the weight associated with it, ensuring that worst actions are unlikely to be chosen. When using softmax, an agent takes a random action a while in state s with a probability $\mathcal{P}(a|s)$ as defined in equation (2).

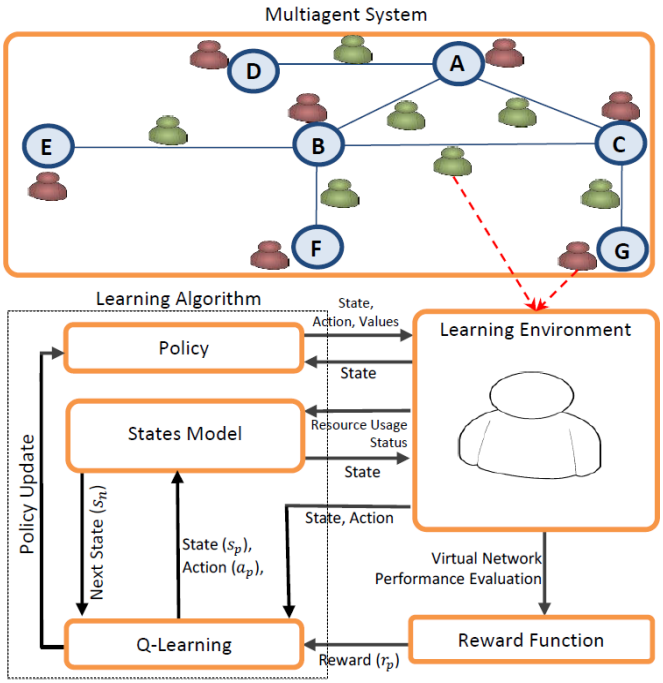

Fig. 2. Reinforcement Learning Model

$$\mathcal{P}(a|s) = \frac{\exp\{Q(s,a)/\tau\}}{\sum_{\hat{a}\neq a} \exp\{Q(s,\hat{a})/\tau\}} \tag{2}$$

where τ is a positive parameter called the temperature. High temperatures cause the actions to be almost equiprobable.

When more than one agent interact with each other, the resulting system is called a multiagent system (MAS) [5]. A detailed description of the modeling of the different aspects of reinforcement learning in the context of dynamic resource management in virtual networks is the subject of Section V.

V. RL Model For Dynamic Resource Allocation

Virtual network embedding allocates resources to virtual nodes and links based on the specification in the VN requests. Stopping at the embedding stage would result in a static allocation in which a fixed amount of substrate network resources is reserved for each virtual link and node irrespective of actual utilisation. This would lead to under utilisation in situations where the substrate network rejects new VN requests while the already embedded ones are lightly loaded. The approach proposed in this paper is to dynamically adjust the resource allocation using RL. To this end, we start by modeling the overall system showing the interaction of the different elements as shown in Fig. 2. The modeling mainly involves the learning environment, the learning algorithm, and a reward function to evaluate the effectiveness of the agents' learning.

A. Learning Environment

The learning environment consists of all the agents that represent the substrate network (the multiagent system). Specifically, each substrate node and link is represented by a node

TABLE I. ACTION DEFINITIONS AND VARIABLE STATES

(a)

Code	Percentage Value
000	$0 <$ Variable ≤ 12.5
001	$12.5 <$ Variable ≤ 25
010	$25 <$ Variable ≤ 37.5
011	$37.5 <$ Variable ≤ 50
100	$50 <$ Variable ≤ 67.5
101	$67.5 <$ Variable ≤ 75
110	$75 <$ Variable ≤ 87.5
111	$87.5 <$ Variable ≤ 100

(b)

Action	Description
A_0	Decrease allocated resources by 50.0 percent
A_1	Decrease allocated resources by 37.5 percent
A_2	Decrease allocated resources by 25.0 percent
A_3	Decrease allocated resources by 12.5 percent
A_4	Maintain Currently allocated resources
A_5	Increase allocated resources by 12.5 percent
A_6	Increase allocated resources by 25.0 percent
A_7	Increase allocated resources by 37.5 percent
A_8	Increase allocated resources by 50.0 percent

agent $n_a \in \mathcal{N}_a$ and a link agent $l_a \in \mathcal{L}_a$, where $\mathcal{N}_a$ and $\mathcal{L}_a$ are the sets of node agents and link agents respectively. The node agents manage node queue sizes while the link agents manage link bandwidths. The agents dynamically adjust the resources allocated to virtual nodes and links, ensuring that resources are not left underutilised, and that enough resources are available to serve user requests. We consider that each $n_a \in \mathcal{N}_a$ has information about the substrate node resource availability as well as the resource allocation and utilisation of all virtual nodes mapped onto the substrate node. In the same way, we expect that each $l_a \in \mathcal{L}_a$ has information about substrate link bandwidth as well as the allocation and utilisation of these resources by all virtual links mapped to it. In case a given virtual link is mapped onto more than one substrate link, then each of the $l_a \in \mathcal{L}_a$ agents coordinate to ensure that their allocations do not conflict.

