A REINFORCEMENT LEARNING APPROACH TO SELF-CONFIGURING EDGE WIRELESS NETWORKS

A Thesis

by

MASON CHRISTOPHER RUMULY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Srinivas Shakkottai
Co-Chair of Committee,	Jean-Francois Chamberland
Committee Members,	Dileep Kalathil
	Natarajan Gautam
Head of Department,	Miroslav M. Begovic

May 2019

Major Subject: Electrical Engineering

ABSTRACT

Wireless Internet access has brought legions of heterogeneous applications all sharing the same resources. However, current wireless edge networks that cater to worst or average case performance lack the agility to best serve these diverse sessions. Simultaneously, software reconfigurable infrastructure has become increasingly mainstream to the point that dynamic per packet and per flow decisions are possible at multiple layers of the communications stack. Exploiting such reconfigurability requires the design of a system that can enable a configuration, measure the network performance statistics (Quality of Service), learn the impact on the application performance (Quality of Experience), and adaptively select a new configuration. The goal of this work is to design, develop and demonstrate a reinforcement learning approach to self-configuring wireless edge networks that in instantiates this feedback loop. Our context is that of reconfigurable queueing, and we use the popular application of video streaming as our example. Through simulation and experimental validation, we show how measurement, learning and control are combined to enable high QoE video streaming on our platform.

DEDICATION

To my dearly departed sister Aly.

ACKNOWLEDGMENTS

I would like to thank Dr. Srinivas Shakkottai, who has graciously mentored and assisted me since I was an undergraduate student. He has given me access to many opportunities to expand my horizons and participate in many research endeavors, with boundless patience and understanding as I learned the ropes. His help was indispensable.

Many thanks to my collaborators on this project: Rajarshi Bhattacharyya, Desik Rengarajan, and Archana Bura. Their insights, effort, ongoing encouragement, and friendship have been invaluable.

My gratitude abounds upon my committee-members, Dr. Jean-Francois Chamberland, Dr. Dileep Kalathil, and Dr. Natarajan Guatam for their guidance and for humoring my eccentricities in pursuing this achievement.

Special thanks to my friends and family for their support and love, and to all my teachers, instructors, and mentors for their contributions over my blessed life to who I am today.

Finally, ultimate thanksgiving and glory to God almighty, creator of heaven and earth; to Jesus Christ, who is the Way, the Truth, and the Life; and to the Holy Spirit.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of adviser Dr. Srinivas Shakkottai and co-advisor Dr. Jean-Francois Chamberland and Dr. Dileep Kalathil of the Department of Electrical and Computer Engineering, and Dr. Natarajan Gautam of the Department of Industrial and Systems Engineering.

The experiments and analysis contained in this work were carried out in collaboration with Rajarshi Bhattacharyya, Desik Rengarajan, and Archana Bura.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a research assistanceship from Texas A&M University in cooperation with the Texas Engineering Experiment Station.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

The Internet has produced a wide variety of heterogeneous, resource-intensive applications which subject networks to growing demands for the resources to provide users with a high Quality of Experience(QoE)[1]. This problem is compounded by the increasingly mobile nature of the devices used to access the Internet, which concentrates strain on wireless edge networks. Access at the wireless edge is mediated through wireless Access Points (AP), which have many potential degrees of freedom over which to tailor the Quality of Service (QoS)[2] used to deliver bytes to clients, such as bandwidth apportionment and packet scheduling schemes. Current algorithms focus on best-effort and application-agnostic performance, an approach which is ill-suited to optimizing achievable QoE in real time with constrained resources.

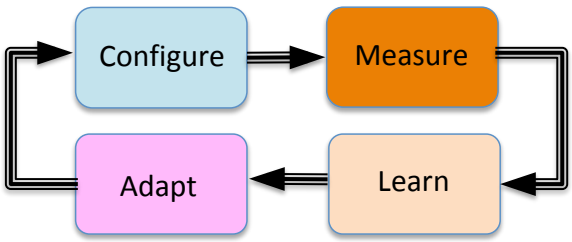

Figure 1.1: AP Self-Configuring Update Cycle

An AP under such conditions must autonomously discover the optimal actions in its dynamic, error-prone environment in order to deliver the best possible QoE to its users given limited resources. Such a goal requires the AP to proceed through a

[1]This is defined as an application-specific value in the interval $[1, 5]$, with 1 indicating the lowest and 5 indicating the highest satisfaction of the human end-user.

[2]This is defined as a vector of statistical connection properties consisting of [*throughput*, *delay*, *jitter*, *lossrate*] at queues.

feedback loop to continually adjust its strategy: it must choose a configuration of its operating parameters, measure the impact of this configuration on the client applications, learn the implications of this result, and adapt its configuration in response, as in Figure 1.1. A solution must be able to consider the long-term consequences of any decision in addition to the immediate results.

Many advances have been made in the field of Reinforcement Learning (RL) on autonomous solutions to problems of this form. In recent advances, Deep Neural Networks are employed in conjunction with Time-Difference bootstrapping methods and various control policies in order to estimate the value associated with taking an action at any given time, with their prowess in complex systems shown a battery of Atari video games [1]. These agent architectures are capable of learning new systems without preexisting domain knowledge, and do not depend on modeling the systems to which they are applied; we aim to apply these methods to the domain of Self-Configuring Networks(SCN).

In order to demonstrate the effectiveness of this approach, we will focus on video-streaming applications, which form a large part of real network traffic and have exacting service requirements to deliver satisfactory QoE [2]. Our goal in this work is to design, develop and demonstrate a framework for adaptive network configuration via differentiated QoS that is provided to flows via the QoS statistics of the queues that they are exposed to. We will see that intelligently utilizing the diversity of QoS provided by such queueing policies can significantly enhance QoE foe all flows.

1.1 Related Work

Our work brings together several different areas ranging from SDN, QoS, QoE and online reinforcement learning:

OpenFlow and Configuration: There has been much recent interest in extending

the SDN idea to other layers. For example, CrossFlow [3, 4] uses SDN OpenFlow principles to control networks of Software Defined Radios. In ÆtherFlow [5], the SDN/OpenFlow framework is used to bring programmability to the Wireless LAN setting. They show that this type of system can handle hand-offs better than the traditional 802.11 protocol. These SD-X extensions (X being the MAC layer in this case) focus on centralized configuration of the hardware and do not provide sample statistics on performance that we desire. Closer to our theme, systems such as AeroFlux [6] and OpenSDWN [7] develop a wireless SDN framework for enabling prioritization rules for flows belonging to selected applications (such as video streaming) via middle-boxes using packet inspection. However, they do not tie such prioritization to the impact on application QoE or end-user value across competing applications from multiple clients. Nor do they use measured QoS statistics as feedback.

Queueing: There has been significant work on QoS as a function of the scheduling policy at queues, with a sequence of fundamental results starting with [8]. Follow-on work in the wireless context has resulted in algorithms such as backpressure-based scheduling and routing in wireless networks [9], and more recently, deficit-based scheduling [10, 11] that ensures that strict delay guarantees are met. Most of these works aim at maximizing throughput or loss rate, but they do not consider all the elements of QoS together.

QoE Maps: The map between QoS and QoE has been studied recently, particularly on the wired network. The work in this space attempts to determine the QoS properties of a network, and then based on data obtained directly from an application, match the observed QoS to the corresponding QoE. Mok et al. [12] describe a method for determining the QoE for HTTP Streaming, focusing on the choice of initial streaming rate for maximizing QoE. Other work focuses on different applications, such as Skype [13] or general Web services [13], to identify conditions that are

sufficient to meet the average QoE targets for those applications.

Online Reinforcement Learning: An online learning approach is natural for the control of systems with measurable feedback under each configuration. The idea of using Thompson sampling has received increasing attention, with empirical studies (e.g., [14]) indicating its superior learning rate to prior approaches, and the derivation of fundamental optimal regret bounds [15]. However, we are unaware of work that uses this promising approach for network configuration. The idea of using reinforcement learning in the context of adaptive video rate selection has recently been explored [16]. However, the context of adaptive network configuration for high QoE application perfromance has not been considered earlier.

SDN-based Video Streaming: A number of systems have been proposed to improve the performance and QoE of video streaming with SDN. One direction is to assign video streaming flows to different network links according to various path selection schemes [17] or the location of bottlenecks detected in the WAN [18]. In the home network environment, the problem shifts from managing the paths of video traffic to sharing the same network (link) with multiple devices or flows. VQOA [19] and QFF [20] employ SDN to monitor the traffic and change the bandwidth assignment of each video flow to achieve better streaming performance. However, without an accurate QoS-to-QoE map to predict the QoE, the controller can only react to QoE degradation passively.

