Expert Systems with Applications 42 (2015) 1376–1390

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A neuro-fuzzy approach to self-management of virtual network resources

Rashid Mijumbi [a,*], Juan-Luis Gorricho [a], Joan Serrat [a,*], Meng Shen [b], Ke Xu [b], Kun Yang [c]

[a] Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
[b] Department of Computer Science, Tsinghua University, 100084 Beijing, PR China
[c] School of Computer Science and Electronic Engineering, University of Essex, Essex CO4 3SQ, UK

ARTICLE INFO

Article history:
Available online 30 September 2014

Keywords:
Neural networks
Fuzzy systems
Neuro-fuzzy systems
Reinforcement learning
Network virtualisation
Dynamic resource allocation
Multi-agent systems
Autonomous systems
Future Internet

ABSTRACT

Network virtualisation promises to lead to better manageability of the future Internet by allowing for adaptable sharing of physical network resources among different virtual networks. However, the sharing of resources is not trivial as virtual nodes and links should first be mapped onto substrate nodes and links, and thereafter the allocated resources managed throughout the lifetime of the virtual network. In this paper, we design and evaluate reinforcement learning-based neuro-fuzzy algorithms that perform dynamic, decentralised and coordinated self-management of substrate network resources. 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 algorithms are evaluated through comparisons with a Q-learning-based approach as well as two static resource allocation schemes.

1. Introduction

Network virtualisation (Fischer, Botero, Till Beck, de Meer, & Hesselbach, 2013) continues to be a focus of the research community due to its possibility to allow infrastructure providers (InP) who own substrate networks (SN) to lease out chunks of their physical resources to service providers (SP). The SPs then use these resources to create virtual networks (VN), which are used to either provide end-to-end services to end users or in a more general case, also lease these resources out to lower tier SPs. In network virtualisation, a VN is made up of a set of virtual links and nodes which are supported by SN physical paths and nodes respectively.

One of the important steps involved in initialising VNs is the efficient sharing of SN resources among the set of VNs to be supported by the SN. This can normally be divided into two steps (Mijumbi et al., 2014). The first step performs the mapping of virtual nodes and links of a new VN request to substrate nodes and links respectively, subject to a set of pre-defined constraints (e.g. topology, node queue size and link bandwidth). This step is known as virtual network embedding (VNE) (Fischer et al., 2013). The second step involves the management of resources allocated to VNs throughout their lifetime, aiming at efficient resource utilisation while meeting certain quality of service requirements (e.g. delay, jitter and packet drop rate).

A number of state-of-the-art proposals exist for resource allocation in network virtualisation. Some of these approaches perform a static mapping (only VNE) without any considerations for possibilities of adjustments to initial mappings, while those that propose dynamic solutions do allocate a fixed amount of node and link resources to the virtual networks through out their life time. In general, there is only a limited number of decentralised and dynamic solutions to VNE (Fischer et al., 2013). Since network loading due to user traffic varies with time, allocating a fixed amount of resources based on peak loadings could lead to an inefficient utilisation of overall substrate network resources, whereby, during periods when the virtual nodes and/or links are lightly loaded, substrate resources are still reserved for such virtual nodes/links, while possibly rejecting new requests for such resources. This would have a negative impact on the revenue of the InPs, and could hinder the practical advancement of network virtualisation.

In our previous work (Mijumbi et al., 2014), we proposed a dynamic and decentralised scheme for dynamic resource allocation in virtual networks, which was based on Q-learning (Even-Dar & Mansour, 2004) and a look-up table-based policy. However, as the state space of the resource allocation and utilisation in virtual

* Corresponding authors at: UPC Campus Nord, Building C3, Sor Eulalia d'Anzizu, 08034 Barcelona, Spain. Tel.: +34 93 4016786; fax: +34 93 4017200.

E-mail addresses: rashid@tsc.upc.edu (R. Mijumbi), juanluis@entel.upc.edu (J.-L. Gorricho), serrat@tsc.upc.edu (J. Serrat), shenmengnetlab@gmail.com (M. Shen), xuke@mail.tsinghua.edu.cn (K. Xu), kunyang@essex.ac.uk (K. Yang).

http://dx.doi.org/10.1016/j.eswa.2014.08.058
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

networks is continuous, a look-up table representation would suffer from the curse of dimensionality (Qi-ming, Quan, Zhi-ming, & Yu-chen, 2009), and hence limit solution scalability in terms of memory requirements for storage of continuous state-action values. Therefore, Mijumbi et al. (2014) discretised the state space to limit the number of states. This comes at a cost in terms of resource allocation efficiency as it limits the sensitivity of resource re-allocation actions to changes in resource utilisation levels.

This paper significantly extends (Mijumbi et al., 2014) by proposing an adaptive (changes with variations in the resource utilisation), hybrid (uses both supervised and unsupervised learning), distributed (represented by a multi-agent system), cooperative (between agents) approach to resource management in VNs that takes as input a continuous state space, and outputs a continuous action space. We model the substrate network as a multi-agent system (Stone & Veloso, 2000) in which each node or link is represented by an agent. However, instead of the agents directly using reinforcement learning (RL) (Even-Dar & Mansour, 2004) as was proposed in Mijumbi et al. (2014), the agents in this paper use an adaptive hybrid neuro-fuzzy system (NFS) (Nürnberger, Nauck, & Kruse, 1999) which learns by use of a reinforcement learning-like reward. The objective of each agent is to dynamically adapt its knowledge base so as to efficiently utilise the resources of the substrate network (avoid that virtual nodes/links have high percentages of unutilised resources), while ensuring that the QoS requirements of the VNs (measured in terms of packet drop rate, delay, jitter) are not negatively affected (by ensuring that at any point the VNs have the resources they require).

The contributions of this paper are as follows:

(1) an adaptive neuro-fuzzy system that dynamically learns allocation of substrate resources to virtual networks,
(2) a hybrid learning mechanism which uses supervised learning to initialise the rule base and then uses unsupervised learning to adapt the rule base and fuzzy sets of each rule to achieve efficient resource allocation,
(3) A cooperation scheme that allows the substrate network agents to coordinate their actions so as to avoid conflicts and to share their knowledge so as to enhance their learning speed and improve action selection efficiency.

The rest of the paper is organised as follows: We define the dynamic resource allocation problem in the context of network virtualisation in Section 2. Section 3 briefly introduces neural networks, fuzzy systems, neuro-fuzzy systems and reinforcement learning while the proposed NFS is described in Section 4. The proposed evaluative feedback and rule base initialisation approaches are presented in Sections 5 and 6 respectively, while we define the agent cooperation scheme in Section 7. The evaluation of performance is given in Section 8. Related work is presented in Section 9, and the paper is concluded in Section 10.

2. Resource allocation problem description

2.1. Virtual and substrate network modelling

The allocation of SN resources to a given VN is initiated by a SP specifying resource requirements for both virtual nodes and links to the InP. The specification of VN resource requirements is usually represented by a weighted undirected graph denoted by $G_v = (N_v, L_v)$, where N_v and L_v represent the sets of virtual nodes and links respectively. Each virtual link $l_{ij} \in L_v$ connecting the virtual nodes i and j has a maximum delay D_{ij} and bandwidth B_{ij}, while each virtual node $i \in N_v$ has a proposed location $L_i(x, y)$, a constraint on deviation from its proposed location $\Delta L_i(\Delta x, \Delta y)$ which specifies the maximum allowed deviation for each of its x and y coordinates, and a queue size Q_i, which is a measure of the maximum number of packets (or Bytes) a given node can have in its buffer before dropping packets. Similarly, a substrate network can be modelled 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 $L_u(x, y)$.

2.2. Virtual network embedding (VNE)

The VNE problem involves the mapping of each virtual node $i \in N_v$ to one of the possible substrate nodes within the set $\Theta(i)$, where $\Theta(i)$ is a set of all substrate nodes $u \in N_s$ that have enough *available queue size* and are *located* within the maximum allowed deviation $\Delta L_i(\Delta x, \Delta y)$ of the virtual node. 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 bandwidth 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. For avoidance of doubt, we state that our consideration is that VNs arrive one at a time and hence the embedding is online, initialising every VN request on arrival. VNE is out of the scope of this work. Any of the static approaches in Section 9.1 can be used for this step. In this paper, two static approaches are used for comparisons in the evaluations. The first is VINEYard (Chowdhury, Rahman, & Boutaba, 2012) and the other is a baseline optimal mathematical programming formulation that performs both node and link embedding in one step. The mathematical programs implementing the algorithms in both approaches are solved using ILOG CPLEX 12.5 (IBM, 2014).

2.3. Dynamic resource management (DRM)

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 in form of packets is transmitted over the VN. Actual usage of allocated resource 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 new VN requests are not rejected when resources reserved to already mapped VNs are idle. It is however a delicate trade-off to ensure that the VNs always have enough resources to guarantee that QoS parameters are kept as established in the corresponding SLAs. In Section 4, we detail the proposed neuro-fuzzy DRM approach.

3. Learning neuro-fuzzy systems

This Section introduces the four main background topics – fuzzy systems, neural networks, neuro-fuzzy systems and reinforcement learning – which constitute the proposed solution.

3.1. Fuzzy systems (FS)

FSs are rule-based expert systems, which use fuzzy rules and fuzzy inference (Kasabov, 1996). These systems are usually made up of three functional blocks: fuzzification, knowledge base and inference, and defuzzification. At the fuzzification step, the values of numerical inputs are compared with membership functions (MFs) so as to determine their degree of membership in the respective fuzzy sets.[1] The knowledge base and inference block contains the fuzzy rules (rule base) which represent knowledge and skills, a database of fuzzy sets (represented by membership functions) used to represent the fuzzy rules, and a decision making unit which performs inference on the fuzzy rules. A fuzzy rule represents knowledge and skills, and is of the form:

3.2. Neural networks (NN)

NNs are computational methodologies inspired by networks of biological neurons to perform multifactorial analysis (Dayhoff & DeLeo, 2001). These networks are made up of simple interconnected computing nodes known as *neurons* which operate as summing devices. A neuron receives one or more inputs, which are first multiplied by *weights* along each connection, and then *summed* to produce a single number. This number is then passed through a (usually non-linear) transfer function, which determines the input–output behaviour of the neuron. If information flows in only one direction (from input towards output), the neural network is called a feed forward network. In NNs, neurons are arranged in layers. Each layer consists of one or more neurons. The most commonly used structure of NNs is made up of 3 layers; an input layer, a hidden layer and an output layer (Smith, 1999). The learning ability of neural networks lies in their ability to gradually minimise an error, which is defined as the difference between a desired output and actual output. Therefore, using neural networks requires that for every input state, a desired output state must be known so as to determine the error.