B. Learning Algorithm

1) Policy: The policy is implemented by means of a lookup table which, for each state, maintains an updated evaluation of all the possible actions. Since we have 9 possible actions and 512 possible states (as explained in the next two subsections), the size of our policy is $9 \times 512 = 4608$ state-action values.

States: The state of any agent is a vector **S** with each term $s \in \mathcal{S}$ representing the state of one of the virtual links/nodes mapped onto it. The states in this work are discrete. We consider that the total resource demand of each virtual node or link can be divided into at least 8 resource chunks, each representing 12.5% of its total resource demand. For example, a virtual node could be allocated 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, 87.5% and 100% of its total demand. It is important to remark that these re-allocations are performed after a successful embedding. Therefore, all embeddings are performed based on the total demand of any given virtual node or link.

The state $s \in \mathcal{S}$ of any given virtual resource is represented by a 3-tuple, $s = \left(R_a,\, R_x^v,\, R_x^s\right)$, where R_a is the percentage resource allocation, R_x^v is the percentage unused virtual resources, and R_x^s is the percentage unused substrate resources. Each of the 3 variables is allowed to take up 8 different states, each made up of 3 bits, e.g., $[010]$. These values are based of the relationship between a current value and a benchmark, for example, if a virtual node is allocated between 37.5% and 50.0% of its total demand, then $R_a = 011$. The complete set of these variables is shown in Table I(a), which is valid for R_a, R_x^v and R_x^s. Therefore, each term of the state vector has 9 bits e.g. $(001, 100, 111)$, implying that we have $n = 2^9 = 512$ possible states.

Actions: The output of each agent is a vector **A** indicating an action $a \in \mathcal{A}$ for each of the virtual nodes/links mapped onto it. An agent can choose to increase or decrease the resources (queue size or bandwidth) allocated to any virtual node or link respectively. Specifically, as shown in Table I(b), at any point, each agent can choose 1 of the 9 possible actions, $a \in \mathcal{A} = (A_0, A_1, \ldots, A_8)$ each of which leads to a discrete change in resource allocation.[3]

2) States Model: The states model mimics the behaviour of the environment. When provided with a given status of the substrate and virtual networks resource allocation and utilisation levels i.e. the values R_a, R_x^v, and R_x^s, a states model returns a state $s \in \mathcal{S}$. In the same way, when provided with a given state $s_p = \left(R_{a_p},\, R_{x_p}^v,\, R_{x_p}^s\right)$ and an action a_p, the states model provides the next state $s_n = \left(R_{a_n},\, R_{x_n}^v,\, R_{x_n}^s\right)$. It is in general a model of the substrate and virtual network resources and how the different possible actions affect the allocation of substrate resources to virtual networks.

3) Q-Learning: In this paper, we propose a decentralised Q-learning based algorithm to iteratively approximate the state-action value, and then use these values to *select actions* for the allocation of substrate resources to the virtual nodes and links. As shown in algorithm 1[4] the learning algorithm is made up of three major steps; policy initialisation, policy update and action selection. We briefly describe each of these steps in the following subsections.

C. Reward Function

When an agent takes an action, the networks are monitored, recording the link delays, packet drops and virtual and substrate network resource utilisation so as to determine a reward. Specifically, the reward resulting from a learning episode of any agent is a vector **R** in which each term $r(v)$ corresponds to the reward of an allocation to the virtual resource[5] v, and is dependent on the percentage resource allocation R_a, the percentage resource utilisation R_u, the link delay $\hat{D}_{ij}$ in case of $l_a \in \mathcal{L}_a$ and the the number of dropped packets $\hat{P}_i$ in the case of $n_a \in \mathcal{N}_a$.

$$r(v) = \begin{cases} -100 & \text{if } R_a \leq 0.25 \\ \nu R_u - \left(\kappa \hat{D}_{ij} + \eta \hat{P}_i\right) & otherwise \end{cases}$$

Where ν, κ and η are constants aimed at adjusting the influence of the variables R_u, $\hat{D}_{ij}$ and $\hat{P}_i$ to the overall reward. In this paper, the values $\nu = 100$, $\kappa = 1000$ and $\eta = 10$ are used. These values have been determined through simulations, for example, by noting that the values of $\hat{P}_i$ are about 100 times more than those of $\hat{D}_{ij}$ (See Figs. 8 and 10). We therefore aim at scaling them to comparable magnitudes so that they can have the same effect on $r(v)$. $\hat{D}_{ij}$ and $\hat{P}_i$ are measures of

[3] In all cases, the percentage change is with respect to the total demand of the virtual node or link.

