QFlow: A Reinforcement Learning Approach to High QoE Video Streaming over Wireless Networks

Rajarshi Bhattacharyya*, Archana Bura*, Desik Rengarajan*, Mason Rumuly*,

Srinivas Shakkottai*, Dileep Kalathil*, Ricky K. P. Mok‡, Amogh Dhamdhere‡

*Texas A&M University, College Station ‡CAIDA, San Diego
{rajarshibh, archanabura, desik, masondataminer, sshakkot, dileep.kalathil}@tamu.edu
{cskpmok,amogh}@caida.org

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 impact on the application performance (Quality of Experience), and adaptively select a new configuration. Effectively, this feedback loop is a Markov Decision Process whose parameters are unknown. The goal of this work is to design, develop and demonstrate QFlow that instantiates this feedback loop as an application of reinforcement learning (RL). Our context is that of reconfigurable (priority) queueing, and we use the popular application of video streaming as our use case. We develop both model-free and model-based RL approaches that are tailored to the problem of determining which clients should be assigned to which queue at each decision period. Through experimental validation, we show how the RL-based control policies on QFlow are able to schedule the right clients for prioritization in a high-load scenario to outperform the status quo, as well as the best known solutions with over 25% improvement in QoE, and a perfect QoE score of 5 over 85% of the time.

CCS CONCEPTS

• **Networks** → **Programmable networks**; *Wireless access points, base stations and infrastructure*; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Reconfigurable queueing, Video Streaming, Reinforcement Learning

ACM Reference Format:
Rajarshi Bhattacharyya*, Archana Bura*, Desik Rengarajan*, Mason Rumuly*,, Srinivas Shakkottai*, Dileep Kalathil*, Ricky K. P. Mok‡, Amogh Dhamdhere‡. 2019. QFlow: A Reinforcement Learning Approach to High QoE Video Streaming over Wireless Networks. In *The Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc '19), July 2–5, 2019, Catania, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3323679.3326523

1 INTRODUCTION

From software defined networking (SDN) at the network layer on commercial routers, to different sub-layers of PHY/MAC on software defined radio (SDR) platforms, it is becoming increasingly easier to reconfigure networking equipment. Optimization over these layers will be needed in upcoming dense, small cell deployments in WiFi and 5G networks that are expected to support high and diverse loads. Among these sub-layers, a fundamental entity that impacts per-packet and per-flow performance is the behavior of queues at the router, which impacts statistical QoS performance metrics, such as throughput, RTT, jitter and loss rate that flows experience. Indeed, the fundamental nature of queueing is the reason for much effort on the design and evaluation of throughput or delay optimal scheduling mechanisms [3, 7, 22, 25].

Even when differentiated queueing mechanisms are available, exploiting them for maximizing system-wide benefit requires a feedback control loop of the kind shown in Figure 1. First, we need to *configure* the system in terms of assigning flows to queues. Second, we need to *measure* the impact of the configuration on QoE [1] and relevant application state at the end-user. Third, we need to *learn* what is the relation between realized QoE and the congfiguration used (using a combination of offline and online learning). Finally, we need to *adapt* the policy used for configuration as we learn in order to maximize performance goals. Note that such a control loop applies to problem of choosing optimal reconfigurations at all layers of network stack. However, the fundamental nature of queueing implies that first order gains might be best attained through such adaptive queue control.

Posed in this manner, the application QoE and other measurable application-specific parameters (such as buffered seconds of video)

This research was supported in part by NSF grants CNS-1149458, CNS-1513847, AST-1443891 and NSF-Intel CNS-1719384.

Mobihoc '19, July 2–5, 2019, Catania, Italy

ACM ISBN 978-1-4503-6764-6/19/07…$15.00
https://doi.org/10.1145/3323679.3326523

[1] This is a number in the interval [1, 5] that indicates end-user satisfaction, with a QoE of 5 being the best.

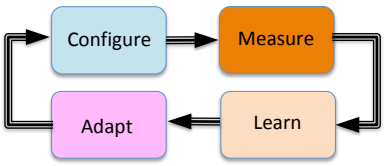

Figure 1: Feedback loop for configuration selection.

is the observable application state of the system, whose evolution is mediated through the assignment of flows to queues. The network QoS statistics of each queue are hidden variables that cause transitions to the application state, potentially in a stochastic manner. The decision of which flows to assign to what queue determines the state transitions that a particular application is exposed to, and must be done in a manner that maximizes QoE over all applications. Thus, the control loop in Figure 1 can be interpreted as a Markov Decision Process (MDP) whose transition kernel is unknown, and which could potentially be discovered using reinforcement learning.

In this work, our goal is to design, implement, and evaluate QFlow, a platform for reinforcement learning that instantiates the feedback control loop described above on a WiFi access point that faces a high demand. Performance over high capacity wired backhaul links is near-deterministic, and resources constraints apply tot he last hop wireless link. 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 [2].

2 MAIN RESULTS

Queue Configuration: We enable reliable delivery of configuration commands to hardware that can support re-configuration. We extend the OpenFlow protocol (currently limited to the network layer) in a generic manner that enables us to use it reconfigure queueing mechanisms. We select 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. Details are presented in Section 5.

Measurement of Application State and QoE: We enable continuous monitoring of client-specific application state consisting of 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. We continuously predict the QoE of the ongoing application (video streaming) flows as a function of the application state using existing maps of the relationship between video events (such as stalls) and QoE. Details are presented in Sections 4, 5.

Model-Free Reinforcement Learning: We develop a model-free reinforcement learning (RL) method that enables adaptation to the current QoE and application state over all users to maximize the discounted sum of QoEs. We design a simulator that approximates the evolution of the underlying system, and its impact on application state and QoE. We use the simulator to train a Q-Learning approach in an offline manner, with non-linear function approximation using a neural network. This so-called Deep Q Network (DQN) is able to account for state space explosion across the users and provides a Q-function approximation for all states. Details are presented in Section 6.