1.2 Overview of Approach and Results

This thesis builds upon an system for reconfigurable edge networks entitled QFlow, currently under development in our group. The main idea behind QFlow is to instantiate a platform for reliable delivery of configuration commands to hardware that can support re-configuration. It extends the OpenFlow protocol (currently

limited to the network layer) in a generic manner that enables us to use it reconfigure queueing mechanisms. Using commercially available WiFi routers with Gigabit ethernet backhaul as the wireless edge hardware, reconfigurable queueing is attained by leveraging differentiated queueing mechanisms available in the Traffic Controller (tc) package by installing OpenWRT (a stripped-down Linux version). Here, we can choose between queueing disciplines and set filters to assign flows to queues.

QFlow also enables continuous monitoring of flows at the packet level to obtain per-queue and per station network performance metrics, such as throughput, latency, and RSSI. Furthermore, it allows for monitoring of client-specific application state such as buffered seconds of video and stall duration (when the video re-buffers). These monitors at the WiFi router and the mobile station, are compatible with our OpenFlow extensions, and use the protocol to periodically send statistics to the OpenFlow controller for processing. In order for the statistics ot be meaningful, we reconfigure the system at intervals of 10 seconds, and collect statistics throughout each such interval. Details on QFlow are presented in Chapter 4.

The goal of this thesis is to utilize the reconfiguration functionalities provided by QFlow in a online manner to ensure high QoE video streaming over multiple connected clients. Our setup consists of two queues at an AP over which we employ token-bucket based differentiated services to enable a higher QoS for one queue over another. Three Intel NUCs are used to instantiate up to 9 clients (YouTube sessions). Note that each such session can be associated with multiple TCP flows, and we treat all the flows associated with a particular YouTube session identically. QoS values are measured across the queues at the AP and communicated back to the database, whereas QoE observed at each station (NUC) is calculated based on a decision tree that both accounts for the QoS received, video stalls (re-buffering) and the amount of buffered video of the YouTube session.

Given this experimental setup, our objective is to complete the control loop in Figure 1.1 by designing intelligence into the system in terms of choosing between configurations automatically, i.e., which clients to assign to which queues at each reconfiguration interval. Such a controller can be seen as sampling the state space of cross-layer mechanism combinations as the channel conditions and offered load change dynamically. Our goal then reduces to modeling the system as a Markov Decision Process (MDP), where the state consists of the state of the video players at each client (including the current QoE), and designing a controller that suggests optimal actions (choice of how to assign clients to queues). However, the transition probabilities of this MDP are unknown, which motivates the need to employ Reinforcement Learning (RL) to jointly learn the system and to identify the optimal decision at each time. Distributed computing capabilities enables the efficient collection and processing of sample statistics to inform policy decisions. However, the huge dimensionality of the configuration space implies that reducing the dimensionality of the search space and low complexity of learning algorithms is crucial. Thus, we desire lightweight reinforcement learning algorithms that can learn quickly and accurately.

We approach the problem of RL using a an $\epsilon-$greedy approach under which the policy uses a random action with probability ϵ, and the greedy optimal action with probability $1 - \epsilon$. The value of ϵ is gradually decayed to zero. Given the history of state action pairs seen until the current time, the actual learning aspect of RL lies in determining the value of each possible action using this available history. A simple method for learning is the so-called Q-learning, under which the value of each action under a given policy is determined. Since the state space in our problem is overly large for standard Q-learning, we take recourse to approximation of the Q-function using an Artificial Neural Network (ANN), with appropriate input sequencing using

experience replay and *target network* modifications. The end result is trained ANN that correctly identifies Q-functions in the context of network reconfiguration.

We then evaluate our system using both simulations and over an experimental deployment. Our results on adaptive flow assignment reveal that the vanilla approach of treating all flows identically has significantly worse average QoE than adaptive approaches. More interestingly, flows in both the high and low priority queues in the adaptive approaches outperform the base case, indicating that by selecting flows in need of QoE improvement (due to high likelihood of stalls in the near future), adaptive flow assignment improves QoE for all flows. Finally, the RL approach turns out to be superior to all other approaches, and yields the highest QoE over all methods as it correctly learns the impact of prioritization on each client and selects the appropriate client at each step.

2. SYSTEM MODEL AND PROBLEM STATEMENT

We consider a model in which clients are connected to an wireless Access Point (AP) in a high demand situation. We choose video streaming as the application of interest using the case study of YouTube, since video has stringent network requirements and occupies a majority of Internet packets today [21]. Our goal is to maximize the overall QoE of all the clients in this resource constrained situation. The model used to determine QoE will be explained further in this chapter.

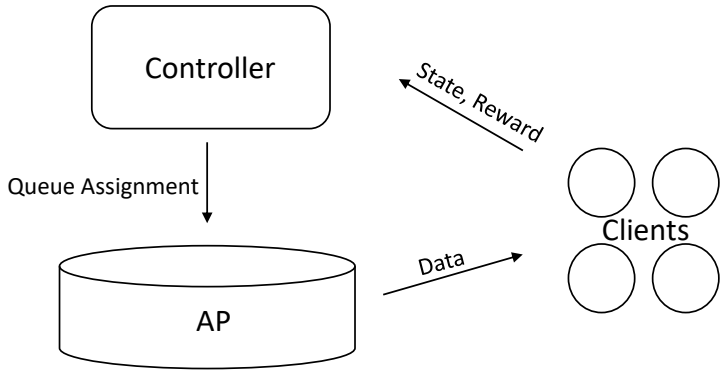

Figure 2.1: Simplified System Model

A simplified model of the system is shown in Figure 2.1. The AP has a high priority and low priority queue. Clients assigned to the high priority queue typically experience a better QoS (higher bandwidth, lower latency etc.) when compared to the clients assigned to the low priority queue. The goal of the controller is to strategically assign clients to each of these queues at every decision period (DP) such that the overall QoE of all clients is maximized. Determining the optimal

strategy is complex, since the controller does not have prior knowledge of the system model. Hence, the controller must *learn* the system model.

Consider a discrete time system where time is indexed by $t \in \{0, 1, ...\}$. At each DP ($t = 0, 1, 2..$) the controller makes an assignment of clients to queues, and observes the system. Based on its observation and previous assignment, the controller makes an assignment in the next DP, eventually learning the system model empirically.

This class of problem falls within the Reinforcement Learning(RL) paradigm, and thus can be abstracted to a general RL framework consisting of an *Environment* that produces *states* and *rewards* and an *Agent* that takes *actions*; these are designated as follows:

Environment: The environment is composed of clients and the AP. Let $\mathcal{C}$ denote the set of clients.

State: Each client keeps track of its state which consists of its current buffer (the number of seconds of video that it has buffered up), the number of stalls it has experienced (i.e., the number of times that it has experienced a break in playout and consequent re-buffering), and its current QoE. The state of the system is the union of the states of all clients. Let s_t^c denote the state of client c at time t and s_t denote the state of the system,

$$s_t^c = \textit{[Current Buffer State, Stall Information, Current QoE]}\ \forall c \in \mathcal{C}$$

$$s_t = [\bigcup_{\forall c \in \mathcal{C}} s_t^c]$$

Agent: The controller is the agent, which takes an action a_t (queue assignment) every decision period in order to maximize its *expected discounted reward*.

Reward: The reward $R(s_t, a_t)$ obtained by taking action a_t at state s_t is the average QoE of all clients in state s_{t+1}.

