Computer Communications 154 (2020) 331–346

Contents lists available at ScienceDirect

Computer Communications

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

Wireless control using reinforcement learning for practical web QoE

Henrique D. Moura *, Daniel F. Macedo, Marcos A.M. Vieira

Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, MG, Brazil

ARTICLE INFO

Keywords:
Wireless network
Software defined network
Reinforcement learning
Q-Learning
Multi-armed bandit
Quality of Experience

ABSTRACT

Wireless networks show several challenges not found in wired networks, due to the dynamics of data transmission. Besides, home wireless networks are managed by non-technical people, and providers do not implement full management services because of the difficulties of manually managing thousands of devices. Thus, automatic management mechanisms are desirable. However, such control mechanisms are hard to achieve in practice because we do not always have a model of the process to be controlled, or the behavior of the environment is dynamic. Thus, the control must adapt to changing conditions, and it is necessary to identify the quality of the control executed from the perspective of the user of the network service. This article proposes a control loop for transmission power and channel selection, based on Software Defined Networking and Reinforcement Learning (RL), and capable of improving Web Quality of Experience metrics, thus benefiting the user. We evaluate a prototype in which some Access Points are controlled by a single controller or by independent controllers. The control loop uses the predicted Mean Opinion Score (MOS) as a reward, thus the system needs to classify the web traffic. We proposed a semi-supervised learning method to classify the web sites into three classes (light, average and heavy) that groups pages by their complexity, i.e. number and size of page elements. These classes define the MOS predictor used by the control loop. The proposed web site classifier achieves an average score of $87\% \pm 1\%$, classifying 500 unlabeled examples with only fifteen known examples, with a sub-second runtime. Further, the RL control loop achieves higher Mean Opinion Score (up to 167% in our best result) than the baselines. The page load time of clients browsing heavy web sites is improved by up to 6.6x.

1. Introduction

Over the past 20 years, IEEE 802.11-based (Wi-Fi) wireless networks have become commonplace on office and campus environments, and it is the most common method of Internet access for home stations [1]. However, the wireless medium is subject to performance problems, such as packet loss, delay, and low connection speed, often caused by interference of other Wi-Fi networks, signal propagation issues, and misconfiguration. Vendors sell commercial wireless controllers that centralize Wireless Local Area Network (WLAN) configuration, but they are closed platforms, which rely on the vendor's initiative to implement new features. Furthermore, network operations and management remain cumbersome. Comsa et al. [2] highlight that the main challenge in wireless networks is to achieve higher user satisfaction. Furthermore, it is important to provide a good QoE for the user, because 40% of visitors will abandon a web site that takes more than 3 s to load, and from those, 80% will not return [3]. As a consequence, expert management systems are required to compensate for the dynamic nature of the wireless medium, and to maintain an acceptable user experience at all times, demanding little to no human interaction [4].

Auto-configuration and auto-optimization capabilities can be added to the wireless network using Software Defined Networking (SDN) with minimal or no human administration [4,5]. There are two prerequisites for auto-optimization: a control loop algorithm that reads the environment and acts on the network, and metrics representative of the QoE experienced by the users. A control loop can employ RL for sequential optimization problems, allowing an agent to interact with an unknown environment, without a system model. Other methods such as Bayesian learning could be used, however, these methods work better when the amount of data is large [6], and they require some assumptions, e.g. the prior distribution. However, a model-free approach that performs satisfactorily with a little amount of data is preferable, since users do not tolerate under-performing systems for even short periods.

There are still many open challenges to build an automatic control loop based on QoE. First, it is necessary to map how the network characteristics influence the QoE perceived by the user. QoE is subjective, and the characteristics of the flows change over time with the evolution of the services. Further, the flows must be classified using a very small subset of labeled data. Once we can map the network

* Corresponding author.
E-mail addresses: henriquemoura@dcc.ufmg.br (H.D. Moura), damacedo@dcc.ufmg.br (D.F. Macedo), mmvieira@dcc.ufmg.br (M.A.M. Vieira).

https://doi.org/10.1016/j.comcom.2020.02.032
Received 21 October 2019; Received in revised form 7 February 2020; Accepted 9 February 2020
Available online 25 February 2020

characteristics into QoE, we must overcome the second challenge: the intelligent system should learn online, while minimizing the disruptions to the users. Learning requires exploring new actions, and some of them may reduce the QoE. Thus, the system must reduce its impact on the production network by learning effectively. Finally, the third challenge deals with distributed decision making. In a WLAN, several intelligent APs will compete with each other for wireless medium resources. Hence, what is the processing cost and performance in a centralized, decentralized or cooperative approach? The main advantages of the proposed method to the literature are that the RL method used does not need to know the system model, the learning method reacts to changes in the network conditions (adapts to a non-stationary environment), and seeks to continuously improve user satisfaction.

This article presents a closed control loop for Wi-Fi networks, using RL and SDN, which optimizes the Web QoE. To the best of our knowledge, our work is the first that uses SDN to control 802.11-based wireless networks using RL with a QoE metric as feedback. The main contributions of this article are:

- We propose a power control and channel selection scheme using a model-free approach to maximize the Web QoE. The proposed architecture is shown in Fig. 1. We propose an SDN control loop based on Q-Learning (QL) (item 3 of Fig. 1). It is compared with a fixed configuration and with control loops using greedy approaches. A prototype was developed and tested under three real scenarios: (i) a single controller manages one Access Point (AP); (ii) two APs are managed by independent controllers; and (iii) one controller coordinates two APs. The RL approaches showed lower regret (better QoE) than the baselines considered in this article. Our best result improved the QoE by 167% when compared to the baselines. QL showed worse values than Multi-armed Bandit (MAB) (best improvement of 84%), but faster convergence time. The control loops improve the QoE metric as well as the global average page load time by 5.6x in a SA scenario, and 6.6x in MA scenario.
- We investigate the trade-offs of a distributed versus centralized decision approach in Sections 3.3 and 3.4, showing better performance in a centralized approach, since the controller has a better view of what happens to the network. However, the distributed approach needs fewer controller resources, since the search space increases significantly in the centralized approach. The decentralized approach reached the maximum MOS faster in all six combinations, but the centralized approach got better regret in four out of six combinations of web site types.
- We propose a web site classifier based on the similarity to three labeled web sites in [7], employing a semi-supervised learning algorithm called Transductive Support Vector Machine Algorithm Based on Spectral Clustering (TSVMSC) [8]. This classifier is applied to the control loop in two steps: an online, and an off-line process, to provide faster responses to user requests. They correspond to items 1 and 4 of Fig. 1. This classifier selects the predictor proposed by [7] (item 2 of Fig. 1). To the best of our knowledge, this is the first time this approach is applied to site classification. Also, we incorporated state-of-the-art enhancements to the original TSVMSC proposal: (a) inferring the number of clusters by rotating on the eigenvectors of the similarity matrix, (b) using centroid distance of clusters with labeled and unlabeled data to define the class approximation, and (c) using a fast heuristic to obtain the Transductive Support Vector Machine (TSVM) margin. The web site classifier is evaluated using experimental data, varying the regularization parameters and the size of the test set. It achieves an accuracy score of $87\% \pm 1\%$ with 500 unlabeled examples and 15 labeled examples, in a one-second execution, which shows the viability of the classifier. It is also compared to classical machine learning classification methods.

Table 1
Key symbols used in the article.

Symbol	Description
ν	Number of features of the TSVMSC classifier
X	Features for the labeled examples
Y	Labels of the labeled examples
$\grave{n}$	Number of unlabeled examples in the TSVMSC classification
$\mathbf{x}_i^*$	Vector of features on the unlabeled data i
X^*	Matrix $\mathbb{R}^{\grave{n}\times\nu}$ of unlabeled data $\mathbf{x}_i$ as rows
σ	Parameter that controls the width of the neighborhood in the distance matrix generation
C	Regularization parameter for labeled data in TSVMSC classification
C^*	Regularization parameter for unlabeled data in TSVMSC classification
c_n	Centroids for the labeled data, where the index $n \in \{light, average, heavy\}$
c_u	Centroids for the unlabeled data, where the index $u \in \{1, \ldots, K\}$
K	Number of clusters discovered by the spectral clustering procedure
c	Loop parameter that represents the evaluated class

This work is an extension of [9]. The main differences are: (i) in [9] we assume that the web sites are already classified into different types, based on their User Interface (UI) style. This article removes this requirement, as it proposes and evaluates a web site classifier. (ii) we propose a new control mechanism, based on Q-Learning; and (iii) we evaluate a second baseline, based on a greedy algorithm; and (iv) we evaluate the gains in MOS during the learning phase as well as the stationary state, in which the learning algorithms have a more refined hypothesis.

The remainder of this article is organized as follows. Section 2 describes the proposed architecture. Section 3 discusses the results of our prototype using three case studies. Related work is discussed in Section 4. Finally, Section 5 concludes the article.

2. System architecture

This section presents the proposed architecture. We have used an SDN-based approach to implement an intelligent control loop based on a QoE predictor. By decoupling the data plane from the control plane, the SDN controller can run more complex algorithms, which would be prohibitive in the limited processors of existing APs. Further, the SDN controller has a full view of the network, furthermore, it can reach better decisions than distributed algorithms. Section 3 compares the performance of the proposed solution in a centralized scenario (one control loop for the entire network) against a distributed scenario (one control loop for each AP). For clarification, we list in Table 1 the key symbols used in this section.

The control loop is built on top of a software-defined wireless network controller (in our case, the *Ethanol* [10] communication layer). Both programs run in the same host, and both run as POX [11] modules. Note that the control loop is independent of *Ethanol*, so any southbound interface could be employed to control the APs and wireless stations. The main parts of the SDN controller are numbered in Fig. 1, and explained below:

1. **On-line classification of the web sites requested by users**. In this step, the requested web site is classified by the proposed site classifier into one of the three classes (*light*, *average* or *heavy*) proposed in Hora et al. [7]. This classification will be detailed in Section 2.2.
2. **MOS Predictor**. QoE represents the user's satisfaction or discomfort using a given service [12], and it is perceived subjectively. It is measured using a MOS value.[1] Because we cannot inquire the users about their satisfaction, we use a MOS predictor, which estimates the QoE that will be obtained when the network

[1] MOS is a measure used in QoE domain [12] that ranges from 1 to 5.

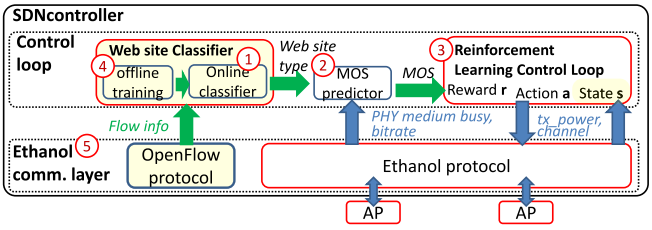

Fig. 1. Proposed architecture. Boxes in yellow did not exist in [9]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

configuration is changed. The predictor used in our work was proposed in [7]. This predictor map the PHY rate and medium busy time into the expected MOS of the web page using Support Vector Regression (SVR). The first parameter represents the link quality, while the second indicates medium availability. Hora et al. [7] have identified three distinct classes of web sites, which have different responses to the PHY rate and medium busy time. They are called *light*, *medium* and *heavy*, and are related to the number of images and other elements in the web page. Hence, the authors derived three SVR models, one for each class. In their evaluation, the authors have shown that the predictors adhered to real values up to 93% in a validation set with data from APs deployed in 4880 residential customers of a large Asian-Pacific Internet Service Provider (ISP), consisting of 23,000 devices and a total of 180 million samples.