Model-Based Reinforcement Learning: We next develop a model-based RL approach based on the observation that the state evolution of an individual client is independent of others given the action (queue assignment). We first use measurements conducted over the system using a range of control policies to empirically determine the transition probabilities on a per-client basis, and then use the independence observation to construct the system transition kernel (this applies to the vector of all client states taken together). While doing so, we reduce the system state space by discretization and aggregation to a subset of frequently observed system states. Finally, we solve the MDP numerically to obtain the model-based policy. Details are presented in Section 7.

Experimental results: The experimental configuration consists of a single queue in the base (vanilla) case, and two reconfigurable queues in the adaptive case. We conducted experiments in both a static scenario of 6 clients, as well as a dynamic one in which anywhere between 4 and 6 clients are in the system at a given time. Apart from model-free and model-based RL, we also implemented round-robin assignment, greedy maximization of expected QoE, and greedy selection of the clients with lowest video buffers (this policy has been shown to ensure low probability of stalling [19]). Our results on adaptive flow assignment (Section 8) reveal that the vanilla approach of treating all flows identically has significantly worse average QoE than adaptive approaches.

More interestingly, both the model-based and model-free approaches manage to ensure that any given client experiences a perfect QoE of 5 over 85% of the time, whereas the best that any other policy is able to achieve is only about 60%, while vanilla manages even less at about 50%. This impressive performance improvement of about 25-30% indicates that by selecting flows in need of QoE improvement (due to high likelihood of stalls in the near future), RL-based adaptive flow assignment improves QoE for the majority of clients.

3 RELATED WORK

Our work brings together several different areas ranging from SDN, QoS, QoE and machine learning.

OpenFlow and Configuration: There has been much recent interest in extending the SDN idea to other layers. For example, CrossFlow [17, 18] uses SDN OpenFlow principles to control networks of Software Defined Radios. In ÆtherFlow [24], 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 [16] and OpenSDWN [15] 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.

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. [11] 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 [20] or general Web services [20], to identify conditions that are sufficient to meet the average QoE targets for those applications. Different from these, we desire a continuous estimate of QoE as a function of player state (such as buffered video seconds) and network state (QoS statistics).

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 [8] or the location of bottlenecks detected in the WAN [12]. 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 [13] and QFF [5] employ SDN to monitor the traffic and change the bandwidth assignment of each video flow to achieve better streaming performance. However, without an accurate map of action to QoE, the controller can only react to QoE degradation passively.

Reinforcement Learning: An RL approach is natural for the control of systems with measurable feedback under each configuration. The idea of using model-free RL in the context of video streaming rate selection was explored in [9]. The work can be seen as the complement of our own. Whereas we are interested in allocating network resources (at the wireless edge) to suit concurrent video streams, their goal is to choose the streaming rate to suit the realized network characteristics.

4 SYSTEM MODEL AND ARCHITECTURE

We consider a system 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 [2]. Our goal is to maximize the overall QoE of all the clients in this resource constrained situation.

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 controller assigns clients to each of these queues at every decision period (DP; 10 seconds in our implementation). 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 and/or control law.

4.1 Markov Decision Process

We 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*.

Environment: The environment is composed of clients and the AP. Let 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 rebuffering), and its current QoE (a number in $[1, 5]$ that represents user satisfaction, with 5 being the best). 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,

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

Agent: The controller is the agent, which takes an action $a_t \in \mathcal{A}$ (queue assignment) in every decision period in order to maximize its *expected discounted reward*. Let a_t^c denote the action taken on client c at time t,

$$a_t = [\cup_{\forall c \in C} a_t^c]$$

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}.

The goal of the agent is to maximize the overall QoE of the system. 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,

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

4.2 Measuring QoE for Video Streaming

Considerable progress has been made in identifying the relation between video events such as stalling, and subjective user perception (QoE) [4, 6, 26] via laboratory studies. However, these studies are insufficient in our context, since they do not consider the network conditions (QoS statistics) that gave rise to the video events. Nevertheless, we can leverage these studies by using them as models of human perception of objectively measurable video events. We considered three models in this context, namely Delivery Quality Score (DQS) [26], generalized DQS [4], and Time-Varying QoE (TV-QoE) [6]. 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 similar, and we choose DQS as our

candidate. Note that DQS has been validated using 183 videos and 53 human subjects [26], and we do not repeat these experiments.

The DQS model weights the impact according to duration of the impairments to better model human perception. For example, the impact on QoE of stall events during playback is greater than that of initial buffering. Similarly, the first stall event produces less dissatisfaction than repeated stalling. The state diagram of the model is shown in Figure 2. The increases and the decreases in perceived QoE are captured by 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 3, where the two stall events result in degradation of QoE. Recovery of QoE from each stall event becomes progressively harder.

4.3 QFlow System Architecture

The system architecture is illustrated in Figure 4. The three main units are, (i) an off-the-shelf WiFi access point running the OpenWRT operating system, (ii) a centralized controller hosted on a Linux workstation, and (iii) multiple wireless stations. We denote each software functionality with both a color and a circled number. These functionalities pertain to ① queueing mechanisms, ② QoS policy (configuration selection), ③ Reinforcement Learning, and ④ Policy Adaptation, which we overview below. Tying together the units are ⑤ Databases at the Controller (to log all events), and at each station (that obtains a subset of the data for local decision making). The final components are ⑥ Network Interfaces and ⑦ User Application, which are unaware of our system. We refer to the user application as a client or session, which is composed of one or more flows that are treated identically.

① **Per-Packet Queueing Mechanisms:** At the level of data packets, we utilize the MAC layer of software defined infrastructure, namely, reconfigurable queueing. Multiple Layer 2 queues can be created, and different per-packet scheduling mechanisms can be applied over them. When such mechanisms are applied to aggregates of flows, the resulting QoS statistics at the queue level can be varied, with higher priority queues getting improved performances. In turn, this results in state and QoE changes at the application.