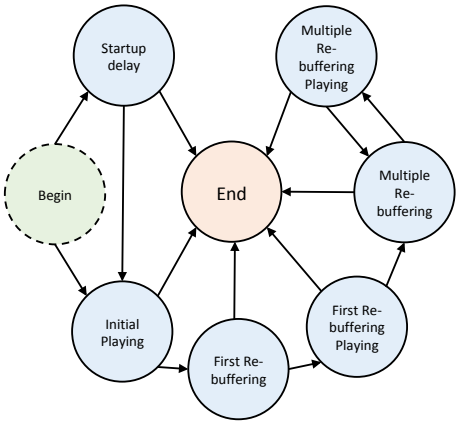

Figure 2.2: DQS State Machine

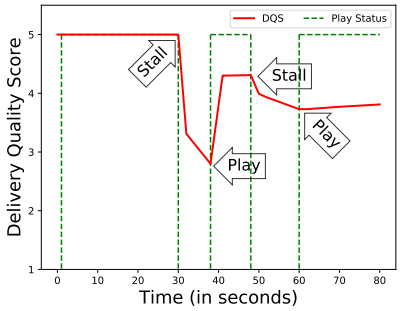

Figure 2.3: DQS Evolution with Stall and Play Events

The goal of the controller is to maximize the overall QoE of the system, which implies maximizing the average QoE over an infinite time horizon. This goal can be formulated as maximizing the expected discounted reward over an infinite horizon. Let $\pi(a_t|s_t)$ denote the probability of taking action a_t given the current state (called the policy) and γ denote the discount factor. Then the goal is to find π^*, the policy that maximizes the expected discounted reward,

$$\pi^* = \text{argmax}_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|s_0 = s, a_t \sim \pi(\cdot|s_t)\right].$$

2.1 QoE Model

We require a model to determine the human perception of QoE. We studied three models in this context, namely Delivery Quality Score (DQS) [22], generalized DQS [23], and Time-Varying QoE (TV-QoE) [24]. All of the three models are based on the same features (stall event information) if there is no rate adaptation. Since our goal is to support high resolution video without degradation, we fix the resolution so as to prevent video rate adaptation. Under this scenario, all three models are

fundamentally similar, and we choose DQS as our candidate.

Without rate adaptation, the impairments in video streaming that have the greatest impact on the QoE are startup delay and stalling events [25, 26, 27] . The DQS model weights the impact according to duration of the impairments to better model the human perception. For example, the impact of stall events during playback is greater on the QoE than that of initial buffering. Similarly, the first stalling event is looked at with less dissatisfaction than repeated stalling. The state diagram of the model is shown in Figure 2.2. The increases and the decreases in perceived QoE are captured by a function which is a combination of raised cosine and ramp functions. This enables it to model greater or lesser changes in the perceived QoE according to the time it spends in a particular state. The behavior of the predicted QoE by the model in the presence of a particular stalling pattern can be seen in Figure 2.3, where the two stall events result in degradation of QoE. Recovery of QoE from each stall event becomes progressively harder.

3. REINFORCEMENT LEARNING APPROACH TO EDGE CONFIGURATION

The goal of an RL agent is to learn how to behave in an uncertain environment in a way that maximizes its objective. The agent learns by interacting with the environment, via state, actions and reward. State is used to describe the agent with regard to the environment it is acting in. Given a state that the environment is in, the agent applies an action a, and obtains a reward r from the environment. The action can be thought of as a change applied to the environment in order to achieve its goal. Reward measures the immediate response of the environment to the action applied when it is in state s.

The interaction of agent with the environment is modeled as a set of tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ over time. It represents a transition from one state to another state when an action is applied, and the reward obtained from the environment. Based on these interactions, the RL algorithm needs to extract a policy π that recommends an action to take, given a state, in order to maximize its long term cumulative reward. In other words, it has to act in such a way that minimizes mistakes over time.

3.1 Exploration-Exploitation in RL

The RL algorithm faces the fundamental dilemma between exploring the environment, i.e., sampling new states and actions, and exploitation of the knowledge it accrued so far. This trade-off is instantiated in the ϵ-greedy algorithm. The algorithm operates in the following manner: At any time, the agent takes a random action with ϵ probability, and an action with respect to the greedy policy (i.e., one that provides maximum value given the current knowledge of the system) with probability $1 - \epsilon$. Here, ϵ aids in exploration. The value of ϵ is gradually reduced from a

maximum value to a minimum value. This process means that the algorithm exploits the knowledge it has gained about the system with more and more certainty as the time progresses.

3.1.1 Q-learning

Each state-action pair (s, a) under a policy π can be mapped to a scalar value, using a Q-function. $Q(s, a)$ is the expected reward of taking an action a in a state s and following the policy π from there on.

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | s_0 = s, a_0 = a\right] \tag{3.1}$$

, where $\gamma \in [0, 1]$ is the discount factor, which quantifies our preference for immediate rewards, in an infinite time horizon. Thus, maximizing the cumulative reward is equivalent to finding a policy that maximizes the Q-function. The Q-learning algorithm [28] is aimed at this. The control algorithm can be represented by a Markov Decision Process. The optimal Q-function satisfies the Bellman equation,

$$Q(s_t, a_t) = R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}), \forall s_t, a_t. \tag{3.2}$$

Q-learning can be implemented by an iterative procedure, as follows:
Given samples $s_0, a_0, R_0, s_1, a_1, R_1, \ldots$ where $s_{k+1} \sim P(.|s_k, a_k)$ where P is the transition model of the system, which is unknown to the RL agent.

$$Q_{k+1}(s, a) = \begin{cases} (1 - \alpha_k) Q_k(s, a) + \alpha_k \left(R_k + \gamma \max_b Q_k(s_{k+1}, b)\right), & \text{if } s = s_k, a = a_k \\ Q_k(s, a), & \text{otherwise} \end{cases} \tag{3.3}$$

The convergence of Q-learning depends upon visiting each state action pair infinitely

often. Hence, one approach is to use ϵ-greedy policy in conjunction with the above iterative algorithm. In the literature, these techniques are combined with table look up for Q-values for the state transitions. In the next section, we argue that such a scheme does not scale well with the large dimensionality of our problem, and we propose a different approach.

3.2 A Discussion on the State Space and RL Algorithms

Figure 3.1 depicts the learning scenario in an edge configuration problem. The individual client states are combined to form a comprehensive state. The aggregate reward is the reward of all clients combined. The learning agent observes the states and rewards, and outputs an action. The environment then moves to the next state, yielding a reward.

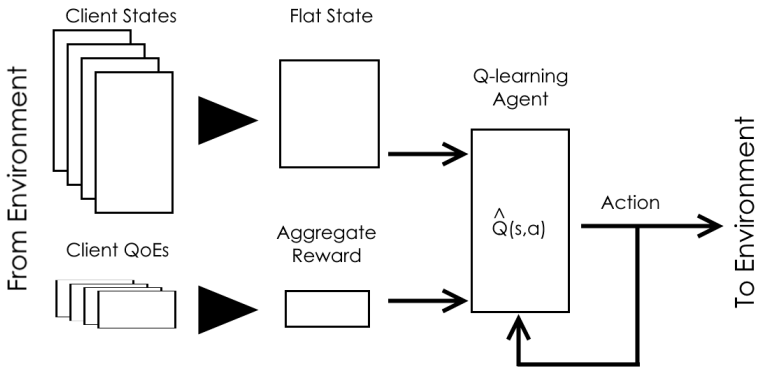

Figure 3.1: Comprehensive Learning Diagram

The goal of our work is to model the learning agent for Edge Configuration problem. Using RL to solve the Edge Configuration problem poses a number of difficulties. Most importantly, the state space corresponding to Edge Configuration problem is huge, compared to many traditional RL formulations. As discussed in

section 2, our state s_t comprises of union of states of all clients at time t. The state of each client is a tuple consisting of its buffer state, stall information, and its QoE at t. Buffer state and QoE are considered to be real numbers, and thus can take an uncountable number of values. In such large state spaces, tabular methods discussed in the previous section are very memory inefficient.

Furthermore, a change in state observed by the agent depends upon the QoS of the queues on the AP in which the agent places the flows. The QoS of a queue at the AP is random due to the dynamic nature of the wireless environment (effects of fading). The RL agent has to act in this uncertain environment.

To overcome these challenges, we propose a Deep Reinforcement Learning framework that makes use of the following approaches: Q-learning with Function approximation, experience replay, and target network. We now explain each of these components.

