Neural Network-based Autonomous Allocation of Resources in Virtual Networks

Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat
Universitat Politècnica de Catalunya,
08034 Barcelona, Spain

Maxim Claeys, Jeroen Famaey, Filip De Turck
Ghent University - iMinds,
B-9050 Gent, Belgium

***Abstract*—Network virtualisation has received attention as a way to allow for sharing of physical network resources. Sharing resources involves mapping of virtual nodes and links onto physical nodes and links respectively, and thereafter managing the allocated resources to ensure efficient resource utilisation. In this paper, we apply artificial neural networks for a dynamic, decentralised and autonomous allocation of physical network resources to the virtual networks. The objective is to achieve better efficiency in the utilisation of substrate network resources while ensuring that the quality of service requirements of the virtual networks are not violated. The proposed approach is evaluated by comparison with two static resource allocation schemes and a reinforcement learning-based approach.**

Keywords*—Artificial neural networks, network virtualisation, resource allocation, reinforcement learning, autonomous systems.

I. Introduction

Network Virtualisation [1] allows substrate network (SN) owners to lease out part of their infrastructure as a service to service providers who create virtual networks (VN) to provide end-to-end services to end-users. A VN is made up of a set of virtual links and nodes which are supported by SN physical paths and nodes respectively. Efficient sharing of SN resources amoung VNs can be achieved in two steps [2]. The first, known as virtual network embedding (VNE) [1], involves mapping of virtual nodes and links to substrate nodes and paths, subject to a set of pre-defined constraints (e.g. topology, node queue size and bandwidth). The second step dynamically manages the allocation of resources to virtual nodes and links throughout the lifetime of a VN.

While many approaches have been proposed for management of resources in virtual networks, the number of decentralised and dynamic solutions is still limited [1]. In a previous work [3], we proposed a decentralised scheme for dynamic resource allocation (DRA) in VNs based on reinforcement learning (RL) [4] and a look-up table based policy. Since a look-up table representation suffers from the curse of dimensionality [5], the state and action space was discretised to limit the size, as well as the high memory for reading, writing and storage of the learning policy. This however comes at a cost of efficiency, as the learning algorithm is constrained in terms of perception and action granularity.

In this paper, we improve the efficiency of [3] by proposing an autonomous system based on artificial neural networks (ANN) [6] to achieve an adaptable allocation of resources to virtual networks, without restricting the input-output space dimensions. We start by representing each substrate node and link as an ANN whose input is the network resource status and the output an allocation action. We then use a reinforcement-like error function to evaluate the desirability of ANN outputs, and hence perform online training of the ANN.

The rest of the paper is organised as follows: We present related work in Section II. Section III gives a brief theoretical background on the resource management problem in network virtualisation, and ANNs. The proposed approach is presented in Section IV and evaluated in Section V. Finally, Section VI concludes the paper, giving an outlook for future work.

II. Related Work

A comprehensive survey on the state-of-the-art in VNE can be found in [1]. Most of the approaches perform a static embedding without any considerations for possibilities of adjustments to initial embeddings, while those that propose dynamic solutions do allocate a fixed amount of node and link resources to the VNs throughout their life time. Since network load varies with time due to non-uniform user traffic, allocating a fixed amount of resources based on peak load could lead to an inefficient utilisation of overall SN resources, especially during periods when the virtual nodes and/or links are lightly loaded.

Existing work on DRA is based on three main approaches: control theory, performance dynamics modelling and workload prediction. For example, [7] is a control theoretic approach, [8] is based on performance dynamics, while [9] uses workload prediction. The difference between our proposal and these works is not only with respect to the solution tool (ANN), but also in application domain (network virtualisation). DRA in VNs 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. In addition, in a VN environment, the managed resources are dependent on each other, for example, a given virtual link can be mapped on more than one substrate link, and the resources allocated to a virtual node may affect the performance of virtual links attached to it, say in terms of increased routing delays.