② **QoS Policy and Statistics:** Policy decisions are used to select configurations (which clients are assigned to which queue). Decisions are made at a centralized controller that communicates using the OpenFlow protocol. We create a custom set of OpenFlow messages for QFlow. The Access Point runs QFlow, which interprets these messages and instantiates the queueing mechanisms and configurations selected by the controller. The access point periodically collects statistics related to QoS, including signal strengths, throughput, and RTT and returns those back to the controller (these statistics are used for sanity checks).

A smart middleware layer at clients is used to interface with QFlow in a manner that is transparent to the applications (such as YouTube) and the end-user. The middleware determines the foreground application, and samples the application to determine its state (stalls, and buffered seconds on YouTube). QoE is calculated using the DQS model. The client middleware contacts the Controller Database to periodically send the application state and QoE.

③ **Reinforcement Learning Agent:** Application state and configuration decisions (state-action pairs) are used to train RL agents. in the case of the Model-Free approach, a simulation environment duplicating the QFlow setup is used for offline training, and online training continues on the actual system. In the case of Model-Based RL, state-action pairs (resulting from various different policies) stored in the controller database are used for learning the model.

④ **Policy Adaptation:** Policy Adaptation has to do with implementing the policy as empirical data accumulates. An assignment algorithm (policy) matches sessions to queues every 10 seconds, and obtains a sample of client state each time it does so. This state-action pair is captured in the database, and a new action is obtained form the database (as determined by the RL agent). The assignment algorithm is geared towards discounted QoE maximization.

Interactions: The Client Middleware at each wireless client captures the state and calculates the corresponding QoE values specific to the foreground application. These realized QoE and state values from all participating clients are sent to the Controller, which performs a policy decision for flow assignment. These policy decisions are sent to the Access Point using OpenFlow Experimenter messages. QFlow interprets and implements these policy decisions. These steps are executed once every 10 seconds.

5 QFLOW IMPLEMENTATION DETAILS

Our implementation of QFlow extends the OpenFlow protocol using experimenter messages. We exploit the separation of control and data planes of OpenFlow to implement policy decisions using QFlow. Further, our choice of using experimenter messages ensures that we do not require changes at the controller. We use an off-the-shelf TP-Link WR1043ND v3 router with OpenWRT Chaos Calmer as the firmware for our implementation. We choose OpenWRT because of its support for Linux based utilities like `tc` (Traffic Control) for implementing per packet mechanisms. Since OpenWRT does not natively support SDN, we use CPqD SoftSwitch [1], an OpenFlow 1.3 compatible user-space software switch implementation.

We next extend SoftSwitch to include QFlow capabilities. Such capabilities include the ability to modify packet-handling mechanisms. Our goal is to enable configuration changes, in addition to the collection of statistics related to the implemented per packet mechanisms and the connected clients. We construct two types of QFlow commands for implementing the described capabilities, *Policy commands* and *Statistics commands*. The rationale behind this separation is to differentiate policy decisions from statistics collection. The controller uses Experimenter messages to communicate these commands to the Access Point using OpenFlow.

5.1 Policy Commands

We design Policy commands to allow us to choose between available mechanisms at different layers. Every time a Policy command is sent, it is paired with a *Solicited response* that is generated by the receiver and sent to the controller using an experimenter message. A Solicited response message thus provides us with feedback from the intended receiver. We define the format of the policy experimenter messages as shown in Figure 5 (left). The Controller packs a policy command, and sends it to the Access Point using

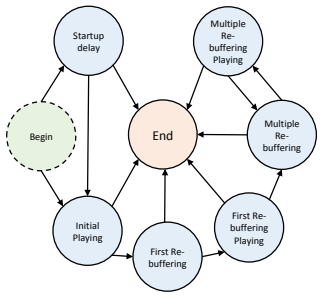

Figure 2: DQS state machine

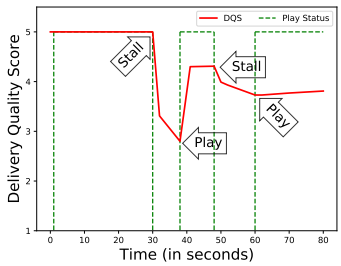

Figure 3: Sample DQS evolution.

Experimenter ID: QFlow
Type: QFlow Policy Command
Command ID
Command Length
Command

Policy Command Packet

Experimenter ID: QFlow
Type: QFlow Client Statistics
Client ID
Average RTT (in ms)
RSSI (in dBm)
Application specific info

Client State Packet

Figure 5: Packet formats in QFlow

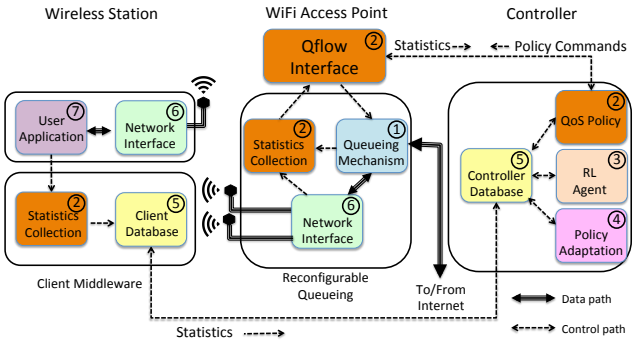

Figure 4: The system architecture of QFlow.

OpenFlow. On receiving the message, QFlow unpacks it, identifies the specific policy command using the type field, and performs the corresponding operation. Using this framework, we implemented policy commands for the MAC layer.

Data Link Layer Queue Command: At the data link layer, we need a means of providing variable queueing schemes. 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. We selected *Hierarchical Token Bucket* (`htb`) 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, we use *Filter*s to classify and enqueue packets into classes.

