QFlow: A Learning Approach to High QoE Video Streaming at the Wireless Edge

Rajarshi Bhattacharyya, Archana Bura, Desik Rengarajan, Mason Rumuly, Bainan Xia, Srinivas Shakkottai, *Senior Member, IEEE*, Dileep Kalathil, *Senior Member, IEEE*, Ricky K. P. Mok, *Member, IEEE*, and Amogh Dhamdhere

***Abstract*— The predominant use of wireless access networks is for media streaming applications. However, current access networks treat all packets identically, and lack the agility to determine which clients are most in need of service at a given time. Software reconfigurability of networking devices has seen wide adoption, and this in turn implies that agile control policies can be now instantiated on access networks. 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 develop QFlow, a platform that instantiates this feedback loop, and instantiate a variety of control policies over it. We use the popular application of video streaming over YouTube as our use case. Our context is priority queueing, with the action space being that of determining which clients should be assigned to each queue at each decision period. We first develop policies based on model-based and model-free reinforcement learning. We then design an auction-based system under which clients place bids for priority service, as well as a more structured index-based policy. Through experiments, we show how these learning-based policies on QFlow are able to select the right clients for prioritization in a high-load scenario to outperform the best known solutions with over 25% improvement in QoE, and a perfect QoE score of 5 over 85% of the time.**

***Index Terms*— Reinforcement learning, wireless edge networks, video streaming, auction mechanisms, OpenFlow.**

Manuscript received May 4, 2020; revised November 27, 2020 and May 19, 2021; accepted July 19, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Rao. This work was supported in part by the National Science Foundation under Grant CNS-1955696, Grant CRII-CPS-1850206, and Grant NSF-Intel CNS 1719384; in part by the Army Research Office (ARO) under Grant W911NF-19-1-0367 and Grant W911NF-19-2-0243; and in part by the Defense Advanced Projects Research Agency (DARPA) under Grant CA HR00112020014. *(Corresponding author: Srinivas Shakkottai.)*

Rajarshi Bhattacharyya was with Texas A&M University, College Station, TX 77843 USA. He is now with Aruba (a Hewlett Packard Enterprise Company), San Jose, CA 95002 USA (e-mail: rajrc11@gmail.com).

Archana Bura, Desik Rengarajan, Srinivas Shakkottai, and Dileep Kalathil are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: archanabura@tamu.edu; desik@tamu.edu; sshakkot@tamu.edu; dileep.kalathil@tamu.edu).

Mason Rumuly was with Texas A&M University, College Station, TX 77843 USA. He is now with Arista Networks, Austin, TX 78746 USA (e-mail: masonminer1552@gmail.com).

Bainan Xia was with Texas A&M University, College Station, TX 77843 USA. He is now with Breakthrough Energy LLC, Kirkland, WA 98033 USA (e-mail: ericxbn@gmail.com).

Ricky K. P. Mok and Amogh Dhamdhere are with the Center for Applied Internet Data Analysis (CAIDA), University of California at San Diego, San Diego, CA 92093 USA (e-mail: cskpmok@caida.org; amogh@caida.org).

Digital Object Identifier 10.1109/TNET.2021.3106675

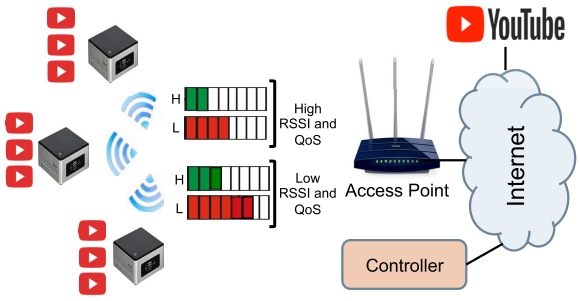

Fig. 1. Ensuring high QoE video streaming via adaptive prioritization.