A combination of ANNs and RL has been applied to many problems such as [10], [11], [12]. In these proposals, ANNs are used as function approximators for the RL policy. The proposal in this paper differs from these works on two fronts: (1) we use RL to train the ANN rather than using ANNs to approximate the RL policy. This, remarkably, allows us the possibility to do away with the need for training examples and/or target outputs

usually needed for learning in neural networks[1], and (2) we apply the combination ANN and RL to a network virtualisation environment.

III. Theoretical Background

This Section introduces the two main steps−virtual network embedding (VNE) and dynamic resource allocation (DRA)−involved in resource management in VNs. We also introduce artificial neural networks (ANN).

A. Virtual Network Embedding (VNE)

VNE involves mapping of VNs onto a SN, and is initiated by a service provider (SP) specifying resource requirements for both nodes and links to an infrastructure provider. The specification of VN resource requirements is usually represented by a weighted undirected graph $G_v = (N_v, L_v)$, where N_v and L_v represent the sets of virtual nodes and links respectively. Each virtual link $l_{ij} \in L_v$ or virtual node $i \in N_v$ usually has requirements such as maximum delay, CPU, queue size, bandwidth etc. In a similar way, the SN node and link capacities can be represented. For a successful mapping, all the VN node and link mappings should be in accordance to the VNE constraints [13]. VNE is out of the scope of this work. Any of the static approaches in [1] can be used for this stage.

B. Dynamic Resource Allocation (DRA)

The next step, which is the focus of this paper, follows a successful VNE. It involves the lifecycle management of resources allocated/reserved for the mapped VN, and is aimed at ensuring optimal utilisation of overall SN resources. Our consideration is that SPs reserve resources to be used for transmitting user traffic, and therefore, after successful mapping of a given VN, user traffic is transmitted over the VN. Actual usage of allocated resources is then monitored and based on the level of utilisation, we dynamically and opportunistically adjust allocated resources. The opportunistic use of resources involves carefully taking advantage of unused virtual node and link resources to ensure that other VN requests are not rejected when resources reserved to already mapped VNs are idle. It is however a delicate balancing approach that ensures that while VNs should not have idle resources, the allocated resources are sufficient to ensure that the quality of service parameters such as packet drop rate and delay for the VNs are not affected. In Section IV, we detail the proposed ANN approach.

C. Artificial Neural Networks (ANN)

ANNs are collections of computing nodes known as *neurons* which operate as summing devices, interconnected by *links* [14]. A neuron receives one or more inputs, which are first multiplied by *weights* along each link, and then *summed* to produce an output. The output is then passed through an activation function (such as the logistic function [6]), which determines the input-output behaviour of the neuron. In ANNs, neurons are arranged in layers, with each layer consisting of one or more neurons. The most commonly used structure of ANNs is made up of 3 layers; an input layer, a hidden layer, and an output layer [14]. The learning ability of neural networks lies in their ability to adjust their weights. This is achieved by gradually minimising an *error*, defined as the difference between an actual output and a target output. The most popular method for learning in ANNs is called back-propagation (BP) [14]. In BP, after an output is obtained, an *error signal* - which is the difference between actual output and target output - is determined. The error signal is then "propagated backwards" from the output layer to the input layer, adjusting the network weights. Therefore, learning in ANNs requires that for every input, a target output must be known so as to determine the error. An introduction to ANNs, and the back propagation algorithm (and how it is used for learning in ANNs) can be found in [14].

[1]While we still use some training examples in our proposal (see Section IV-B3), it is only aimed at guiding in the ANN structure design as well as ensuring a faster convergence of the algorithm (through problem specific weight initialisation) rather than as a requirement as would have been in a typical ANN learning scenario.