3.2.1 Q-learning with Function Approximation

To address the problem with high dimensionality of the state space, we use function approximation techniques to approximate the Q-function. Since we can not exhaustively visit every state and action pair to learn its Q values, function approximation generalizes the Q-function from seen states to unseen states.

$$Q(s, a) \approx Q(s, a; w) \tag{3.4}$$

i.e, the Q function is parameterized by a weight vector w, and we learn w. Then, given any state s and action a, we can obtain the Q values. Hence, we need a good class of function approximators. Artificial Neural networks (ANN) are empirically proven to be able to provide a generic class of function approximators that scale well given the size of the problem domain. Thus, our natural choice is to adopt ANN's to

estimate the Q-functions. Performing Q-learning with function approximation using ANN's is via the Deep Q Network (DQN) algorithm [1]. The parameter update equation for Q-learning with FA is given as the following gradient descent equation, with step size α.

$$w = w + \alpha \nabla Q_w(s_t, a_t)(R_t + \gamma \max_b Q_w(s_{t+1}, b) - Q_w(s_t, a_t)) \tag{3.5}$$

3.2.2 Experience Replay

In using Q-learning with function approximation, sequential states are used to approximate the Q function. i.e., take action a_t, and obtain the data point $e = \{s_t, a_t, R_t, s_{t+1}\}$. Clearly, the transitions obtained this way are highly correlated. Hence, Q-learning may not converge. An attractive solution to this problem is using experience replay [1]. Experience replay uses the idea of supervised learning. As the agent explores the environment, it adds the transitions $\{s_t, a_t, R_t, s_{t+1}\}$ to the replay buffer. To learn the Q-function, a random mini batch of K data points, sampled from the replay buffer are utilized. This way, it is ensured that the data points are not correlated.

3.2.3 Target Network

Another problem with Q learning with function approximation as seen in equation (3.5) is that the Q-learning target $R_t + \gamma \max_b Q_w(s_{t+1}, b)$ is moving along with the parameter w. Hence, the stochastic gradient descent may not converge. To overcome this, we fix the for the target for multiple steps. With the target network, the

parameter w update equation becomes

$$w = w + \alpha \nabla Q_w(s_t, a_t)(R_t + \gamma \max_b Q_{w^-}(s_{t+1}, b) - Q_w(s_t, a_t))$$

$$w^- = w \quad \text{after every } N \text{ steps.}$$

The DQN algorithm, using above mentioned techniques, suffers from overestimation of Q values. As seen from equation (3.5), Q learning uses $\max_b Q_w(s_{t+1}, b)$ as an estimate for $\max_b \mathbb{E}\left[Q_w(s_{t+1}, b)\right]$. To overcome this problem, we propose to use a Double DQN [29] based algorithm. Similar to DQN, Double DQN also uses two Q-networks, one to select actions, and the other to improve the actions, which is called the target network. Double DQN algorithm improves the DQN's target network in the following manner.

3.2.4 Target Network for Double DQN algorithm

The Double DQN's parameter update equation is shown below:

$$w = w + \alpha \nabla Q_w(s_t, a_t)(R_t + \gamma Q_{w^-}(s_{t+1}, \arg\max_b Q_w(s_{t+1}, b)) - Q_w(s_t, a_t)) \quad (3.6)$$

Here, the target is computed at the target network based on the greedy action with respect to the current Q network. In DQN, the target is computed solely based on the greedy action with respect to the target Q network. This way, double DQN overcomes the over estimation problem in DQN. Below, we show the pseudo code for Double DQN algorithm implemented for the Edge Configuration problem.

Algorithm 1 A Double DQN algorithm for Edge configuration

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights w

Initialize target action-value function Q^- with weights $w^- = w$

Initialize state s_1

for $t = 1, \ldots, T$ **do**

With **probability** ϵ select a random action a_t.

Otherwise select **greedy action** $a_t = \arg\max_a Q_w(s_t, a)$.

Play action a_t and observe reward R_t and state s_{t+1}.

Store **Experience** (s_t, a_t, R_t, s_{t+1}) in D.

Sample random mini batch of **Experience** (s_j, a_j, R_j, s_{j+1}) from D.

Set **Target** $\gamma_j = R_j + \gamma Q_{w^-}(s_{j+1}, \arg\max_b Q_w(s_{j+1}, b))$.

Perform gradient descent step on $(\gamma_j - Q_w(s_j, a_j))^2$ with respect to the parameters w.

Every C steps reset $w^- = w$.

end for

Algorithm 1 uses ideas of ϵ-greedy policy, experience replay and target network, as discussed earlier. Though no convergence guarantees exist theoretically, empirical evidence suggests that these techniques are shown to perform very well in several games [29]. The details of the Double DQN implementation for edge configuration problem are discussed in Chapter 5.

4. A SOFTWARE DEFINED ARCHITECTURE FOR NETWORK CONFIGURATION

The architecture of the Self Configuring Network (SCN) is shown in Figure 4.1. The system consists of three main parts, an off-the-shelf 802.11n Wireless Access

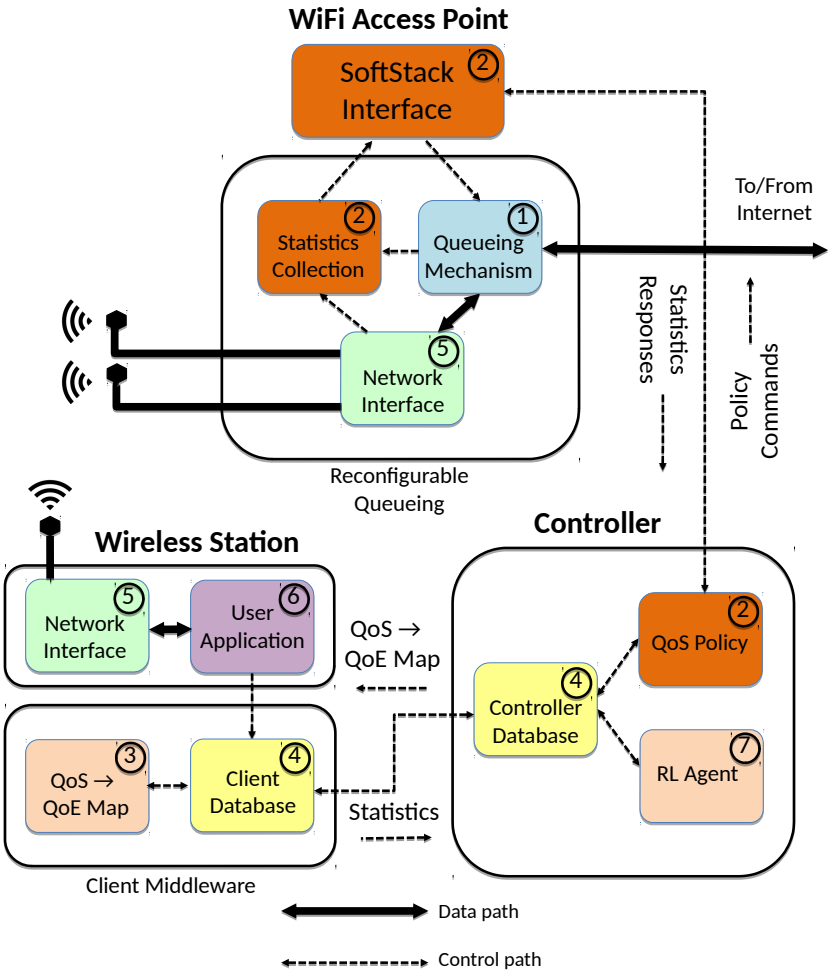

Figure 4.1: System Architecture

Point, multiple wireless stations and a centralized controller. Each of these units contain multiple components, which have been represented with different colors and numbers to identify functionality:

① **Per-Packet Queueing Mechanisms:** We chose queues at the MAC layer as a reconfigurable mechanism for our system because of the impact they have on the performance of flows. The configuration options include creation of multiple queues, allocation of bandwidth and application of different scheduling algorithms. This enables us to apply mechanisms with different priorities/capabilities to different sessions, resulting in contrasting performance at the end users.

② **QoS Policy:** A policy decision determines the assignment of sessions to queues. Each such assignment results in the realization of QoS vectors for each of the queues created. Such decisions are made at the centralized controller and are forwarded to the Access Point using OpenFlow commands. The SoftStack interface interprets these commands and reconfigures the queueing mechanisms accordingly. Statistics corresponding to the queues, which include Throughput, Round Trip Time and Drop Rate, are periodically sent back to the Controller as custom OpenFlow messages.

③ **Application QoE:** The Client middleware layer abstracts the system from the end user. It is responsible for identifying and collecting data specific to the foreground application (like current QoE, state of the buffer, stall information). Available QoS options are retrieved from the Controller database. Regression techniques are then employed to predict the QoE for the application, given the current application state and the QoS vector. The updated state of the application is sent to the Controller database.