I. INTRODUCTION

A MAJOR fraction of the traffic carried by wireless access (edge) networks today is related to media streaming, and has relatively stringent constraints on the required quality of service (QoS) provided by the network. These QoS metrics typically are measured as link statistics such as $[Throughput, RTT, Jitter, LossRate]$. The impact of such QoS on user satisfaction is identified in terms of Quality of Experience (QoE). QoE indicates user-satisfaction and is quantified as a number in the interval $[1, 5]$, which can depend on the application and its evolving state. For example, the application can be video streaming over the Web, with the state being the number and duration of stalls (re-buffering events) that have been experienced thus far. Supporting a large number of concurrent streams of this kind, while ensuring high QoE for all clients is a major challenge.

As a concrete example, consider Figure 1 that shows 9 simultaneous YouTube clients that are supported over a wireless access network. This setup with simultaneous YouTube sessions is used for our laboratory experiments, and can emulate a range of load and channel conditions at the access point. The traditional (vanilla) approach is to maintain a single queue, and to treat all packets identically regardless of the importance of the packets to the QoE of the clients. So a session that has already buffered up many seconds of video might get equal service as one that is near stalling. While this approach might be acceptable when the number of streams is limited, the need to support multiple high quality streams motivates the desire to do better.

Given that queuing behavior is fundamental to all elements of the QoS statistics mentioned above, differentiated queuing at the access point immediately suggests itself. Token-bucket-based shaping can be used to create high-priority and low-priority queues, with the QoS statistics of the former being superior to that of the latter. We can also create multiple "bins" of queues as shown in Figure 1, with each bin corresponding

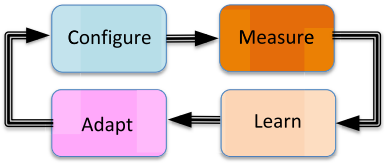

Fig. 2. Feedback loop for configuration selection.

to similar client channel conditions, and allocate them similar time-spectrum resources. At the client end, middleware can be used to gather and share the application QoE and state. Then a basic question is that of periodically deciding client schedules: *Given the current QoE and video state at each client, how should the controller assign clients to queues for the next decision period in order to attain system-wide benefits?*

A policy that can attain system-wide benefits requires a feedback control loop of the kind shown in Figure 2. 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 and application state at the end-user. Third, we need to *learn* what is the relation between realized QoE and the configuration used (using offline and online learning). Finally, we need to *adapt* the policy used for configuration in order to maximize performance goals.

Posed in this manner, the application QoE and other measurable application-specific parameters (such as buffered seconds of video) are the observable state of the system, whose evolution is mediated through the assignment of flows to queues. The underlying network QoS statistics cause stochastic transitions to the application state. The decision of which flows to assign to what queue determines the state transitions that a particular application incurs, and must be done in a manner that maximizes QoE. Thus, the control loop in Figure 2 can be interpreted as a Markov Decision Process (MDP) whose transition kernel is unknown, and which could potentially be discovered using reinforcement learning. We note that existing approaches such as max-weight scheduling [1], [2] or deficit-weight based scheduling [3] are Markov algorithms that use state information such as queue length, deficit in service and channel state. Other notions of fair scheduling may compare average channel statistics and the current realization, in which case the running average is used as a state variable. Thus, although these schedulers are not directly generated as the optimal solution produced by solving an MDP, they are Markov algorithms that fit into our general view.

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 to the 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 [4].

A. Main Results

1) Measurement of Application State and QoE: We implement simple middleware for monitoring of client-specific application state consisting of buffered seconds of video and stall duration (when the video re-buffers). The middleware periodically sends statistics to the OpenFlow controller for processing. Here, 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 III, VIII.

2) 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 develop a simulator that approximates the evolution of the underlying system, and train a Q-Learning algorithm 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. As with many model-free approaches, training takes many samples and we use 200,000 in our case. Details are presented in Section IV.

3) 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 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. Exploiting this structure reduces the training samples needed significantly to 3600. Finally, we solve the MDP numerically to obtain the model-based policy.

4) Auction: The above approaches require that the state of each client be supplied by the clients themselves, which implies that strategic clients could obtain more than their fair share of resources through appropriate declarations. We hence develop an incentive compatible (truth-telling) auction approach. Here, bids for the auction are placed via a middleware algorithm *(the human end-user need not engage with the system)*, and can be interpreted as the number of cents that the bidding algorithm is willing to pay for high priority service for the next 10 seconds.[1] The agents are provided the model (transition kernel) as in model-based RL, as well as the empirical bid-distribution of all the agents, and use these to obtain the best response bid, consistent with our earlier work on a mean field game approach to scheduling [5], [6]. The per-agent bid computation under this regime is straightforward, and details are presented in section VI.