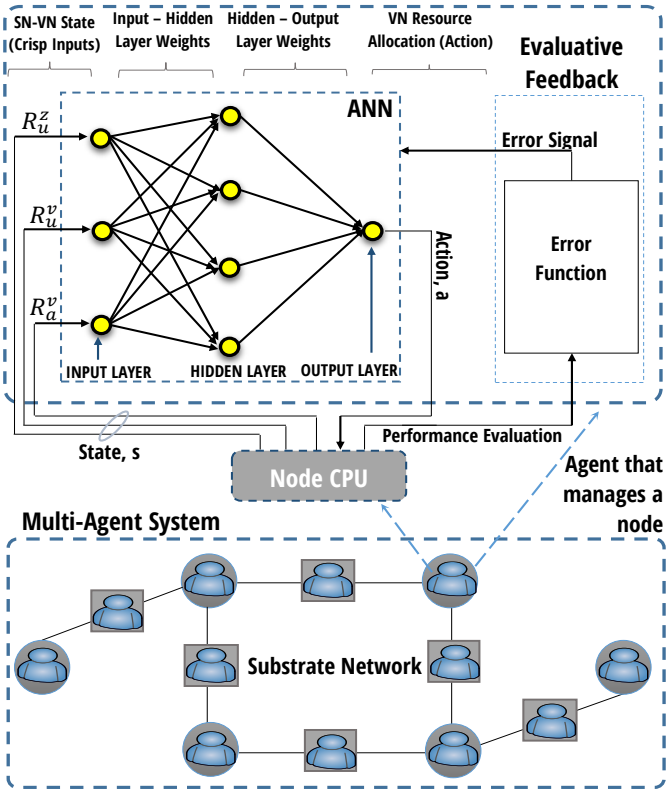

Fig. 1. Artificial Neural Network-based Resource Allocation Model

IV. ANN-based Dynamic Resource Allocation

The system model used for our proposal is shown in Fig. 1. As can be observed from the figure, there are three main components: the multi-agent system representing the substrate network, the ANN that represents the internal components of each agent, and the evaluative feedback block that produces the error signal. In the following subsections, each of these elements of the model is detailed.

A. Multi-Agent System

The multi-agent system consists of all the agents that represent the SN. Specifically, each substrate node and link

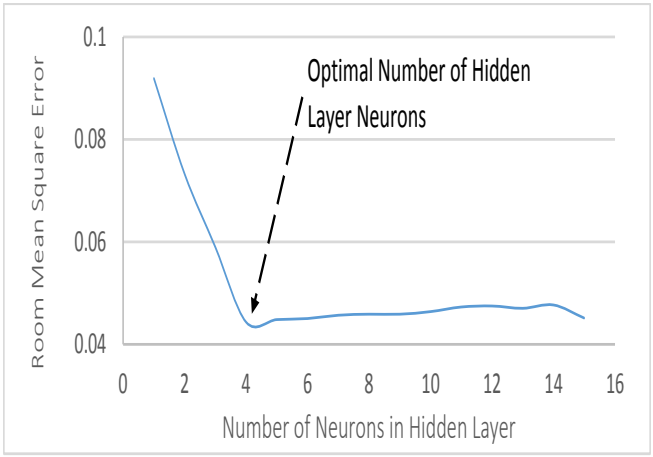

Fig. 2. Variation of RMSE with Number of Neurons in Hidden Layer

is represented by a node 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 under-utilised, and that enough resources are available to meet VN requirements. As shown in Fig. 1, a given agent receives as input the state, s of the substrate node it manages, and outputs an action, a.

B. Artificial Neural Network

Our proposal uses a 3-layer ANN. An important design issue of any ANN is determining network topology, i.e. number of neurons in each of the network layers.

1) Input Layer: We model the state of any virtual resource (node queue size or link bandwidth) v hosted on a substrate resource z, by a 3-tuple, $s = (R_a^v,\ R_u^v,\ R_u^z)$, where R_a^v is the percentage of the virtual resource demand currently allocated to it, R_u^v is the percentage of allocated resources currently unused, and R_u^z is the percentage of total substrate resources currently unused. Therefore, the input layer consists of 3 neurons, one for each of the variables R_a^v, R_u^v and R_u^z.

2) Output Layer: During each learning episode, an agent should perceive the state s and give an output. This output, a, is a scalar that indicates which action should be taken to change the resource allocation for the virtual resource v under consideration. The action may be aimed at increasing (if it is positive) or reducing the resources allocated to any virtual node or link respectively. Therefore, the output layer consists of 1 neuron, representing the action, a. To illustrate the effect of an action, if a given virtual resource, v has total allocated resources v_r and the agent action is a (where $-1 \leq a \leq +1$), then the resulting resource allocation is: $v_r = v_r + a \times v_t$, where v_t is the total initial demand of the virtual resource (as specified in the VN request before the VNE).