⑦ **RL Agent:** The Reinforcement Learning (RL) agent makes the policy decision for scheduling all the participating clients on the available reconfigurable queue-

ing mechanisms. It extracts the current states of all the clients from the Controller database and feeds this extended state as an input to the learning algorithm to obtain the policy decision (assignment). This assignment, which is saved to the Controller database, is subsequently retrieved by the QoS Policy component and sent to the Access Point for implementation.

Tying together the units are ④ **Databases** at the Controller (to capture the MAC layer state), and at each client (to capture the application specific state). ⑤ **Network Interface** and ⑥ **User Application** are unaware of our system.

The interactions between the different components described above are shown in Figure 4.2 in a chronological order. The Client Middleware sends a request to the Controller to retrieve the list of available QoS vectors, which is then used to predict the perceived QoE of the client. The state of the client, which comprises of this QoE value along with application state, is then sent to the Controller. The RL Agent receives the state of all the clients and then uses this cumulative state information as an input to the RL algorithm. The output of the algorithm is the policy (assignment of sessions to queues), which is sent to the Access Point using OpenFlow. The SoftStack component interprets these commands and performs the implementation of the policy. The resultant QoS vector for each of the two queues is then sent back to the Controller using OpenFlow messages.

4.1 Implementation Details of SCN

In this section, we take a closer look at the the different functional components described in the previous section. OpenFlow is a communication protocol which empowers a centalized controller to modify the forwarding tables of network routers and switches. It abstracts away the vendors specific details of network devices and enables separation of the previously tightly coopled control and data planes. We

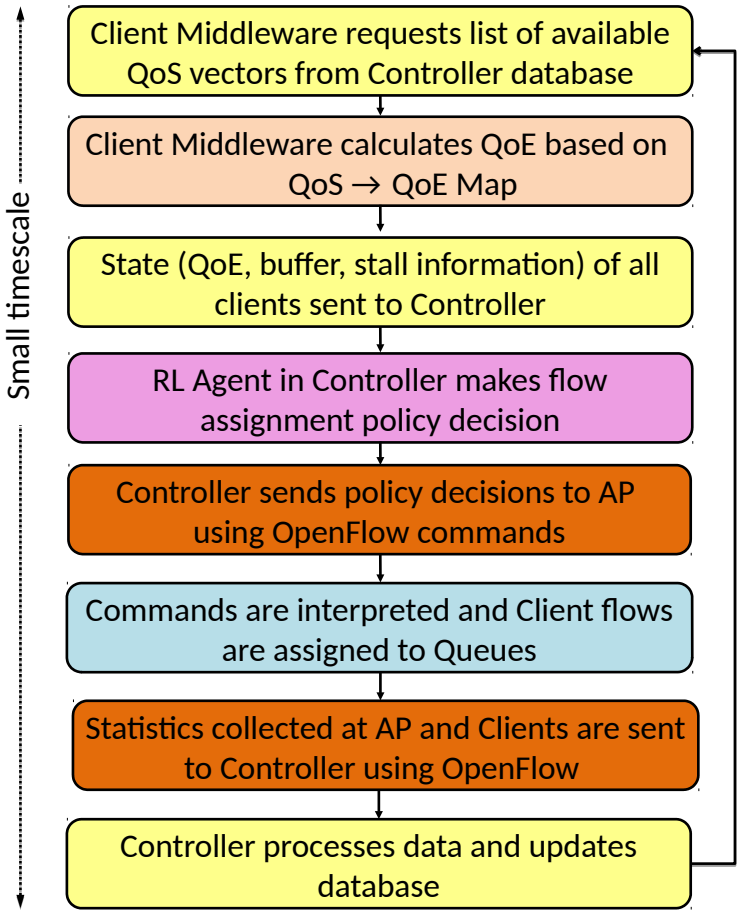

Figure 4.2: Order of Interactions Between the Components of the System.

use experimenter messages to forward custom SCN commands to an off-the-shelf TP-Link WR1043ND v3 router. The router is installed with OpenWRT Chaos Calmer as the operating system. OpenWRT supports Linux based utilities like `tc` (Traffic Control) which simplifies implementation of reconfigurable queueing mechanisms. We use CPqD SoftSwitch [30], an OpenFlow 1.3 compatible user-space software switch implementation, to enable support for OpenFlow.

We next require capabilities to create and modify the queueing mechanisms.

The extension of SoftSwitch, which we name SoftStack, empowers us to make the necessary mechanism changes. It also allows us to collect both queue and client specific statistics. We define two types of SCN commands, *Policy commands* and *Statistics commands* to implement these capabilities.

4.1.1 Queuing Mechanisms

Traffic control (`tc`) is a Linux utility that enables us to configure the settings of the kernel packet scheduler by allowing us to *Shape* (control the rate of transmission and smooth out bursts) and *Schedule* (prioritize) traffic. Each network interface is associated with a *qdisc* (Queueing discipline) which receives packets destined for the interface. *qdisc*s can be Classful or Classless. Classless *qdisc*s perform basic traffic management tasks like reordering, slowing down or dropping packets. Classful *qdisc*s, on the other hand, enable us to create children classes, each of which can be assigned a Classful/Classless *qdisc*, and are meant for more complicated traffic scenarios, like when different flows have to be treated differently. We selected *Hierarchical Token Bucket* (`htb`) which is a Classful *qdisc*, for our experiments because of the versatility of the scheme. It performs shaping by specifying *rate* (guaranteed bandwith) and *ceil* (maximum bandwidth) for a class, with sharing of available bandwidth between children of the same parent class, and can also prioritize classes. Finally, *Filter*s are used to classify and enqueue packets into classes. We use `tc` with `htb` to create a hierarchy of classes with specified *rate*s and *ceil*s to provide different levels of service during the course of our experiments.

4.1.2 Policy Commands

Policy commands enable the implementation of a policy decision, which is to select the appropriate mechanism (from the list of available ones) for each client. The packet format of a policy messages is shown in Figure 4.3. The QoS Policy

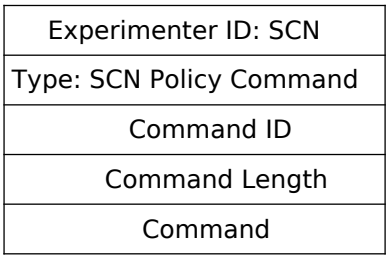

Figure 4.3: Policy Command Packet

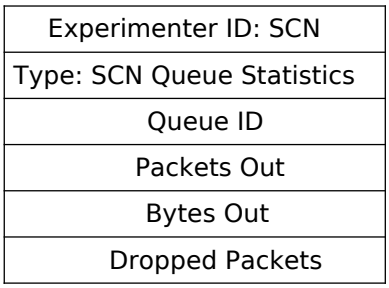

Figure 4.4: Queue Statistics Packet

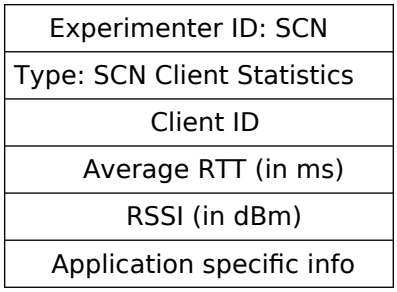

Figure 4.5: Client-specific Statistics Packet

component encapsulates a policy decision in the format, and sends it to the Access Point using OpenFlow. When SoftStack receives the message, it verifies the identity of the message from the headers, interprets the policy command, and implements the requested reconfiguration. We have implemented policy commands for reconfiguring the queuing mechanisms at the MAC layer.

We require mechanisms with different capabilities at the MAC layer to provide prioritized service to the clients. In our experiments, we create two queues with different token rates using `htb`. We enable sharing of tokens between queues to prevent wastage in scenarios of excess bandwidth. We also create a default queue to handles any background traffic.

Decisions at the data link layer include assigning flows to queues, changing the bandwidth allocated to the queues, and enabling or disabling sharing of excess (un-

used) bandwidth between them.

4.1.3 Statistics Commands

Implementation of a policy decision at the Access Point results in variations of the QoS vector (for the queues). These variations are captured using Statistics commands and communicated back to the Controller. Statistics related to the MAC layer queues include cumulative counts of downlink packets, bytes and dropped packets. In addition to the queue statistics, client state is also affected by a policy decision. Client-specific statistics like Round Trip Times (RTT), signal strength (RSSI) and Application specific statistics like buffer state, stall information and video bitrate are sent also periodically (once every second) to the controller.