[4] We remark that this algorithm is slightly different from the "conventional" Q-learning algorithm [2] because instead of getting a reward immediately, in our case the reward of a given learning episode are used just before the following episode after a performance evaluation has been made.

[5] We use the term virtual resource to mean either a virtual node queue or virtual link bandwidth.

Algorithm 1 *Agent Learning Algorithm*

1: **POLICY INITIALISATION:**
2: **for** $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
3: Initialize the Q-table values $Q(s, a)$
4: **end for**
5: Determine current state s_c
6: previous state, $s_p = s_c$, previous action, $a_p = A_0$, next state, s_n.
7: **repeat**
8: Wait(Learning Interval)
9: **POLICY UPDATE:**
10: Read s_p, a_p, s_n
11: Observe Virtual Network Performance and Determine reward for previous action r_p.
12: Update the Q-Table using the equation (1)
13: **ACTION SELECTION:**
14: Determine current state, s_c.
15: Choose an action, $a_c \in \mathcal{A}$, for that state using a given *action selection criterion*
16: Take the action, a_c and determine next state s.
17: Set $s_p = s_c$, $a_p = a_c$, $s_n = s$
18: **until** Learning is stopped

the performances of link agents and node agents respectively. Therefore, for $n_a \in \mathcal{N}_a$, $\hat{D}_{ij} = 0$ while $\hat{P}_i = 0$ for $l_a \in \mathcal{L}_a$. The objective of the reward function is to encourage high virtual resource utilisation while punishing $n_a \in \mathcal{N}_a$ for dropping packets and $l_a \in \mathcal{L}_a$ for having a high delay. We also assign a punitive reward of -100 to resource allocations below 25% to ensure that this is the minimum allocation to a virtual resource and therefore avoid adverse effects to QoS in cases of fast changes from very low to high VN loading.

Policy Initialisation

Before learning can start, we need to initialise the learning policy. One possible approach is to assign random or constant values to all states and actions. However, since Q-learning requires all state-action pairs to be visited *at least once* so as to reach optimality, using random or constant initial values may lead to a slow convergence especially for a policy with many state-action values like we have in our approach. The idea is to start with a Q-table with values that more easily represent the expected actions of the agents. We therefore propose an initialisation approach that improves the rate of convergence. We initialise every possible state-action value using equation (3).

$$Q(s, a) = \frac{a}{\Psi} \times (s - 255) \tag{3}$$

Where Ψ is a constant aimed at scaling the $Q(s, a)$ values to the required ranges. The formula in equation (3) is based on observing that the free substrate and virtual resources increase as we move from state $(000, 000, 000)$ to $(111, 111, 111)$. Therefore, the rationale behind equation (3) is to generally bias the agents to increase resource allocation to the virtual network whenever it finds itself in a state closer to $(000, 000, 000)$ and reduce the allocation while in states closer to $(111, 111, 111)$. To this end, we represent each of the states $s \in \mathcal{S}$ with integers $[0, 511]$ and all the actions $a \in \mathcal{A}$ with integers $[0, 8]$. We then divide the total state space into two; such that while in states $[0 - 255]$ the agents in general allocate more resources

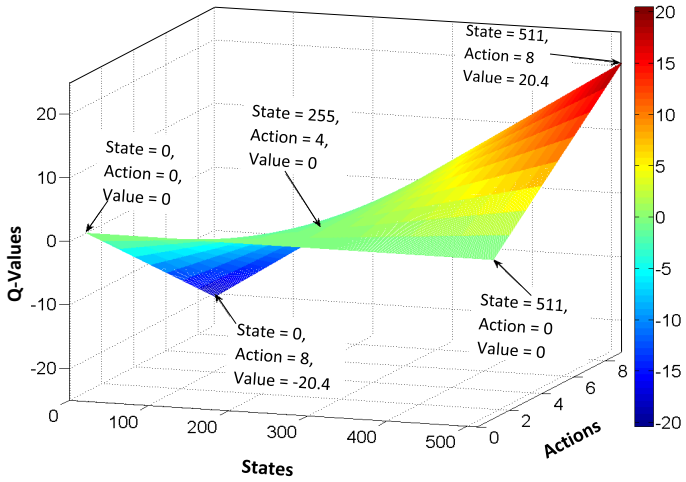

Fig. 3. Policy initialisation function

to the virtual network and then allocate less while in the states $[256 - 512]$. In Fig. 3, we show the different possible combinations with their respective values. As shown in the figure, for the same state $(000, 000, 000)$, action A_0 has a Q-value of 0 while action A_8 has -20.4. The evaluation of the proposed initialisation method is presented in Section VI.