3) Hidden Layer: The optimal number of neurons in a hidden layer of any ANN is problem specific, and is still an open research question [6]. In this paper, we determine this number by experimentation. We perform a search from number of hidden layer neurons, $N_{HL} = 1$ to 15.

In order to achieve this, we need a test dataset. The dataset used for this purpose was saved from the q-table of a RL approach proposed in [3]. This q-table was a result of a learning system for a similar DRA task and it gives the state-action-values for the learning task. This dataset contains 512 entries, each showing the best action value in each of the possible 512 states.

With the above training set, a $10-$fold cross-validation was performed in Weka 3.6 [15], using the default parameters (learning rate, validation threshold, momentum, etc.) for the multilayer perceptron in Weka. Fig. 2 shows the average root mean square error (RMSE) values for 20 experiments. From the figure, the optimal number of neurons for the hidden layer is 4. The reason for choosing to use 4 neurons is not only to allow for a low RMSE, but also to avoid a possibility of over-fitting which could be caused by a network with many neurons. This experimentation also provides initial weights that are used to initialise the neural network, and hence avoid the slow learning characteristics (and hence slow convergence) of BP.

C. Evaluative Feedback

After each learning episode, the affected substrate and virtual nodes/links are monitored, taking note of average utilisation of substrate resources, the delay on virtual links and packets dropped by virtual nodes due to buffer overflows. These values are fed back to the agent in form of a *performance evaluation*, used by the error function to produce an error signal, which is used by the BP algorithm to adjust the weights of the ANN, and hence improve future actions.

Error Function: The error $e(v)$, is an indication of the deviation of the agent's actual , from a target action. The objective of the error 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 high delays. Good actions by an agent are characterised by an $e(v)$ equal or close to 0, while any deviations indicate undesirable actions. Therefore, the value of $e(v)$ gives the degree of desirability or undesirability the agent's action, and is dependent on resources allocated to the virtual resources, unutilised resources, link delay in case of $l_a \in \mathcal{L}_a$ and the number of dropped packets in the case of $n_a \in \mathcal{N}_a$. The proposed error function is shown in (1).

$$e(v) = \begin{cases} \left(R_u^v + \alpha P_v\right) & \forall n_a \in \mathcal{N}_a \quad (1a) \\ \left(R_u^v + \beta D_v\right) & \forall l_a \in \mathcal{L}_a \quad (1b) \end{cases}$$

where α and β are constants aimed at ensuring that the magnitudes of the two terms in each of (1a) and (1b) are comparable. The values $\alpha = 0.05$ and $\beta = 40$ used in this paper, were determined by simulations. For example, looking at Fig. 5 shows that the maximum value of P_v is about 20. Therefore, to make these values comparable to $0 \leq R_u^v \leq 1$, we divide each value by 20 (multiply it by $\alpha = 0.05$). P_v is the number of packets dropped by node $n_a \in \mathcal{N}_a$ from the time the allocation action was taken, and D_v is the $extra$ delay encountered by a packet using $l_a \in \mathcal{L}_a$. The extra delay is calculated as the difference between actual delay and the theoretical delay. We define theoretical delay as the delay

TABLE I. NETWORK TOPOLOGY PARAMETERS

Parameter	Substrate Network	Virtual Network
Name (Model)	Router Waxman	Router Waxman
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

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. SN AND VN PROPERTIES

Parameter	Substrate Network	Virtual Network
Minimum Number of Nodes	25	5
Maximum Number of Nodes	35	15
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

TABLE IV. COMPARED ALGORITHMS

Code	Resource Allocation Approach
D-ANN	Dynamic, based on Artificial Neural Networks [Our Contribution]
D-RL	Dynamic, based on Reinforcement Learning[3]
S-CNMMCF	Static, Coordinated Node Mapping and MCF for link mapping[13]
S-OS	Static, link based optimal one shot Virtual Network Embedding[3]

the virtual link would have if it was allocated 100% of its bandwidth demand[2]. The actual delay is determined as the difference between when a packet is received at one end of the link, to when it is delivered to the other end. Once the error is determined, the ANN weights are adjusted using BP.