3. **Adjustment of the wireless network parameters to improve Web QoE**. Using RL, the controller adjusts the transmission power and the wireless channel so that it maximizes the QoE for the given web site type. This is explained in detail in Section 2.3.
4. **Periodic update of the web site classifier**. The web site classification model is recalculated periodically, using off-line training, as shown in Section 2.2. This update is necessary because the proposed classifier is trained with very few labeled data, and new web sites improve the precision of the classifier. Online learning was avoided due to the need for fast responses, as described in Appendix.
5. **AP configuration interface/API**. The architecture's data plane uses *Ethanol* [10], an SDN architecture for IEEE 802.11 networks that configures and manages wireless resources. *Ethanol* controls IEEE 802.11 APs as well as wireless stations that fully implement the IEEE 802.11/2016 standard. The control plane can configure any IEEE 802.11 parameter that *Ethanol* exports, such as the channel frequency, the transmission power, among others. Other southbound interfaces could be used, however *Ethanol* was chosen because of the author's familiarity with the platform and because it provides the functionality necessary to carry out the proposed experiments.

2.1. The choice of a model-free approach

Models of the IEEE 802.11 channel dynamics, such as those of Bianchi and others, usually model aspects such as throughput and delay in specific situations (e.g. on a saturated channel, or non-saturated channel with a fixed packet generation rate). Beyond that, physical layer effects such as multipath fading are very hard to model accurately. Another aspect of channel dynamic models is the fact that they rely on optimization models. Such models are expensive to re-run on real-time. Meanwhile, RL is less expensive to run, since the updates of the state transitions are performed locally (i.e. only at the current state).

Next, channel dynamics models can infer the Quality of Service (QoS) of the channel and not the QoE. QoE depends on user perception, so to use WiFi channel models in a QoE-aware control loop, one would first have to derive models that map QoS into QoE perception.

Instead of relying on two separate models (i.e. a channel dynamics model and a QoS to QoE mapping model), this work directly maps the effect of actuation into the end-user QoE. Since there are no pre-existing models in the literature to map the QoE of an application into actionable parameter changes, we adopted a model-free approach in which the system learns the relation of states and actions.

2.2. Classifying the web site type

The first step in the control loop is the selection of which MOS predictor to use. In our work, we used the predictors proposed by Hora et al. [7]. The authors defined three predictors, one for each type (class) of a web site. They divided the web sites into three types: *light*, *average*, and *heavy*. The *light* web site represents a web site with very lightweight pages, such as the front page of search engines (e.g. Google). On the other hand, a *heavy* web site represents pages with many objects/images, e.g. Amazon. An *average* web site represents a site with intermediary complexity, like Facebook. Each web site type presents a different response to the controlled wireless parameters, hence the type and intensity of the controller's actuation in the network depend on the web site type.

Hora et al. [7]'s proposal has two shortcomings: they exemplify each class with only one web site (as shown above), and they do not classify arbitrary web sites into the defined classes. Hence, it is critical for a production system to classify other web sites based on similarities with the examples provided by the authors.

Because of the reasons above, we developed a web site classifier. Since there are few labeled examples, the classifier must deal with a problem called low-density separation, which attempts to place boundaries in regions where there are few data points (labeled or unlabeled) [13]. We tackled this problem through semi-supervised learning using TSVMSC.

Our system periodically probes the reference web sites (Google, Amazon, and Facebook), collecting information such as page load time, page size, and Round-Trip Time (RTT). This step is labeled (4) in Fig. 1. These probes provide the labeled data. The number of probes must be kept low, so that this learning process does not to compete with normal system activity.

During the operation of the control loop, due to the need for rapid responses, a target web site is classified online using the TSVMSC trained model whenever the user accesses a web site. This is performed in the box labeled (1) in Fig. 1. TSVMSC is a classifier that is trained in two steps: the first clusters the unlabeled data using Spectral Clustering (SC), and the second uses a modified TSVM to label the data in each cluster. These two steps are explained in more detail below. Because this method considers both labeled data and unlabeled data, it is expected that its results are better than classic supervised and unsupervised methods. We compared the classification results of TSVMSC with some classic methods in Appendix.

The proposed web site classifier uses TSVMSC, which is composed of two phases: (1) a clustering phase – SC and (2) a classification phase – TSVM. They are described below:

1. **SC**: The SC [14] phase generates clusters among the unlabeled data. It uses the eigenvalues of the data similarity matrix to perform dimensionality reduction before clustering. The similarity matrix is the normalized Euclidean distance between every unlabeled data. The number of clusters generated by SC is selected using a technique proposed in §3 in [15], which analyzes the eigenvectors of the similarity matrix. The Laplacian is rotated so there is more than one non-zero entry in some of the rows. The authors define a cost function based on the Laplacian values, and minimizing this cost over all possible rotations will provide the best alignment with the canonical coordinate system. The number of clusters is the one that provides the minimal cost.

Notice that this number can be greater than three (the number of web site classes). The classifier then obtains three centroids c_n using the labeled data, using Euclidean distance. The centroid c_u is then calculated for each cluster of unlabeled data, and the Euclidean distance between c_u and the labeled data centroid c_n determines the cluster label (*light*, *average* or *heavy*).

2. **TSVM**: The clustered unlabeled data provides an approximation of the proportion of positive examples for each class among the unlabeled examples, which is necessary for the next step – TSVM [16]. TSVM generates a classifier for each class, using both labeled and unlabeled examples. Since this problem is NP-hard [16], the *QN-S*^{3}*VM* heuristic [17] was used. This is a modified Support Vector Machine (SVM) where the unlabeled data patterns are taken into account by searching for a partition of these patterns into two classes. This step interactively anneals the influence of the unlabeled part in the SVM cost function. An estimated number of positive examples is needed to run *QN-S*^{3}*VM*. Among the labeled examples, those of the class c in evaluation receive a value $y_i = 1$, while the others receive a value $y_i = -1$. Among the unlabeled examples, those belonging to the clusters classified in class k are counted, providing an estimate of the number of positive examples among the unlabeled set. Therefore, we generate three classifiers using a One-vs-Rest approach. TSVM has two regularization parameters, C and C^*. The first one accounts for the margin over the label data, i.e, it is the cost of the SVM. The second parameter creates a margin over the unlabeled data. These parameters are analyzed in Appendix.

Ties between the classifiers are broken by classifying the web site into the class with more rigorous QoE requirements, i.e., *heavy* and *average*, in this order. This step is $O(v)$ because it depends only on the number of features – v. Also, the features obtained from the last accesses are stored for the off-line step, which retrains the classifier, improving the results.

Appendix shows the evaluation of the classifier used in our experiments, and also compares it with other traditional machine learning methods. The proposed classifier shows an accuracy of 0.8720, an F1 score of 0.8944, a precision of 0.9912, and a recall of 0.8148. The proposed classifier presented the best accuracy among the tested algorithms. The second best obtained an accuracy of 0.553, about 40% less than the proposed classifier. However, its training time was only better than Gaussian Process (GP), however this should not be a problem, since the classifier is trained offline.

2.3. QoE maximization using reinforcement learning

Once the web sites are classified by our proposed classifier, we can use Hora et al. [7]'s predictors to infer which action should be taken to improve the user's QoE. We use RL for that end. For each flow, the predictor provides an estimate of the MOS perceived by the user. Our proposal uses a reward function based on those estimates (i.e. the average of the MOS of the users of an AP) to provide feedback to the learning algorithm. Table 2 shows the symbols introduced in this section.

Fig. 2 shows the proposed control loop. The agent (in our case the controller) receives information about the environment's state **s**. Based on this information, the agent takes an action **a**, the environment changes to a new state **s'**, and the agent receives a reward **r**. These terms will be explained in the following sections. The control loop maximizes the QoE of Web traffic using Q-Learning (QL).

The exploration technique used in this article is called ϵ–greedy [18]. It balances exploration (searches for better responses), and exploitation (reuses the best results so far). The control loop follows with a probability equal to $(1-\epsilon)$ a greedy strategy that considers the action that maximizes the discounted return in the current state. On the other hand, with a probability equal to ϵ, it selects a random action, thus testing other actions that may be better than the best action known to date. Because the environment may not be stationary, we do not anneal the ϵ, which guarantees that the algorithm will always explore [19,20]. However, adjusting the value of ϵ allows the control loop to follow a better approximation of the best action if the system is stable for a long time. This aspect needs further research and will be addressed in future work.

Table 2
Symbols used in the article (continuation).

Symbol	Description
s, s	Current state
s', s'	Next state (after the action a)
a, a	Selected action in state **s**
r, r_t, r	Reward received when in state **s** perform the action **a**
t	Current timestep
ϵ	Controls the exploration rate in the ϵ-greedy algorithm
$MOS_{i,t}$	MOS perceived by the station i at the timestep t
M	Number of connected wireless stations with web flow
Q^*	Optimal Q-value
$Q(s,a)$	Estimated Q-value for state s and action a
α	Learning rate
γ	Discount rate

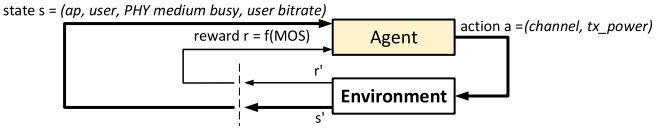

Fig. 2. The agent-environment interaction in reinforcement learning.
Source: Adapted from Sutton and Barto [18].

QL is a simple method, which has convergence guarantees. The same cannot be said of approaches with approximation functions such as Deep QL. Since the states are discretized in our approach, the state matrix grows linearly with the number of APs. For 200 APs, the Q-matrix occupies about 1 MB of memory, which can be handled by our devices. Thus, for medium-sized deployments, this approach does not suffer from the curse of dimensionality. Also, the adopted approach allowed us to see that the convergence is fast using QL. However, an approximation function would help the exploration phase, since extrapolates the estimates to unexplored points in the state space. This approach will be explored in future work.

The control loop is modeled as follows:

2.3.1. Reward

For a time t, we define the reward as the average MOS of the connected stations'. Given that a wireless station i perceives the $MOS_{i,t}$, which is the QoE approximation calculated in time t using Hora et al.'s predictor, the reward used in the control loop is: $r_t = \frac{1}{M}\sum_{i=1}^{M} MOS_{i,t}$, where M is the number of connected wireless stations with web flows. Another option, which we intend to evaluate in future work, is the use of the concept of fairness proposed by Jain [21], to guarantee a more balanced MOS between the stations.

Because Hora et al. signed an NDA, they could not disclose the trained SVM predictor parameters. However, they provided the heat map values generated by these predictors for each of the classes, i.e., we have an approximation of the prediction function. The heat maps were provided as a two-dimensional array of size 63 (PHY medium busy) x 23 (average PHY bitrate). Since the MOS values provided by the authors are discrete, and the values read from the APs are continuous, the MOS used in the experiments is calculated as a linear regression of neighbor points, as shown in Fig. 3. The figure shows how this process is done. When the control loop reads the current state of the environment (P_{env}), it identifies the closest neighbor points (suppose they are P_1 to P_4). These points define a plane, and the MOS value that corresponds to P_{env} is interpolated from this plane. The MOS predictor is used in our

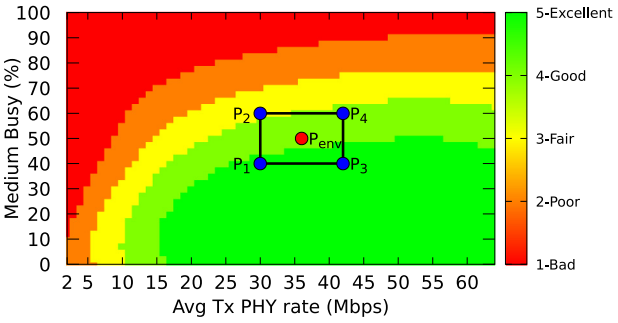

Fig. 3. How MOS is calculated in our evaluation.

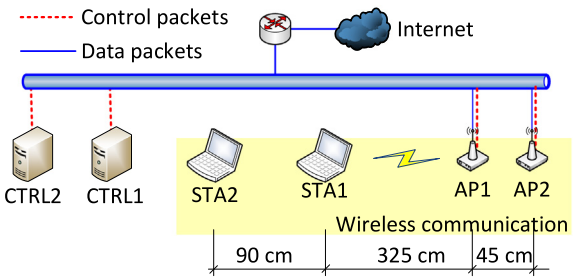

Fig. 4. Network layout of the experiment.

proposal as a black box, thus if a better or more suitable predictor appears, it can be used instead.