Policy Update

The idea of learning is to gradually improve the policy until an optimal or near optimal policy is reached. This is achieved by updating the policy table after every learning episode. In this paper, the policy table is updated using the Q-learning equation (1).

Action Selection

An agent can select one out of the 9 possible actions. Since the suitability of any of the two action selection methods described in Section IV depends on the nature of the task, in this paper, we evaluate both of them with respect to our specific learning task, and their respective performances discussed in Section VI.

Time Complexity

We now formally analyse the time complexity of Algorithm 1. The initialisation step in Line 2 requires initilisation of the learning policy and can be solved in $O(|N_{s-a-v}|)$, where N_{s-a-v} is the number of state-action-values (4608 in this paper). Lines 5, 14 and 16 may each require iteration through all possible states in the worst case and can therefore be solved in $O(|\mathcal{S}|)$. Finally, the for loop in Line 15 runs in time $O(|\mathcal{A}|)$. Therefore, each episode of the proposed algorithm can be solved in linear time determined by the policy size.

Cooperation between Agents

Since a virtual link can be mapped to more than one substrate link, the agents $l_a \in \mathcal{L}_a$ that support the given virtual link must cooperate to avoid conflicting resource allocations. We accomplish this by allowing the agents to exchange messages. We consider that each agent $l_a \in \mathcal{L}_a$ maintains a

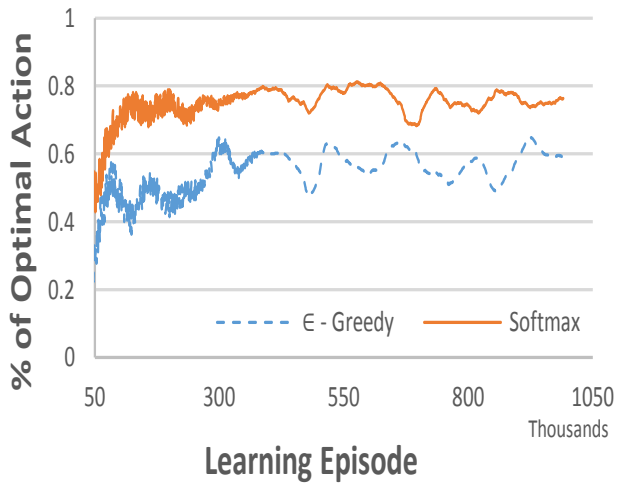

Fig. 4. Performance Comparison of ϵ-greedy and Softmax

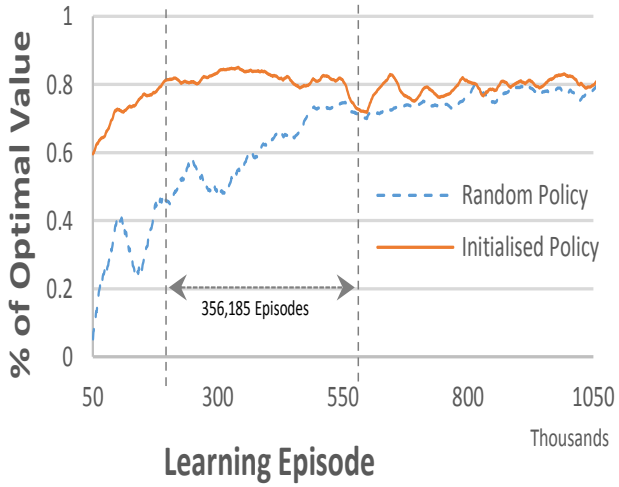

Fig. 5. Effect of biased policy initialisation

TABLE II. NS3 PARAMETERS

Parameter	Value
Queue Type	Drop Tail
Queue drop Mode	Bytes
Maximum Queue Size	6,553,500 Bytes
Maximum Packets Per VN	3500 Packets
Number of VNs	1024
Network Mask	255.255.224.0
IP Adress Range	10.0.0.0 – 10.255.224.0
Network Protocol	IPv4
Transport Protocol	TCP
Packet MTU	1518 Bytes
Packet Error Rate	0.000001 per Byte
Error distribution	Uniform (0, 1)
Port	8080