V. PERFORMANCE EVALUATION

A. Simulation Environment

To evaluate our proposal, SN and VN topologies were generated using Brite [16] with settings shown in Table I. Thereafter, VN requests arrive, one at a time to the SN. Whenever a VN request is accepted by the SN, the VN topology is created in NS3 [17] (we added a network virtualisation module to NS3 based on parameters in Table II). Our NS3 module allows us to create a traffic application for each accepted VN request, and the traffic application starts transferring packets over the VN. The traffic application generates packets based on real traffic traces from CAIDA anonymised Internet traces [18]. This dataset contains anonymised 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 [18]. 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 VNs, so that each VN receives about 3500 packets. These traces are used to obtain packet sizes and time between packet arrivals for 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. Simulations were run on an Ubuntu 12.04 LTS Virtual Machine with 4.00GB RAM and 3.00GHz CPU specifications.

[2]The simulations in this paper determine this value from a parallel simulation using a virtual network with 100% resource allocation.

B. Simulation Parameters

Both substrate and virtual networks were generated on a 250×250 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 minimum and maximum values shown in Table III. Link delays are as determined by Brite. Each virtual node is allowed to be located within a uniformly distributed distance $75 \leq x \leq 150$ of its requested location, measured in grid units. We assumed that VN requests arrive following a Poisson distribution with an average rate of 1 per minute. The average service time of each VN is 60 minutes and is assumed to follow a negative exponential distribution. The ANN algorithm runs every minute[3].

C. Compared Algorithms

We compare the performance of our proposal with 3 representative state-of-art solutions. The first, [3], uses RL for DRA; the second performs a coordinated node and link mapping [13]; and the third is also a static baseline formulation that performs a one shot mapping, and also used in performance evaluations in [3]. The solution in [13] was adapted to fit into our formulation of the problem. In particular, for [13] the link delay requirements were neglected at the embedding stage, and for this reason, it is not used in QoS evaluations. In addition, our consideration in this paper is for unsplittable flows. We identify and name the compared approaches in table IV. The mathematical programs in all proposals are solved using CPLEX 12.5 [19].

[3]It is worth remarking that while this paper has not studied the effect of the frequency of running the algorithm, we expect that a lower running frequency would make the dynamic allocation become comparable to the static one, while a higher frequency might negatively impact system stability.