5) Index Policy: The results from the auction approach suggest that a indexing of state in the manner of the Whittle index [7] is possible, under which each client state is associated with a real-number index. The optimal policy simply picks the clients with largest indices to promote to the high priority queues. We empirically validate this hypothesis by using the value function of a given state derived from the auction as its index, and find that such an index policy performs as well or better than all others, lending credence to the indexing claim. Details are presented in Section VII.

6) Platform Implementation: 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

[1]We calculate that the eventual dollar price will be consistent with current cellular data prices of about $5-10 per GB.

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 VIII.

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 with a fixed number of clients, as well as a dynamic one in which the number varies with time. Apart from auction-based, model-free and model-based RL, we also implemented channel-binned round-robin assignment (approximating proportional fairness), 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 [8]). Our results on adaptive flow assignment (Section IX) reveal that the vanilla approach of treating all flows identically has significantly worse average QoE than adaptive approaches.

Interestingly, the model-based, model-free and auction-based approaches 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.

8) Limitations: We focus on network-level adaptations, and so fix the video bitrate at 1080P in our experiments. However, such birate adaptation is compatible with our RL-based approach in two ways, namely, (i) treating it as part of the transition kernel and accounting for it implicitly during learning, or (ii) jointly choosing video bitrate and network priority, which, however, would require coordination across the content distributor and network manager. Another consideration is of Sim2Real mismatch when we transfer a simulation-trained controller in the model-free approach to the real system. However, the online training that occurs naturally during experiments appears to be sufficient to obtain the same performance of the model-based approach that is trained on offline empirically collected data.

An earlier version of this work appeared in [9], in which we presented basic RL approaches over the QFlow platform. The differences between our earlier work and this paper are: (i) we develop an auction platform for clients to compute and bid their perceived valuations, and show empirically that it attains similar (slightly better) performance than the basic RL approaches, (ii) we explore the idea that policies can have structure, and show empirically that a simple index-type of policy might be optimal, and (iii) we show empirically that the indices (state ordering by value) developed for a larger number of clients follow a similar order to those for a smaller number of clients, meaning that a single set of indices work well without retraining, even under a dynamically changing number of clients with time varying channel conditions.

II. Related Work

A. Optimal Queueing

There has been significant work on QoS as a function of the scheduling policy, e.g., a sequence of work starting with [1], and follow on work in the wireless context that resulted in algorithms such as backpressure-based scheduling and routing in wireless networks [2] and more recently [3] 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. Also, they do not map received QoS to application QoE.

B. 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 [10] or the location of bottlenecks detected in the WAN [11]. 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 [12] and QFF [13] 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.

C. Reinforcement Learning

An RL approach is natural for the control of systems with measurable feedback under each configuration. The idea of using RL in the context of video streaming rate selection has been explored in [14]–[17]. Different control theoretic methods, such as model predictive control [18] and PID control [19] have also been used for adaptive video streaming. This body of 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. Finally, deep reinforcement learning algorithms have been used to solve a number of problems in communication network applications, although not in our problem space; see [20] for a survey.

D. Auctions and Scheduling

There has been much work on using price or auction-based resource allocation in the wireless context. On the analytical side, [21] considered the problem of auction-based wireless resource allocation. It was shown that with finite number of users, a Nash Equilibrium exists and the solution is Pareto optimal. In [5], [6], an auction framework is presented in which queues (representing apps on mobile devices) repeatedly bid for service in a second-price auction. They show that under a large system scaling (called the mean field game regime), the result of the auction would ensure fair service for all. Our design of auction-based scheduling algorithms are motivated by these ideas. In the context of experiments, a recent trial of a price-based system is described in [22]. Here, day-ahead prices are announced in advance to users, who can choose to use their cellular data connection based the current price. Thus, the decision makers are the human end-users that essentially have an on/off control. Furthermore, the prices are not dynamic and have to be determined offline based on historical usage.