TABLE III. BRITE NETWORK TOPOLOGY GENERATION PARAMETERS

Parameter	Substrate Network	Virtual Network
Name (Model)	Router Waxman	Router Waxman
Number of nodes (N)	25	[5 – 10]
Size of main plane (HS)	250	250
Size of inner plane (LS)	250	250
Node Placement	Random	Random
GrowthType	Incremental	Incremental
Neighbouring Nodes	3	2
alpha (Waxman Parameter)	0.15	0.15
beta (Waxman Parameter)	0.2	0.2
BWDist	Uniform	Uniform
Minimum BW (BWMin)	2×10^6 bps	1×10^6 bps
Maximum Dev. (BWMax)	8×10^6 bps	1×10^6 bps

record of other agents $l'_a \in \mathcal{L}_a$ with which it is managing the resources of a given virtual link. This set of collaborating agents changes dynamically for each agent as new virtual networks are embedded and old ones leave. To ensure that the agents $l_a \in \mathcal{L}_a$ do not perform conflicting actions, only one of them learns at any given time. This is achieved by starting the learning processes of each agent at different times on their creation and thereafter performing learning at regular intervals. After each learning episode, if an agent $l_a \in \mathcal{L}_a$ needs to change an allocation, and the virtual link under consideration is mapped onto more than one substrate link, a message is sent to all the other affected substrate link agents $l'_a \in \mathcal{L}_a$ with information about the proposed allocation. This allows for a synchronised allocation of virtual link resources. This is reasonable since all agents belong to the same organisation (the SN) and learn the same policy; as they cannot have conflicting objectives. It would however be interesting to consider a more advanced cooperation protocol that allows for possibilities of agents accepting or rejecting proposals of other agents, which would be ideal in heterogeneous environments where the agents belong to different organisations and hence have different objectives.

Scalability: It is worth noting that in general, a virtual link is mapped to $2-3$ substrate links. This means that at any point, a given agent only needs to send update messages to about $1-2$ other agents. We consider that this number of update messages is manageable, and would not congest the network. In addition, since the communicating agents represent substrate links that are part of a simple substrate paths, they should be connected to each other, and hence the update messages are restricted to small regions even for big network sizes.

VI. PERFORMANCE EVALUATION

A. Simulation Setup

To evaluate the performance of the proposed approach, we added a network virtualisation module to NS3 [25]. Table II shows the NS3 parameters used in our simulations. The implemetation is such that every time a virtual network request is accepted by the substrate network, the virtual network topolgy is created in NS3, and a traffic application starts transfering packets over the virtual network. The traffic used in this paper is based on real traffic traces from CAIDA anonymised Internet traces [26]. This data set contains anonymized passive traffic traces from CAIDA's equinix-chicago and equinix-sanjose monitors on high-speed Internet backbone links, and is mainly used for research on the characteristics of Internet traffic, including flow volume and duration [26]. The trace source used in this paper was collected on 20th December 2012 and contains over 3.5Million packets. We divide these packets amoung 1000 virtual networks, so that each virtual network receives about 3500 packets. These traces are used to obtain packet sizes and time between packet arrivals for