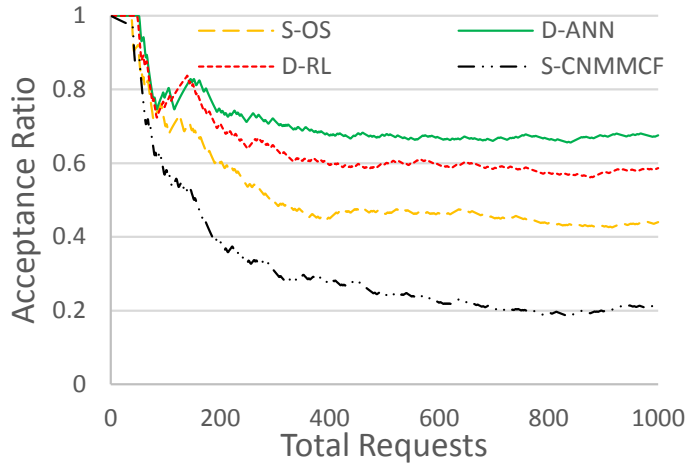

Fig. 3. Acceptance Ratio

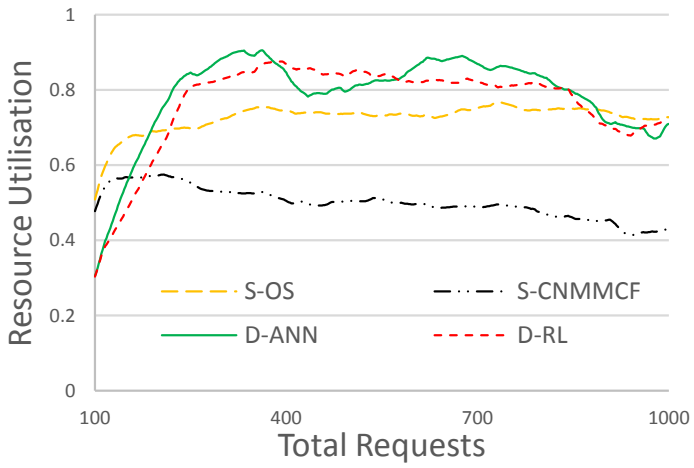

Fig. 4. Resource Utilisation

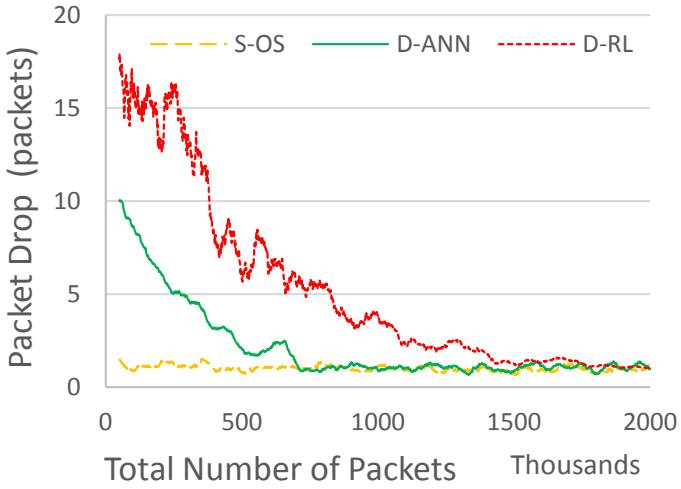

Fig. 5. Number of Dropped Packets

Fig. 6. Delay Variation

D. Performance Metrics

1) Embedding Quality: We define embedding quality as a measure of how efficiently the algorithm uses the SN resources for accepting VN requests. This is evaluated using the acceptance ratio and the average level of utilisation of SN resources. The acceptance ratio is a measure of the long term number of VN requests that are accepted by the substrate network. The average level of utilisation of substrate resources is a measure of how efficiently the SN resources are used.

2) Quality of Service: We use both packet drop a well as delay variation as indications of the quality of service. As shown in Table II, we model the networks to drop packets due to both node buffer overflow as well as packet errors. We determine the packet delay variation as the difference in delays encountered by packets transmitted over the network over two successive time intervals, while packet drop is the number of packets dropped by a given VN during a given time interval. For these evaluations, the time interval used to update these measurements corresponds to the transmission of every 100 packets.

E. Discussion of Results

The simulation results are shown in Figs. 3 – 6. As can be seen from Fig. 3, while both dynamic approaches perform better than the static ones in terms of VN acceptance ratio, the ANN approach outperforms all three. The reason for the dynamic approaches performing better than the static ones is that in former cases, the substrate network always has more available resources than in the later case, which is a direct result of allocating and reserving only the required resources for the virtual networks. The fact that ANN outperforms the RL approach can be attributed to the fact that the ANN approach models the states and actions with better granularity i.e. without restricting the states and actions to few discrete levels. We also note that S-OS has a better acceptance ratio than S-CNMMCF. This is due to the fact that since S-CNMMCF performs node and link mapping in two separate steps, link mappings could fail due to locations of already mapped nodes.

Fig. 4 shows the average utilisation of SN resources. It can be observed that except for S-CNMMCF, the other three approaches on average use the same amount of SN resources. The fact that S-CNMMCF has a lower resource utilisation is expected as a result of having slightly more resource

requests rejected either due to a node mapping that makes link mapping impossible, or for previous link mappings using more resources. The fact that S-OS, D-RL and D-ANN all have on average the same resource utilisation profile is mainly due to all of them having the same initial mapping algorithm (which is S-OS). It can however be noted that while S-OS, D-RL and D-ANN all have similar resource utilisation levels, D-ANN uses these resources to achieve a higher acceptance of VNs, which further confirms the extra resource allocation efficiency of the ANN approach.