E. OpenFlow Extensions

There has been much work on OpenFlow extensions for cross-layer wireless configuration. In this context, CrossFlow [23], [24] uses the SDN framework for configuring software defined radios. Similarly ÆtherFlow [25], extends OpenFlow for enabling remote configuration of WiFi access points. Finally, recent systems such as AeroFlux [26] and OpenSDWN

[27] enable packet prioritization for flows that are identified by packet inspection as belonging to high priority applications, such as video streaming. However, these are all offline static policies in that they do not relate the prioritization policy with the state of the application.

III. 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 [4]. 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. Here, we mean that 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.

A. Markov Decision Process

We consider a discrete time system where time is indexed by $t \in \{0, 1, \ldots\}$. At each DP ($t = 0, 1, 2 \ldots$) 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*.

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

2) 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 (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,

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

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

3) Scheduler Action: The scheduler is the agent that takes queue assignment actions in every decision period in order to maximize its *expected discounted reward*. Let $a_t^c \in \{0, 1\}$ denote the action taken on client c at time t, where 1 and 0 indicate assignment to the high and low priority queue, respectively. Let the set of overall actions be denoted $\mathcal{A}$. Each such overall action is of form $a_t = [a_t^1, a_t^2, \cdots a_t^{|\mathcal{C}|}]$. The scheduler may assign only N clients to the high priority queue, i.e., $\sum_{c \in \mathcal{C}} a_c^t = N$.

4) Reward: The per-client reward $R(s_t^c, a_t^c)$ resulting from taking action a_t at state s_t is the QoE of client c in state s_t^c. The overall reward $R(s_t, a_t)$ is the average QoE of all clients in state s_{t+1},

$$R(s_t, a_t) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} R(s_t^c, a_t^c)$$

5) Transition Kernel: Let $P(s_{t+1}|s_t, a_t)$ denote the system transition kernel.

6) Policy: 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,

$$\pi^* = \text{argmax}_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_t \sim \pi(\cdot|s_t)\right]. \quad (1)$$

The infinite horizon discounted reward models the idea that a video might conclude or the user might terminate it with some probability at any time. Empirically, the discount factor is chosen such that the mean lifetime matches the average video length in the library of videos that we use for testing.

B. Auction

We consider a market wherein clients bid for high priority service periodically. In each discrete time instant, a fixed number of clients N are assigned to the high priority queue. Clients participate in an $(N+1)^{th}$ auction to compete for admission to the high priority queue. The N winners who obtain high priority service will pay a price that is equal to the $(N+1)^{th}$ highest bid, and the rest of the clients will be assigned to the low priority queue. We model the system in a Mean Field approach as described below,

1) Bid: The bid submitted by the client in each auction is denoted by $b \in \mathcal{B}$, where $\mathcal{B}$ is a set containing discrete bid values. The bids can be seen as the price each client c is willing to pay to obtain high priority service. Note that the human end user plays no role in selecting these bids.

2) Bid Distribution: The clients must place their bid based on the beliefs of their competitors. We denote the assumed bid distribution in the market as ρ.

3) Auction Outcome: The probability that a client c wins at the auction depends on the event that its bid is greater than the $N + 1^{th}$ bid. Under the mean field model, the client assumes that competing bids are all drawn in an IID manner from ρ. We denote the probability of winning under this assumption when a client places a bid b by $p_{win}(b)$, where we have dropped the dependence on ρ for ease of notation. Thus, $p_{win}(b)$, is the probability that b lies within the top N values of $|\mathcal{C}| - 1$ independent draws from ρ. Accuracy of the mean field approximation in the regime where there is a large pool of clients from which a small subset is drawn at each auction is available in [5], [6]. The model might apply to the situation at a coffee shop or other public access point, where the devices in the whole town are possible clients, but only a few of them are in the coffee shop using the access point at a given time.

4) Payment: The (random) amount paid after each auction is denoted by pay. The payment distribution in our system upon winning is the distribution of the $(N+1)^{th}$ highest bid.

5) Scheduler Action: As in the previous case, the scheduler decides on which clients to assign to each queue. However, here the actions are taken based on the outcome of the auction. The actions $a_t^c = 1$ and $a_t^c = 0$ correspond, respectively, to winning and losing at the auction by the client c.