In our experiments, we create queues with different token rates using `htb`. Tokens may be borrowed between queues, meaning that queues will share tokens if they have no traffic. We also create a default queue that handles any background traffic. Decisions at the data link layer include assigning flows to queues, setting admission limits, changing the throughput caps queues, and enabling or disabling sharing of excess (unused) throughput between them.

5.2 Statistics Commands

Policy commands result in changes to the QoS statistics of the queues. We define Statistics commands to collect these results and send them back to the controller for analysis. Queue statistics include cumulative counts of downlink packets, bytes and dropped packets. Client-specific statistics consist of average Round Trip Times (RTT; which includes both the RTT from the base station to the client as well as the RTT from the base station to the wide-area destinations with which the client communicates), signal strength (RSSI) and Application specific statistics like buffer state, stall information and video bitrate. Since statistics are sent periodically (once every second) to the controller, we label such messages as *Unsolicited response messages*.

Similar to Policy commands, we define the structure of both Queue and Client-specific Statistics messages. After collecting the respective statistics, QFlow packs the data and sends them to the Controller using OpenFlow. On receiving these messages, the Controller unpacks them, identifies the type from the header information and then saves the extracted data to the database. The packet formats of the Client Statistics messages is shown in Figure 5 (right).

QFlow thus is capable of generating state-action, and measuring the resultant rewards in terms of QoE. The details of using the system for RL will be described in the next two sections.

The client-specific statistics, together with statistics of the queue it is placed in, constitute the Quality of Service (QoS) for a client.

6 MODEL-FREE RL

We describe a model-free RL based approach for learning a control algorithm for the system described in Section 4. More specifically, the objective is to learn a control policy for the MDP when the system model (transition probability kernel of the MDP) is unknown. Model-free RL algorithms learn the optimal control policy directly via the interactions with the system, without explicitly estimating the system model. The interaction of the RL agent with the system is modeled as a set of tuples $(s_t, a_t, R_{t+1}, s_{t+1})$ over time and the goal of the RL agent is to learn a policy π that recommends an action to take given a state, in order to maximize its long term expected cumulative reward. We will employ a specific model-free RL algorithm known as Q-learning algorithm.

6.1 Q-Learning

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

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

where $\gamma \in (0, 1)$ is the discount factor. Maximizing the cumulative reward is equivalent to finding a policy that maximizes the Q-function. The optimal Q-function, Q^* satisfies the Bellman equation

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s'}[\max_b Q^*(s', b)], \forall s, a.$$

The objective of the Q-learning algorithm is to learn this optimal Q^* from the sequence of observations $(s_t, a_t, R_{t+1}, s_{t+1})$. The optimal policy π^* can be computed from Q^* as,

$$\pi^*(s) = \arg\max_a Q^*(s, a).$$

Q-learning algorithm is implemented as follows. At each time step k, the RL agent updates the Q-function Q_k as

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

where α_k is the learning rate. If each-state action pairs is sampled infinitely often and under some suitable conditions on the step size, Q_k will converge to the optimal Q-function Q^* [21].

6.2 Deep Q-Learning

Using a standard tabular Q-learning algorithm as described above to solve our problem is infeasible due to the huge state space associated with it. Figure 6 depicts our learning problem. The individual client states are combined to form a joint 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.

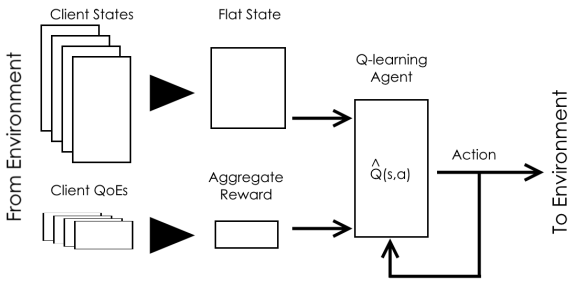

Figure 6: RL Framework

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. Even if we quantize, the number of states increases exponentially with the dimension and the number of clients. Tabular Q-learning approaches fails in such scenarios.

To overcome this issue due to the curse of dimensionality, we address this problem through the framework of deep reinforcement learning. In particular, we use the double DQN algorithm in [23] that achieved the state of the art performance in many tasks including Atari games. This approach is a clever combination of three main ideas: Q-function approximation with neural network, experience replay, and target network. We give a brief description below.

Q-function approximation with neural network: To address the problem of large and continuous state space, we approximate the Q-function using a multi-layer neural network, i.e., $Q(s, a) \approx Q_w(s, a)$ where w is the parameter of the neural network. Deep neural networks have achieved tremendous success in both supervised learning (image recognition, speech processing) and reinforcement learning (AlphaGo games) tasks. They can approximate arbitrary functions without explicitly designing the features like in classical approximation techniques. The parameter of the neural network can be updated using a (stochastic) gradient descent with step size α as

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

Experience Replay: Unlike supervised learning algorithm, the data samples $\{s_t, a_t, R_t, s_{t+1}\}$ obtained by an RL algorithm is correlated in time due to the underlying system dynamics. This often leads to a very slow convergence or non-convergence of the gradient descent algorithms like (1). The idea of experience replay is to break this temporal correlation by randomly sampling some data points from a buffer of previously observed (experienced) data points to perform the gradient step in (1) [10]. New observation are then added to the replay buffer and the process is repeated.

Target Network: In (1), the target $R_t + \gamma \max_b Q_w(s_{t+1}, b)$ depends on the neural network parameter w, unlike the targets used for supervised learning which are fixed before learning begins. This often leads to poor convergence in RL algorithms. To addresses this issue, deep RL algorithms maintain a separate neural network for the target. The target network is kept fixed for multiple steps. The update equation with target network is given below.

$$w = w + \alpha \nabla Q_w(s_t, a_t)(R_t + \gamma \max_b Q_{w^-}(s_{t+1}, b) - Q_w(s_t, a_t))$$
$$w^- = w \quad \text{after every } N \text{ steps.}$$

The combination of neural networks, experience replay and target network forms the core of the DQN algorithm [10]. However, it is known that DQN algorithm suffers from overestimation of Q values. Double DQN algorithm [23] overcomes this problem using slightly modified updated equation as

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

The target network is updated after every N steps as before.