3.3. Neuro-fuzzy system (NFS)

Since fuzzy systems are created from explicit knowledge in form of rules and membership functions (MFs), applying them in dynamic systems requires a way of automatically tuning these parameters (rules and MFs), or even changing the structure of the system. One of the most popular ways of dynamically tuning these parameters and/or adapting the structure of fuzzy systems is by use of neural networks. A combination of neural networks and fuzzy systems leads to NFSs. NFSs benefit from both the learning characteristics of NNs as well as interpretation and clarity of systems representation found in fuzzy systems. A NFS can be viewed as a special 3-layer feedforward neural network in which the neurons use t-norms and t-conorms instead of the usual neural network activation functions (Nürnberger et al., 1999). The first layer represents the input variables, the hidden layer represents the fuzzy rules and the third layer represents output variables. The fuzzy sets are encoded as (fuzzy) connection weights. The knowledge and hence accuracy of the network is determined by its structure (the different connections), as well as the fuzzy weights on these connections. We model the interactions between the different components of the designed NFS, as well as with the multi-agent system for the application to dynamic virtual network resource management in Section 4.

[1] While we have done our best to make this paper self-contained, a reader who is not familiar with such terms as fuzzy variable, fuzzy set, fuzzy logic, fuzzy rule, t-norm, t-conorm, membership functions is referred to (Kasabov, 1996) for an introduction.

3.4. Reinforcement learning (RL)

RL is a technique from artificial intelligence (Russell & Norvig, 2009) in which an agent placed in an environment performs actions from which it gets numerical rewards. For each learning episode, 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. It can learn to do this over time by systematic trial and error, guided by a wide variety of learning algorithms such as Q-learning.

4. NFS model for VN resource allocation

As mentioned in Section 1, VNE approaches allocate resources to each virtual node and link as specified in VN requests. In static allocation schemes, the amount of allocated resources is kept fixed irrespective of actual utilisation, which leads to inefficient utilisation of substrate network resources (Mijumbi et al., 2014). Our approach dynamically adjusts the resources allocated to each virtual node and link using a neuro-fuzzy system. The overall system model used for this purpose is shown in Fig. 2. As can be observed from the model, we highlight the multi-agent system representing the substrate network and the learning neuro-fuzzy system that represents the internal components of each agent. We further split the learning neuro-fuzzy system into four functional modules: fuzzifier, RDI (rule base, database, inference), defuzzifier, and learning (EF, AC, ARW). In the following subsections, each of these elements of the model and their respective interactions is detailed.

4.1. Multi-agent environment

The multi-agent environment consists of all the agents that represent the substrate network. Specifically, 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. As shown in Fig. 2, a given link agent receives as input the state of the substrate link it manages.

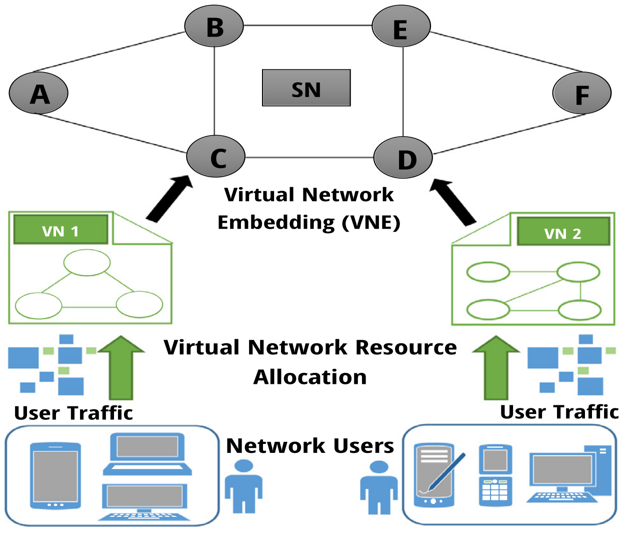
B
E
A
SN
F
C
D
Virtual Network Embedding (VNE)
VN 1
VN 2
Virtual Network Resource Allocation
User Traffic
User Traffic
Network Users

Fig. 1. VNE and DRM.

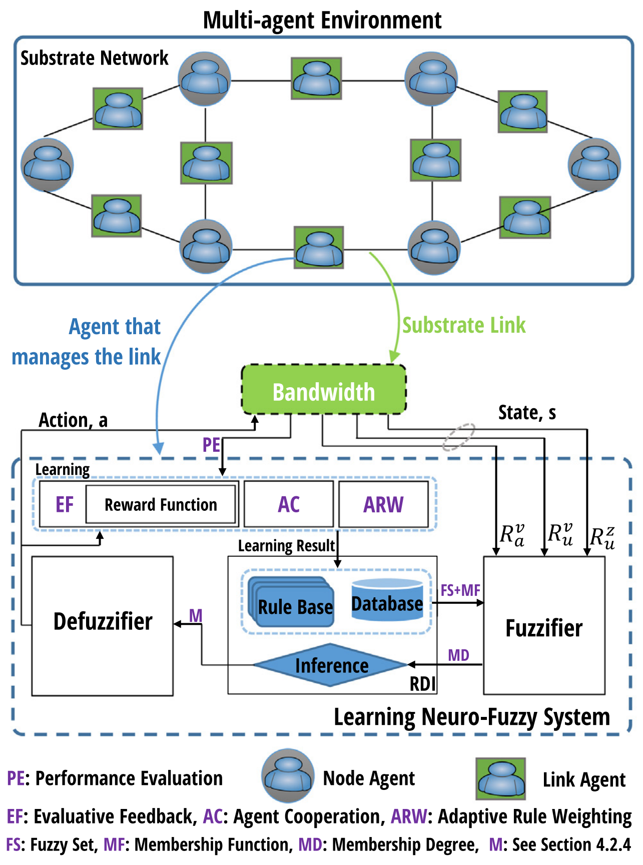
Multi-agent Environment
Substrate Network
Agent that manages the link
Substrate Link
Bandwidth
Action, a
State, s
PE
Learning
EF
Reward Function
AC
ARW
Learning Result
R_a^v
R_u^v
R_u^z
Defuzzifier
M
Rule Base
Database
FS+MF
Fuzzifier
Inference
MD
RDI
Learning Neuro-Fuzzy System
PE: Performance Evaluation
Node Agent
Link Agent
EF: Evaluative Feedback, AC: Agent Cooperation, ARW: Adaptive Rule Weighting
FS: Fuzzy Set, MF: Membership Function, MD: Membership Degree, M: See Section 4.2.4

Fig. 2. Learning neuro-fuzzy system VN resource allocation model.

The state of any resource[2] is a vector **S** with each term $s \in \mathcal{S}$ representing the state of one of the virtual links/nodes mapped onto it. More specifically, this state, for any given virtual resource v hosted on a substrate resource z, is represented 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. While in a given state, s, the agent gives as output, an action. The action of each agent is a vector **A**, where each term $a \in \mathcal{A}$ indicates which action should be taken to change the resources for each of the virtual node/link mapped onto it. The action may be aimed at increasing or reducing the resources (queue size or bandwidth) allocated to any virtual node or link respectively. Each element in **A** corresponds to a unique element in **S**. Based on how well a given action drives the objectives of the system, the agent gets a performance evaluation (PE), which is used to learn so as to improve future actions. 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.

4.2. Learning neuro-fuzzy system (NFS)

4.2.1. Database

The database is a definition of the fuzzy sets (FS) and the membership functions (MF) that represent them, and is used in the

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

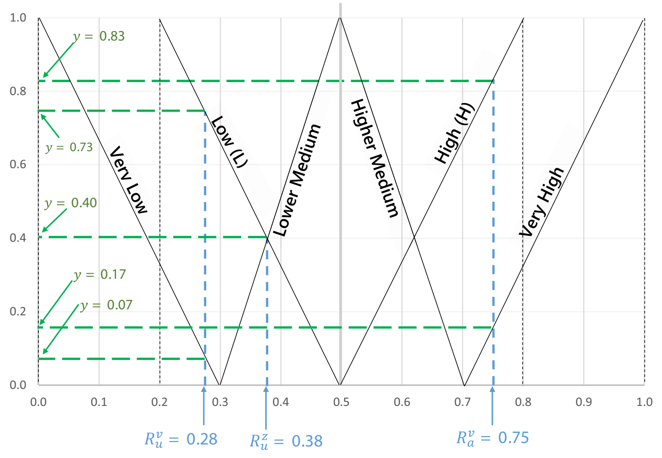

Fig. 3. Monotonic MFs for input (state) fuzzy sets.

creation of fuzzy rules. We have defined six fuzzy sets into which the input variables can fall. These are: very low (VL), low (L), lower medium (LM), higher medium (HM), high (H) and very high (VH). These fuzzy sets are represented by the monotonic membership functions in Fig. 3. Each membership function $y = \mu(x)$ in Fig. 3 is characterised by two parameters p and q such that $\mu(p) = 0$ and $\mu(q) = 1$ as defined in Eq. (1). The MFs are used to determine a value or degree of membership, which quantifies the grade of membership of a given variable to the fuzzy sets.

$$y = \mu(x) = \begin{cases} \frac{p-x}{p-q} & \text{if } ((p \leqslant q) \wedge (x \in [p,q])) \quad (a) \\ \frac{p-x}{p-q} & \text{if } ((p > q) \wedge (x \in [q,p])) \quad (b) \\ 0 & \text{otherwise} \quad (c) \end{cases} \tag{1}$$

In the same way, we have defined eight fuzzy sets for the output variable. These are: negative large (NL), negative medium (NM), negative small (NS), negative zero (NZ), positive zero (PZ), positive small (PS), positive medium (PM) and positive large (PL). These fuzzy sets are represented by the monotonic membership functions in Fig. 4, and have similar definitions as in (1).

4.2.2. Rule base

The rule base consists of the rules that are used by the agents to take actions while in given states. These rules are based on the input and output fuzzy sets defined in Figs. 3 and 4. As an example, (2) presents four possible rules that could be formulated for the system under consideration.

$$R_1: \text{ if } R_a^v \text{ is VH and } R_u^v \text{ is L and } R_u^z \text{ is LM then } O \text{ is PZ} \tag{2a}$$
$$R_2: \text{ if } R_a^v \text{ is L and } R_u^v \text{ is H and } R_u^z \text{ is HM then } O \text{ is NZ} \tag{2b}$$
$$R_3: \text{ if } R_a^v \text{ is H and } R_u^v \text{ is L and } R_u^z \text{ is L then } O \text{ is PS} \tag{2c}$$
$$R_4: \quad R_u^v \text{ is VL then } O \text{ is PS} \tag{2d}$$

In words, rule (2a) states that: if the resource allocation to the virtual node/link v is very high AND a low percentage of these resources are unused AND the substrate node/link z onto which it is embedded has a low−medium percentage of free resources, then the action should be to increase the amount of resources allocated to v by an amount determined by *positive zero*. In general, each agent will have more than one rule at any given time. Therefore, since we have 3 input variables each with 6 possible fuzzy sets and 1 output variable with 8 possible fuzzy sets, the maximum number of rules we can have in the system is $6 \times 6 \times 6 \times 8 = 1728$. Initialising a system with this number of rules would take a long time before the NFS has reduced

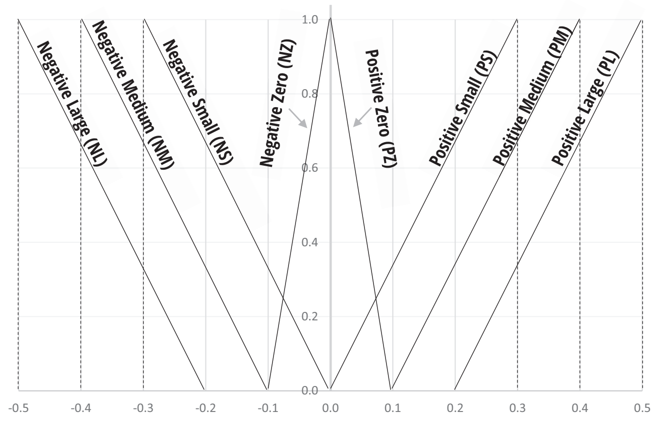

Fig. 4. Monotonic MFs for output (action) fuzzy sets.

the rules to the *necessary* ones.[3] In this paper, we propose supervised learning-based rule initialisation scheme that is aimed at enhancing the learning speed of the agents. This scheme is presented in Section 6.