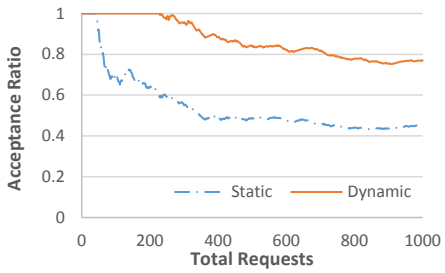

Fig. 6. VN Acceptance Acceptance Ratio

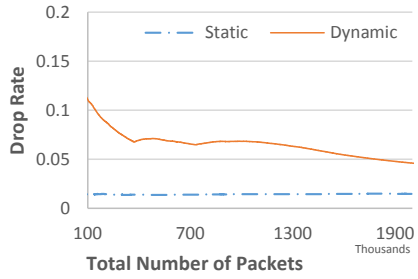

Fig. 7. Node Packet Drop Rate

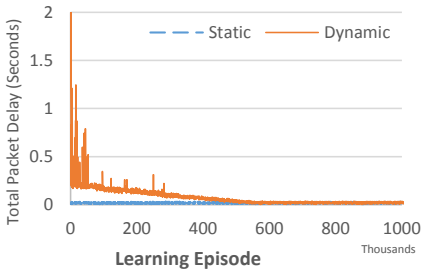

Fig. 8. Link Packet Delay

Fig. 9. Number of Accepted Virtual Networks

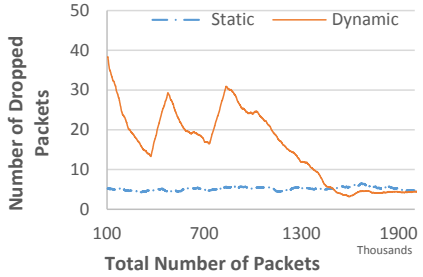

Fig. 10. Node Packet Drop Rate Variation

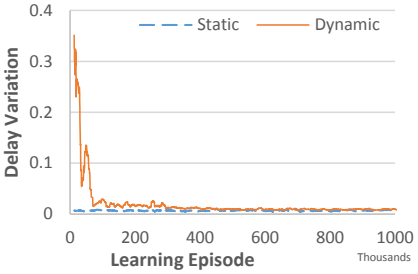

Fig. 11. Link Packet Delay Variation

TABLE IV. SUBSTRATE AND VIRTUAL NETWORK PROPERTIES

Parameter	Substrate Network	Virtual Network
Minimum Number of Nodes	25	5
Maximum Number of Nodes	25	10
Minimum Node Queue Size	(100×1518) Bytes	(10×1518) Bytes
Maximum Node Queue Size	(200×1518) Bytes	(20×1518) Bytes
Minimum Link Bandwidth	2.0Mbps	1.0Mbps
Maximum Link Bandwidth	10.0Mbps	2.0Mbps

each VN. As the source and destination of the packets are anonymised, for each packet in a given VN, we generate a source and destination IP address in NS-3 using a uniform distribution.

The substrate and virtual network topologies are generated using Brite [27] with settings shown in Table III. Simulations were run on an Ubuntu 12.04 LTS Virtual Machine with 4.00GB RAM and 3.00GHz CPU specifications. Both substrate and virtual networks were generated on a 25×25 grid. The queue size and bandwidth capacities of substrate nodes and links as well as the demands of virtual networks are all uniformly distributed between values shown in Table IV. Link delays are as determined by Brite. Each virtual node is allowed to be located within a uniformly distributed distance $7.5 \leq x \leq 15$ of its requested location, measured in grid units. We assumed that virtual network requests arrive following a Poisson distribution with an average rate of 1 per minute. The average service time of each virtual network is 60 minutes and is assumed to follow a negative exponential distribution.

B. Initial Evaluations

The initial evaluations are aimed at determining the appropriate action selection method for our task, as well as the effectiveness of the proposed policy initialisation scheme. Both of these evaluations are based on a comparison of agent actions with optimal actions. We define an optimal action for an agent as that action that would lead to a resource allocation equal to what the network is actually using. The deviations in these evaluations are therefore with reference to actual resource usage in a similar network that is not performing dynamic allocations.

Fig. 4 compares the performance of the action selection methods ϵ-greedy and softmax. It is evident that for this task, softmax performs better than ϵ-greedy. The difference in performance can be attributed to the fact that for ϵ-greedy, when random actions are chosen, the worst possible action is just as likely to be selected as the second best, yet softmax favours actions with better values. This could also explain why softmax actions appear to be relatively stable as compared to those by ϵ-greedy. In Fig. 5, we show the effect of the proposed initialisation method (action selection based on softmax). We observe that an initialised policy requires about $350,000$ learning episodes less to converge than a random policy. This can be attributed to the agents not having to explore all possible actions in all states as initialisation makes some actions more valuable than others.

For these evaluations as well as those in the next subsection the reinforcement learning parameters used are: learning rate, $\alpha = 0.8$, discount factor, $\lambda = 0.1$ and temperature, $\tau = 1$. We remark that based on the results of the evaluations in this subsection, the rest of the simulations in this paper are based on an initialised policy and the action selection method is softmax.

C. Performance Metrics

We evaluate the performance of our proposal on two fronts; the quality of the embeddings, as well as the quality of service to the virtual networks. The idea is that the opportunistic use of virtual network resources should not be at the expense of the service quality expectations of the network users.

1) Embedding Quality: This is evaluated using the acceptance ratio and total instantaneous accepted virtual networks. The acceptance ratio is a measure of the long term number of virtual network requests that are accepted by the substrate network. The total instantaneous accepted virtual networks is

a measure of the embedding cost incurred by a given substrate network, as a substrate network that incurs a lower embedding cost normally has more extra resources at any point and hence is able to have many embedded virtual networks at any point.