6.3 Training the RL Algorithm

We implemented the double DQN algorithm using the TensorForce library [14]. Hyperparameters are selected via random search. The final configuration and hyperparameter of the RL algorithm is specified in Table 1.

For the faster training of our RL algorithm, we first implement a simulation environment which closely mimics the dynamics of the physical testbed. The environment simulates each video including its bitrate, buffer, length, and QoE. The bitrate and length of each video is generated according to a normal distribution; buffer is stored in terms of time, rather than bits. Each client continuously plays one video after another, stalling where its buffer runs out and building up a buffer of 10 seconds before attempting playing again.

Table 1: Selected hyperparameters for RL agent

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

Queues are serviced with a constant total bandwith, but the fairness of queue's service among flows assigned to that queue is chosen in each decision period (DP) according to a Dirichlet distribution. Each DP is of duration 10 seconds. The simulation environment uses a high-priority queue with 11 Mbps bandwidth and a low-priority queue with 4.3 Mbps. In the static network configuration, six clients are specified that draw video bit-rates from a truncated $\mathcal{N}(2.9, 10)$ distribution in Mbps, and draw video lengths from a truncated $\mathcal{N}(600, 50)$ distribution in seconds.

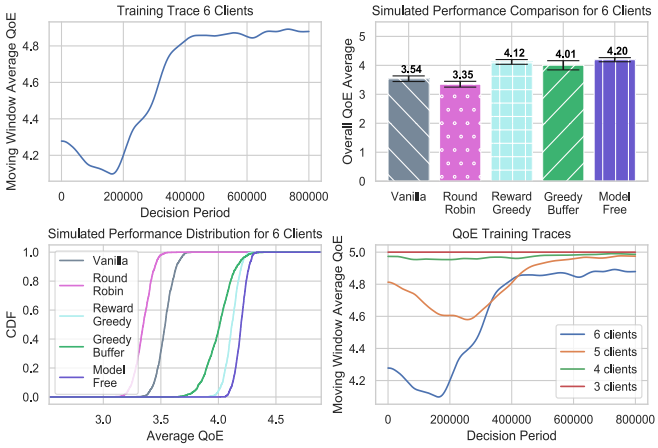

Figure 7: Training model-free RL via simulations

For hyperparameter search, the system was simulated for 200 DP per episode for 1000 episodes. 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. The evolution of value during the training process is shown in Figure 7 top-left. As is seen, the trained controller achieves a high QoE of near 5.

Next, we compare the performance of different policies in the simulation environment. Figure 7 top-right shows the average QoE attained by different policies, which suggests that perhaps the model-free approach, while best, may not give substantial performance improvements. The QoE CDFs in Figure 7 bottom-left, however, indicate that model-free RL achieves a higher QoE for a larger fraction of clients, suggesting that it might be more robust to resource constraints. Indeed, we will see in experiments in Section 8 that it attains quite substantial gains over the other approaches in practice under a bandwidth constrained environment.

6.4 Dynamic Number of Clients

In the above description, we assumed that the number of clients in the system is static. The timescale at which the number of clients change is very large (several tens of minutes; this models human mobility) when compared to the decision period (10 seconds). Including a dynamic number of clients into training would require augmenting the state space with the number of connected clients, and a Markov model of transitions in this value. Since this increases the state space and training duration, we instead obtain the optimal static policy for the system with 4 to 6 clients using the model-free approach. Figure 7 bottom-right shows the evolution of value over the training process over the different cases. We can then choose the the right policy based on number of clients in the system. Interestingly, there appears to be enough structure in our problem that a policy developed for a larger number of clients can simply be used for a smaller number (setting non-existent clients to have large QoE and buffer values), since the relative priorities of clients is all that matters. This simplifies training considerably.

7 MODEL-BASED RL

In this section, we discuss the scenario in which the dynamics of the system (transition kernel) are first determined, i.e., given the current state s_t of the system and the action taken a_t, we find the transition probabilities to the next states s_{t+1}. Given the transition kernel of the system $\mathcal{P}$, we can use policy or value iteration to solve for the optimal policy π^*. The model-based approach is particularly interesting because of its special structure, since the state transitions of a client given its current state and action are independent of the states and actions of other clients in the system. In other words,

$$\mathbb{P}(s_{t+1}|a_t, s_t) = \prod_{\forall c \in C} \mathbb{P}(s^c_{t+1}|a^c_t, s^c_t)$$

It must also be noted that the state transitions of all clients in the system given their current states and actions are identical. Thus, we can determine the transition kernel of the system using the transition kernel of each individual client.

7.1 Static Model

In what follows, we determine the transition kernel of the system with a fixed number of clients, and obtain the optimal policy.

Experimental Traces. We generate state (s^c_t), action (a^c_t) and next state (s^c_{t+1}) tuples for all clients by running the system under Round Robin, Greedy Buffer, Random, Model Free and Vanilla policy for a duration of 10 hours.

Discretizing the state space. The state of each individual client s^c_t and hence the state of the system s_t have elements that are (non-negative) real numbers. In order to calculate the transition kernel of the client in atractable manner, we discretize the state space of the client according to table 2. Since the state of a client is 3 dimensional (Buffer, Stall, QoE) we encode it to obtain a label for each client state as follows, Let *NSB* and *NQB* denote the number

Table 2: Client State Space Discretization

Parameter	Range	Bins
Buffer	[0,20]	21
Stalls	[0,5]	5
QoE	[1,5]	9

of stall and QoE bins respectively,

$$s_t^c = \text{Buffer} \times \text{NSB} \times \text{NQB} \ + \text{QoE} \times \text{NSB} \ + \text{Stall}$$

The discretized and encoded state space of a client $\mathcal{S}_c$ has a cardinality of 945.

