Learning Algorithms for Dynamic Resource Allocation in Virtualised Networks

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

***Abstract*—Network virtualisation is being proposed as a key technology to support dynamic service composition and diverse protocol suites in the future Internet. It involves a sharing of physical resources by multiple virtual networks. Every virtual network can then use its share of the resources to provide its users with a custom set of services and protocol stacks. 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 employ machine learning to achieve 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.**

Keywords*—Machine learning, reinforcement learning, network virtualisation, dynamic resource allocation, future Internet.

I. Introduction

Network Virtualisation is the decoupling between physical and user network resources. It involves the splitting of a role traditionally taken by Internet service providers (ISPs) into two; such that we have infrastructure providers (InPs) who concentrate on deploying and managing physical resources in form of substrate networks (SNs), and service providers (SPs) who use these resource to create virtual networks (VNs), so as to provide end-to-end services to end users. Efficient sharing of SN resources amoung VNs can be achieved in two steps [1]. The first, known as virtual network embedding (VNE) [2], 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 VNE is well studied, decentralised and dynamic management of resources in virtualised networks still requires attention [2]. In particular, most current approaches do allocate a fixed amount of resources to virtual nodes and links throughout the lifetime of the virtual networks. Since user traffic is non-uniform, static resource allocation could have a negative impact on the resource utilisation efficiency, especially if it leads to the InP rejecting requests to map more virtual networks while reserving resources to VNs that are not utilising their allocation.

In this paper, we propose a decentralised learning system in which autonomous agents are used to perform a dynamic resource allocation (DRA) to virtual nodes and links. The DRA involves monitoring virtual nodes and links for actual resource utilisation levels, and adjusting allocations to reduce, as much as possible, the un utilised resources. To this end, we start by representing each substrate node and link as an autonomous agent [3]. These agents use reinforcement learning [4] to learn optimal resource allocation policies with two objectives: (1) to ensure that virtual networks do not keep idle resources, (2) to ensure that virtual networks always have the resources needed to satisfy their QoS requirements.

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 and briefly introduces reinforcement learning. We present the proposed learning approach in Section IV, evaluate it in Section V, and conclude the paper in Section VI.

II. Related Work

A comprehensive survey on the state-of-the-art in VNE can be found in [2]. 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, [5] is a control theoretic approach, [6] is based on performance dynamics, while [7] uses workload prediction. The difference between our proposal and these works is not only with respect to the solution tool (RL), 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.

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 reinforcement learning (RL).

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 [8]. VNE is out of the scope of this work. Any of the static approaches in [2] 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 RL approach.

C. Reinforcement Learning (RL)

RL is a technique from artificial intelligence [9] in which an agent placed in an environment performs actions from which it gets numerical rewards. For each learning episode [4], 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 [4]. It can learn to do this over time by systematic trial and error, guided by a wide variety of *learning algorithms* [10]. One such learning algorithm is Q-learning. This is a temporal difference [4] 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 [4]. 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 [4]. A detailed description of the modeling of the different aspects of reinforcement learning in the context of dynamic resource allocation in virtualised networks is the subject of Section IV.

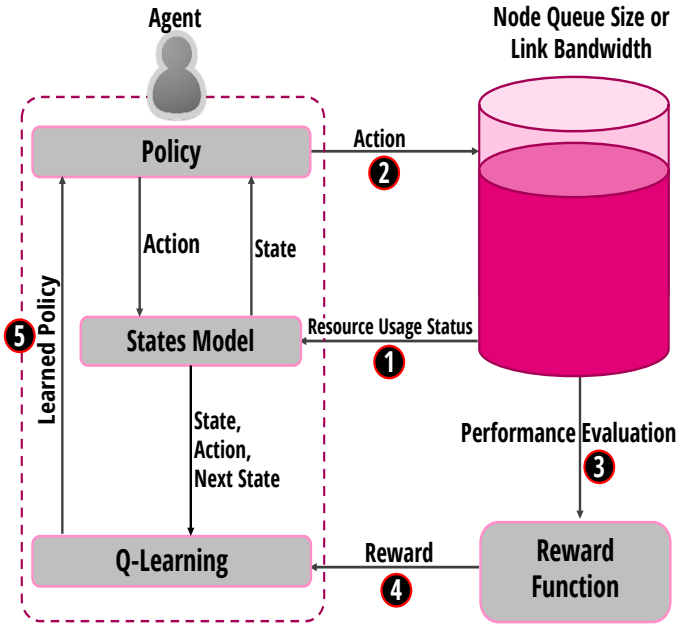

Fig. 1. Reinforcement Learning-based Dynamic Resource Allocation Model

IV. Reinforcement Learning-based DRA

VNE 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. The approach proposed in this paper is to dynamically adjust the resource allocation using RL. To this end, we represent the substrate network as a multi-agent system, in which each substrate node and link 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 serve user requests.