2) Quality of Service: We use the packet delay and drop rate as indications of the quality of service. We define the packet delay as the total time a packet takes to travel from its source to its final destination. The drop rate is defined as the ratio of the number of packets dropped by the network to the total number of packets sent. As shown in Table II, we model the networks to drop packets due to both node buffer overflow as well as packet errors. In addition, as it is more important in some applications, we define the variations of these two parameters. The jitter (delay variation) is defined as the difference between delays during different time periods, while the drop rate variation is defined as the variation between packet drops in different time periods. The time interval to update the measurements corresponds to the transmission of 50 packets.

D. Discussion of Results

The simulation results are shown in Fig. $6-11$. As can be seen from Fig. 6, the dynamic approach performs better than the static one in terms of virtual network acceptance ratio. This can be attributed to the fact that in the dynamic approach the substrate network always has more available resources than in the static case, as only the resources needed for actual transfer of packets is allocated and/or reserved for virtual networks. This is further confirmed by Fig. 9 which shows that at any given point a substrate network that dynamically manages its resources is able to embed more VNs than a static one.

Fig. 7 shows that the packet drop rate of the static approach is in general constant (due to packet errors as well as buffer overflows) while that of the dynamic approach is initially high, but gradually reduces. The poor performance of the dynamic approach at the start of the simulations can be attributed to the fact that at the beginning of the simulation when the agents are still learning, the virtual node queue sizes are allocated varying node buffers that lead to more packet drops. In fact, this initial number of packet drops affects the rate at which the overall drop rate reduces towards the one for the static approach. This can be confirmed by observing the actual periodic drops in packets as shown in Fig. 10 which show that the total number of packets dropped by both approaches is comparable towards the end of the simulation.

Similarly, Fig. 8 shows that the packets in the dynamic approach initially have higher delays than those in the static approach. Once more, the reason for this is the initial learning period of the agents. This is again confirmed by observing that the delay variations in Fig. 11 easily converge to those of the static approach. It is however worth noting that unlike the packet drop rate (Fig. 7), the actual delay (Fig. 8) of the dynamic approach finally converges to that of the static approach. Again, this could confirm that the slow convergence of the drop rate is due to the initial packet drops, since initial packets delays would not affect the delays of other packets, yet initial packet drops remain factors in the final drop rate.

We are however mindful that it could require a much higher number of learning episodes for the overall drop rate in Fig. 7 to finally converge to that of the static approach. This is because we used a learning policy with 4608 state-action values. With this high number of state-action values, the agents require a lot of time to learn an optimal policy. Moreover, it could improve the accuracy and precision of the agents' actions even more if the state-action values were increased. It would therefore be better to use function approximation or a more compact parametrised function representation to model the agents' policy other than a look-up table. We will investigate this approach more in the future.

VII. Conclusion

This paper has proposed a dynamic approach to the management of resources in virtual networks. We used a distributed reinforcement learning algorithm to allocate resources dynamically. We also proposed a method of initialising the learning policy that enhances the convergence rate of the learning algorithm. We have been able to show through simulation that our proposal improves the acceptance ratio of virtual networks, which would directly translate in revenue for the substrate network providers, while ensuring that, after the agents have learnt an allocation policy, the quality of service to the virtual networks is not negatively affected.

However, a number of future research directions can be considered. Implementing our proposed algorithm in real networks could pose more questions, e.g., the ease of having distributed network loading information, whether a dedicated network would be needed for communication between the agents, e.t.c. In future, we will study these issues and develop a prototype LAN where the agents are based on a real agent development platform. In addition, dynamic virtual network resource management in a multi-domain virtual network environment may raise more challenges since it may require a clear communication protocol, negotiations and agreements between competing agents that support inter-domain substrate paths. It could also be interesting to study the possible improvement in the agents' learning policy for example by using function approximation techniques such as artificial neural networks.

Acknowledgment

This work was partly funded by Flamingo, a Network of Excellence project (318488) supported by the European Commission under its Seventh Framework Programme. We also acknowledge support from the EVANS project (PIRSES-GA-2010-269323) as well as project TEC2009-14598-C02-02 from Ministerio de Ciencia y Educación and grant 2009-SGR-1242 from Generalitat de Catalunya.

References

[1] A. Fischer, J. Botero, M. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey", IEEE Communications Surveys Tutorials, pp. 1-19, 2013

[2] S. Sutton G. Barto, "Reinforcement Learning: An Introduction", Cambridge MA USA MIT Press, 1998.