Determining the transition kernel of a client. We determine the transition kernel of a single client by fitting an empirical distribution over the state, action, and next state tuples collected from experimental traces, i.e., we empirical determine,

$$\mathbb{P}(s_{t+1}^c | a_t^c, s_t^c) \quad \forall s_{t+1}^c, s_t^c \in \mathcal{S}_c \quad \forall a_t^c \in \mathcal{A}_c$$

from experimental traces. $\mathcal{A}_c$ is the set of all actions for a client c.

Identifying Frequent States of the system. The state of the system (s_t) is the union of states of all clients (s_t^c) in the system. If there are N clients in the system, the state of the system is a N dimensional vector, where each dimension corresponds to the state of a client. Let $\mathcal{S}$ denote the discretized state space of the system. The cardinality of $\mathcal{S}$ is of the order of 945^N. Solving an MDP with 945^N states is intractable. Hence, based on experimental traces, we identify the most frequent states $\mathcal{S}_p$ of the system, and approximate all other states to these popular states using the L^2 norm, i.e., given a state in $\mathcal{S}$, we approximate it by a state in $\mathcal{S}_p$ with the least Euclidean distance.

Calculating the transition kernel of the system. The state space of our system has now reduced from $\mathcal{S}$ to $\mathcal{S}_p$. To obtain the transition kernel of this system, we empirically sample one hundred state transitions for each state in $\mathcal{S}_p$ under each action in $\mathcal{A}$ using the transition kernel of individual clients. If the generated state transitions are outside $\mathcal{S}_p$, we approximate it with the state in $\mathcal{S}_p$ which is closest in Euclidean distance. Thus, we obtain state, action, next state tuples for the system with state space $\mathcal{S}_p$. We fit an empirical distribution over these tuples to obtain the transition kernel of the system. Hence, we empirically determine

$$\mathbb{P}(s_{t+1} | a_t, s_t) \quad \forall s_{t+1}, s_t \in \mathcal{S}_p \quad \forall a_t \in \mathcal{A}.$$

Obtaining the optimal policy We obtain the optimal policy π^* by running value iteration over the transition kernel generated for $\mathcal{S}_p$. It must be noted that the reward obtained by taking action a_t in state s_t is the average QoE of state s_{t+1} which is a part of s_{t+1} and hence need not be calculated explicitly.

7.2 Dynamic Number of Clients

In the previous subsection, we assumed that the number of clients in the system are static. To deal with a dynamic number of clients, we follow an approach similar to the one described in section 6. We obtain the optimal policy for the system with 4-6 clients using the static model approach described in the previous subsection. In the same manner as the model-free case, we may also use the policy for 6 clients for a smaller number of clients by just comparing their relative priorities.

8 EVALUATION

An off-the-shelf WiFi router installed with QFlow is used as the Access Point and three Intel NUCs are used to instantiate up to 6 clients (YouTube sessions) for our experiments. 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. The three NUCs are equipped with 5th generation i7 processors with 8 GB of memory, each capable of running multiple traffic intensive sessions simultaneously. Relevant session information such as ports used by an application, play/load progress, bitrate and stall information for YouTube sessions is collected every second and written to the database.

We setup a scenario with two downlink queues, one with a higher bandwidth allocation (resulting in better QoS) than the other using token bucket queueing. A default queue is used for any background traffic. Two clients may be allowed into the high priority queue. For the no differentiation case, we just set up a single queue with the same total throughput limit as that of the two queues in the previous scenarios. Our control problem is to determine which sessions to assign to which queues.

8.1 Policies

In addition to described Model-based and Model-free policies, we consider four additional policies for choosing these assignments.

Vanilla: This is the base case with a single queue that is allocated the full bandwidth, and with no differentiation between clients.

Round Robin: As the name suggests, we assign clients to the high priority queue in turn. Although it is computationally inexpensive, work-conserving and prevents starvation, it might lead to the wrong clients (those who have no hope of significantly increasing their QoE) being considered for the high-quality service instead of clients who might benefit much more from the service.

Reward Greedy: This policy computes the expected one-step reward on a per-client basis, and assigns clients so as to maximize the sum of rewards. We can think of this as a myopic version of model-based RL. This might starve sessions that were unlucky and stalled at some point, since QoE growth rates reduce after stalls.

Greedy Buffer: The smooth playout of a video depends on the size of the playback buffer. When the buffer is empty, the client experiences a stall and the perceived QoE drops. This approach promotes the clients with the lowest buffered video to the high priority queue to prevent this from occurring. This policy might promote the wrong agents who have low buffers because they are at the end of their videos, or those that have stalled multiple times and can never recover high QoE.

8.2 Static Network Configuration

In our static configuration, each NUC hosts two YouTube sessions to simulate a total of 6 clients. The QoE performance comparison of the different policies is shown in Figures 8, 9 and 10. We first compare the average QoE of the various policies in the first figure. It is clear that the Model-based and the Model-free policies outperform the other policies. This gap in performance becomes even more evident when we compare the CDFs of the individual and the average QoE of the different policies in Figures 9 and 10. For example, we can observe from Figure 9 that the Model-based and the Model-free