4.2.3. Fuzzifier

Given a state s represented by the real-valued variables R_a^v, R_u^v and R_u^z, a fuzzifier determines the set of rules $R\prime \subseteq R$ that are satisfied (fired), where R is the rule base. Then, for each satisfied rule, fuzzification involves determining the membership degree (MD) of each input variable in the respective fuzzy set. This is the result of the function $\mu(x)$ for a particular input variable, rule and membership function, and is calculated using Eqs. (1). To illustrate this, consider that rule (2a) has been satisfied by a given input state. Then, the fuzzifier would have to determine the membership degrees $\mu_{5a}^{\mathrm{VH}}(R_a^v), \mu_{5a}^{\mathrm{L}}(R_u^v)$ and $\mu_{5a}^{\mathrm{LM}}(R_u^z)$ to which the inputs R_a^v, R_u^v and R_u^z belong to the fuzzy sets VH, L and LM respectively. As can be noted from Fig. 3, a given input variable can belong to one or more MFs. For example, the variable $R_a^v = 0.75$ belongs to both H and VH with membership degrees $y = \mu_i^{\mathrm{H}}(0.75) = 0.83$ and $y = \mu_i^{\mathrm{VH}}(0.75) = 0.17$ respectively. In a similar way, given values of $R_u^v = 0.28$ and $R_u^z = 0.38$, their degrees of membership to the respective fuzzy sets can be determined as shown in Fig. 3.

4.2.4. Inference

The inference block receives, for each rule $R_i \in R\prime$, a membership degree $y_i^{R_a^v}, y_i^{R_u^v}$ and $y_i^{R_u^z}$ for each of the variables R_a^v, R_u^v and R_u^z respectively. Ideally, standard inference would involve a *max* – *min* operation on these membership degrees (Kasabov, 1996). However, due to the nature of the defuzzification function used in this paper (see Section 4.2.5), inference only involves the min operation. This operation results into the minimum membership degree $y_{min} = min\left(y_i^{R_a^v}, y_i^{R_u^v}, y_i^{R_u^z}\right)$ for each rule. Then, the inverse $\lambda^{-1}(y_{min})$ of each y_{min} is determined from (3).

$$x = \lambda^{-1}(y) = p - y(p - q) \tag{3}$$

Eq. (3) is similar to (1), re-arranged to make x the subject. A matrix **M** is then created with the each row containing the value y_{min} in the first column and $\lambda^{-1}(y_{min})$ in the second column. Therefore, the matrix **M** has as many rows as the number of fired rules. It is this matrix that is used at the defuzzification step.

4.2.5. Defuzzifier

The output of the inference block is a matrix **M** of membership degrees and their respective inverse values. The role of the defuzzifier is to convert this matrix into an output action a for the agent. In this paper, we adopt a non-standard defuzzification approach proposed in (Nauck & Kruse, 1992), in which the crisp output a is given by (4).

$$a = \frac{\sum_{i=1}^{n}(m_{i1} \times m_{i2})}{\sum_{i=1}^{n} m_{i1}} \tag{4}$$

where $n = |R\prime|$ is the number of fired rules, and m_{i1} and m_{i2} are the elements in the first and second columns respectively of the i^{th} row of M. The rationale behind (4) is to determine a weighted average of the potentially applicable actions by their corresponding membership values. It is worth noting that this simplified defuzzification process is a direct result of the monotonic membership functions used to define the output fuzzy sets. These membership functions make it unnecessary to make any translations of the inverse $\mu_{y_{min}}$ of each rule as these values are already crisp (Nauck & Kruse, 1992). The use of monotonic membership functions for the input fuzzy sets is for simplicity.

4.2.5.1. Example: fuzzification, inference, defuzzification.

To illustrate the processes described above with regard to the VN resource allocation problem, we revisit the four rules defined in (2). Consider that the state of a given substrate resource z and a virtual resource v is such that $R_a^v = 0.75, R_u^v = 0.28$ and $R_u^z = 0.38$ as shown in Fig. 3. In the *fuzzification* step, we note that the input variable $R_a^v = 0.75$ lies in two fuzzy sets VH and H with membership degrees $\mu VH(R_a^v) = 0.17$ and $\mu^H(R_a^v) = 0.83$ respectively. In a similar way, the membership degrees of $R_u^v = 0.28$ and $R_u^z = 0.38$ to their respective fuzzy sets can be determined. We see that the rule R_2 is not satisfied (e.g. since R_a^v does not belong to the fuzzy set L) while rules R_1, R_3 and R_4 are satisfied. In Table 1, we show these details for each input variable. For R_1, the *inference* step is $y_{min}^1 = min(0.17, 0.73, 0.4) = 0.17$, that for R_3 is $y_{min}^3 = 0.4$, and that for R_4 is $y_{min}^4 = 0.07$. We therefore have the membership degrees for the three output fuzzy sets: PZ, PS and PS for rules R_1, R_3 and R_4 respectively. We now find the inverse for each y_{min} using (3). As can be seen from Fig. 4, for $R_1, p = 0.1$ while $q = 0.0$. Therefore, $\lambda^{-1}(y_{min}^1) = 0.1 - 0.17 \times (0.1 - 0.0) = 0.083$. In a similar way, $\lambda^{-1}(y_{min}^3) = 0.12$ and $\lambda^{-1}(y_{min}^4) = 0.021$. Finally, the input into the output layer node is a $n \times 2$ *matrix* where each row i contains the output y_{min}^i of the i^{th} rule (R_i) in the first column and its inverse $\lambda_i^{-1}(y_{min}^i)$ in the second column. For our example, the matrix, M below will be the input to the output node (defuzzifier).

$$M = \begin{bmatrix} 0.17 & 0.0830 \\ 0.40 & 0.1200 \\ 0.07 & 0.0210 \end{bmatrix}$$

The last step is *defuzzification* and uses Eq. (4) on the contents of **M** to produce the agent action. The action a in this case would be

$$\frac{(0.17 \times 0.083) + (0.4 \times 0.12) + (0.07 \times 0.021)}{0.17 + 0.4 + 0.07} = 0.11$$

This would mean that the resources allocated to the virtual resource in question has to be increased by 11% of its total demand.

4.2.6. Learning

After an agent takes an action, we evaluate the action so as to determine if it led to a better utilisation of substrate resources without negatively impacting the QoS requirements of the virtual node or link. Therefore, as shown in Fig. 2, the learning module receives as input the agent's action, and an evaluation of the performance (PE) that resulted from this action. Then, the module outputs a "learning result" that is aimed at adjusting the knowledge base, and hence lead to better actions in future. The agents designed in this work perform learning to achieve one or more of three objectives, which are aimed at (1) deleting rules deemed

[3] We refer to rules as being necessary if they are often required by an agent, and their actions usually lead the agent to efficient resource allocations.

Table 1
Running example – fuzzification.

Input variable	Fuzzy sets	Satisfied rules	Membership degrees
$R_a^v = 0.75$	VH, H	R_1, R_3, R_4	$\mu_1^{VH}(0.75) = 0.17,\ \mu_3^{H}(0.75) = 0.83$
$R_u^v = 0.28$	VL, L	R_1, R_3, R_4	$\mu_1^{L}(0.28) = 0.73,\ \mu_3^{L}(0.28) = 0.73,\ \mu_4^{VL}(0.28) = 0.07$
$R_u^z = 0.38$	LM, L	R_1, R_3, R_4	$\mu_1^{LM}(0.38) = 0.4,\ \mu_3^{L}(0.38) = 0.4$

unnecessary, (2) adding rules expected to be useful in future, and (3) adjusting the membership functions. In order to achieve these objectives, the learning module is made up of three sub-modules, each of which is associated to one of the objectives as detailed in what follows.

4.2.6.1. Adaptive rule weighting (ARW). Adaptive rule weighting involves adjusting a weight w_i. The weight w_i is defined, initialised and attached to each rule R_i during the rule base initialisation step (see Section 6). After each learning episode, the weight w_i is adjusted in two ways; First, w_i is decremented by a constant $\varphi_1 = 1$ if rule R_i was fired, and incremented by a constant $\varphi_2 = 0.5$ if the rule was not fired. The reason for adjusting w_i based on whether a rule was fired or not is that if a given rule is consistently not used by an agent, then it is more likely that the rule is unnecessary, and should therefore be dropped from the rule base. In this paper, rules with lower values of w_i are considered more important than those with higher values; in fact, in our proposal, when the value of w_i is greater or equal to a constant Ψ_1, the agent can consider the rule unnecessary or counterproductive and hence the rule R_i is deleted from the rule base.

In addition, for all fired rules, w_i is changed according to Eq. (5).

$$w_i^{new} = w_i - r(v) \tag{5}$$

where $r(v)$ is a dynamic evaluation reward defined in (6). The objective of (5) is to reduce the weight w_i whenever the rule contributes to a correct action ($r(v)$ is positive), and increase it otherwise.

4.2.6.2. Agent cooperation (AC). While adaptive rule weighting helps an agent get rid of unnecessary rules, it does not help the agent acquire more rules that are expected to be important to future actions. Agent cooperation can be used for this purpose. Our proposal in this regard involves a *cooperation* between different agents in two ways. First, the agents coordinate to avoid conflicting actions, and then, at predefined times, agents share information aimed at improving their individual performances. By sharing knowledge, the agents can either add or delete rules from their rule bases. We describe cooperation between agents in Section 7.

4.2.6.3. Evaluative feedback (EF). Both ARW and AC can only add or delete a rule from the rule base. However, it is also necessary to be able to adjust a given rule, by changing the parameters of the membership functions so as to improve the output of those rules that remain in the rule base. This is achieved by using a reinforcement learning-based evaluative feedback mechanism that uses a reward function to adjust the membership functions. We describe the reward function used in this paper in Section 5.1, and then derive the rule updating mechanism used to learn the parameters of rule membership functions in Section 5.2.