The MOS predictor uses two features (Medium Busy, and transmission PHY bitrate), and uses an SVR with non-linear kernel. The models learn similar boundaries in all different classes. The predictor in Hora et al. [7] predicts MOS equal 5 for the web sites when the Wi-Fi conditions are perfect, *i.e.*, medium busy near 0 and PHY bitrate is at maximum. Similarly, when either PHY bitrate is close to 0 or Medium Busy is close to 100%, the predicted MOS is 1. However, the requirements for MOS > 3 differ significantly per visited web site. Complex web sites require much better Wi-Fi conditions than light ones. The area where the predicted MOS is maximum (for example, the green area in Fig. 3) is much bigger using the *light* predictor than the other ones, meaning (as expected) that this site type requires fewer network resources to provide good results to the user.

2.3.2. States

A discrete representation of the system state variable is a three-dimensional vector, which is represented by the tuple (AP_{id}, *PHY medium busy*, *average bitrate*), where AP_{id} is the controlled APs, and the other two values represent the inputs of the QoE predictor, respectively the current level of medium usage (in percentage) and the average bitrate of all connected users in AP_{id} (in percentage of the maximum bitrate supported by the AP). *PHY medium busy*, and *average bitrate* are discretized considering the heat map values provided by Hora et al. As an example, consider a scenario with two APs. The controller reads from these APs the following $PHY\ medium\ busy$, and $average\ bitrate$ values: $(0.25, 0.75)$, and $(0.25, 0.76)$, respectively. The tuple $(1, 25, 75)$ would represent the first AP and its environmental condition discretized to the nearest value in the heat map, while the tuple $(2, 25, 75)$ would represent the state of the second AP. Thus, the state matrix used by the QL algorithm has $n \times 63 \times 23$ states, where n is the maximum number of APs.

2.3.3. Actions

Actions are selected in a parameterized timely basis (1 s). Two factors need to be considered to define the interval between actions in the controller: (1) the total control loop execution time, which consists of communication and processing time in the controller and devices; and (2) If the controller makes a channel change decision, it is necessary to wait for the Channel Switch Announcement message (CSA) time. Thus we have adopted a conservative period of $1\ s$, so that these activities can be performed successfully. However, this period can be adjusted by the user.

The controller alters two Wi-Fi parameters, so an action a is represented by the tuple ($transmission\ power$, $transmission\ channel$). For each controlled parameter, the control loop can **increase**, **decrease** or **maintain** the current value. The transmission power is altered by 1 dBm increments, while the channel is changed by 1. Thus, for example, if the AP is in channel 9 with transmission power equal to 10 dBm, an action that increases the transmission power, and decreases the channel, changes the AP to channel 8 with transmission power equal to 11 dBm. Hence, there are nine[2] possible actions in $\mathbf{A}$, where $a_i = (channel, tx\ power)$. When the parameter reaches the maximum (or the minimum) of the wireless network card, the action does nothing. The range of valid transmission power values is from 1 dBm to 15 dBm and the channel can assume values between 1, and 11 (the valid 2.4 GHz range in Brazil). These ranges can be adjusted by the administrator. An invalid move receives a penalty of -1.

2.3.4. Q -value and its representation

The Q-value is important in the QL method because it allows the agent to estimate the optimal value denoted by Q^* from its experiences. The agent selects actions based on their Q-values, and $Q(s, a)$ is bootstrapped using the Bellmann Equation [18,20]: $Q(s, a) = (1-\alpha)Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a')\right)$. s is the current state where the agent takes an action a, which makes the environment transit to a new state s'. α is the learning rate, and γ is the discount rate used to keep the recurrent sum bounded.

We use a matrix to represent $Q(s, a)$, thus it maps a state and an action to the expected return Q^*. It keeps the value for each state s and action $a \in \mathbf{A}$. The Q-value is stored in a four-dimensional matrix – three dimensions for state, plus one for action, because we have flattened the action space in an $\mathbf{A}$ set, defined in the previous section.

3. Control loop evaluation

A prototype was created containing two *Ethanol* APs and two wireless stations. We use stations that recognize CSA messages, which are used by an AP in a Basic Service Set (BSS) to advertise its new channel before changing channels. This way the station can accelerate the migration to the new channel, reducing the disconnection time.

In the experiment, the station continuously downloads a live web page from the Internet, requesting a new page when the previous one ends downloading. Fig. 4 shows a diagram of the network. The controllers and the APs are connected to a gigabit Ethernet network, which forwards control data. The APs also use the Ethernet links to forward the station traffic to/from the Internet using NAT. The devices are aligned, and the distances are shown in Fig. 4.

The controllers are dual-core Intel i5 PCs with 16 GB of RAM. The stations are PCs with Dual Core Pentium 2.4 GHz CPUs, 2 GB of RAM, and a RT3062 802.11n wireless card. Because *Ethanol* runs on Linux, the APs are ASUS notebooks with Intel i7 CPU @ 1.80 GHz, 8 GB of RAM, and Atheros AR9485 wireless NICs. This equipment is shown in Fig. 5.

The experiments are performed in an environment with more than 60 other APs. We have no control over these APs, which operate concurrently with our experiments. The control loop must handle co-channel interference generated by these devices. Also, in our two-APs scenarios, the control loop must also deal with co-channel interference

[2] There are two types of actions, and three actions each, generating nine distinct combinations.

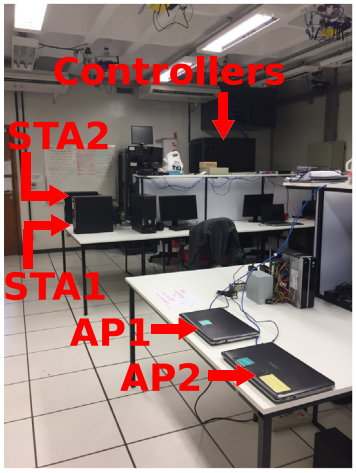

Fig. 5. Equipment.

of managed APs during startup, because control loop experiments start both APs on the same channel and power, and, during the experiment, the control loop can also select actions that put both APs on interfering channels.

Hora et al. rank web sites into three classes, and the experiment has two clients, thus there are six possible traffic combinations. The traffic is not simulated, that is, the stations download live pages over the Internet using a modified version of the Ghost.py web browser (https://github.com/jeanphix/Ghost.py). The stations download pages from the example web sites defined by Hora et al.: Google, Facebook, and Amazon. For each combination, the experiment is repeated 30 times, stopping the run if both stations reach the maximum MOS or if a timeout occurs (30 min). Three metrics were collected: convergence time,[3] average MOS, and regret.[4] Results in this section show the average as well as the 95% confidence interval using Student's t-distribution.

The experiments begin with the system having no previous knowledge – all values in $Q(s, a)$ are equal to 0. α is set to 0.5, so there is an equilibrium between what is already learned and the new reward, $\epsilon = 0.05$, so 5% of the control loop's choices are exploratory, and $\gamma = 0.8$. Due to the lack of space, we are not showing the tests performed to obtain these values. Each iteration of our learning algorithm occurs with one-second intervals, thus the state and the reward are also collected with this granularity.

3.1. Baselines

To compare our results we came up with two baselines described below. To collect the data for each combination of web site types, the experiment was run 30 times. Each run takes 20 min.

- Many commercial controllers provide an "Auto Channel" feature, i.e, they automatically configure the APs to mitigate the interference between them. This baseline thus gives an idea of how our proposed method compares with a commercial controller.
  This baseline is called "Automatic Channel Selection (ACS) baseline" in the figures. It uses a heuristic provided in the *hostapd* code, called ACS (see https://wireless.wiki.kernel.org/en/users/documentation/acs for details). This heuristic evaluates the valid channels' interference and occupation at the *hostapd* initialization, and selects the channel with lowest interference factor.
  Every 5 s, *Ethanol* requests the controlled AP to measure the inputs for this heuristic. If there is a better channel (i.e. less busy), it requests the AP to switch to this channel.
  The period to run the ACS procedure was manually selected, trying to beat the QL performance. This implementation depends heavily on the amount of time spent gathering the survey data. It should be noticed that this is a costly procedure, because when the AP is scanning the channel it cannot forward traffic. In the current hostapd implementation, this procedure is only executed during hostapd startup.
  ACS can be suboptimal because short traffic bursts may be missed. It also has to be performed several times to obtain statistically meaningful data. Thus, there is a tradeoff between channel interference measurement, the quality of the information obtained and the AP's traffic forwarding performance. In our implementation, there are five samples, one for each second. Some commercial solutions use a secondary wireless card for scanning. This feature, however, is present only in high-end devices.
- We also use a simpler setup that we called "Fixed baseline", used in [9]. This baseline corresponds to the behavior of most SOHO wireless installations, where the administrator configures each device individually, selecting a channel during its setup, and that remains static for a long period, *i.e.*, until the administrator reconfigures the AP. APs are started on a randomly selected channel, and with maximum transmission power. This step resembles the operations of APs using *hostapd* (https://w1.fi/hostapd/).
  This baseline is justified because using the maximum power on a single AP is the best local strategy, since it increases the Signal to Interference plus Noise Ratio (SINR) for its stations, regardless of the station's flow characteristics. The existence of other APs, managed by other administrators who do not interact with each other, allows us to model the configuration task as a game similar to the prisoner's dilemma, where administrators greedily employ the maximum power (defect) or selflessly lower their device's transmission power (cooperate). This game's dominant equilibrium is for both players (AP administrators) to defect, i.e., both APs are configured at maximum power. Further, this setting is typical for non-technical users, since standalone commercial APs usually ship with maximum transmit power.
- In a previous work [9], we proposed an RL control loop based on MAB. MAB is a special class of RL for sequential optimization problems, where the agent only keeps track of the return distributions of each action, thus there is no state. The method used is based on confidence intervals [22], *i.e.*, the algorithm learns the reward distribution for each action, but at the same time estimates the confidence of that distribution. Thus, it deals with the exploration-exploitation dilemma without hyperparameters. It explores actions that it has the least confidence, thus learning more about the reward for this action (and increasing the confidence in its estimate). On the other hand, it exploits actions with higher reward estimates.

Some commercial wireless controllers have auto power capabilities, but these solutions, such as Meraki (https://documentation.meraki.com/MR/Radio_Settings/Auto_TX_Power), control only APs from their brand. In this way, those controllers do not cooperate with other wireless systems in the same coverage area. Thus, we decided to set the baselines with a fixed transmission power.

3.2. Single agent experiment

This experiment uses one controller that manages one AP. Two wireless clients connect to this AP. The control is centralized, so this is an RL scenario with a single agent (SA), resembling a home network.

[3] The convergence time is the time it taken to reach the maximum MOS for all stations.

[4] It measures how much reward, on average, the system is losing by not obtaining the maximum MOS Since the maximum reward is five, we can compare the strategies using QL and the baseline to an optimal strategy that would always ensure the maximum reward (i.e. regret equals to zero).

Table 3 shows the number of iterations when both wireless stations reach MOS = 5. It compares the results obtained using QL, and MAB. The table rows are the combinations of the three web site classes, shown in the first and second columns. The next columns show the average number of iterations with the 95% confidence interval and the regret for each control loop and the baselines. As each iteration occurs in approximately one second, these values also approximate the time spent to reach this limit. We observe that the scenarios with at least one *light* page presented the shortest convergence times using QL. This occurs because the page download requires fewer network resources, so the controller can quickly reach the maximum MOS for this client. On the other hand, the scenarios with *heavy* web sites presented the longest convergence times and largest confidence intervals. Notice that, on average, QL is faster to reach the maximum MOS, but, in most cases, MAB obtains lower medians.

The confidence interval presents large variations mostly due to very few runs (outliers). For example, in a *light–heavy* case, one run took 481 iterations to converge, while all others took at most 61 iterations. Notice also that none of the baselines reached the maximum MOS in either of the stations after 30 min of experimentation.