We define the structure of both Queue and Client-specific Statistics messages for validation and correct interpretation of the messages at the Controller. The packet formats of the two types of Statistics messages are shown in Figure 4.4 and Figure 4.5. At the Access Point, SoftStack encapsulates the raw statistics in the indicated format and sends them to the Controller using OpenFlow. When the Controller receives a Statistics message, it verifies the type of messsage by reading the header information and then saves the extracted statistics to the database.

The client-specific statistics, collectively with the statistics of the queue it is assigned to, comprise the Quality of Service (QoS) for a client, and is a measure of the service provided by the network. QoS is a vector which contains [$throughput$, RTT ($mean \,\&\, median$), $jitter$, $drop\,rate$, $buffer\ state$, $stall\,information$, $video\,bitrate$], and is used to estimate the Quality of Experience (QoE) of the application.

5. EXPERIMENTAL EVALUATION

We evaluated our approach in order to show the advantages of online adaptive approach when compared to a fixed policy. We focused on the questions of how quickly the agent is able to find the optimal policy, as well as the performance of its optimal policy as compared to a selection of other scheduling algorithms.

The evaluation consisted of three parts: first, the implementation of an appropriate RL algorithm; second, performance evaluation on a simulated system; finally, performance evaluation on a physical wireless testbed. Each system uses a Decision Period (DP) timescale of 10 seconds; that is, the agent observes the environment and chooses its next action every ten seconds.

5.1 Implementation Approach

5.1.1 RL Agent

We initially implemented a custom RL agent based on DQN which included certain later upgrades. However, we ultimately opted to use the TensorForce library implementation of DQN in order to take advantage of the generalizability, efficiency, and optimization present in the existing work and which it would not be productive to duplicate [31].

Random search for hyperparameters of the agent was carried out using the simulated system; see subsection 5.2.1 for examples of the comparative performance. The final agent configuration chosen for evaluating the efficacy of this approach is detailed in Table 5.1.

Hyperparameter	*Chosen Value*
Discount	0.9999
Network Hidden Layers	$(64, 32)$
Network Optimizer	Adam, Learning Rate 0.001
Replay Buffer	500000
Replay Batch	32
Target Sync Period	100000
Huber Loss	1.0
Double Learning	On
Control Policy	ϵ-greedy, Decay ϵ from 1.0 to 0.01 over 1000000 steps

Table 5.1: RL Hyperparameter Selection

5.1.2 Other Algorithms

We also implemented the following policies in order to serve as comparative benchmarks for the agent:

Vanilla This policy serves as the lower bound on performance. It is set apart by having only one queue available, to which all resources are allocated and to which all clients are given access; no assignment actions are available.

Round-Robin This policy chooses each assignment in turn, giving all flows equal time in the high-priority queue. Although computationally cheap at 0(1), it does not take into account the system state at all. As such, it may serve clients which are unable to make use of the service due to buffer fill or being in a state where no amount of service can significantly increase experience quality.

Greedy Buffer This policy chooses the two clients with the smallest buffer to assign to the high-priority queue. This is still a relatively cheap algorithm, linear in the number of clients because it computes the minima of an unsorted list. It also improves on Round-Robin by focusing on the client which is most likely

to stall next, forming the most intuitive of the heuristic policies; this policy is similar to the one found by the Bandit and RL agents in exploration. More formally, it treats the buffers built up by the clients as anti-deficit, where $d_t = min(0, d_{t-1} + 10 - D_t)$ is the deficit at time t where D_t seconds of video are delivered at a given time; the highest two deficits, corresponding to the lowest two buffers, are given high-priority service.

Reward Greedy This policy is identical to the RL agent, with the single modification of setting the future value decay $\gamma = 0$. This produces a myopic agent which performs single-step maximization of the reward like a contextual multi-arm-bandit. This policy is included to demonstrate the worth of considering long-term value.

5.2 Simulation

In order to quickly iterate on and validate algorithms and implementations, as well as pre-train agents for the physical testbed, we implemented a simulation environment which closely mimics the dynamics of the physical testbed with YouTube clients. The environment simulated each video including its bitrate, buffer, length, and QoE separately. The bitrate and length of each video was generated according to a normal distribution; buffer was stored in terms of time, rather than bits. Each client would continuously play one video after another, stalling where its buffer runs out and building up a buffer of 10 seconds before attempting playing again. Queue performance was kept to a constant total bandwith, but the fairness of queue's service among flows assigned to that queue was chosen in each DP according to a Dirichlet distribution.

For these simulations, the environment uses a high-priority queue with 11 Mbps bandwidth and a low-priority queue with 4.3 Mbps (Vanilla simulation used a sin-

gle 15.3 Mbps queue). Six clients are specified which draw video bit-rates from a $\mathcal{N}(2.9, 10)$ distribution in Mbps, and draw video lengths from a $\mathcal{N}(600, 50)$ distribution in seconds.

5.2.1 Hyperparameter Validation

For hyperparameter search, the system was simulated for 200 DP per episode for 1000 episodes, as seen in each plot of Figure 5.1. Note that increasing the number of units or layers in the network used for value estimation after $(64, 32)$ does not significantly affect the convergence curve; however, the magnitude of the learning rate creates large differences in the performance to which the agent ultimately converges. Further, a single layer is incapable of learning to the performance achieved by the two-layer network. We therefore choose the $(64, 32)$ configuration for our agent.

5.2.2 Online Learning

Upon choice of these hyperparameters, a new agent with the chosen parameters was allowed to run with learning on a single 5 Million DP episode and all other settings identical to those used for validating the hyperparameters, the results of which is shown in Figure 5.2 in both raw reward at each time-step and with an exponential moving average applied. It stabilizes at an approximate 4.25 average QoE by approximately 500000 DP (about 1.9 months of simulated time). This demonstrates that the agent requires significant training in order to be useful on a real, dynamic system within a reasonable period of time, but also that it is capable of learning online, so may be able to take advantage of transfer learning to warm-start with a good policy and tune it to specific contexts.