5. Evaluative feedback

5.1. Reward function

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 (PE). The reward resulting from a learning episode of any agent is therefore a vector **R** in which each term $r(v)$ corresponds to the reward of an allocation to the virtual resource v. This reward is an *indication* of the deviation of the agent's *actual action* from a *desired action*, and is therefore aimed at minimising this deviation. 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 high delays. The reward function defined in this paper is designed so as to carry two pieces of information; a *magnitude* and a *direction*. If the agent's action was desirable, $r(v)$ is positive, otherwise it is negative. The magnitude of $r(v)$ gives the degree of desirability or undesirability of 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 resulting reward function is presented in (6).

$$r(v) = \begin{cases} \kappa\tau & \text{if } (\tau < 0.0) & \text{(a)} \\ (R_v - D_v) & \forall l_a \in \mathcal{L}_a & \text{(b)} \\ (R_v - P_v) & \forall n_a \in \mathcal{N}_a & \text{(c)} \end{cases} \tag{6}$$

where κ is a constant, and τ is an index of the correctness of the action adopted by the agent. Specifically, τ gives an indication of whether the action a taken by the agent should have been *increasing* resource allocation or *reducing* it, and can take on positive or negative values based on the perceived expected direction of action. It is worth noting that (6a) takes precedence over (6b) and (6c), implying that whenever it is satisfied, the reward is calculated from it. Eqs. (6b) and (6c) respectively apply for link and node agents. The value of τ is defined in Eqs. (7).

$$\tau = \begin{cases} a & \text{if}\left((R_a^v \leqslant \xi_1) \wedge (a < 0.0)\right) & \text{(a)} \\ a & \text{if}\left((R_u^v \leqslant \xi_2) \wedge (a < 0.0)\right) & \text{(b)} \\ -a & \text{if}\left((R_u^v \geqslant \xi_3) \wedge (a > 0.0)\right) & \text{(c)} \\ 0 & \text{otherwise} & \text{(d)} \end{cases} \tag{7}$$

where ξ_1, ξ_2, and ξ_3 are constants, and a is the crisp output value of the agent. The definition of τ is aimed at ensuring that actions that could possibly lead an agent away from its objective receive a negative feedback. In fact, Eq. (7a) is aimed at ensuring that resource allocations below ξ_1 are avoided as this could easily have a negative impact on the QoS requirements of the virtual resource. In particular, (7a) states that: if the resource allocated to v is already below a given minimum ξ_1 and the agent's action is to further reduce the allocation ($a < 0.0$), then the reward for this agent should be negative, proportional to the amount by which resource allocation was reduced. In the same way, Eq. (7c) states that if at least ξ_3 of the resources allocated to the virtual resource v are unutilised and the agent decides to increase the resource allocation ($a > 0.0$), then this action takes the agent in the wrong direction, and as a result, τ should be negative. The constant κ in Eq. (6a) can be adjusted to result into higher/lower negative rewards so as to guide the agent away from situations where $\tau < 0.0$. The values $\kappa = 1, \xi_1 = \xi_2 = 0.25$ and $\xi_3 = 0.75$ were used in this paper.

R_v is the utilisation of resources allocated to v and is derived from R_u^v, while D_v and P_v are measures of the performances of link agents and node agents respectively, and their values are derived from the link delay D_{ij} and number of lost packets P_i.

$$R_v = 1 - R_u^v, D_v = \frac{D_{ij}}{0.1} \text{ and } P_v = \frac{P_i}{100}$$

The reason for scaling D_v and P_v is to ensure that $0 \leqslant D_v \leqslant 1$ and $0 \leqslant P_v \leqslant 1$ which is also part of the effort to ensure that $-1 \leqslant r(v) \leqslant 1$. The choice of the values 100 and 0.1 is based simulations in which we observed that $0 \leqslant D_{ij} \leqslant 0.1$ and $0 \leqslant P_i \leqslant 100$. In any case, the values of D_v and P_v are capped at a value 1 and if the scaling results into a value greater than 1, then the maximum value 1 is used.

5.2. Membership function learning

With the overall reward determined, we now need to determine the contribution of each fired rule $R_i \in R\prime$ to the reward, and use it in the learning process to update the membership functions. We can derive the contribution of each rule as a gradient descent (8) on squared error (11), and thereafter adjust a parameter, ε of the membership functions belonging to the rule R_i so as to reduce the general error.

$$\Delta\varepsilon = -\alpha\left(\frac{\partial E}{\partial \varepsilon}\right) \tag{8}$$

where $\Delta\varepsilon$ is the amount by which parameter ε should be changed so as to reduce the error E in its action, $0 \leqslant \alpha \leqslant 1$ is referred to as *learning rate*, and it determines how fast learning occurs. Therefore, an updated value ε_{new} of the parameter can be determined from its current value ε using Eq. (9).

$$\Delta\varepsilon = \varepsilon_{new} - \varepsilon \tag{9}$$

Combining (8) and (9) results into the general learning rule for ε shown in (10)

$$\varepsilon_{new} = \varepsilon - \alpha\left(\frac{\partial E}{\partial \varepsilon}\right) \tag{10}$$

The error E is a measure of the difference between the expected optimal action a^* and the actual action a, and is usually given by a square of the difference between a^* and a as shown in (11).

$$E = \frac{1}{2}(a^* - a)^2 \tag{11}$$

In order to determine a learning procedure over ε, we start by determining the error rate $\partial E/\partial\varepsilon$, which can be derived using the chain rule shown in (12).

$$\frac{\partial E}{\partial \varepsilon} = \frac{\partial E}{\partial a} \times \frac{\partial a}{\partial \mu} \times \frac{\partial \mu}{\partial \varepsilon} \tag{12}$$

Using (11), $\partial E/\partial a = -(a^* - a)$, and as defined in 5.1, the difference between expected and actual actions $(a^* - a)$ is given by the reward function $r(v)$. $\partial a/\partial\mu$ is the contribution of membership function μ to the overall action a (and hence its contribution towards the error). For this work, the value $\lambda^{-1}(y_{min})$ as defined in Section 4.2.4 as the inference result of the antecedent of the rule under consideration is used. Finally, if the parameter ε is along the horizontal axis of the membership function, then $\partial\mu/\partial\varepsilon$ is the gradient of the membership function (i.e. $\partial\mu/\partial\varepsilon = \partial y/\partial x$) and is given by $1/(q-p)$ (using Eq. (3)). Therefore, (12) becomes

$$\frac{\partial E}{\partial \varepsilon} = -r(v) \times \lambda^{-1}(y_{min}) \times \frac{1}{(q-p)} \tag{13}$$

Substituting (13) into (10) gives (14), which is the membership parameter learning equation used in this paper.

$$\varepsilon_{new} = \varepsilon + \alpha\left(\frac{r(v)}{(q-p)} \times \lambda^{-1}(y_{min})\right) \tag{14}$$

In this paper, the membership parameter ε to be learnt is chosen as q. This is based on the observation, from Eq. (14), that for negative values of $r(v)$ (i.e. the action taken by the agent was undesirable), the new value ε_{new} increases if $p > q$ and reduces otherwise. Choosing to learn the value of q ensures that whenever an undesirable action is taken, the value $\Delta x = |q - p|$ is reduced (the gradient of the membership function is increased), hence making the rule under consideration less likely to be fired by inputs in the same range. On the other hand, for positive values of $r(v)$ (a good action was taken), we do increase Δx, making the rule more likely to be used by inputs in a close range. To avoid possibilities of division by zero (when $p = q$), we add a small constant δ_0 to the value $(q - p)$ in the denominator of (14). Therefore, the final membership function parameter learning rule is given in (15).

$$q_{new} = q + \alpha\left(\frac{r(v)}{(q-p) + \delta_0} \times \lambda^{-1}(y_{min})\right) \tag{15}$$

Since the crisp value of the output is based on a combination of the different input membership functions, adjustments in input MFs directly affect future outputs of the same rule. For this reason, in this proposal, the updating of the membership functions is restricted to antecedents of the rules.

5.3. Neuro-fuzzy system network structure

With the overall model defined, the next step is to design the actual neuro-fuzzy network. We propose a 3-layer feedforward network with an input layer, a rule (hidden) layer and an output layer. In Fig. 5 we show such type of network, designed for the

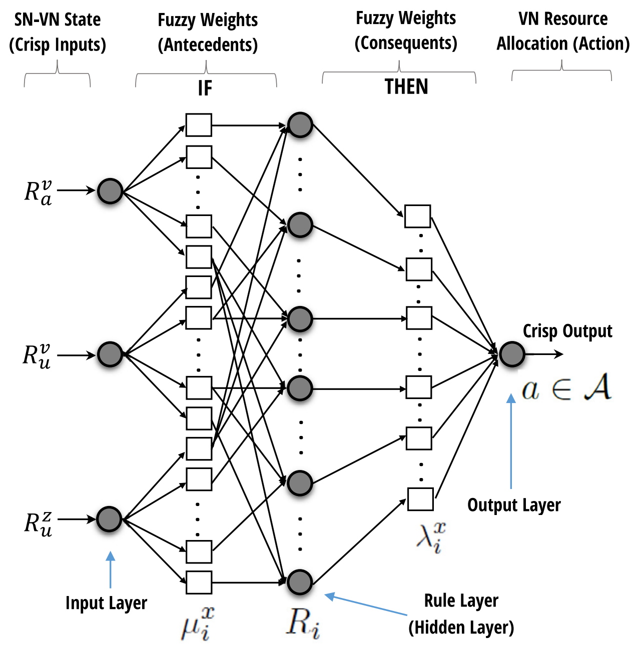
SN-VN State (Crisp Inputs)
Fuzzy Weights (Antecedents)
Fuzzy Weights (Consequents)
VN Resource Allocation (Action)
IF
THEN
R_a^v
R_u^v
R_u^z
Crisp Output
$a \in \mathcal{A}$
Output Layer
λ_i^x
Input Layer
μ_i^x
R_i
Rule Layer (Hidden Layer)

Fig. 5. Neuro-fuzzy network for VN resource allocation.

resource allocation model represented in Fig. 2. As can be seen, the input layer contains 3 neurons and has as inputs the three variables R_a^v, R_u^v and R_u^z which we use to define the state of system resources. The output layer contains a single neuron, and its output is an action $a \in \mathcal{A}$ aimed at changing resource allocation. The rule (hidden) layer contains fuzzy if-then rules that are used by the system to make resource allocation decisions. On the left side of the rule layer are fuzzy weights μ_i^x that represent the weight of each connection from an input to a rule node R_i, and using the fuzzy set x. Similarly, the right hand side of the rule layer has another set of fuzzy weights λ_i^x connecting a rule R_i to the output, represented a fuzzy set x. While it is possible for different rule nodes to share some weights on either side, the network designed in this paper creates a unique fuzzy weight for each rule. This way, the rule addition/deletion described in Section 4.2.6 involves creating/cutting connections between the appropriate input-rule-output nodes of the network. In the same way, membership function learning involves adjusting these weights for the appropriate connections/rules.