6) Client Reward: As before, reward $R(s_t^c, a_t^c)$ resulting from action a_t^c at state s_t^c is the QoE of client c in state s_t^c. Note, however, that each agent is only concerned with its own reward, unlike the average case discussed earlier.

7) Client Transition Kernel: Let $P(s_{t+1}^c|s_t^c, a_t^c)$ denote the client transition kernel. Thus the probability of transitioning to state s_{t+1}^c is jointly defined by the probability of winning the auction when bidding b, $p_{win}(b)$ and $P(s_{t+1}^c|s_t^c, a_t^c)$.

8) Policy: The agent (client) must place a bid at each time, accounting for its progression of state. Following the same methodology as [5], [6], we formulate the optimal policy (bid decision) problem of the corresponding MDP:

$$\begin{aligned} b^*(s_t^c) = \text{argmax}_{b\in\mathcal{B}}\Big\{ & p_{win}(b)\Big[R(s_{t+1}^c, a_t^c = 1) - pay \\ & + \sum_{s_{t+1}^c} P(s_{t+1}^c|s_t^c, a_t^c = 1)\gamma v(s_{t+1}^c)\Big] \\ & + (1 - p_{win}(b))\Big[R(s_{t+1}^c, a_t^c = 0) \\ & + \sum_{s_{t+1}^c} P(s_{t+1}^c|s_t^c, a_t^c = 0)\gamma v(s_{t+1}^c)\Big]\Big\}, \end{aligned} \tag{2}$$

where $v(.)$ is the optimal client value function.

C. 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) [28]–[30] 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) [28], generalized DQS [29], and Time-Varying QoE (TV-QoE) [30]. The three models are based on the features of stall events and video bitrate adaptation, if any. Since our focus is on network adaptation, with the goal of supporting high resolution video, we fix the resolution to prevent video bitrate adaptation. All three models are similar in this case, and we choose DQS as our candidate. We note that DQS has earlier been validated using 183 videos and 53 human subjects [28].

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 QoE model is shown in Figure 3, and the model can account for rate adaptations by augmenting the state with video bitrate. 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 4, where the two stall events result in degradation of QoE. Recovery of QoE from each stall event becomes progressively harder.

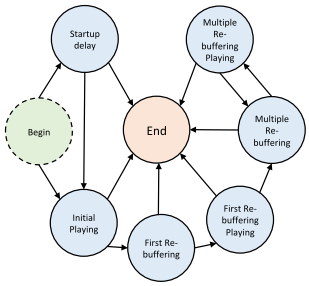

Fig. 3. DQS state machine.

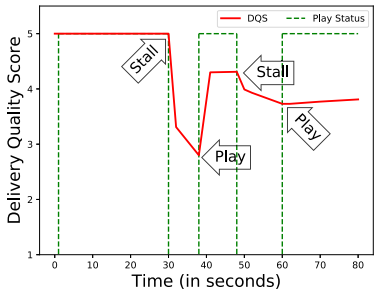

Fig. 4. Sample DQS evolution.

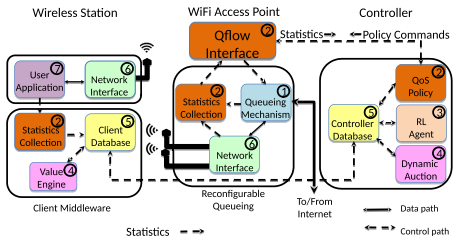

Fig. 5. The system architecture of QFlow.

D. QFlow System Architecture

The system architecture is illustrated in Figure 5. 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 End User Value and Auction, 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 *End User Value and Auction:* Clients are offered feasible QoS vectors under an market framework. The decision on which N flows to admit to a high-priority queue is taken via an $N+1^{th}$ price auction using a local currency (a token allowance), which is conducted every 10 seconds. The resultant policy decisions in turn lead to a realization of the offered QoS. End-users setup priorities for different applications (at the timescale of weeks or months), and the Controller Database provides statistics of current market conditions (bid distribution), using which a Value Engine at the client middleware determines what the value of winning and losing would be. Finally, a Bid Generator places a bid for service. Auction results translate into QoS policies that remain in place for 10 seconds.