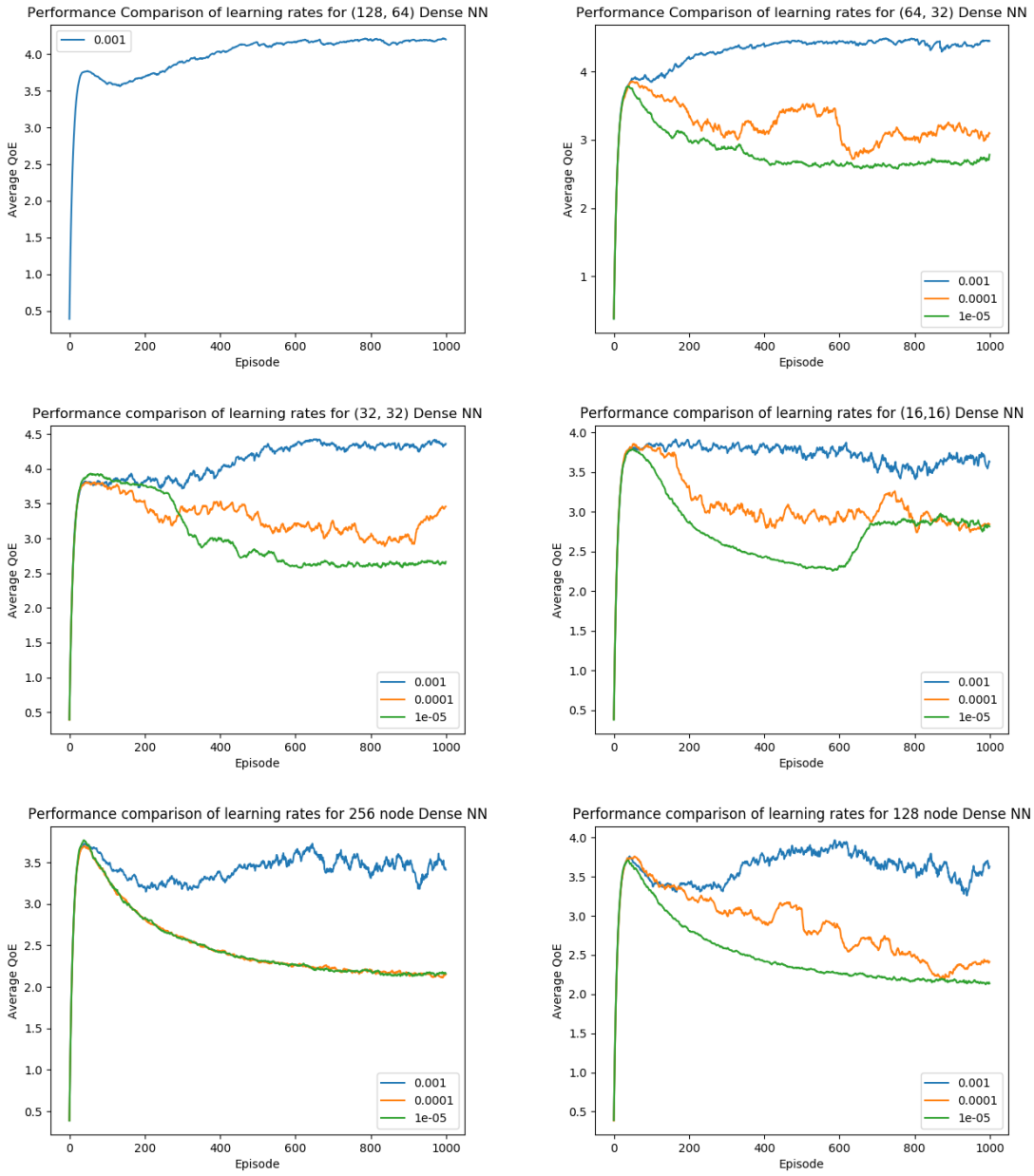

Figure 5.1: RL Selected Hyperparameter Training Comparison

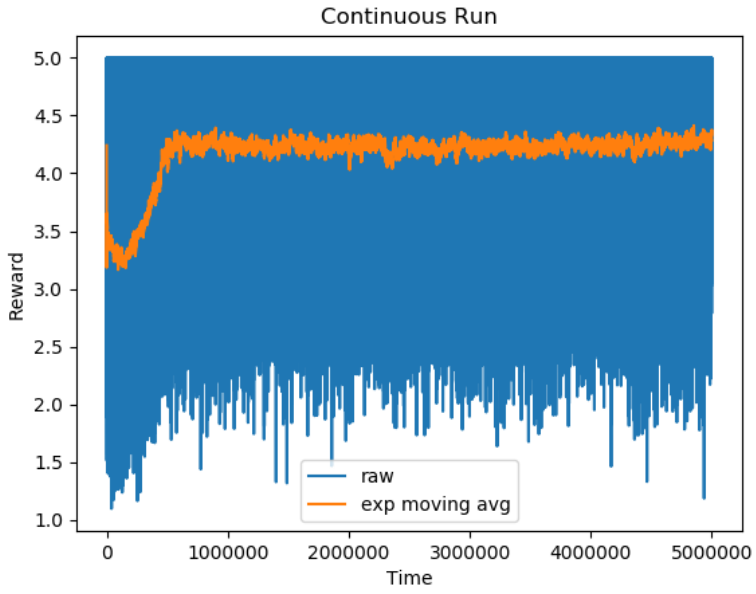

Figure 5.2: Single Episode RL Agent Long Run

Figure 5.3: Average Episodes for Compared Policies

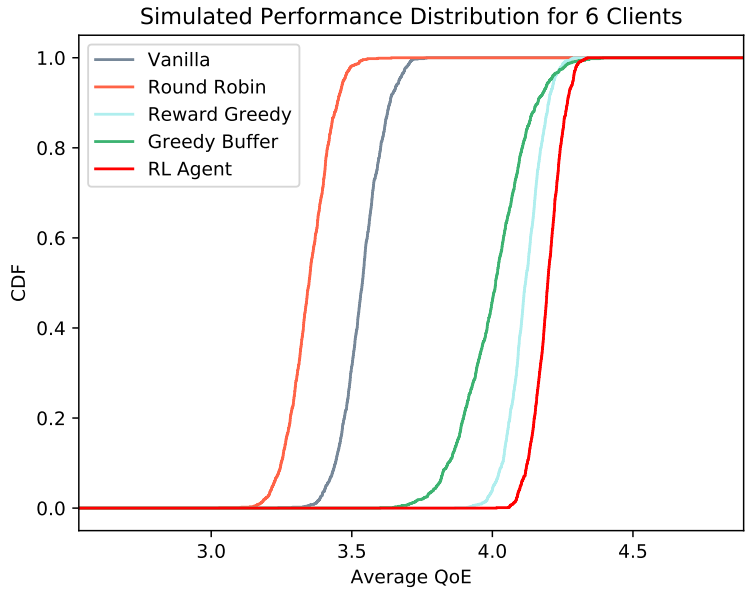

Figure 5.4: Virtual Performance Distribution for Compared Policies

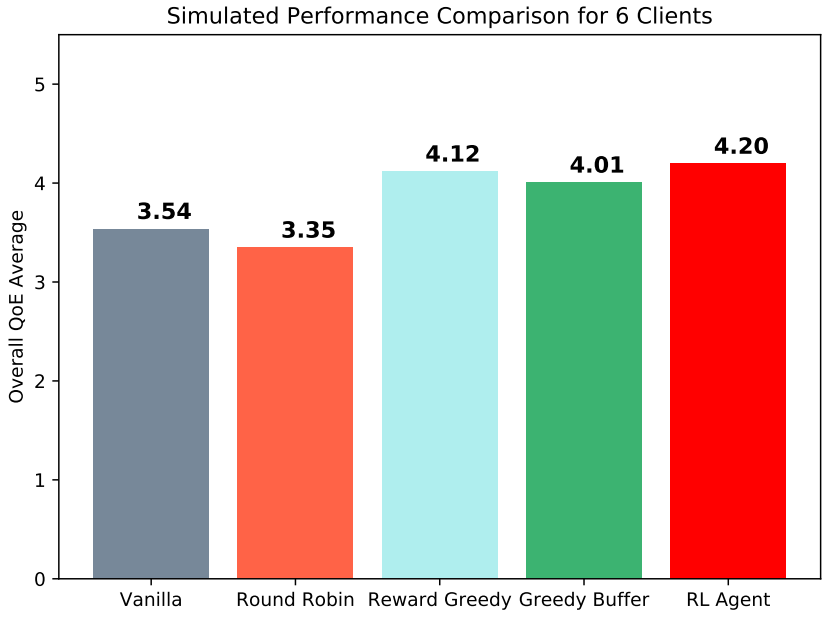

Figure 5.5: Virtual Performance Averages for Compared Policies

5.2.3 Policy Comparison

Finally, a 1000-run average was taken on this virtual system for a 5000 DP episode to compare this trained RL agent to the alternative policies detailed above. Figure 5.4 shows the distribution of performance for each policy, summarized to the empirical mean in Figure 5.5, while Figure 5.3 shows the reward trace for an episode averaged over the 1000 runs at each time step. The trained RL agent outperforms all other policies as well as providing a QoE which does not vary greatly with time. Notice the prominent transient QoE spike at the beginning of the episode, which is due to starting all clients at once with a fresh video; this transient fades over time as the video start times desynchonize due to the normally-distributed durations.

5.3 On-System Performance

A physical testbed was constructed in order to test a practical implementation of the RL policy. SoftStack was installed on a commercial WiFi router to serve as the AP. Three Intel NUCs (equipped with 5th generation i7 processors and 8GB memory) serve as client machines, each running 2 client YouTube sessions while connected wirelessly to the AP. Each YouTube session is associated with multiple TCP flows, which are treated as a bundle for the sake of this experiment; that is, the ability to separate out content flows from advertisement or other flows was considered out-of-scope. Each client reported relevant session information to a central database, such as ports used, play/load state, bitrate, stall information, and QoE. Videos randomly chosen with replacement from a selected list of videos were played sequentially as long as the system was active. YouTube was chosen as the experimental application due to its popularity and free access; this method can be applied to easily to other sites which employ MPEG-DASH streaming technology, such as Netflix and Hulu, with little modification.