6. Rulebase initialisation

At the beginning of the learning process, the system has no rules. Therefore, we need to define a way of establishing an initial rule base. One rule initialisation possibility is a decremental rule learning proposed in Nürnberger et al. (1999) in which all the possible rules are initialised into the system and then subsequently reduced as the agent learns. As already mentioned our system can work with up to 1728 i.e. $(6 \times 6 \times 6 \times 8)$ rules, and starting the learning process with this high number of rules would slow down the learning process. Our proposal starts by creating the maximum possible rule base with 17208 rules. A weight $w_i = 0$ is then attached to each rule R_i, and is used to perform a weighting and pruning process that is based on expert knowledge. The initialisation proposed in this paper includes three sequential steps as described below.

(1) The first step takes into account the likelihood that a rule may not be required when in optimal operation. This is based on the design objectives of the system. Specifically, since we would like the system to be mindful of the QoS requirements of VNs, the system is unlikely to be in states where R_a^v = VL. Similarly, efficient substrate resource utilisation would ensure that states with R_u^v = VH and/or R_u^z = VH are less likely to occur. For each of such rules, the weight w_i is incremented by δ_1.

(2) The second takes into account the likelihood that a given rule could cause the agent to take a wrong action. Examples of such rules could be in situations where a given resource allocation is very high, but the selected action is to increase the allocation even more by a *very high* percentage. In particular, rules in which R_a^v = VH with actions PM and PL, R_a^v = VL with actions NM and NL and those R_u^v = VL with actions NM and NL are likely to lead to wrong actions. For each of such rules, the weight w_i is incremented by δ_2.

(3) In the final step, each of the rules is evaluated based on an input–output dataset, and the weight w_i adjusted again. The dataset used for this purpose was saved from the q-table of a reinforcement learning approach proposed in Mijumbi et al. (2014). This q-table was a result of a resource allocation learning system for a similar resource allocation task and it gives the state-action-values for the learning task. The table is made up of 3 columns, one for the *state* (which is also defined by three variables R_a^v, R_u^v and R_u^z), another for a possible *action*, and the other for a *value* that shows the desirability of taking the action while in the given state. However, before the dataset can be used for the pruning proposed in this paper, we need to process it so as to put in a form similar to fuzzy rules. In Section 6.1, we describe the preprocessing steps taken. After this process, all rules for which the weight w_i is greater than a pre-established constant Ψ_2 are pruned from the rule base.

6.1. Dataset preprocessing

The training dataset is made up of 4608 entries (resulting from 512 possible states and 9 possible actions), each showing the value of every possible action while in each state. The first step is to choose only those entries corresponding to the best possible action (actions with the best values) for each state. This leaves us with 512 entries. We then convert these state-action-values into fuzzy rules. However, this requires a mapping from the state and action codes used in Mijumbi et al. (2014) to the fuzzy sets used in this paper. In Table 2 (a) and (b) we show the mapping that has been performed on the states and actions. However, since each of the state MFs can only take on 6 values for the 3 input variables, the maximum number of possible *unique* antecedent combinations is $6 \times 6 \times 6 = 216$. Therefore, we again prune the training dataset eliminating entries with the same antecedent (remaining with one rule with the highest original state-action-value for all duplicate rules). After this final step, we have a training rule set R_{RL} with 216 rules that we can use as a training set for an initial pruning of the rule base.

6.2. Initial rule base pruning

For each rule $r_j \in R_{RL}$, each rule $R_i \in R$ in the rule base is evaluated so as to determine the correctness of its consequent. To this end, a *rule matching* procedure is performed. This determines whether the antecedent of r_j is the same as that of R_i. The rule matching step involves comparing the fuzzy sets representing the input and/or output variables. For this initialisation step, two rules match if all the three corresponding antecedent fuzzy sets are the same. The result of a rule matching process is either a success if the rules match, or a failure otherwise. If R_i matches r_j then an *error*, which is a measure of how the consequent of r_j differs from that of R_i, is evaluated. To achieve this, we model each of the 8 possible consequents with an integer value. These values

Table 2
Action and state to membership function mapping.

(a) Previous action (%)	MF
Maintain R_a^v unchanged	PZ
Decrease R_a^v by 50.0	NL
Decrease R_a^v by 37.5	NM
Decrease R_a^v by 25.0	NS
Decrease R_a^v by 17.5	NZ
Increase R_a^v by 17.5	PZ
Increase R_a^v by 25.0	PS
Increase R_a^v by 37.5	PM
Increase R_a^v by 50.0	PL
(b) Previous state (% value)	**MF**
$0 <$ Variable $\leqslant 12.5$	VL
$12.5 <$ Variable $\leqslant 25$	L
$25 <$ Variable $\leqslant 37.5$	LM
$37.5 <$ Variable $\leqslant 50$	LM
$50 <$ Variable $\leqslant 67.5$	HM
$67.5 <$ Variable $\leqslant 75$	HM
$75 <$ Variable $\leqslant 87.5$	H
$87.5 <$ Variable $\leqslant 100$	VH

Table 3
Output membership function to integer mapping.

NL	NM	NS	NZ	PZ	PS	PM	PL
−4	−3	−2	−1	1	2	3	4

are shown in Table 3. The *absolute* value, of the difference between the value a_j for the consequent of rule r_j and a_i for rule R_i is then used to increment the weight w_i for rule R_i.

We show the pseudocode for the rule initialisation in Algorithm 1. In the algorithm, $\mu_i^1, \mu_i^2, \mu_i^3$ and λ_i^1 are the *initial* fuzzy sets for the variables R_a^v, R_a^v, R_a^v and O respectively. The fuzzy sets shown Figs. 3 and 4 are used for the initialisation stage, but as learning progresses, these sets change for the respective rules. After the weighting and pruning stage, the proposed initialisation algorithm reduces the initial 1728 rules to 1215 rules.[4]rulebase

Algorithm 1. Rule base initialisation

```
Initial Rule Base R Creation
1: Initialise rule weight: $i = 1$
2: for $\mu_i^1 \in \mu$ do
3:   for $\mu_i^2 \in \mu$ do
4:     for $\mu_i^3 \in \mu$ do
5:       for $\lambda_i^1 \in \lambda$ do
6:         Create Rule: $R_i$ if ($R_a^v$ is $\mu_i^1$ and $R_u^v$ is $\mu_i^2$
7:           and $R_u^z$ is $\mu_i^3$) then $O$ is $\lambda_i^1$
8:         Initialise weight: $w_i = 0$
9:         Increment rule weight: $i{+}{+}$
10:      end for
11:    end for
12:  end for
13: end for
   Rule Weighting and Pruning
14: for $R_i \in R$ do
    Step 1: Efficiency and QoS Awareness
15:   if ($\mu_i^1 = VL$) or ($\mu_i^2 = VH$) or ($\mu_i^3 = VH$) then
16:     $w_i = w_i + \delta_1$
17:   end if
    Step 2: Protection against wrong actions
18:   if
19: ($\mu_i^1$ = VH and $\lambda_i^1$ = PM) or...or ($\mu_i^2$ = VL and $\lambda_i^1$ = NL)
20:   then
21:     $w_i = w_i + \delta_2$
22:   end if
    Step 3: Learning from Dataset
23:   for $r_j \in R_{RL}$ do
24:     Perform Rule Matching $(r_j, R_i)$
25:     if $(Matching = Success)$ then
26:       $w_i = w_i + (|o_j - o_i|)$
27:       if $w_i >= \Psi_2$ then
28:         DELETE $R_i$
29:       end if
30:     end if
31:   end for
32: end for
```

6.3. Time complexity: rule base initialisation

The initial rule base creation stage in Lines $2-13$ involves three basic operations (Lines 6, 8 and 9) for each of the possible rules R. Therefore, this step can be performed in time $O(|3R|)$. The rule matching operation involves at most four operations (3 for the antecedent and 1 for the consequent). Therefore, Lines $24-30$ includes at most six operations, implying that the for loop from Lines $23-31$ runs in time $O(|6R_{RL}|)$. Including the two operations on Lines 16 and 21 leads to the time $O(|(6R_{RL}+2)R|)$ for the Lines $14-32$. Therefore, the dominating factor in Algorithm 1 is $O(|R_{RL} \times R|)$. It is worth noting that both $|R|$ and $|R_{RL}|$ are predefined as 1728 and 216 respectively.

7. Agent cooperation

In this paper, the substrate node or link agents can cooperate on two fronts. The first is an action coordination aimed at conflict prevention, while the other is a knowledge sharing aimed at learning enhancement. We briefly describe both of them below.

7.1. Coordination among agents

From the VNE problem formulation, a given virtual link l_{ij} may be mapped onto more than one substrate link. This creates a possibility of more than one substrate link agent dynamically managing the resources allocated to such a virtual link. In this case, the set of agents $L_a^{l_{ij}} \subset \mathcal{L}_a$ that are able to change the resource allocation to l_{ij} must coordinate their actions to avoid conflicting resource allocations. The first step in the conflict prevention proposed in this paper is the creation of the agent set $L_a^{l_{ij}}$. After every VNE step, each substrate link agent that participated in the embedding determines – for each new embedded virtual link – the set of other substrate link agents that manage the virtual link resources. Since we consider that all the agents in our model belong to the same organisation (the infrastructure provider), we consider that this kind of information is readily available to all agents. The next step is to allow each agent $l_a \in L_a^{l_{ij}}$ to communicate with every other agent in the same set, sharing resource allocation information every time an allocation is performed. Therefore, after each learning episode, if an agent $l_a \in L_a^{l_{ij}}$ decides to change the amount of resources allocated to l_{ij}, it sends an update to other agents in $L_a^{l_{ij}}$ providing information about the action a to be taken as well as the final percentage resource allocation $R_a^{l_{ij}}$ resulting from the action. All the agents in $L_a^{l_{ij}}$ therefore perform the resource change at the same time. Once again, the fact that a given agent is able to trust and take actions based on decisions of another agent is reasonable since all these agents belong to the same organisation and as such, they cannot have conflicting objectives. Finally, in order to ensure that the actions and information sharing of the agents $L_a^{l_{ij}}$ is synchronised, 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.

7.1.1. Scalability consideration

It is worth noting that in general, if the link mapping algorithm is efficient, the agent set $L_a^{l_{ij}}$ will contain an average of $2-3$ agents.[5] This means that at any point, a given agent $l_a \in L_a^{l_{ij}}$ 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, we specifically avoid the

[4] The initial rule base, training dataset and final initialised rule base can be downloaded from: http://www.maps.upc.edu/rashid/files/nfsrules.rar.