[3] J. Lu, and J. Turner, "Efficient mapping of virtual networks onto a shared substrate", DCSE department, Washington University in St Louis, Technical Report, Vol.35, pp. 1-11, 2006.

[4] M. Chowdhury , M. Rahman and R. Boutaba, "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping", IEEE/ACM Transactions on Networking, VOL. 20, NO. 1, pp. 206-219, 2012

[5] S. J. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall, Englewood Cliffs, New Jersey, Third Edition, 2010.

[6] I. Houidi, W. Louati and D. Zeghlache, “A Distributed Virtual Network Mapping Algorithm”, IEEE International Conference on Communications, pp. 5634 - 5640, 2008.

[7] J. Infuhr, and G. R. Raidl, “Introducing the virtual network mapping problem with delay, routing and location constraints”, 5th international conference on Network optimization, Springer-Verlag, pp. 105 - 117, 2011.

[8] A. Jarray and A. Karmouch, “VCG auction-based approach for efficient Virtual Network embedding”, IFIP/IEEE International Symposium on Integrated Network Management, pp. 609-615, 2013.

[9] C. Marquezan, L. Granville, G. Nunzi, and M. Brunner, “Distributed autonomic resource management for network virtualization, IEEE Network Operations and Management Symposium (NOMS), pp. 463 470, 2010.

[10] Z. Cai, F. Liu, N. Xiao, Q. Liu, and Z. Wang, “Virtual network embedding for evolving networks, IEEE Global Telecommunications Conference, pp. 1 5, 2010.

[11] M. Yu , Y. Yi , J. Rexford and M. Chiang, “Rethinking virtual network embedding: Substrate support for path splitting and migration”, ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 17-29, 2008.

[12] M. R. Rahman and R. Boutaba, “SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization”, IEEE Transactions on Network and Service Management, VOL. 10, NO. 2, pp. 105-118, 2013.

[13] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “VNR algorithm: A greedy approach for virtual networks reconfigurations, IEEE Global Telecommunications Conference, pp. 1 5, 2011.

[14] G. Sun, H. Yu, V. Anand, L. Li, “A cost efficient framework and algorithm for embedding dynamic virtual network requests”, Future Generation Computer Systems 29, pp. 12651277, 2013.

[15] M. F. Zhani, Q. Zhang, G. Simon and R. Boutaba, “VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds”, Proc. of the IFIP/IEEE Integrated Network Management Symposium (IM). Ghent (Belgium), pp. 18 - 25, 2013.

[16] N. F. Butt, M. Chowdhury, R. Boutaba, “Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding”, Proc. of the 9th International IFIP Networking Conference (NETWORKING), Springer Berlin, pp. 27-39, Chennai (India), May 11-15, 2010.

[17] W. Pan, D. Mu, H. Wu, and L. Yao, “Feedback control-based QoS guarantees in web application servers, IEEE International Conference on High Performance Computing and Communications, pp. 328-334, 2008.

[18] T. Patikirikorala, A. Colman, J. Han, and L. Wang, “A multi-model framework to implement self-managing control systems for QoS management, 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 218-227, 2011

[19] R. Han, L. Guo, M. Ghanem, and Y. Guo, “Lightweight resource scaling for cloud applications, 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 644-651, 2012.

[20] W. Lai, M. Chiang, S. Lee, T. Lee, “Game Theoretic Distributed Dynamic Resource Allocation with Interference Avoidance in Cognitive Femtocell Networks”, IEEE Wireless Communications and Networking Conference, pp. 3364-3369, 2013.

[21] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, “Resource provisioning for cloud computing,” Conference of the Center for Advanced Studies on Collaborative Research, pp. 101111, 2009.

[22] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, J. Lilius, “Prediction-Based Dynamic Resource Allocation for Video Transcoding in Cloud Computing”, 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 254-261, 2012.

[23] L. P. Kaelbling, M. L. Littman and A. W. Moore, “Reinforcement Learning: A survey”, Journal of Artificial Intelligence Research 4, pp. 237-285, 1996.

[24] IBM, “IBM ILOG CPLEX Optimizer”, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/about/

[25] NS-3 Consortium, “Network Simulator 3”, http://www.nsnam.org/

[26] The CAIDA Anonymized Internet Traces 2012 - 20 December 2012, equinix sanjose.dirB.20121220-140100.UTC.anon.pcap.gz, http://www.caida.org/data/passive/passive_2012_dataset.xml

[27] A. Medina, A. Lakhina, I. Matta, J. Byers, “BRITE: An Approach to Universal Topology Generation”, 9th IEEE International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 346-353, 2001.