Fig. 6 shows the Cumulative Distribution Function (CDF) of the MOS. This Figure also shows the results obtained using MAB [9]. Due to space limitations, we only show the worst (Fig. 6-A) and the best (Fig. 6-B) results. The worst and best curves were selected based on the comparison with the "Fixed baseline". RL approaches achieve the worst results in Fig. 6-A. In this Figure, the "ACS baseline" outperforms QL and MAB, but both are better than the "Fixed baseline". However, when both stations access *light* web sites, only 40% of the QL's measurements are better than the "Fixed baseline". This behavior is expected, since the APs in this baseline work at maximum power, improving the bitrate. Meanwhile, QL and MAB perform exploration, thus these algorithms select worse settings during the experiment, which impacts the overall result. Improving the exploration mechanism should provide better results, but this is still an open topic in RL. However, for the "Fixed baseline" configuration, the stations are unable to reach the maximum MOS during the execution. On the other hand, in cases where one station accesses a *heavy* web site, the control loop results are better than the fixed baseline. The overall MOS for the QL control loop is, on average, improved by 47.5% if compared to the "Fixed baseline", and 10.5% compared to the "ACS baseline".

Table 3 shows the average regret with a 95% confidence interval. This table shows that QL-controlled experiments provide lower average regret in all cases when compared to the "Fixed baseline", and particularly in *heavy* scenarios. Notice that a 100% reduction is equal to an optimal regret (equal to zero). The the *light–average* case achieved the highest reduction of the experiment, equal to 76%. QL has worse performance than MAB and beats the "ACS baseline" in half of the cases. Comparing the results with "ACS baseline", we notice that the regret is reduced to 1.77 in the *light–heavy* case using QL. MAB [9] achieves smaller regrets when compared to QL for all combinations, however, QL converges, on average, faster for all combinations. This is due to the exploration strategies used in each method. QL explores faster, in this way it reaches a state where both stations get MOS maximum faster, but because the environment is dynamic, MAB obtains better MOS than QL. Dealing with the exploration and exploitation phases is a dilemma in RL. We think that better controlling this behavior should improve QL results. It is important to highlight that there is no learning period for the baselines, because they use a fixed configuration. In this way, the results in Table 3 could be improved if we consider only the values when the system stabilizes. This topic is discussed in Section 3.5.

The controller and the AP exchange, on average, 193 ± 92 control messages per experiment (about 509.2 kB).[5] 7.7 MB were downloaded in the experiment with lower traffic, both *light* web sites, while the higher traffic experiment downloads 280 MB. Those values correspond to all web content downloaded in the experiment. In the worst case overhead, i.e both *light* web sites, is about 6.5%, and in the best case, it is less than 0.2% of the useful traffic. Notice that the control messages are not transmitted in the wireless medium, only the (web) data is transmitted in the wireless medium, and the off-line web site classifier traffic only affects the controller.

[5] This value considers the sizes of all the frames transmitted in the network, which are captured using *tcpdump*, including the three-way TCP handshake.

3.3. Multi-agent experiment

This scenario simulates a deployment with multiple APs, such as a shopping center, a condominium, etc., and each AP is managed by a different administrator who does not exchange information with other administrators. This experiment uses two independent controllers that do not exchange information. The APs are started on the same channel and with the same power.

Table 4 shows the number of iterations until convergence. The convergence time increases as the stations demand more resources from the Internet, so the scenarios where the *light* web site is involved present smaller convergence time than *heavy* web site cases. In the *average–heavy* configuration, for example, the confidence interval is larger because there are two extreme cases where the convergence time was very long. This situation is expected, since the algorithm is always started without any knowledge of the environment, the channel is selected randomly, and the power starts with the minimum.

Fig. 7 shows the worst (7-A) and best (7-B) results for the MOS CDF, when compared to the "Fixed baseline". The RL approach reaches greater MOS than the baseline in most cases. The exceptions occur in less than 10% of (*light–light*) and (*average– average*) experiments. Note that the baseline used maximum transmission power and has a higher probability of APs being in non-interfering channels.

Table 4 shows the average regret with a 95% confidence interval for the MA experiment. QL provides lower average regret in all cases when compared to the "Fixed baseline", corroborating the results shown in Fig. 7. The overall MOS in this experiment is, on average, improved by 47.6% if compared to the "Fixed baseline". QL wins from "ACS baseline" in three combinations: *light–light*, *light–heavy*, and *heavy–heavy*. Thus QL loses by 1.2% when compared to the "ACS baseline" (averaging all results). However, MAB outperformed the "ACS baseline" on average by 18.1%. MAB showed smaller regrets than QL in all six combinations (Table 4), but QL converges faster, and page load time is smaller in 4 cases, while the other two are statistically a tie. The better convergence time is due to the QL exploration strategy. In this case, QL explores more than MAB, reaching faster a configuration where both stations get maximum MOS. However, the MAB regret is lower, because it obtains better MOS more times than QL.

The application throughput, i.e., the transmission speed, was evaluated. When a controller is used, the page load time is lower and the throughput observed by the stations is higher than the "Fixed baseline". An increase of 6.7x on the average page load time was measured in the (*light*, *average*) case – our best result in this scenario.

During the experiment 278 ± 55 control messages were exchanged, causing an overhead of 743.8 kB in the Ethernet network. Note that more messages are exchanged in MA than in SA. However, the confidence intervals overlap in both cases. For the experiment with lower traffic, 9.8 MB of Web content was downloaded, while for the situation of higher traffic 77 MB was downloaded. Hence, the control messages generate an overhead of 1% to 7.4%.

This is a non-cooperative multi-agent scenario, where agents do not communicate but have to cope with cross-interference. The system may become non-stationary from the perspective of an individual agent, since each agent independently changes its behavior while learning takes place. This may lead to oscillations [23]. Future work will test an adapted Payoff Propagation method, turning the setup into a cooperative scenario.

Table 3
SA experiment - Number of steps to reach maximum MOS using the controller and the regret obtained during the experiment.

Client1	Client2	Number of iterations		Regret			
		Q-Learning	MAB	MAB	QL	*Fixed*	*ACS*
light	light	22.3 ± 7.46	32.8 ± 56.46	0.743 ± 0.004	1.486 ± 0.072	1.611 ± 0.005	**0.671** ± 0.015
light	average	**16.4** ± 3.10	166.6 ± 126.84	**0.711** ± 0.003	1.568 ± 0.104	1.904 ± 0.006	0.936 ± 0.052
light	heavy	**36.4** ± 31.76	195.7 ± 177.47	**0.440** ± 0.003	0.642 ± 0.053	2.706 ± 0.011	1.724 ± 0.005
average	average	**15.1** ± 2.60	16.9 ± 8.08	**1.013** ± 0.005	1.468 ± 0.108	2.386 ± 0.009	1.263 ± 0.081
average	heavy	**32.0** ± 21.60	243.5 ± 279.73	**0.634** ± 0.005	1.246 ± 0.049	3.108 ± 0.013	1.838 ± 0.037
heavy	heavy	**54.8** ± 26.45	100.7 ± 44.15	**1.411** ± 0.014	1.454 ± 0.038	3.164 ± 0.014	3.066 ± 0.008

Table 4
Number of iterations to reach maximum MOS, and the regret in MA experiment considering the control loops and the baselines.

Client1–Client2	Number of iterations		Regret			
	Q-Learning	MAB	MAB	QL	*Fixed*	*ACS*
light–light	**48.3** ± 12.64	1356.4 ± 233.31	**0.675** ± 0.006	1.070 ± 0.045	1.579 ± 0.007	1.011 ± 0.007
light–average	**37.1** ± 7.23	1309.7 ± 243.56	**0.610** ± 0.006	1.101 ± 0.063	1.843 ± 0.009	1.102 ± 0.009
light–heavy	**63.2** ± 24.05	1123.9 ± 213.27	**1.034** ± 0.008	1.423 ± 0.048	3.019 ± 0.006	1.433 ± 0.006
average–average	**43.8** ± 5.35	860.2 ± 171.32	**0.771** ± 0.010	1.535 ± 0.060	2.477 ± 0.008	1.004 ± 0.008
average–heavy	**71.4** ± 42.17	804.4 ± 150.03	**1.388** ± 0.012	1.790 ± 0.063	3.583 ± 0.006	1.611 ± 0.006
heavy–heavy	**63.7** ± 14.52	635.5 ± 108.77	**1.770** ± 0.013	2.334 ± 0.058	3.780 ± 0.003	2.581 ± 0.003

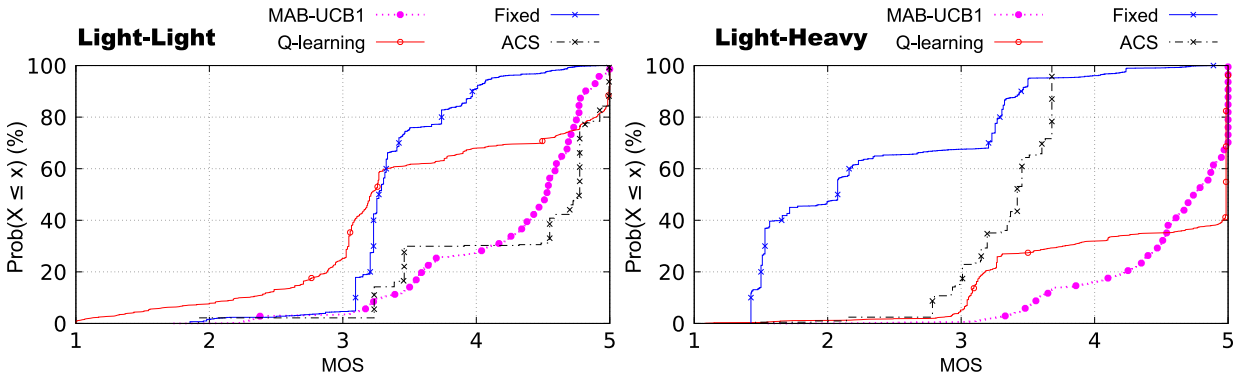

Fig. 6. SA experiment – Cumulative distribution function of MOS – curves more to the right mean better overall MOS.

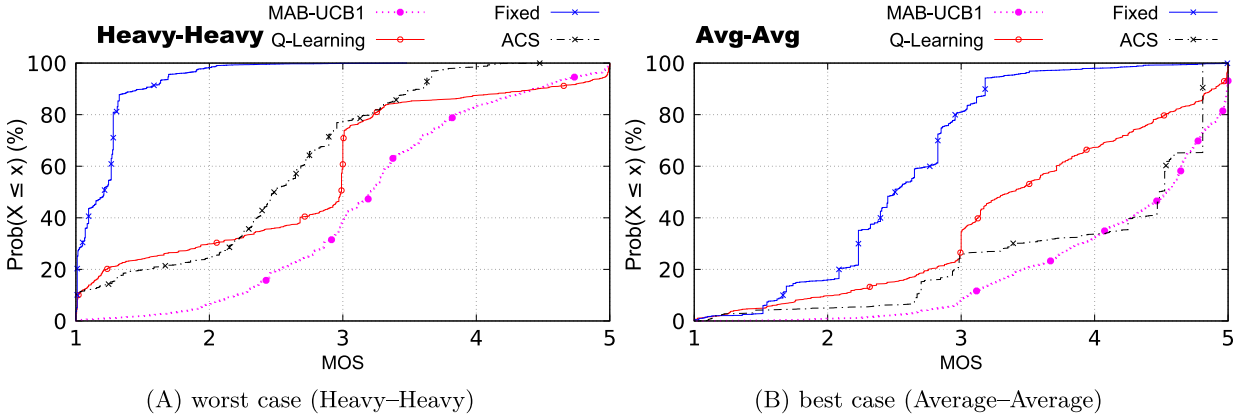

Fig. 7. MA experiment - Cumulative distribution function of MOS.