[5] Based on simulations carried out using the S-OS algorithm (see Table 7) for average link bandwidth utilisation levels between 50% and 70%.

exchange of “acknowledge” messages to diminish as much as possible the traffic among agents, instead preferring to use an update message that also includes the final resource allocation to l_{ij}. This ensures that if for any reason a given agent does not get an update message, this can be corrected at the next learning episode.

7.2. Knowledge sharing among agents

The second form of cooperation between agents involves sharing of their knowledge bases. In this proposal, each learning agent $a_s \in (\mathcal{L}_a \cup \mathcal{N}_a)$ periodically shares its rule base R_{a_s} as well as database of membership functions f_{a_s} with other agents $a_t \in A_t$, where $A_t \subset (\mathcal{L}_a \cup \mathcal{N}_a)$. There are two advantages that are derived from this cooperation. First, it leads to performance enhancement if it leads to addition of new knowledge to the base or to improvement of the membership functions of existing rules and then it allows a faster convergence to optimal network structure in case it leads to deletion of some rules. For each newly received rule $R^i_{a_s} \in R_{a_s}$, the integration into the rule base is a four-step matching and elimination process. The receiving agent a_t compares $R^i_{a_s}$ to each of the rules in its rule base, performing a rule matching (described in Section 6.2) over both the input and output variable fuzzy sets. If the result of matching for the whole rule base is a failure (the rule $R^i_{a_s}$ does not match any of the rules in the rule base), then the agent adds the rule $R^i_{a_s}$ together with its membership functions to its knowledge base. On the other hand, if the result of matching is a success (there is a rule that is similar to $R^i_{a_s}$), then the agent learns from the membership functions of $R^i_{a_s}$. This is achieved by replacing the $p-values$ of the membership functions of the matching rule R_i with a weighted sum defined in (16).

$$p_{new} = (\gamma \times p_1) + (\beta \times p_2) \tag{16}$$

where p_1 is the old *p-value*and p_2 is the *p-value* of the received rule. γ and β are constants intended to bias the sensitivity of the agent to knew information. $\gamma + \beta = 1$. In this paper, the values $\gamma = 0.7$ and $\beta = 0.3$ are used. Needless to mention, the overall learning scheme proposed in this paper learns both the p and q parameters of the membership functions. The q parameter is learnt through the evaluative feedback described in Section 5.2 while this subsection has defined a learning procedure for the p values.

Algorithm 2. Neuro-fuzzy learning algorithm

```
Initialisation
 1: Initialise Rule Base, $R$
 2: Determine current state, $s_c$
 3: Define: previous state, $s_p = s_c$, previous action, $a_p = 0.0$,
    next state, $s_n = s_c$, set of fired rules $F = \emptyset$.
  Thread1: Learn from others (Knowledge Sharing)
 4: repeat
 5:     Wait(Cooperation Interval)
 6:    Receive rule base $R_x$ from other agents
 7:    for $R_j \in R_x$ do
 8:      if $R_j \in R$ then
 9:        for $R_i \in R$ do
10:          Update $p_i$ as described in Section 7.2
11:        end for
12:      else
13:        Add $R_j$ to $R$
14:      end if
15:    end for
16: until Learning is stopped
  Thread2: Learn from actions (Evaluative Feedback)
17: repeat
18:    Wait(Learning Interval)
19:    Read $s_p$, $a_p$, $s_n$, $F$
20:    Determine $r(v)$ using Eq. (6)
21:    for $r_i \in F$ do
22:      for $\lambda_i^x \in r_i$ do
23:        Determine $q_{new}^{\lambda_i^x}$ using Eq. (15)
24:      end for
25:      Update weight $w_i$ as using ARW in Section 4.2.6
26:    end for
27:    Set $F = \emptyset$
28:    for $R_i \in R$ do
29:      if $w_i \geqslant \Psi_1$ then
30:        Delete $R_i$
31:      end if
32:    end for
33:    Determine current state $s_c$, add all fired rules to $F$
34:    Determine action, $a \in A$ as explained in Sections 4.2.3–
       4.2.5
35:    Take action $a$, and determine next state, $s'_n$
36:    Set $s_p = s_c$, $a_p = a$, $s_n = s'_n$
37: until Learning is stopped
```

7.2.1. Scalability consideration

Allowing every agent $a_s \in (\mathcal{L}_a \cup \mathcal{N}_a)$ to communicate and share knowledge with every other agent in the system would be non-scalable. In this paper, we restrict each agent to only share knowledge with its direct neighbours. This means that any link agent $l_a \in \mathcal{L}_a$ can only share experience with at most two node agents at either end of the link. In the same way, any given node agent $n_a \in \mathcal{N}_a$ can only share knowledge with only those link agents that directly connect it to its adjacent nodes.

The complete learning algorithm proposed in this paper is shown in Algorithm 2. As can be seen, the learning process is made up of different steps as already described. However, we note that

Table 4
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 address 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 5
Brite network topology generation 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

some of the processes take place in parallel, for example, the agents continuously learn from each other (knowledge sharing) independently of the learning achieved from the evaluative feedback.

7.3. Time complexity: neuro-fuzzy learning algorithm

The initialisation in Line 1 can be performed in time $O(|R_{RL} \times R|)$ as established in Section 6.3. Lines $7-15$ require at most $O(|R|^2)$ time, while the for loop in lines $21-26$ can run in time $O(|5R|)$. Finally, Lines $28-32$ can be performed in time $O(|R|)$. Since $R > R_{RL}$, the overall time complexity of Algorithm 2 is $O(|R|^2)$. Therefore, both algorithms proposed in this paper run in polynomial time.

8. Performance evaluation

8.1. Simulation model

The simulation model used in this paper is based on Fig. 1. We start by creating a SN topology, and thereafter VN requests arrive one at a time to the SN. The substrate and virtual network topologies are generated using Brite (Medina, Lakhina, Matta, & Byers, 2001) with settings shown in Table 5. Whenever a virtual network request is accepted by the substrate network, the virtual network topology is created in NS3 (2014) (we added a network virtualisation module to NS3 based on parameters in Table 4). Our NS3 Module allows us to create a traffic application for each accepted VN request, and the traffic application starts transferring packets over the virtual network. The traffic application generates packets based on real traffic traces from CAIDA anonymised Internet traces (CAIDA, 2014). 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 (CAIDA, 2014). The trace source used in this paper was collected on 20th December 2012 and contains over 3.5Million packets. We divide these packets among 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 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 4.00 GB RAM and 3.00 GHz CPU specifications.

8.2. 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 6. Link delays are as determined by Brite. Each virtual node is allowed to be located within a uniformly distributed distance $75 \leqslant x \leqslant 150$ 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 min and is assumed to follow a negative exponential distribution.

Table 6
Substrate and virtual network 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	2.0 Mbps	1.0 Mbps
Maximum link bandwidth	10.0 Mbps	2.0 Mbps

Table 7
Compared algorithms.

Code	Resource allocation approach
D-NFS	Dynamic, based on neuro-fuzzy system [our contribution]
D-RL	Dynamic, based on reinforcement learning (Mijumbi et al., 2014)
S-CNMMCF	Static, coordinated node mapping and MCF for link mapping (Chowdhury et al., 2012)
S-OS	Static, link based optimal one shot virtual network embedding (Mijumbi et al., 2014)

8.3. Comparison against alternatives

We compare the performance of our proposed solution with closely related solutions. In particular, three representative solutions from the literature are chosen. The first performs a dynamic resource allocation using one shot VNE for the first step and reinforcement learning for resource management (Mijumbi et al., 2014); the second is a static allocation approach that performs a coordinated node and link mapping (Chowdhury et al., 2012); and the third is also a static baseline formulation that performs a one shot mapping, and also used in performance evaluations in Mijumbi et al. (2014). The solution in (Chowdhury et al., 2012) was adapted to fit into our formulation of the problem. In particular, for (Chowdhury et al., 2012) 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 solutions in Table 7. We also compare different variations of our proposal to determine the effect of initialising rule bases as well as sharing knowledge between the agents. Details of these variations are shown in Table 8.

8.4. Performance metrics

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

8.4.1. Embedding quality

We define embedding quality as a measure of how efficiently the algorithm uses the substrate network resources for accepting virtual network requests. This is evaluated using the acceptance ratio and the 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

Table 8
Compared approaches – Initialisation and agent cooperation.

Code	Resource allocation approach
D-RL	Reinforcement learning (Mijumbi et al., 2014)
I-C	Initialised rule base, cooperating agents
I-NC	Initialised rule base, non cooperating agents
NI-C	Non initialised rule base, cooperating agents
NI-NC	Non initialised rule base, non cooperating agents

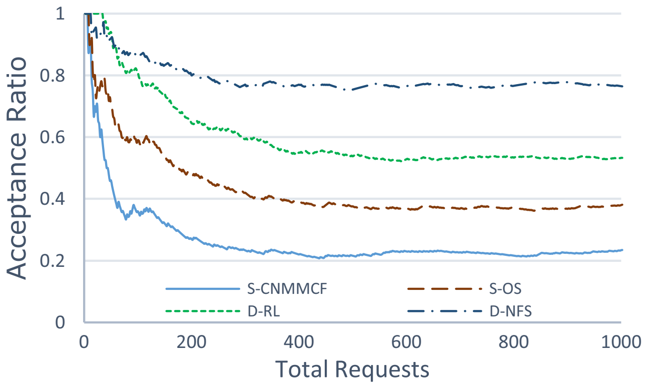

Fig. 6. VN Acceptance ratio.

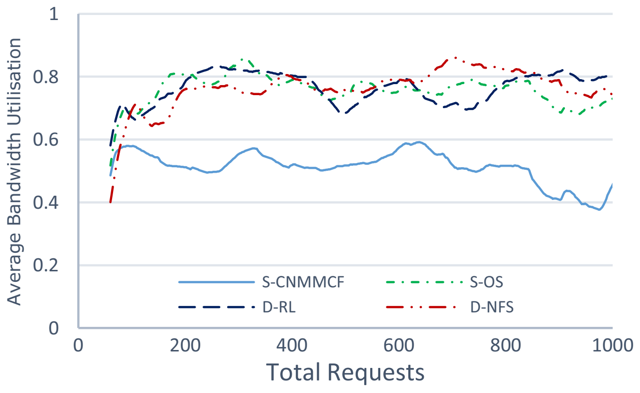

Fig. 8. Average SN bandwidth utilisation.

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.

8.4.2. 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 4, 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.