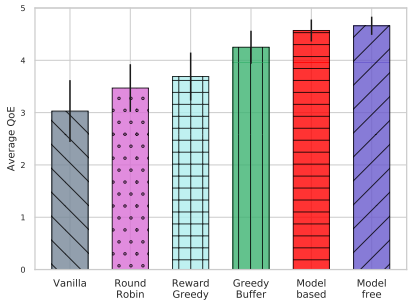

Figure 8: Comparison of average QoE

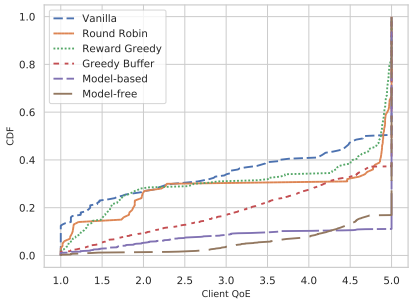

Figure 9: Comparison of client QoE CDF

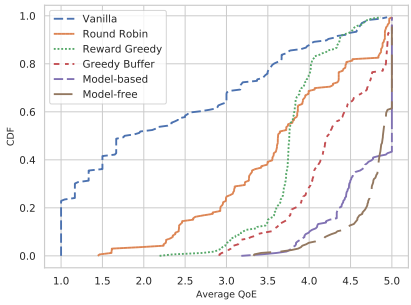

Figure 10: Comparison of average QoE CDF

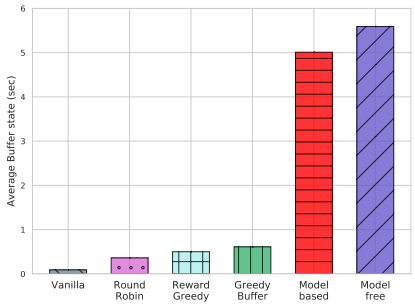

Figure 11: Comparison of average Buffer

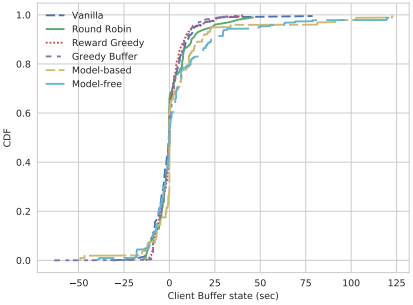

Figure 12: Comparison of client Buffer CDF

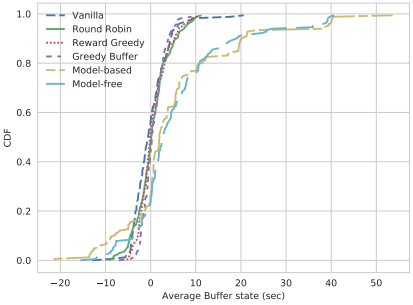

Figure 13: Comparison of average Buffer CDF

policies are able to provide a QoE of 5 for almost 90 and 85% of the time for all clients, whereas it is only about 65% of the time for the next best policy. Similarly, it can be deduced from Figure 10 that the Model-based and the Model-free policies are able to achieve an average QoE of 5 for all participating clients in the system about 55 and 40% respectively. The value of this metric for the next best policy is about 5%.

The QoE experienced by a client is affected by the buffer state of the client and the stalls experienced during video playback. Hence, we study the buffer state and the stall durations experienced by the clients under the different policies. Similar to the QoE plots, we compared the averages, the CDFs of the individual and the average values for both these features in Figures 11 to 16. Again, it is evident from the figures that the Model-based and the Model-free policies ensure better buffer state and lower stall durations (both individual and average) than the other policies under consideration.

8.3 Dynamic Number of Clients

We next study the performance of the policies in a scenario with a varying number of clients. We choose the number of active clients in the system to vary between 4 and 6, while keeping the bandwidth allocation same as that of the static configuration.

We consider a larger timescale of 30 minutes for changing the number of clients participating in the system. We start with 6 clients in the system and then remove 1 client each for the next two time periods. At the end of the third period, we introduce two more clients in the system. The evolution of the average QoE for each of the policies for the above scenario is shown in Figure 17. It is seen that Model-based and Model-free policies perform well irrespective of the number of users in the system, whereas other policies only do well when there are relatively fewer clients in the system.

Since the bandwidth allocation is the same, reducing the number of clients implies relaxation of the resource constraints and hence, other policies see an improvement in performance. This can be seen in Figures 18 and 19, where the CDF curves of the other policies are closer to those of the Model-based and the Model-free polciies. Even so, Model-based and Model-free policies exhibit the best performance, which reinforces their superiority over other policies in both static and dynamic client scenarios.

9 CONCLUSION

In this paper, we considered the design, development and evaluation of QFlow, a platform for reinforcement learning based edge network configuration. Working with off-the-shelf hardware and open source operating systems and protocols, we showed how to couple queueing, learning and scheduling to develop a system that is able to reconfigure itself 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.