3.4. Centrally controlled agent experiment

This scenario simulates a deployment with multiple APs managed by a single administrator, such as in a company or a campus. The control loop is executed on a state space that simultaneously contains both stations and APs, and therefore the system seeks a global reward, equivalent to the average of the MOS obtained by the stations. This experiment is similar to SA because the controllers can only obtain information about its two stations, but now the controller coordinates both APs, and therefore the controller can handle the cross-interference that is generated by them. We added a second greedy baseline in this scenario. Alves et al. [24] proposed a greedy algorithm to adjust the channel of WLAN networks, which they called HNR. The proposed algorithm calculates the interference generated by the other controlled APs as well as the interference of devices in the neighborhood using the channel quality and the throughput measured in the AP's wireless interface. We empirically tuned the threshold parameter used in the algorithm. For that end, we ran the experiment 20 times for 30 min,

Table 5
Number of iterations to reach maximum MOS, and the regret in CA experiment considering the control loops and the baselines.

Client1–Client2	Number of iterations		Regret				
	Q-Learning	MAB	MAB	QL	*Fixed*	*ACS*	*HNR*
light–light	**73.6** ± 17.77	383.5 ± 168.94	**0.869** ± 0.005	1.153 ± 0.032	1.579 ± 0.007	1.082 ± 0.007	1.337 ± 0.007
light–average	91.7 ± 12.35	**60.9** ± 37.33	**0.788** ± 0.004	0.899 ± 0.032	1.843 ± 0.009	0.908 ± 0.009	1.252 ± 0.009
light–heavy	**76.3** ± 8.66	226.2 ± 57.65	**1.055** ± 0.006	1.306 ± 0.038	3.019 ± 0.006	1.421 ± 0.006	1.302 ± 0.006
average–average	114.7 ± 30.73	**18.9** ± 6.38	**0.969** ± 0.006	1.685 ± 0.037	2.477 ± 0.008	1.273 ± 0.008	1.265 ± 0.008
average–heavy	**83.4** ± 14.67	211.6 ± 42.50	**1.260** ± 0.007	1.432 ± 0.046	3.583 ± 0.006	1.469 ± 0.006	1.297 ± 0.006
heavy–heavy	**77.0** ± 9.71	199.7 ± 37.65	1.737 ± 0.008	2.135 ± 0.042	3.780 ± 0.003	2.239 ± 0.003	**1.710** ± 0.003

and the best threshold among these runs was used. The HNR curve and regret shown in this section were obtained running 30 repetitions using the best threshold.

Table 5 shows the number of iterations for this experiment. Again, QL provides the lowest average regret in all cases, if compared to the fixed baseline. CA showed the worst page load times in most of the cases, although the runs get lower throughput than the baseline. We also noticed this behavior in MA, but with less emphasis. We believe that is related to the format of the MOS metric: the control loop maximizes the bitrate, which only refers to the radio data rate (of only a part) of the packets [25], and not the actual throughput of the web flows. For example, a station can have a high bitrate, even if it is not transmitting.

MAB showed in [9] smaller regrets in all six combinations, as displayed in Table 5. However, QL converges faster in all combinations, and the page load time is smaller in one case, while all other cases are statistically a tie. The reasons for the observed convergence time and regret are the same ones discussed in the previous section. If compared to the "ACS baseline", QL obtains better results in half of the cases. It is important to highlight that this baseline was manually configured to beat QL. Altering the scanning interval or the network setup affects the baseline performance, while QL can learn how the environment behaves and adapts the APs' configuration to it. The HNR performance depends on the flow threshold. This is a weak point of their proposal, since the threshold should be adjusted for each configuration. Despite that, HNR provides reliable and consistent performance for their clients, and outperforms QL, and MAB (by a narrow margin) in the *heavy–heavy* case.

The experiment demands an overhead of 1070 kB in the Ethernet network due to the control messages sent from the controller to the APs. Notice that in the CA experiment 44% more messages are exchanged than on the MA experiment, because the control loop takes more time to converge. The 95% confidence intervals in both cases overlap, so statistically, both values are the same. The experiment with lower traffic transfers 1.4 MB, while in the higher traffic, 21 MB were downloaded by the stations (an overhead of 75%).

Fig. 8 shows the MOS's CDF. This figure shows the CDFs for the two proposed RL methods – QL and MAB-UCB1, the baselines described in Section 3.1, and the results obtained using the algorithm proposed in [24] – HNR. In the worst case presented (Fig. 8A), QL obtains worse results than MAB for more than 75% of the cases. However, QL reaches the maximum MOS faster, and the curves intersect at the end, causing QL to have more cases with values closer to 5. Despite that, the MAB regret is better (i.e. overall regret value is smaller) than QL. The overall MOS in this experiment using QL is, on average, improved by 55.1% if compared to the "Fixed baseline", and 2.3% when compared to the "ACS baseline". However, MAB improves on average by 13.3% and 71.7%, respectively.

The control loop achieves higher MOS values than both baselines in most cases. This is shown in the regret columns in Table 5. All cases show a reduction in regret when compared to the baseline.

During the learning process, the APs sometimes reaches a local maximum. The increase in the number of controlled APs implies an increase in the actions set size and in the Q-value matrix size. The size of the Q-value matrix is $|Q| \propto |A||S|$. Hence, in the MA experiment, each controller explores (and fills) simultaneously both Q-value matrices of size $|Q|$. On the other hand, in the CA experiment there is only one matrix with size $|Q'| \propto |2A||2S| \propto 4|Q|$. Analyzing the executions in both scenarios, we observed that on average the agent needs to explore more states in the CA scenario than in MA. In the MA scenario, the agent explores, on average, about 0.0042% of the total space, while in CA the agent explores 0.0051% of the space. This shows the importance of how Q-value is modeled. These values indicate that (1) in MA the agent needs to explore further the search space to achieve a result similar to CA, and (2) the Q-value matrix is sparse, thus the agent is unaware of the effect of similar actions.

3.5. Exploration vs. exploitation

RL mechanisms are subject to a trade-off between two choices: exploration, where the agent seeks more information about the environment to verify the existence of better decisions, or exploitation when it uses the best decision known to date. More exploration means that the agent may select many sub-optimal actions, and therefore this impairs its performance. On the other side, if the agent performs only exploitation, it may be unaware of other actions that can provide better long-term results. The rewards obtained by the control loops with RL shown in Figs. 6, 7, and 8 consider the rewards obtained during the entire experiment. This means that results are measured from the initialization (when the agent knows nothing of the environment) until the stations reach a maximum MOS.

To illustrate how the initialization affects the rewards, Fig. 9 shows the MOS obtained in ten executions of the experiment when the clients accesses *light*-type web sites. Three curves are shown. The first one corresponds to the cumulative distribution of the rewards during the first 200 s of 10 runs. The others correspond respectively to the 200 and 400 s following this initialization phase. The results after the initialization are better as expected, since these curves are more to right in the graph than the curve obtained with the first 200-second results. Further, they also have a shorter initial tail than the curve in blue.

The average MOS and the 95% confidence interval are, respectively, 4.36 ± 0.02 for the 200-second intervals, 4.61 ± 0.01 for the next 200 s and 4.60 ± 0.01 for the next 400 s. This means that the first interval is different from the last two intervals (using the Student's t-test). Comparing the averages, one can notice a gain of 5.8% and 5.5% from the intervals of the next 200 and 400 s respectively to the initialization phase (first interval). Another way to highlight the difference of the results produced by the initialization phase and the following seconds is to check the ratio between the areas under both curves and perform the Kolmogorov–Smirnov (KS) test for two samples [26]. The KS test verifies if both curves come from the same distribution based on the distance between the CDFs. The CDFs from the initialization and the other ones are different with 95% confidence (using the test of the difference of the means with unknown variance). Comparing the initialization phase with the subsequent 200 s, there is a 40% difference in the area. This ratio indicates that the agent gets a cumulative performance in the subsequent 200 s of almost one-and-a-half times better than the initialization phase. The initialization, when compared to the next 400 s, provides a difference on the order of 34.4%. Thus, during the initialization period, when the agent knows very little about the environment, the actions return worse results than the subsequent seconds.

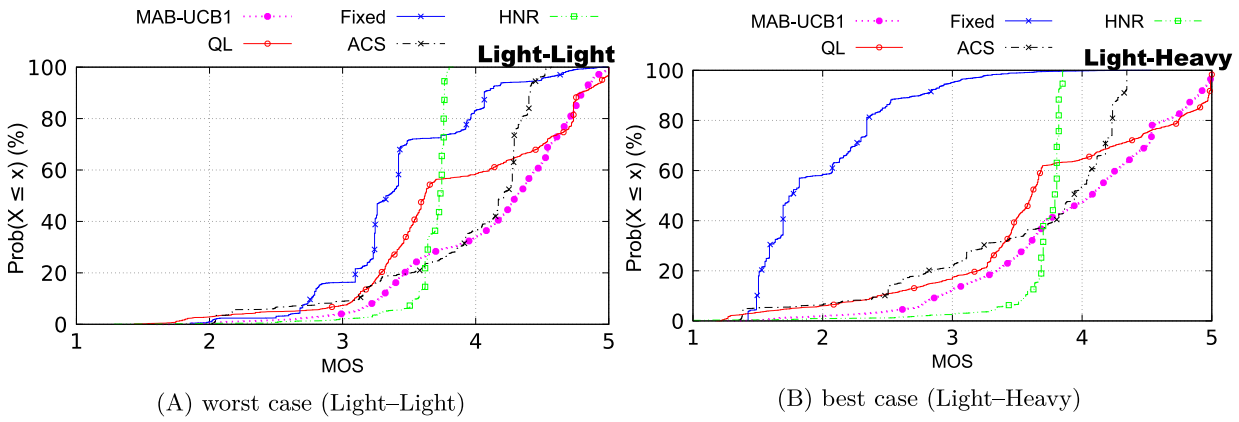

Fig. 8. CA experiment – Cumulative distribution function of MOS.

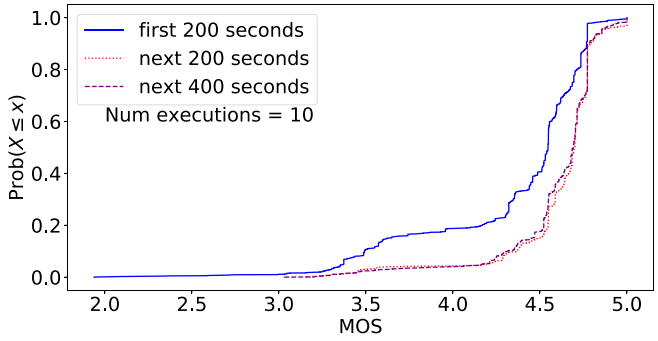

Fig. 9. Comparison of MOS during initialization for (*light–light*) experiment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.6. Misclassification problems

Table 6 shows the effect on the regret if the classifier misclassifies the web site class. To view this effect, we used the SA scenario described in Section 3.2, however, we forced the classifier to misclassify one or both web sites. This result also indicates the effect of site classification mismatches on Web 2.0 web sites, in which the type of demand may change over time according to what is displayed on-screen at the moment.

The results shown in the table correspond to 10 executions of 30 min for each combination. The rows in the table correspond to the estimated web site classification. The columns indicate the actual classes of both web sites. The cells show the average regret and the confidence interval with 95% confidence for each result. The results on the main diagonal of the table correspond to the values obtained when the classifier makes the classification for both web sites correctly, and therefore correspond to the results presented in Table 3. We colored the table's cells so that lighter colors correspond to lower regrets (better results), while darker colors correspond to higher regrets. The reader should compare the results obtained on the same line.

From the table, one can see that the regret values obtained when the classifier classifies the web site type with higher requirements into a web site type with lower requirements (values in the row to the left of the main diagonal) are worse than the respective value when the classification of both sites is correct. In this case, the system thinks it is obtaining better results because the MOS for a class with fewer requirements is reached more easily (in graphical terms, the green area in Fig. 3 is wider for less demanding classes). Thus, the agent stops improving the performance, wrongly deducing that the MOS is already maximal for both stations. We also see that the values above the main diagonal (when the classifier "upgrades" the flow to a more demanding site type) also show worse results. For example, when both stations are downloading the *light* web site (first line of the table), but the classifier indicates that the second station is downloading a *heavy* web site (first row, third column), the result is also worse.