8.5. Discussion of results

The simulation results are shown in Figs. 6–14. As can be seen from Fig. 6, while both dynamic approaches perform better than the static ones in terms of virtual network acceptance ratio, the neuro-fuzzy 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 NFS outperforms the RL approach can be attributed to two factors: (1) the NFS system models the states and actions with better granularity i.e. without restricting the states and actions to few discrete levels, and (2) the NFS system is more dynamic in the sense that it continuously changes its knowledge base by adding rules, modifying them, and deleting others. 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. In addition, the link mapping phases could potentially use more resources than if both steps are performed once. A similar performance pattern can be noted from Fig. 9 which further confirms that at any given point, the substrate networks that dynamically manage resources are able to embed more VNs than the static ones, and that D-NFS performs better than D-RL and S-OS better than S-CNMMCF. Figs. 7 and 8 show the average utilisation of substrate node queue and link bandwidth respectively. It can be observed that except for S-CNMMCF, the other three approaches on average use the same amount of substrate network resources. The fact the 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-NFS all have on average the same utilisation is mainly due to all of them having the same initial mapping algorithm (which is S-OS). However the interesting point from looking at Figs. 6–9 is that while S-OS, D-RL and D-NFS all have a similar resource utilisation levels, D-NFS uses these resources to serve a higher number of VNs at any given time, which confirms the extra efficiency introduced by the proposed approach.

Fig. 10 shows that S-OS has an almost constant packet drop rate while that for D-RL and D-NFS is initially high, but gradually

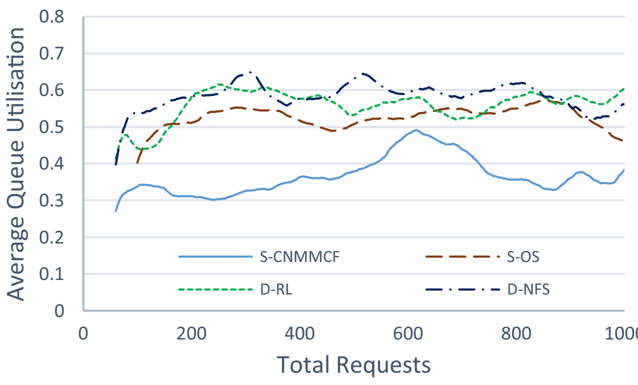

Fig. 7. Average SN queue size utilisation.

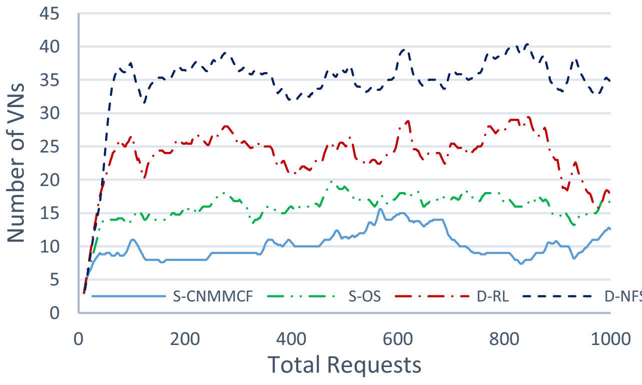

Fig. 9. Number of accepted virtual networks.

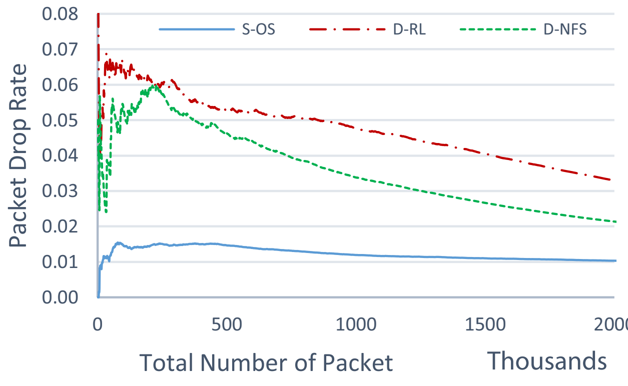

Fig. 10. Node packet drop rate.

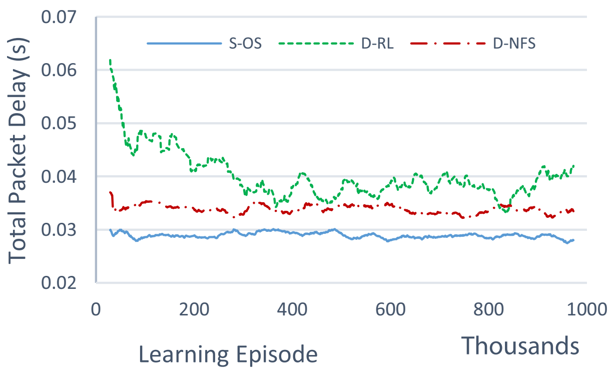

Fig. 11. Link packet delay.

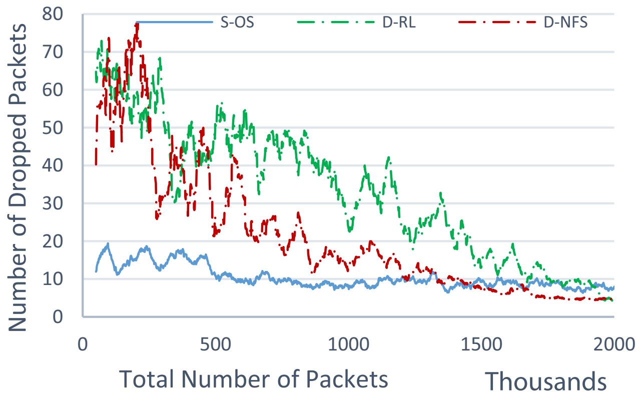

Fig. 13. Node packet drop rate variation.

reduces. The fact that the dynamic approaches initially perform badly is expected, since, at the beginning of the learning processes the agents may vary the queue sizes quite considerably leading to more packet drops. This high initial packet drop also affects the overall speed at which the drop rate converges to the one in the static approach. This can be confirmed by noting from Fig. 13, that in fact, the periodic drops in packets by all approaches finally converge. Once again, the fact that D-NFS has a lower packet drop rate than D-RL over the learning period can be explained since D-NFS has better granularity in perceiving the state of resources and allocation. In a similar way, Fig. 11 shows that the packet delays for the two dynamic approaches is initially higher but reduces over the learning period, while Section 14 shows that in fact the variations in this delay converge to the static approach. Again, these differences are attributed to the initial learning phase, and the difference in D-RL and D-NFS is due to better options in perception and action for D-NFS.

Finally, Section 8 shows how fast the agents learn optimal rules (for D-NFS) and policy (for D-RL). The actions of the agents are compared with *optimal actions*. An optimal action for an agent is that action that would lead to a resource allocation equal to what the network is actually using Mijumbi et al. (2014). The deviations in these evaluations are therefore with reference to actual resource usage in a similar network that is not performing dynamic allocations. The approaches compared in this regard are shown in Table 8. Except for D-RL, all other entries are variations of D-NFS. It can be observed that in general, after convergence, the actions from D-NFS have a slightly higher optimality than those from D-RL. As earlier explained, this is expected due to the granularity of actions taken by D-NFS. However, we also note that D-RL converges to optimal actions slightly faster than D-NFS. This can be explained by the fact that D-RL has a far less state space to learn than D-NFS (due to discretisation). We also note from the graphs that the approaches proposed for rule base initialisation and agent cooperation do enhance the speed at which the agents' actions converge optimal ones. However, as can be seen from the graphs, even without rulebase initialisation and no cooperation between the agents (NI-NC), the actions would finally converge to the optimal ones, albeit at a much later stage than others.

9. Related work

9.1. Static virtual network embedding

In order to cope with the hardness of the VNE problem, some existing research on VNE (Lu & Turner, 2006; Zhu & Ammar,

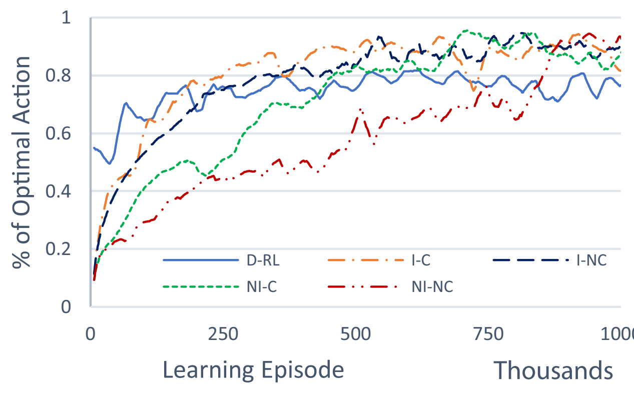

Fig. 12. Initialisation and agent cooperation.

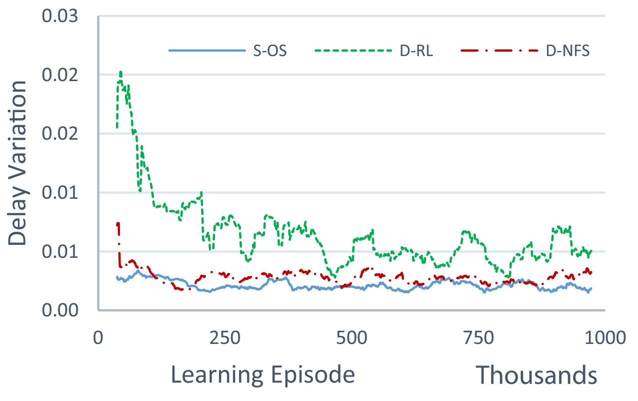

Fig. 14. Link packet delay variation.

2006) assume that all VN requests are known in advance, others (Fan & Ammar, 2006; Lu & Turner, 2006) ignore constraints on virtual nodes and links, while all of Fan and Ammar (2006), Lu and Turner (2006) and Zhu and Ammar (2006) perform the node and link embedding in two uncoordinated steps and assume that the substrate network has infinite capacity to accept all VN requests. VINEYard (Chowdhury et al., 2012) proposes a linear programming based node mapping algorithm that coordinates both node and link mapping. A distributed one-shot embedding solution based on a multi-agent system is proposed in Houidi, Louati, and Zeghlache (2008), while Infhr and Raidl (2011) and Jarray and Karmouch (2013) propose mathematical programming based solutions to VNE. A common feature of all these approaches is that their focus is on the mapping of virtual nodes and links to substrate nodes and paths i.e virtual network embedding. We remark once more that virtual network embedding is not the focus of this paper, and that unlike the proposal in this paper, they all do not make any changes to virtual resource allocations through out the lifetime of the VNs.

9.2. Dynamic virtual network embedding

The authors in Cai, Liu, Xiao, Liu, and Wang (2010) and Marquezan, Granville, Nunzi, and Brunner (2010) study the VN embedding problem when the substrate network is dynamically changing. We differ from these works in that our consideration is on the changes in actual loadings of the virtual networks, rather than on a changing substrate network. In Yu, Yi, Rexford, and Chiang (2008), 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 Rahman and Boutaba (2013) 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 Fajjari, Aitsaadi, Pujolle, and Zimmermann (2011) 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. Sun, Yu, Anand, and Li (2013) proposes algorithms for the problem of efficiently reconfiguring 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. 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. Zhang, Qian, Tang, Wu, and Lu (2011, 2012) proposes opportunistic sharing of substrate network resources among different virtual networks, while Mijumbi et al. (2014) proposes a multi-agent based q-learning for a dynamic and decentralised resource allocation in virtual networks. The work in this paper 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. Our proposals also consider a complete network (not a single node or link as in some works), through out its lifetime.