Fig. 5 shows that S-OS has an almost constant packet drop rate while that for D-RL and D-ANN is initially high, but gradually converges to that of S-OS. At the beginning of the learning processes, the dynamic approaches vary the queue sizes quite considerably leading to more packet drops. The fact that D-ANN has a lower packet drop rate than D-RL over the learning period can be explained since D-ANN has better granularity in perceiving the state of resources and allocation. We also note that the initial drop rate of D-ANN is lower than that of D-RL which can be attributed to the weight initialisation obtained from Weka (See Section IV-B3).

Finally, Fig. 6 shows that packet delay variations for the two dynamic approaches is initially higher but reduces over the learning period. Once more, these differences are attributed to the initial learning period, and the difference between D-RL and D-ANN is due to better options in perception and action for D-ANN, as well as the weight initialisation in D-ANN.

VI. Conclusion

This paper has proposed a distributed and dynamic approach for allocation of resources in virtual networks. We applied artificial neural networks to ensure that the allocation agents perceive a continuous network state and take continuous resource allocation actions. We have been able to show through simulation that our proposal improves the acceptance ratio of virtual networks, which would translate into higher revenue for the infrastructure providers, while ensuring that the quality of service to the virtual networks is not negatively affected.

In future, we intend to extend this proposal to the multi-domain environment, which raises more questions especially with regard to cooperation and trust between agents as well as the need for negotiation. We will also study the possibilities of implementing our proposal in a real network, say, by setting up a server to collect VN requirements and user traffic characteristics, and using this in a prototype LAN.

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) and project TEC2012-38574-C02-02 from Ministerio de Economia y Competitividad.

References

[1] A. Fischer, J.F. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *Communications Surveys Tutorials, IEEE*, 15(4):1888–1906, Fourth 2013.

[2] R. Mijumbi, J.L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck. Contributions to efficient resource management in virtual networks. In *Proceedings of the IFIP 8th International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, AIMS2014, 2014.

[3] R. Mijumbi, J.L. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latre. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, NOMS2014, 2014.

[4] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[5] Fu Qi-ming, Liu Quan, Cui Zhi-ming, and Fu Yu-chen. A reinforcement learning algorithm based on minimum state method and average reward. *Computer Science and Information Engineering, World Congress on*, 5:534–538, 2009.

[6] Jeff Heaton. *Introduction to Neural Networks for Java, 2Nd Edition*. Heaton Research, Inc., 2nd edition, 2008.

[7] Wenping Pan, Dejun Mu, Hangxing Wu, and Lei Yao. Feedback Control-Based QoS Guarantees in Web Application Servers. In *HPCC*, pages 328–334. IEEE, 2008.

[8] Rui Han, Li Guo, M.M. Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 644–651, May 2012.

[9] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu. Resource provisioning for cloud computing. In *Conference of the Center for Advanced Studies on Collaborative Research*, pages 101 –111, 2009.

[10] Junfei Qiao, Ruiyuan Fan, Honggui Han, and Xiaogang Ruan. Q-learning based on dynamical structure neural network for robot navigation in unknown environment. In *ISNN (3)*, volume 5553 of *Lecture Notes in Computer Science*, pages 188–196. Springer, 2009.

[11] Shi chao Wang, Zheng xi Song, Hao Ding, and Hao bin Shi. An improved reinforcement q-learning method with bp neural networks in robot soccer. In *ISCID (1)*, pages 177–180. IEEE, 2011.

[12] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

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

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[15] Stephen R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.

[16] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '01, pages 346–353, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Network Simulator 3. http://www.nsnam.org/. Accessed: 2014-02-17.

[18] 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. Accessed: 2014-02-17.

[19] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/about/. Accessed: 2014-02-17.