At a first glance, it might seem that this should not be the case, as the predicted MOS has a higher demand for the misclassified site: the user would get a better real MOS, because the green area in Fig. 3 for the *light* web site (which is being accessed) is much larger than the area for the *heavy* web site (which the system thinks is being downloaded). However, the problem is exactly the higher demand, which causes the agent to look for better settings (increases exploration) when in fact it has already reached the maximum MOS. Since both stations are connected to the same AP, the actions also affect the results of the station that was correctly classified, *e.g.*, changing channels increases the disconnected time.

3.7. Discussion

Table 7 shows the best results obtained by both learning methods (Column 1) for each of the baselines (Column 2). The values presented in the other columns represent the gain (bigger is better) when the learning method is applied to the baseline. The rows with the HNR baseline only show results in the last column because this baseline was run only in the CA experiment. Recall that HNR aims to reduce co-interference between the controlled APs, so the CA experiment is the only scenario where this can occur. Observe that a 100% reduction means getting the ideal regret, that is, regret is equal to zero.

In general, results show that there is a trade-off when choosing MAB or QL. This occurs because MAB presents smaller regrets, i.e. results closer to the optimal, than QL. However, the rate of learning of QL is faster than that of MAB. As a consequence, MAB would fare better in more stable environments, while QL would work better than MAB in more dynamic networks. To ascertain how long the network would need to converge, we evaluated the distributions of MOS for different periods of the simulation. In our setup, the learning algorithms were able to reach a stationary return after only 200 s of operation (recall Fig. 9).

Using learning to control the wireless network improves network performance by reducing the regret observed during web browsing. Our best results (using MAB) show a reduction of 84%, 74%, and 35% compared to the Fixed, ACS, and HRN baselines, respectively. QL outperforms the "Fixed baseline" in all cases, and the "ACS baseline" in half of the cases, while MAB outperforms the baselines in most of the cases (40 out of 42).

The experiments did not evaluate the relationship between the number of stations and the convergence time, because we did not have enough devices for that. We believe, however, that the convergence time should increase as the complexity of the system increases. We saw

Table 6
Comparing regret results with misclassifications.

Client1–Client2	Classified as					
⇓ real class	light-light	light-average	light-heavy	average-average	average-heavy	heavy-heavy
light-light	1.486 ± 0.072	1.574 ± 0.015	2.166 ± 0.014	1.635 ± 0.015	2.019 ± 0.020	2.150 ± 0.018
light-average	2.412 ± 0.023	1.568 ± 0.104	2.011 ± 0.028	1.674 ± 0.033	2.051 ± 0.023	1.972 ± 0.042
light-heavy	3.122 ± 0.012	2.667 ± 0.016	0.642 ± 0.053	2.964 ± 0.017	2.646 ± 0.012	3.111 ± 0.012
average-average	3.009 ± 0.021	2.085 ± 0.035	2.487 ± 0.045	1.468 ± 0.108	2.556 ± 0.070	2.458 ± 0.032
average-heavy	3.288 ± 0.022	2.520 ± 0.017	2.935 ± 0.021	2.826 ± 0.024	1.246 ± 0.049	3.283 ± 0.040
heavy-heavy	3.677 ± 0.015	3.918 ± 0.007	3.729 ± 0.012	2.827 ± 0.020	3.655 ± 0.019	1.454 ± 0.038

Table 7
Best regret results when applying the proposed learning methods compared to the baselines.

Learning method	Baseline	SA	MA	CA
MAB	Fixed	84%	69%	65%
	ACS	74%	45%	26%
	HNR	–	–	35%
QL	Fixed	76%	53%	60%
	ACS	53%	10%	8%
	HRN	–	–	28%

in SA, and MA experiments that increasing the number of APs increases the convergence time, because this raises the likelihood of co-channel interference.

Results indicate that the agent explores a small range of states (less than 1 percent of the space). We plan to evaluate two ideas in future work to improve the coverage of the learning process. First, we plan to use an approximation function to the Q-value (such as a Bayesian or neural network), for faster learning. A second option is the use of transfer learning, in which the network would be deployed with pre-trained weights. The latter could be evaluated by running the same experiment on a different testbed.

The quality of the classification will determine whether the system will choose the most appropriate predictor for each situation or not. There are two types of misclassifications: (1) the selected predictor imposes higher system demands, for example, it selects a web site as *heavy* when it should be *light*, and (2) the selected predictor imposes lower system demands, for example, it classifies a web site as *light* when it should be *heavy*. In the first case, if the control loop can reach the maximum MOS value, the user will not be affected because the conditions will be favorable to him. In the second case, the effect for the client may be immediate, because the *light* web site demands fewer resources. Hence, the predictor may erroneously estimate that the station is at full MOS, causing the control loop not to look for a better configurations, where the station could achieve the maximum MOS.

4. Related work

Wireless channel conditions can be modeled using a finite-state Markov model [46]. Previous research use RL for routing, channel and power control in ad-hoc and wireless sensor networks [47,48]. Baraković and Skorin-Kapov [49] surveyed the state-of-the-art of QoE management, focusing on wireless networks, and exploring the use of MOS in cellular networks, but they did not provide a QoE metric based on wireless metrics.

A QoE/QoS correlation for IPTV QoE evaluation is provided by Kim and Choi [50]. It creates a normalized QoE value using bandwidth, burst level, delay and jitter. However, the authors evaluated this correlation only on cabled broadband connections, which are much more stable than a wireless network. D-DASH [32] uses a video metric to estimate QoE. They improve the QoE using Deep-QL, but their approach controls only the video server, not the network as in our approach. Zhang et al. [51] propose a cache management approach for HTTP servers using adaptive bitrate streaming in wireless networks. Their proposal aims to maximize the users' QoE. Their work infers the QoE using a logarithmic function of the playback rate, and this metric is validated by 22 users. Amani et al. [52] proposed an offloading mechanism for 5G networks using SDN and wireless networks, and they support the decision using QoS metrics.

RL and QoE were applied to various wireless network problems as summarized in Table 8. These proposals aim to provide better congestion control [53], content delivery [29,31,32], handover [44], medium access control [36], resource allocation and management [38], routing [41], and traffic classification [54]. The table shows approaches that use RL to define the agent's action, for example, to define the delivery rate [29] or the quality of the video to be downloaded [30–32]. Other approaches model the network problem as an auction system, e.g. [39], where RL solves the auction offer problem, which is NP-hard, thus the agent learns the best offers based on the reward received by auctioning. Harishankar et al. [39] considers QoE-sensitive applications, but a QoE is not explicitly considered in the calculations.

All of the work shown in the Table, except [33] and ours, test the proposal using simulators, and even in most of those case, the simulator is omitted. Also, only a few proposals are tested using real traces. The table shows in the first line our proposal. None of the related work improved the client's web access in an 802.11-based wireless network.

5. Conclusion and future work

This article proposed a closed control loop system that optimizes the QoE for web flows. The control mechanism interacts with the environment, and in our case, the proposal uses SDN as a southbound interface. The intelligence is performed by RL algorithms that learn what is the best action for the current system state. Our proposal acts upon the environment by changing the AP's transmission power and channel. The control loop tackles issues such as how to relate network parameters with the user's QoE, and how to achieve the best network configuration without compromising the user's experience, while at the same time minimizing the compute and network resources required.

A crucial aspect of the control is the use of a good QoE estimator. This allows the control loop to weigh the benefit of each action considering the perceived QoE. The proposed semi-supervised estimator classifies web sites based on the similarity to a small number of known web sites. We use TSVMSC, which outperforms six classical machine learning algorithms in terms of accuracy by up to 57%.

We evaluated the proposal with a prototype using three case studies, in which many APs are controlled by a central controller or independently. The use of a learning method to improve QoE has shown a promising future according to our results. Convergence to maximum MOS values is achieved within one to two minutes of operation. In the MA and CA experiments, QL outperforms the fixed baseline in all cases, and the ACS baseline in half of the cases. Meanwhile, MAB outperforms all baselines. Our best result improved the QoE by 167% if compared to the baselines. When compared to the baselines, our proposal improved the page load times by up to 6.6 times, which significantly improves the perceived QoE by the user.

There are two main types of future work, namely improving the practicality of the system, and improving the machine learning algorithms. First, we will advance the system to operate on flows other than web browsing. Regarding improvements in the method, there are many options to be pursued, for example, (a) using function approximation (e.g Deep QL), (b) using a pre-trained system to respond quickly to

Table 8
A summary of RL approaches using QoE in networks.

Application	Type of network	RL method	SDN	Reward	Simulator	Control location	Ref.
Power and channel allocation	**Wi-Fi**	QL/MAB	**Yes**	The predict MOS for **web access**	**Real testbed**	SDN controller	**Ours**
Association control	Cellular network	QL	No	User's QoE formed by a weighted combination of performance indicators	N/A	Small cells	[27]
Content delivery	Cellular network	QL	No	Three different estimated rewards: (1) when a single frame is transmitted or dropped; (2) frames are temporarily keep in the PE buffer; and (3) when several frames are transmitted from the PE buffer to the MAC buffer	N/A	Video encoder	[28]
	Cellular network	QL	No	Weighted sum of the client buffer states	OMNeT++[a] using INET framework[b]	Content server	[29]
	TCP network	QL	No	A function that associates the current video quality level, the oscillation in quality levels during the video playout, and buffer starvation	ns-3[c]	Client	[30,31]
	Wireless networks	Deep QL	No	A function that accounts for the benefit of a higher quality qt of the video, and two negative terms due to quality variations in consecutive frames, and rebuffering events,	N/A	HTTP client	[32]
Game streaming	5G	Actor-critic deep RL	No	A function of the bitrate and stalling time	Implemented in China	Edge node	[33]
Handover	IEEE 802.21	WoLF-PHC (see [34])	No	Relates to the current and the previous (stored) mean user-perceived quality (QoE) level, and length of the time the user has experienced degradation or improvements due to the recent action	ns-2[d]	Mobile nodes	[35]
Medium access control	Wireless networks	Spatial adaptive play	No	An utility function that accounts for the node i satisfaction when node j keeps silent	N/A	Nodes	[36]
Radio resource management	5G	Deep actor-critic	No	Guaranteed Bit Rate (GBR), delay and Packet Loss Rate (PLR) requirements	N/A	Base station	[2]
Rate adaptation	LTE	DQL Rainbow	No	Video chunk QoE based on the bitrate and staling rate	N/A	Node	[37]
Resource allocation	IoT wireless network	DQL	No	Three MOS formulations: file download, video streaming, and IPTV, and VoIP	Discrete simulator ccnSim[e]	Nodes	[38]
	LTE	Episodic Monte Carlo policy iteration	No	Equal to the utility function if agent wins the bit, otherwise zero	SimuLTE and INET	eNodeB	[39]
Routing	Wireless Sensor Network	QL	No	Considers three types of services, and provides a QoE value based on QoS parameters	ns-2	Nodes	[40,41]
	Wireless networks	QL	No	A MOS value based on an MLP classifier based on QoS values	OPNET[f]	Nodes	[42]
Spectrum access	5G CRN	QL	No	A function that reflects the MOS based on the state and action	N/A	Secondary user	[43]
Spectrum handoff	Cognitive Radio Network	QL	No	The predicted MOS of multimedia transmission, for a certain handover	N/A	Nodes	[44]

Notes:
[a] https://www.omnetpp.org/.
[b] https://inet.omnetpp.org/.
[c] https://www.nsnam.org/.
[d] https://www.isi.edu/nsnam/ns/.
[e] See more information in [45].
[f] OPNET is now incorporated into the Riverbed's SteelCentral. https://www.riverbed.com/products/steelcentral/opnet.html.

customer requests and test the model generalization, and (c) on-line adjustment of the exploration phase. It is possible to use a function approximator for Q-value representation, such as neural networks like the ones used in Deep QL. Convergence nor optimality are ensured though, for example, Bertsekas and Tsitsiklis [55] observed that DQN may yield suboptimal solutions or even diverge. There is, however, empirical evidence that near-optimal values can be obtained. This line of research can be pursued in future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Henrique D. Moura: Conceptualization, Methodology, Software, Data curation, Writing - original draft, Writing - review & editing, Visualization, Investigation, Formal analysis, Validation. **Daniel F. Macedo:** Supervision, Conceptualization, Writing - review & editing. **Marcos A.M. Vieira:** Supervision, Conceptualization, Writing - review & editing.