9.3. Dynamic resource management

Most existing works on dynamic resource management are based on three approaches: control theory, performance dynamics modelling and workload prediction. Pan, Mu, Wu, and Yao (2008) and Patikirikorala, Colman, Han, and Wang (2011) are control theoretic approaches while Han, Guo, Ghanem, and Guo (2012) and Lai, Chiang, Lee, and Lee (2013) are based on performance dynamics. The authors in Hu, Wong, Iszlai, and Litoiu (2009) and Jokhio, Ashraf, Lafond, Porres, and Lilius (2013) use workload prediction. An approach that uses fuzzy neural methodology for joint radio resource management is proposed in Giupponi, Agust, Prez-Romero, and Sallent (2008). The major differences between our approach and these works is mainly 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. 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.

9.4. Neuro-fuzzy systems

With regard to NFSs, the most closely related works to our proposal are in (Nauck & Kruse, 1992; Nürnberger et al., 1999). The NEFCON model (Nürnberger et al., 1999) consists of 3 layer units, and weights that based on fuzzy sets. It proposes a neuro-fuzzy system and its implementation in the area of control theory. The model learns and optimizes the rule base of a Mamdani like fuzzy controller online by a reinforcement learning algorithm that uses a fuzzy error measure. In a related work, Nauck and Kruse (1992) also uses a fuzzy error propagation approach to adapt membership functions in a fuzzy set based fuzzy control environment by use of neural network learning principles. Like the proposal in this paper, both these approaches use 3-layer feedforward neural networks whose structures and weights are represented by fuzzy rules. In addition, both use reinforcement based error functions for adjusting the network weights, so as to achieve network learning. In a later effort, Nürnberger (2001) presents a hierarchical recurrent neuro-fuzzy model for time series prediction and analysis of dynamic systems, which reduces the complexity of the structure by allowing for information storage of prior system states internally. The first difference between these works and our work is that in this paper, the learning environment contains multiple learning entities, which is dictated by the problem considered in this paper and the proposed distributed model of the substrate network. Having multiple learning entities in a single system presents several challenges among which includes managing possible conflicting actions as well as the need for cooperation to benefit from actions of others. In addition, the specific application domain of VN resource allocation requires problem specific modelling of input–output relationships, knowledge base, reward function and learning algorithm. To the best of our knowledge, each of these aspects of the problem cannot be trivially adapted from any related works.

10. Conclusion

This paper has proposed an autonomous system that uses a combination of neural networks, fuzzy systems and reinforcement learning to achieve dynamic self-management of resources in network virtualisation. We modelled the substrate network as a distributed system of autonomous, adaptive and cooperative intelligent agents which – unlike previous works – dynamically adapts the actual resource allocations to virtual networks to perceived demand for them. We have proposed a range of algorithms starting from a hybrid initialisation of the knowledge base of these agents, a self-governing cooperation of the multi-agent system, and the general reinforcement learning based algorithm for continuously changing the structure and weights of the designed neuro-fuzzy network. We have been able to evaluate the proposed algorithms through comparisons with state-of-the-art approaches and been able to show that our proposals lead to better utilisation of substrate network resources by accepting about 23% more virtual network requests, which would directly translate into increased revenue for the infrastructure providers. We have also shown that after the learning process, the agents are able to ensure that the QoS requirements of the virtual networks are not negatively impacted.

In future, we will try to enhance the convergence speed of the proposed algorithms, say, by creating an initial offline learning step. We also intend to extend this proposal to the multi-InP environment, which raises more questions especially with regard to cooperation and trust between agents as well as the need for negotiation. It would also be interesting to study how the rules in each agent vary in a given agent over time, with the aim of determining possibilities for enhancing the effect of agent cooperation.

However, there are practical questions that still need to be answered. For example, the ease of representing each network node or link with an agent, as well the actual over-head that would result in message exchanges between these agents. There is also a need to have service level agreements between substrate and virtual network owners that allows for continuous adjustments to resource allocations. As an initial step, we are studying the possibilities of implementing our proposals in a real network, by setting up a server to collect virtual network requirements and user traffic characteristics, and using this in a prototype LAN using the Java Agent Development Framework (JADE) (Bellifemine, Caire, & Greenwood, 2007).

Acknowledgments

This work was partly funded by the EVANS project (PIRSES-GA-2010-269323), project TEC2012-38574-C02-02 from Ministerio de Economia y Competitividad, and Flamingo (318488) – a Network of Excellence project supported by the European Commission under its 7th FP.

References

Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE*. Wiley.

Cai, Z., Liu, F., Xiao, N., Liu, Q., & Wang, Z. (2010). Virtual network embedding for evolving networks. In *GLOBECOM* (pp. 1–5). IEEE.

CAIDA (2014). 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 17.02.14.

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

Dayhoff, J. E., & DeLeo, J. M. (2001). Artificial neural networks: Opening the black box. *Cancer, 91*, 1615–1635.

Even-Dar, E., & Mansour, Y. (2004). Learning rates for q-learning. *The Journal of Machine Learning Research, 5*, 1–25.

Fajjari, I., Aitsaadi, N., Pujolle, G., & Zimmermann, H. (2011). Vnr algorithm: A greedy approach for virtual networks reconfigurations. In *GLOBECOM* (pp. 1–6). IEEE.

Fan, J., & Ammar, M. H. (2006). Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *Proceedings of IEEE INFOCOM*.

Fischer, A., Botero, J., Till Beck, M., de Meer, H., & Hesselbach, X. (2013). Virtual network embedding: A survey. *IEEE Communications Surveys Tutorials, 15*, 1888–1906.

Giupponi, L., Agust, R., Prez-Romero, J., & Sallent, O. (2008). A novel approach for joint radio resource management based on fuzzy neural methodology. *IEEE Transactions on Vehicular Technology, 57*, 1789–1805.

Han, R., Guo, L., Ghanem, M., & Guo, Y. (2012). Lightweight resource scaling for cloud applications. In *2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid)* (pp. 644–651).

Houidi, I., Louati, W., & Zeghlache, D. (2008). A distributed virtual network mapping algorithm. In *ICC* (pp. 5634–5640). IEEE.

Hu, Y., Wong, J., Iszlai, G., & Litoiu, M. (2009). Resource provisioning for cloud computing. In *Conference of the center for advanced studies on collaborative research* (pp. 101 –111).

IBM (2014). IBM ILOG CPLEX Optimizer. <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/about/> Accessed 17.02014.

Infhr, J., & Raidl, G. R. (2011). Introducing the virtual network mapping problem with delay, routing and location constraints. In J. Pahl, T. Reiners, & S. Vo (Eds.), *INOC. Lecture notes in computer science* (Vol. 6701, pp. 105–117). Springer.

Jarray, A., & Karmouch, A. (2013). Vcg auction-based approach for efficient virtual network embedding. In *2013 IFIP/IEEE international symposium on integrated network management (IM 2013)* (pp. 609–615).

Jokhio, F., Ashraf, A., Lafond, S., Porres, I., & Lilius, J. (2013). Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *Proceedings of the 2013 21st Euromicro international conference on parallel, distributed, and network-based processing PDP '13* (pp. 254–261). Washington, DC, USA: IEEE Computer Society.

Kasabov, N. K. (1996). *Foundations of neural networks, fuzzy systems, and knowledge engineering* (1st ed.). Cambridge, MA, USA: MIT Press.

Lai, W.-S., Chiang, M.-E., Lee, S.-C., & Lee, T.-S. (2013). Game theoretic distributed dynamic resource allocation with interference avoidance in cognitive femtocell networks. In *WCNC* (pp. 3364–3369). IEEE.

Lu, J., & Turner, J. (2006). Efficient mapping of virtual networks onto a shared substrate. Technical Report. Washington University, St. Louis.

Marquezan, C., Granville, L., Nunzi, G., & Brunner, M. (2010). Distributed autonomic resource management for network virtualization. In *2010 IEEE network operations and management symposium (NOMS)* (pp. 463–470).

Medina, A., Lakhina, A., Matta, I., & Byers, J. (2001). 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* (pp. 346–353). Washington, DC, USA: IEEE Computer Society.

Mijumbi, R., Gorricho, J., Serrat, J., Claeys, M., De Turck, F., & Latre, S. (2014). 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*. IEEE Press.

Nauck, D., & Kruse, R. (1992). A neural fuzzy controller learning by fuzzy error propagation. In *Proc. NAFIPS'92* (pp. 388–397).

NS3 (2014). Network simulator 3. <http://www.nsnam.org/> Accessed 17.02-14.

Nürnberger, A. (2001). A hierarchical recurrent neuro-fuzzy system. *Joint 9th IFSA world congress and 20th NAFIPS international conference* (vol. 3, pp. 1407–1412). IEEE.

Nürnberger, A., Nauck, D., & Kruse, R. (1999). Neuro-fuzzy control based on the nefcon-model: Recent developments.

Pan, W., Mu, D., Wu, H., & Yao, L. (2008). Feedback control-based qos guarantees in web application servers. In *HPCC* (pp. 328–334). IEEE.

Patikirikorala, T., Colman, A., Han, J., & Wang, L. (2011). A multi-model framework to implement self-managing control systems for qos management. In *Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems SEAMS '11* (pp. 218–227). New York, NY, USA: ACM.

Qi-ming, F., Quan, L., Zhi-ming, C., & Yu-chen, F. (2009). A reinforcement learning algorithm based on minimum state method and average reward. In *World congress on computer science and information engineering* (Vol. 5, pp. 534–538).

Rahman, M. R., & Boutaba, R. (2013). Svne: Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management, 10*, 105–118.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Smith, S. W. (1999). *The scientist and engineer's guide to digital signal processing*. San Diego, CA, USA: California Technical Publishing.

Stone, P., & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots, 8*, 345–383.

Sun, G., Yu, H., Anand, V., & Li, L. (2013). A cost efficient framework and algorithm for embedding dynamic virtual network requests. *Future Generation Computer Systems, 29*, 1265–1277.

Yu, M., Yi, Y., Rexford, J., & Chiang, M. (2008). Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Computer Communication Reviews, 38*, 17–29.

Zhang, S., Qian, Z., Tang, B., Wu, J., & Lu, S. (2011). Opportunistic bandwidth sharing for virtual network mapping. In *Global telecommunications conference (GLOBECOM 2011)* (pp. 1–5). IEEE.

Zhang, S., Qian, Z., Wu, J., & Lu, S. (2012). An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In K. Sohraby & A. G. Greenberg (Eds.), *INFOCOM* (pp. 2408–2416). IEEE.

Zhu, Y., & Ammar, M. (2006). Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM 2006. 25th IEEE international conference on computer communications. Proceedings* (pp. 1–12).