In Fig. 1, we show the proposed RL model for a given substrate node/link. As can be noted, our proposal is made up of five steps. The agent starts by getting a resource usage status, which could have information such as the percentage of substrate node/link resources available and/or the ratio of total virtual node/link demand currently allocated. With this information, the agent takes an action, which could involve increasing or reducing the amount of resource allocated to the virtual node/link. The virtual node/link is then monitored to evaluate its performance e.g. in form of link delays (in case of virtual links) or packet drops (in case of virtual nodes). This evaluation is communicated to the agent in form of a reward, and based on this, the agent adjusts its policy (the Q-Values) so as to ensure that its future resource allocation actions are better. Each of the components of the model in Fig. 1 is described in subsequent subsections.

A. Policy

The policy is implemented by means of a lookup table which, for each state, maintains an updated evaluation (in form of Q-values) of all the possible actions. Since we have 9 possible actions and 512 possible states (as explained in the next two paragraphs), 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.

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. 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 $R_a^v = 011$. 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, at any point, each agent can choose 1 of the 9 possible actions, $a \in \mathcal{A}$, where $A_0 = -50\%, A_1 = -37.5\%, A_2 = -25.0\%, A_3 = -12.5\%, A_4 = 0\%, A_5 = 12.5\%, A_6 = 25.0\%, A_7 = 37.5.0\%, A_8 = 50.0\%$[1].

B. 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^v, R_u^v, and R_u^z, a states model returns a state $s \in \mathcal{S}$. In the same way, when provided with a given state $s_p = \left(R_{a_p}^v, R_{u_p}^v, R_{u_p}^z\right)$ and an action a_p, the states model provides the next state $s_n = \left(R_{a_n}^v, R_{u_n}^v, R_{u_n}^z\right)$. 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.

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

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[2] v, and is dependent on the percentage resource allocation R_a^v, the percentage resource utilisation $\hat{R_u}$ i.e. $\hat{R_u} = 1 - R_u^v$, 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 $n_a \in \mathcal{N}_a$.

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

Where ν, κ and η are constants aimed at adjusting the influence of the variables $\hat{R_u}$, $\hat{D}_{ij}$ and $\hat{P}_i$ to the overall reward. $\hat{D}_{ij}$ and $\hat{P}_i$ are measures of 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.

D. Q-Learning

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 proposal, the policy is updated using the Q-learning and its associated policy update rule [4]. It is however worth noting that the q-learning algorithm used in this paper is slightly different from the "conventional" Q-learning algorithm [4] because instead of getting a reward immediately, in our case the reward of a given learning episode is used just before the following episode after a performance evaluation has been made.

V. Performance Evaluation

To evaluate the performance of the proposed approach, we added a network virtualisation module to NS3 [11]. The implementation is such that every time a virtual network request is accepted by the substrate network, the virtual network topology is created in NS3, and a traffic application starts transferring packets over the virtual network. The traffic used in this paper is based on real traffic traces from CAIDA anonymised Internet traces [12]. These traces are used to obtain packet sizes and time between packet arrivals for each VN. The substrate and virtual network topologies are generated using Brite [13]. We assumed that virtual network requests arrive following a Poisson distribution, and their service times follow a negative exponential distribution. We evaluate the performance of our proposal on two fronts; the quality of the embeddings, and the quality of service. The quality of embeddings is evaluated

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

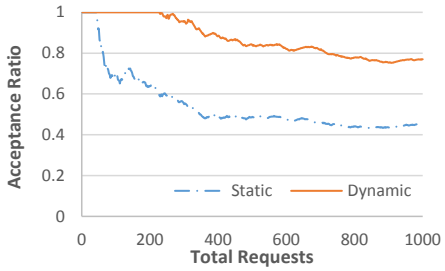

Fig. 2. VN Acceptance Acceptance Ratio

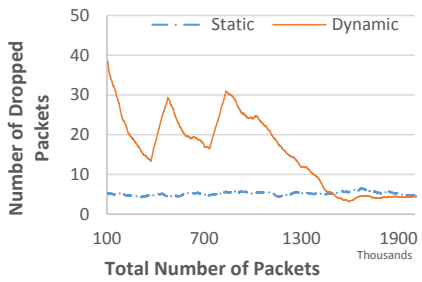

Fig. 3. Node Packet Drop Rate

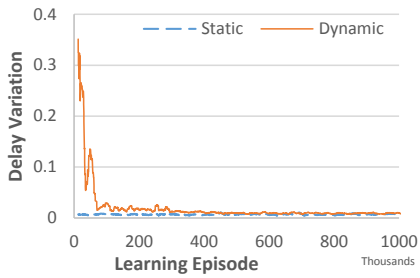

Fig. 4. Link Packet Delay Variation

using the acceptance ratio [2], while the packet delay and drop rate are used as indications of the quality of service.

A. *Discussion of Results*

The simulation results are shown in Fig. 2 – 4. As can be seen from Fig. 2, 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.

Fig. 3 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 relatively bad 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. However, as can be noted, the packet drop rate by both approaches is comparable towards the end of the simulation.

Similarly, Fig. 4 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, as the delay easily converges to that of the static approach after the agents have learnt.

VI. 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 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, for example, our proposal considers that the substrate network, and hence all the node and link agents remain operational at all times. It would be interesting to incorporate possibilities that some of the agents would become un available due failures in physical network resources. In addition, extending the proposal to a multi-domain environment would require consideration of possible competition between the agents.

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] 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.

[2] 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.

[3] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

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

[12] 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.

[13] 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.