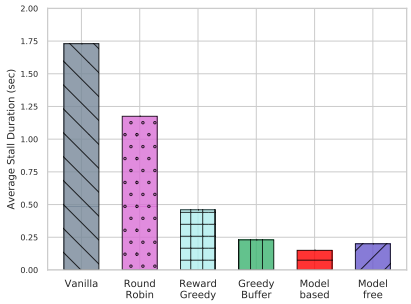

Figure 14: Comparison of average stall duration

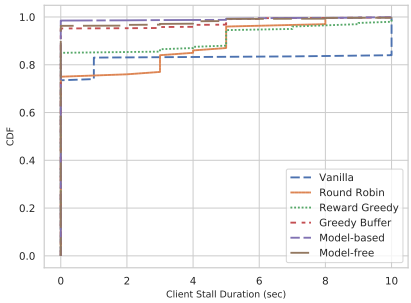

Figure 15: Comparison of stall duration CDF

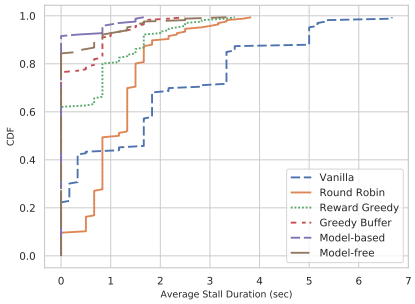

Figure 16: Comparison of average stall duration CDF

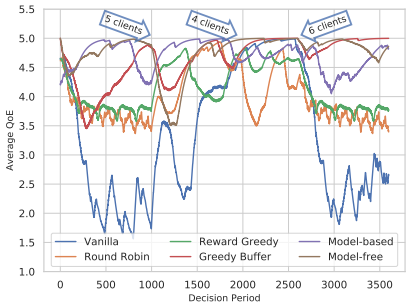

Figure 17: Evolution of QoE for dynamic clients

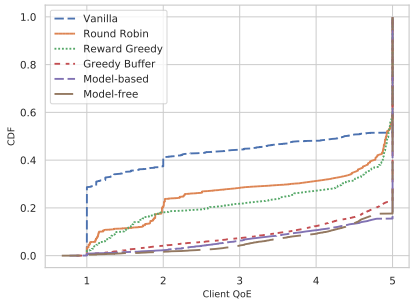

Figure 18: Comparison of client QoE CDF for dynamic clients

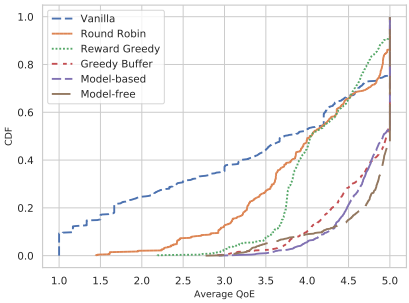

Figure 19: Comparison of average QoE CDF for dynamic clients

REFERENCES

[1] CPqD. 2015. OpenFlow Software Switch. http://cpqd.github.io/ofsoftswitch13/.

[2] Ericsson. 2015. 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.

[3] A. Eryilmaz, R. Srikant, and J. Perkins. 2005. Stable Scheduling Policies for fading wireless channels. *IEEE/ACM Trans. Network.* 13 (April 2005), 411–424.

[4] N. Eswara, K. Manasa, A. Kommineni, S. Chakraborty, H. P. Sethuram, K. Kuchi, A. Kumar, and S. S. Channappayya. 2017. A Continuous QoE Evaluation Framework for Video Streaming over HTTP. *IEEE Transactions on Circuits and Systems for Video Technology* In press (2017). https://doi.org/10.1109/TCSVT.2017.2742601

[5] Panagiotis Georgopoulos, Yehia Elkhatib, Matthew Broadbent, Mu Mu, and Nicholas Race. 2013. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *Proceedings of ACM FhMN.*

[6] D. Ghadiyaram, J. Pan, and A. C. Bovik. 2018. Learning a Continuous-Time Streaming Video QoE Model. *IEEE Transactions on Image Processing* 27, 5 (May 2018), 2257–2271. https://doi.org/10.1109/TIP.2018.2790347

[7] I.H. Hou, V. Borkar, and P.R Kumar. 2009. A theory of QoS for wireless. In *Proceedings of IEEE INFOCOM.*

[8] Michael Jarschel, Florian Wamser, Thomas Hohn, Thomas Zinner, and Phuoc Tran-Gia. 2013. SDN-based Application-Aware Networking on the Example of YouTube Video Streaming. In *Proceedings of EWSDN.*

[9] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication.* ACM, 197–210.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[11] Ricky Mok, Weichi Li, and Rocky Chang. 2016. IRate: Initial Video Bitrate Selection System for HTTP Streaming. *IEEE Journal on Selected Areas in Communications* 34, 6 (June 2016), 1914–1928. https://doi.org/10.1109/JSAC.2016.2559078

[12] Hyunwoo Nam, Kyung-Hwa Kim, Jong Yul Kim, and Henning Schulzrinne. 2014. Towards QoE-aware Video Streaming using SDN. In *Proceedings of IEEE GLOBECOM.*

[13] Sangeeta Ramakrishnan, Xiaoqing Zhu, Frank Chan, and Kashyap Kambhatla. 2015. SDN Based QoE Optimization for HTTP-Based Adaptive Video Streaming. In *Proceedings of IEEE ISM.*

[14] Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke. 2017. TensorForce: A TensorFlow library for applied reinforcement learning. https://github.com/reinforceio/tensorforce.

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

[16] Julius Schulz-Zander, Nadi Sarrar, and Stefan Schmid. 2014. AeroFlux: A Near-Sighted Controller Architecture for Software-Defined Wireless Networks. In *Proceedings of USENIX ONS.*

[17] P. Shome, J. Modares, N. Mastronarde, and A. Sprintson. 2017. Enabling Dynamic Reconfigurability of SDRs Using SDN Principles. In *Proceedings of Ad Hoc Networks.*

[18] P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde, and A. Sprintson. 2015. CrossFlow: A cross-layer architecture for SDR using SDN principles. In *Proceedings of IEEE NFV-SDN.* https://doi.org/10.1109/NFV-SDN.2015.7387403

[19] Rahul Singh and PR Kumar. 2015. Optimizing quality of experience of dynamic video streaming over fading wireless networks. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on.* IEEE, 7195–7200.

[20] T. Spetebroot, S. Afra, N. Aguilera, D. Saucez, and C. Barakat. 2015. From network-level measurements to expected Quality of Experience: The Skype use case. In *Proceedings of IEEE M&N.*

[21] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[22] L. Tassiulas and A.Ephermides. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Automat. Contr.* 37, 12 (1992), 1936–1948.

[23] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI,* Vol. 2. Phoenix, AZ, 5.

[24] M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton. 2015. ÆtherFlow: Principled Wireless Support in SDN. In *Proceedings of IEEE ICNP.* https://doi.org/10.1109/ICNP.2015.9

[25] Simon Yau, Ping-Chun Hsieh, Rajarshi Bhattacharyya, KR Bhargav, Srinivas Shakkottai, I Hou, PR Kumar, et al. 2018. PULS: Processor-Supported Ultra-Low Latency Scheduling. In *Proceedings of ACM MobiHoc.*

[26] H. Yeganeh, R. Kordasiewicz, M. Gallant, D. Ghadiyaram, and A. C. Bovik. 2014. Delivery quality score model for Internet video. In *Proceedings of IEEE ICIP.* https://doi.org/10.1109/ICIP.2014.7025402