Acknowledgments

This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq, Brazil (funding agency from the Brazilian federal government), and FAPEMIG, Brazil (Minas Gerais State Funding Agency).

Appendix. Evaluation of web site type classification

We collected 2880 probes for the three labeled web sites in order to test the performance of the proposed web site classifier. The same data was collected for the Web's ten most visited web sites,[6] excluding the three reference web sites, Google's local search web sites, porn web sites, and web sites that do not respond to *ping* requests, in a total of 40,000 probes.

The accuracy of the classifier is compared to user-defined scores. The unlabeled web sites were classified by five people, who rated from 1 (less likely) to 10 (equal) on how much the web site download is similar to the three known web sites. For each web site the interviewee provided three scores, one for *light*, one for *average* and one for *heavy*. The class with the highest average score is selected for the web site. We tested the classifier with different numbers of unlabeled examples, and with different values of C and C^* (the regularization parameters).

The classification tests employed only 15 labeled examples (5 of each known web site example), {500, 1000, 1500, 2000} unlabeled examples, four values of $C^* = \{0.1, 1, 10, 100\}$, and four values of $C = \{1e{-}07, 1e{-}05, 0.001, 0.1\}$. Each combination of parameters was repeated twenty times. The TSVM classifier (second phase of our proposed web site classifier) used a linear kernel function. The results shown in this section are collected in a regular PC (see controller specification in Section 3).

Classifier performance. Table A.9 shows the average accuracy, the F1 score, the precision, and the recall results in the test set. It also shows the average runtime of the web site classifier, and the 95% Confidence Interval (CI), using Student's t-test. These results were obtained with 500 unlabeled examples, and $C = 1e{-}05$. The training was performed using 3-fold cross-validation. The performance metrics in Table A.9, e.g. F1 score, show very small variation among the lines when C^* is varied. For this reason, Fig. A.10 shows the results when $C^* = 0.1$ and C is altered.

Table A.9
Average accuracy, F1 score, precision, and recall of the proposed classifier using the test set, and the time to train the classifier using a test set size equals to 500 and $C = 1e{-}05$.

C*	Average accuracy	F1 score	Precision	Recall	Average runtime (s)	Confidence interval (95%)
0.1	0.87**20**	0.89**44**	0.9912	0.8148	0.6212	0.0445
1.0	0.87**27**	0.89**44**	0.9911	0.8149	0.6794	0.0454
10.0	0.87**61**	0.89**54**	0.9910	0.8166	0.7356	0.0475
100.0	0.87**85**	0.89**53**	0.9910	0.8165	0.7989	0.0441

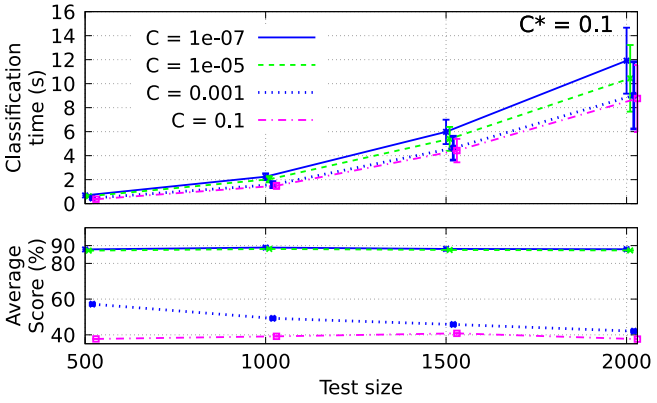

Fig. A.10. Web site classifier accuracy (score), and execution time with $C^* = 0.1$.

Execution time. Fig. A.10 also shows the training time of the (off-line) classifier with $C^* = 0.1$. Training the web site classifier with 500 unlabeled examples takes about one second. Since the TSVM algorithm is not incremental, because there is a loop to fit the labels, a new example implies the complete execution of the algorithm. We observe an exponential growth of the execution time as a function of the size of the test set, thus we prefer to perform the training off-line. Note that further improvements in the execution time can be obtained by using Python compilation, using the label provided by SC as the first estimation of TSVM's $\mathbf{w}$ and b values, or employing the methods presented in [56].

Fig. A.10 shows the average test scores when C and the set size is varied, and C^* is set to a fixed value (0.1). The accuracy is measured from 0 to 100%, also showing the 95% CI for each value. We observe that the C parameter has a strong influence on the accuracy obtained and that, however, the two smaller values $C = 1e{-}05$ and $1e{-}07$ are statistically equal, since their CIs overlap. In this way, the experiments used the classifier with $C = 1 \times 10^{-5}$ and $C^* = 0.1$. In this configuration the accuracy of TSVMSC was 87.5% on average. The classes scored 83.2, 85.4, and 93.8% for *light*, *average* and *heavy*, respectively.

As explained previously, the proposed architecture must retrain the web site classifier from time to time. Since good scores are achievable with 500 unlabeled examples, this was the chosen value for C and C^*. It also allows the model to be retrained more frequently.

Network overhead. In this experiment, each station only accesses one web site during the whole experiment. Thus, in each experiment up to two web sites are accessed, which is atypical in a production network. In real networks, we expect a heavy-tailed distribution of accesses, so the overhead for web site probing will be reduced.

In our setup, during the initialization, the controller makes five accesses to each of the three prototypical web sites to calibrate the classifier. The average sizes of the downloaded content are $36{,}019 \pm 1$ bytes for *light*, $47{,}626 \pm 5884$ bytes for *average*, and $975{,}562 \pm 56{,}208$ bytes for *heavy*,[7] so at start-up the controller downloads 5 MB. Compared

[6] https://en.wikipedia.org/wiki/List_of_most_popular_websites updated on December 28, 2016.

[7] The Google page is the simplest of them. It has some text and CSS elements, and about five images (three icons – page, magnifying glass and

Table A.10
Comparison between the proposed web site classifier and classical classification methods.

Classifier	Cat.	Acc.	Hyperparameters	Ref
k nearest neighbors neighbors.KNeighborsClassifier.html[a]	S	39.75%	K = 500, leaf size = 5, uniform weights	[57]
k-Means cluster.KMeans.html[a]	U	39.96%	K = 4, distances = auto	[58]
Gaussian mixture models mixture.GaussianMixture.html[a]	U	40.02%	components = 4, spherical covariance	[59,60]
SVM svm.SVC.html[a]	S	44.8%	C = 0.1, gamma = 0.1, RBF kernel	[57]
Gaussian Process gaussian_process.GaussianProcessClassifier.html[a]	S	50.7%	RBF kernel, length scale = 16	[61]
Linear regression with gradient descent linear_model.SGDClassifier.html[a]	S	55.3%	alpha = 10^{-5}, epsilon = 0.1, eta0 = 0.0001, L1 penalty, learning rate = 'optimal'	[57]
Web site classifier	SS	87%	Linear kernel	–

Note
[a]The URL starts with "https://scikit-learn.org/stable/modules/generated/sklearn".

with the experiments in the next section, that represents an overhead of 2% in the best case (in the SA experiment), or in the worst case an overhead of 350% (in the CA experiment). Therefore, the frequency of probes directly affects the overhead generated by the web site classifier. For example, if the values are persisted by the loop control, the overhead will be smaller, because the probes will not be needed at start-up. Also if we consider that the value of the RTT and the characteristic of the web sites vary little, the probing interval can be increased to longer periods (days or weeks). The number of probes in each query can also be reduced if the network connection is more stable.

Improving response time. The page loading time (HTML source and all other associated resources) for all three classes is 0.95 ± 0.09 s for *light*, 4.28 ± 0.61 s for *average* and 3.16 ± 0.92 s for *heavy* on our testbed for the first use case. Because for each new web site the system needs to perform a probe to identify the type of the web site, we resort to a "trick" to speed up the first access. In the first request, the web site is classified as *heavy*, while firing a probe. Thus the user request can be answered immediately, without waiting (at least) the probe time, since it can reach several seconds. Upon completion of the probe, the web site class is updated to the class provided by the classifier. Similar "tricks" are common in systems that need to perform online traffic classification, such as intrusion detection systems. Note that the above load time considers the download of all resources associated with the requested page, which is much longer than the rendering time, because current browsers initially load only the resources needed to render the presentation area (window size). Using a metric that uses rendering time would make online classification faster, however, we believe it would still be necessary to use the described "trick" to shorten the user's first-access latency.

Comparing the proposed site classifier with other machine learning methods. We evaluated several other classification methods using supervised and unsupervised learning. Table A.10 shows the best results for each of the evaluated methods and their hyperparameters. The first column shows the name of the method and a link to the implementation description in the Scikit-learn library (https://scikit-learn.org). The second column has an "S" if it is supervised, "U" for unsupervised, and "SS" for semi-supervised learning. The next column shows the best accuracy level in the test set, and the fourth column shows the hyperparameters for this model. Default values are not shown. Due to a lack of space, we will not discuss the methods individually. The last column of the table provides references for more details about each method, including what each hyperparameter means.

Supervised methods require labeled data to be provided, so model training is done using only examples obtained from probes performed on Google, Facebook, and Amazon. The trained supervised model is then used to extrapolate the results to the unknown web sites using the selected features. Meanwhile, unsupervised methods identify similar patterns among the training data. In this case, clusters are generated for *light*, *average* and *heavy* web sites using majority votes, i.e., the classification of the cluster is made by the class with the highest number of known examples. In the case of a tie, we select *heavy* or *average* in this order. If the cluster has no known example, it is joined to the nearest cluster.

The MOS predictor used in our work was proposed by Hora et al. [7]. The authors classify the web sites into three classes, but enumerated in their paper only one reference for each web site. Thus, we have only three URLs as a reference to train our classifier. To get new references we would have to resort to a human panel to label web sites. Obtaining more references for each class could improve the performance of supervised methods, and would also increase the performance of the semi-supervised method used. Getting new references goes beyond the scope of our work, and adds no novelty to our proposal, so we chose to work only with the references provided in [7]. Notice, however, that due to the network varying performance, many samples can be obtained by probing only these three web sites during a long period.

Further, we argue that the predictor in [7] is fairly accurate, due to the methodology employed by the authors. To infer the relationship between WLAN features and QoE, Hora et al. [7] used three data sources. First, they used a WLAN testbed with instrumented commodity APs, where they passively monitored WLAN features and evaluated the correlation of the QoE with the link quality, and the medium availability. Then they collected data for the selected features in a bigger testbed (their office), which was used to train the predictor. Finally, they validated the predictor using data from real users. They used measurements on APs deployed at 4880 residential customers of a large Asian-Pacific ISP, reporting over 23,000 devices to a backend server, collecting a total of 180 million samples.

We observe in Table A.10 that the classical methods showed results between 45% and 65% of those obtained with TSVMSC. The cluster-based methods obtained the worst results, while the methods based on

keyboard), the Google logo, and a small Google advertising image). The Facebook page is a bit more complex. It contains two midsize images (globe connections and company logo), however, it contains a few more HTML text elements because its form is more complex as is the page footer. The Amazon page loads a large number of images (static and animated) and eventually a video. Notice the 56 kB confidence interval. The Amazon page is much bigger than the others. The two other pages are smaller, however, the size difference between them is not negligible, since the test of the difference of means with 99% confidence indicates that the means are different.

probabilities (Gaussian Mixture and GP) obtained intermediate results. Of these two, GP was obtained the best result, however, it takes much more training time than the others. The training time for GP was 11,635 s, while other methods took from 1.2 up to 3.5 s to train.

SVM obtained half of the accuracy of TSVMSC.

We observe a significant improvement in the classification when using the unlabeled data, and that, among the classic methods tested, linear regression obtains the best result. This is a good indication why the best kernel in TSVMSC was the linear one.

Among the algorithms tested, TSVMSC presented the best accuracy, however, its training time was only better than GP. We hoped that TSVMSC would achieve a better result, for example, than SVM, linear regression or unsupervised methods. However, this behavior is not certain, as discussed in Singh et al. [62].