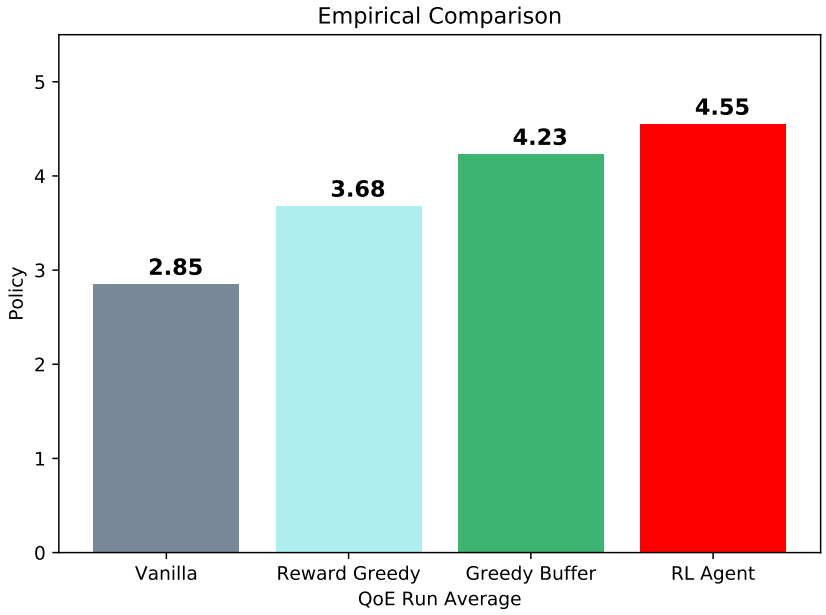

Figure 5.6: Empirical Performance Averages for Compared Policies

The access point was allowed a total of 7 Mbps, with a 2.8 Mbps low-priority queue and a 4.2 Mbps high-priority queue (or a single 7 Mbps queue in the case of the Vanilla policy). These quantities were chosen to produce a constrained system where perfect service, that is, zero stalls and 5.0 episode average QoE, is impossible. Each policy tested was allowed to run continuously for 30 minutes. For the RL performance, the agent trained on the 5M DP run on the simulator was loaded at the beginning of the run and allowed to learn as it controlled the system. The comparison of these results can be found in Figure 5.6. The RL agent outperforms all other policies.

Earlier work [32, 33, 34] has indicated that a threshold approach using the deficit in received packets to determine which clients to prioritize might be the optimal approach to attaining high QoE video streaming. However, this does not account for the fact that QoE follows a Markov process of its own, and choosing the correct client

also depends on how many stalls it has incurred. It is interesting to note that that the RL approach outperforms the deficit approach by likely learning which clients are essentially unrecoverable, and focusing on those that have the maximum impact on average QoE.

6. CONCLUSION

In this thesis, we considered the design, development and evaluation of a reinforcement learning (RL) approach to self configuring networks. We were led by the recent success of RL in a variety of control applications, and our goal was to distill these successes into algorithmic choices that could quickly and accurately learn the Markov Process underlying our system. We chose a focus application of video streaming due to its need for high QoS at certain critical states of the application in order to attain high QoE at the user level.

We developed a Markov Decision Process model of the system, whose parameters are unknown and must be learned in an online manner. We used simulations to winnow out bad candidates, and to determine good choices for the parameter space. Working with off-the-shelf hardware and open source operating systems and protocols, we then showed how implicit learning of queueing behavior via RL able to decide on the correct configuration to best suit the needs of video streaming applications.

As our YouTube observations suggest, such a holistic framework that accounts for this entire chain can reveal efficiencies and interactions that a narrow focus on individual components of the system is incapable of achieving. We believe that the application of our system will be in upcoming small cell wireless architectures such as 5G, and our goal will be to extend our ideas to such settings.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] A. Ericsson, "Ericsson mobility report: On the pulse of the networked society," *Ericsson, Sweden, Tech. Rep. EAB-14*, vol. 61078, 2015.

[3] P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde, and A. Sprintson, "Crossflow: A cross-layer architecture for SDR using SDN principles," in *Proceedings of IEEE NFV-SDN*, 2015.

[4] P. Shome, J. Modares, N. Mastronarde, and A. Sprintson, "Enabling dynamic reconfigurability of SDRs using SDN principles," in *Proceedings of Ad Hoc Networks*, 2017.

[5] M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton, "Ætherflow: Principled wireless support in SDN," in *Proceedings of IEEE ICNP*, 2015.

[6] J. Schulz-Zander, N. Sarrar, and S. Schmid, "AeroFlux: A near-sighted controller architecture for software-defined wireless networks," in *Proceedings of USENIX ONS*, 2014.

[7] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic control over home and enterprise WiFi," in *Proceedings of ACM SOSR*, 2015.

[8] L. Tassiulas and A.Ephermides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Automat. Contr.*, vol. 37, no. 12, pp. 1936–1948, 1992.

[9] A. Eryilmaz, R. Srikant, and J. Perkins, "Stable scheduling policies for fading wireless channels," *IEEE/ACM Trans. Network.*, vol. 13, pp. 411–424, April 2005.

[10] I. Hou, V. Borkar, and P. Kumar, "A theory of QoS for wireless," in *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, April 2009.

[11] S. Yau, P.-C. Hsieh, R. Bhattacharyya, K. Bhargav, S. Shakkottai, I. Hou, and P. Kumar, "Puls: Processor-supported ultra-low latency scheduling," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2018, pp. 261–270.

[12] R. Mok, W. Li, and R. Chang, "IRate: Initial video bitrate selection system for HTTP streaming," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1914–1928, June 2016.

[13] T. Spetebroot, S. Afra, N. Aguilera, D. Saucez, and C. Barakat, "From network-level measurements to expected quality of experience: The Skype use case," in *Proceedings of IEEE M&N*, 2015.

[14] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *Advances in neural information processing systems*, 2011, pp. 2249–2257.

[15] S. Agrawal and N. Goyal, "Near-optimal regret bounds for Thompson sampling," *Journal of the ACM (JACM)*, vol. 64, no. 5, p. 30, 2017.

[16] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.

[17] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of YouTube video streaming," in

Proceedings of EWSDN, 2013.

[18] H. Nam, K.-H. Kim, J. Y. Kim, and H. Schulzrinne, "Towards QoE-aware video streaming using SDN," in *Proceedings of IEEE GLOBECOM*, 2014.

[19] S. Ramakrishnan, X. Zhu, F. Chan, and K. Kambhatla, "SDN based QoE optimization for HTTP-based adaptive video streaming," in *Proceedings of IEEE ISM*, 2015.

[20] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using Openflow-assisted adaptive video streaming," in *Proceedings of ACM FhMN*, 2013.

[21] Ericsson, "Ericsson Mobility Report: On the Pulse of the Networked Society," https://www.ericsson.com/assets/local/mobility-report/documents/2015/ericsson-mobility-report-june-2015.pdf, 2015.

[22] H. Yeganeh, R. Kordasiewicz, M. Gallant, D. Ghadiyaram, and A. C. Bovik, "Delivery quality score model for Internet video," in *Proceedings of IEEE ICIP*, 2014.

[23] N. Eswara, K. Manasa, A. Kommineni, S. Chakraborty, H. P. Sethuram, K. Kuchi, A. Kumar, and S. S. Channappayya, "A continuous QoE evaluation framework for video streaming over HTTP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. In press, 2017.

[24] D. Ghadiyaram, J. Pan, and A. C. Bovik, "Learning a continuous-time streaming video QoE model," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2257–2271, May 2018.

[25] R. Mok, E. Chan, and R. Chang, "Measuring the quality of experience of HTTP video streaming," in *Proceedings of IFIP/IEEE IM*, 2011.

[26] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, "Quantification of YouTube QoE via crowdsourcing," in *Proceedings of IEEE ISM*, 2011.

[27] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs," *IEEE/ACM Trans. Network*, vol. 21, no. 6, pp. 2001–2014, 2013.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, vol. 2. Phoenix, AZ, 2016, p. 5.

[30] CPqD, "OpenFlow Software Switch," http://cpqd.github.io/ofsoftswitch13/, 2015.

[31] M. Schaarschmidt, A. Kuhnle, and K. Fricke, "Tensorforce: A tensorflow library for applied reinforcement learning," *Web page https://github.com/reinforceio/tensorforce*, 2017.

[32] A. ParandehGheibi, M. Medard, A. Ozdaglar, and S. Shakkottai, "Access-network association policies for media streaming in heterogeneous environments," in *Proceedings of the IEEE Conference on Decision and Control*, December 2010.

[33] R. Singh and P. Kumar, "Optimizing quality of experience of dynamic video streaming over fading wireless networks," in *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. IEEE, 2015, pp. 7195–7200.

[34] I.-H. Hou and P.-C. Hsieh, "Qoe-optimal scheduling for on-demand video streams over unreliable wireless networks," in *Proceedings of the 16th ACM In-*

ternational Symposium on Mobile Ad Hoc Networking and Computing. ACM, 2015, pp. 207–216.