According to Zhu [63], there are five types of semi-supervised methods that avoid changes in dense regions, such as TSVMSC. One of them is GP, however as the supervised GP training time was much higher than TSVMSC, we did not consider evaluating semi-supervised GP. The methods based on information regularization, entropy minimization, and graphs require the definition of a prior distribution or the probability distribution of the data. Those methods were not evaluated because we do not know the distribution of all the Internet web sites.

References

[1] L. DiCioccio, R. Teixeira, C. Rosenberg, Characterizing Home Networks with Homenet Profiler, Tech. Rep. CP-PRL-2011-09-0001, UPMC Sorbonne Universits, 2011.

[2] I. Comsa, R. Trestian, G. Ghinea, 360° mulsemedia experience over next generation wireless networks - a reinforcement learning approach, in: QoMEX 2018, 2018, pp. 1–6.

[3] A. Hochstadt, S. Newman, R. Greenberfg, K. Aflalo, Internet trends 2018. stats & facts in the U.S. and worldwide, 2018, https://www.vpnmentor.com/blog/vital-internet-trends/, Accessed: 2018-07-20.

[4] N. Marchetti, N.R. Prasad, J. Johansson, T. Cai, Self-organizing networks: State-of-the-art, challenges and perspectives, in: ICC 2010, 2010, pp. 503–508.

[5] R. Barco, P. Lazaro, P. Munoz, A unified framework for self-healing in wireless networks, IEEE Commun. Mag. 50 (12) (2012) 134–142.

[6] M. Opper, O. Winther, A Bayesian approach to on-line learning, On-line Learn. Neural Netw. (1998) 363–378.

[7] D.N.d. Hora, R. Teixeira, K. van Doorselaer, K. van Oost, Predicting the effect of home Wi-Fi quality on web QoE, in: Internet-QoE '16, 2016, pp. 13–18.

[8] X. Yu, J. Yang, J.-p. Zhang, A transductive support vector machine algorithm based on spectral clustering, AASRI Procedia 1 (2012) 384–388.

[9] H.D. Moura, D.F. Macedo, M.A.M. Vieira, Automatic quality of experience management for WLAN networks using multi-armed bandit, IFIP/IEEE IM (2018) 8.

[10] H. Moura, A.R. Alves, J.R.A. Borges, D.F. Macedo, M.A.M. Vieira, Ethanol: A software-defined wireless networking architecture for IEEE 802.11 networks, Comput. Commun. 149 (2020) 176–188.

[11] S. Kaur, J. Singh, N.S. Ghumman, Network programmability using POX controller, in: ICCCS International Conference on Communication, Computing & Systems, Vol. 138, IEEE, 2014.

[12] I.T. R. P.10/G.100, Vocabulary for performance and quality of service. Amendment 5: New definitions for inclusion in recommendation ITU-T P.10/G.100, 2016, accessed: 2018-03-2. [Online]. Available: https://www.itu.int/rec/T-REC-P.10-201607-I!Amd5/en.

[13] O. Chapelle, A. Zien, Semi-supervised classification by low density separation, in: AISTATS, 2005, pp. 57–64.

[14] A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Advances in Neural Information Processing Systems, 2002, pp. 849–856.

[15] L. Zelnik-Manor, P. Perona, Self-tuning spectral clustering, in: Advances in Neural Information Processing Systems, 2005, pp. 1601–1608.

[16] T. Joachims, Transductive inference for text classification using support vector machines, in: ICML, Vol. 99, 1999, pp. 200–209.

[17] F. Gieseke, A. Airola, T. Pahikkala, O. Kramer, Sparse quasi-Newton optimization for semi-supervised support vector machines, in: ICPRAM (1), 2012, pp. 45–54.

[18] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Vol. 1, MIT press, Cambridge, 1998.

[19] R.S. Sutton, A.G. Barto, Introduction to Reinforcement Learning, first ed., MIT Press, Cambridge, MA, USA, 1998.

[20] E. Alpaydin, Introduction to Machine Learning, MIT press, 2009.

[21] R. Jain, D.-M. Chiu, W.R. Hawe, A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System, Vol. 38, Eastern Research Laboratory, DEC, Hudson, MA, 1984.

[22] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2–3) (2002) 235–256.

[23] J.R. Kok, N. Vlassis, Collaborative multiagent reinforcement learning by payoff propagation, JMLR 7 (2006) 1789–1828.

[24] A.R. Alves, H.M. Duarte, J.R.A. Borges, V.F.S. Mota, L.H. Cantelli, D.F. Macedo, M.A.M. Vieira, HomeNetRescue: an SDN service for troubleshooting home networks, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018.

[25] J. Jun, P. Peddabachagari, M. Sichitiu, Theoretical maximum throughput of IEEE 802.11 and its applications, in: IEEE NCA, 2003, pp. 249–256.

[26] S. Engmann, D. Cousineau, Comparing distributions: the two-sample Anderson-Darling test as an alternative to the Kolmogorov-Smirnoff test, J. Appl. Quant. Methods 6 (3) (2011) 1–17.

[27] M. Jaber, M.A. Imran, R. Tafazolli, A. Tukmanov, A multiple attribute user-centric backhaul provisioning scheme using distributed SON, in: 2016 IEEE Global Communications Conference, IEEE, 2016, pp. 1–6.

[28] N. Changuel, B. Sayadi, M. Kieffer, Online learning for QoE-based video streaming to mobile receivers, in: IEEE Globecom Workshops, 2012, pp. 1319–1324.

[29] F.Z. Yousaf, O. Mämmelä, P. Mannersalo, Reinforcement learning method for QoE-aware optimization of content delivery, in: 2014 IEEE Wireless Communications and Networking Conference, 2014, pp. 3390–3395.

[30] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, F. De Turck, Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming, in: Proceedings of the 2013 Workshop on Adaptive and Learning Agents, Saint Paul (Minn.), USA, 2013, pp. 30–37.

[31] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, F. De Turck, Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client, Connection Science 26 (1) (2014) 25–43.

[32] M. Gadaleta, F. Chiariotti, M. Rossi, A. Zanella, D-DASH: A deep Q-learning framework for DASH video streaming, IEEE TCCN 3 (4) (2017) 703–718.

[33] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, D.O. Wu, Improving cloud gaming experience through mobile edge computing, IEEE Wirel. Commun. (2019).

[34] M. Bowling, M. Veloso, Multiagent learning using a variable learning rate, Artificial Intelligence 136 (2) (2002) 215–250.

[35] B.S. Ghahfarokhi, N. Movahhedinia, A personalized QoE-aware handover decision based on distributed reinforcement learning, Wirel. Netw. 19 (8) (2013) 1807–1828.

[36] J. Yin, Y. Mao, S. Leng, X. Wang, H. Fu, QoE provisioning by random access in next-generation wireless networks, in: 2015 IEEE GCC, IEEE, 2015, pp. 1–7.

[37] J. Liu, X. Tao, J. Lu, QoE-oriented rate adaptation for DASH with enhanced deep Q-learning, IEEE Access 7 (2018) 8454–8469.

[38] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, S. Guo, Green resource allocation based on deep reinforcement learning in content-centric IoT, IEEE TETC (2018).

[39] M. Harishankar, S. Pilaka, P. Sharma, N. Srinivasan, C. Joe-Wong, P. Tague, Procuring spontaneous session-level resource guarantees for real-time applications: An auction approach, IEEE J. Sel. Areas Commun. 37 (7) (2019) 1534–1548.

[40] R. Matos, N. Coutinho, C. Marques, S. Sargento, J. Chakareski, A. Kassler, Quality of experience-based routing in multi-service wireless mesh networks, in: IEEE ICC 2012, IEEE, 2012, pp. 7060–7065.

[41] N. Coutinho, R. Matos, C. Marques, A. Reis, S. Sargento, J. Chakareski, A. Kassler, Dynamic dual-reinforcement-learning routing strategies for quality of experience-aware wireless mesh networking, Comput. Netw. 88 (C) (2015) 269–285.

[42] H.A. Tran, A. Mellouk, S. Hoceini, B. Augustin, Global state-dependent QoE based routing, in: IEEE ICC 2012, 2012, pp. 131–135.

[43] F.S. Mohammadi, A. Kwasinski, QoE-driven integrated heterogeneous traffic resource allocation based on cooperative learning for 5G cognitive radio networks, in: 2018 IEEE 5G World Forum (5GWF), IEEE, 2018, pp. 244–249.

[44] Y. Wu, F. Hu, S. Kumar, Y. Zhu, A. Talari, N. Rahnavard, J.D. Matyjas, A learning-based QoE-driven spectrum handoff scheme for multimedia transmissions over cognitive radio networks, IEEE J. Sel. Areas Commun. 32 (11) (2014) 2134–2148.

[45] Q. Wu, Z. Li, G. Xie, Codingcache: multipath-aware ccn cache with network coding, in: Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking, ACM, 2013, pp. 41–42.

[46] Q. Zhang, S.A. Kassam, Finite-state Markov model for Rayleigh fading channels, IEEE Trans. Commun. 47 (11) (1999) 1688–1692.

[47] A. Forster, Machine learning techniques applied to wireless ad-hoc networks: Guide and survey, in: ISSNIP 2007, IEEE, 2007, pp. 365–370.

[48] R.V. Kulkarni, A. Forster, G.K. Venayagamoorthy, Computational intelligence in wireless sensor networks: A survey, IEEE Commun. Surv. Tutor. 13 (1) (2011) 68–96.

[49] S. Baraković, L. Skorin-Kapov, Survey and challenges of QoE management issues in wireless networks, J. Comput. Netw. Commun. (2013).

[50] H.J. Kim, S.G. Choi, A study on a QoS/QoE correlation model for QoE evaluation on IPTV service, in: ICACT, 2010, Vol. 2, IEEE, 2010, pp. 1377–1382.

[51] W. Zhang, Y. Wen, Z. Chen, A. Khisti, Qoe-driven cache management for HTTP adaptive bit rate streaming over wireless networks, IEEE Trans. Multimed. 15 (6) (2013) 1431–1445.
[52] M. Amani, T. Mahmoodi, M. Tatipamula, H. Aghvami, SDN-based data offloading for 5G mobile networks, ZTE Commun. 12 (2014) 34.
[53] O. Habachi, Y. Hu, M. Van der Schaar, Y. Hayel, F. Wu, MOS-based congestion control for conversational services in wireless environments, IEEE JSAC 30 (7) (2012) 1225–1236.
[54] B. Stefano, D.P. Francesco, G.G. Claudio, M. Salvatore, P. Martina, P. Antonio, R.C. Lorenzo, S. Vincenzo, A multi-agent reinforcement learning based approach to quality of experience control in future internet networks, in: 34th Chinese Control Conference, 2015, pp. 6495–6500.
[55] D.P. Bertsekas, J.N. Tsitsiklis, Neuro-dynamic programming: an overview, in: CDC 2011, Vol. 1, 1995, pp. 560–564.
[56] O. Chapelle, V. Sindhwani, S.S. Keerthi, Optimization techniques for semi-supervised support vector machines, JMLR 9 (2008) 203–233.
[57] S. Rogers, M. Girolami, A First Course in Machine Learning, CRC Press, 2016.
[58] A.K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognit. Lett. 31 (8) (2010) 651–666.
[59] J.A. Bilmes, et al., A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models, Int. Comput. Sci. Inst. 4 (510) (1998) 126.
[60] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning – Data Mining, Inference, and Prediction, Vol. 1, second ed., Springer series in statistics New York, NY, USA, 2009.
[61] C.K. Williams, C.E. Rasmussen, Gaussian Processes for Machine Learning, Vol. 2, MIT Press Cambridge, MA, 2006.
[62] A. Singh, R. Nowak, J. Zhu, Unlabeled data: Now it helps, now it doesn't, in: Advances in Neural Information Processing Systems, 2009, pp. 1513–1520.
[63] X.J. Zhu, Semi-Supervised Learning Literature Survey, Tech. Rep., University of Wisconsin-Madison Department of Computer Sciences, 2005.