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 from the database (as determined by the 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.

IV. Model-Free RL

We describe a model-free RL based approach for learning a control algorithm for the system described in Section III. 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.

A. 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

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

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,

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

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

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

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^* [31].

B. 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. 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.

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 [32]. 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.

1) 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

$$w = w + \alpha \nabla Q_w(s_t, a_t) \times (R_t + \gamma \max_b Q_w(s_{t+1}, b) - Q_w(s_t, a_t)) \quad (3)$$

2) 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 (3). 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 (3) [33]. New observation are then added to the replay buffer and the process is repeated.

3) Target Network: In (3), 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 [33]. However, it is known that DQN algorithm suffers from overestimation of Q values. Double DQN algorithm [32] overcomes this problem using slightly modified updated equation as

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

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

C. Simulation-Based Training the RL Algorithm

We implemented the double DQN algorithm using the TensorForce library [34]. Hyperparameters are selected by evaluating common empirical choices from the user community, as well as sweeping some parameters such as the learning rate over a small interval to find the best one. The final configuration and hyperparameter of the RL algorithm is specified in Table I.

TABLE I
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

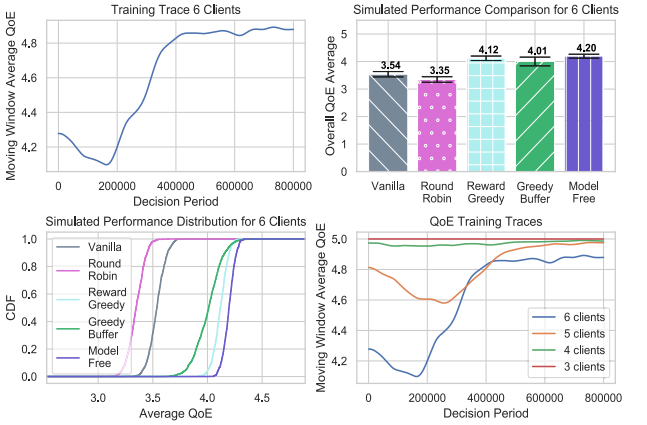

Fig. 6. Training model-free RL via simulations.

For faster training of our RL algorithm, we first implement a simulation environment which closely mimics the dynamics of the physical system. 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 plays videos sequentially, stalling where its buffer runs out and building up a buffer of 10 seconds before attempting to play again. Queues are serviced with a constant total bandwidth, 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.

For hyperparameter search, the system was simulated for 200 DP per episode for 1000 episodes (200,000 samples). 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 6 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 6 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 6 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 IX that it attains quite substantial gains over the other approaches in practice under a bandwidth constrained environment.

D. 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 6 bottom-right shows the evolution of value over the training process over the different cases. We can then choose 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. We discuss this idea further in Section VII.

V. 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 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,

$$P(s_{t+1}|a_t, s_t) = \prod_{\forall c \in \mathcal{C}} P(s^c_{t+1}|a^c_t, s^c_t) \tag{4}$$

We also note 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. Hence, the model-based approach here is attractive because of the conditional independence of the transition matrix, which makes construction of the model much easier. This may not be true of a general problem.

A. Static Model

In what follows, we determine the transition kernel of the system and obtain the optimal policy.

1) Experimental Traces: We generate state (s^c_t), action (a^c_t) and next state (s^c_{t+1}) tuples for clients by running the system under arbitrary policies (Vanilla, Round Robin etc.) for a duration of 10 hours giving 3600 sample points.

2) 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 a tractable manner, we discretize the state space of the client according to table II.

TABLE II
Client State Space Discretization

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

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 of stall and QoE bins respectively,

$$s^c_t = \text{Buffer} \times \text{NSB} \times \text{NQB} \ + \text{QoE} \times \text{NSB} \ + \text{Stall}$$

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

3) 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,

$$P(s^c_{t+1}|a^c_t, s^c_t) \quad \forall s^c_{t+1}, s^c_t \in \mathcal{S}_c \quad \forall a^c_t \in \mathcal{A}_c$$

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

4) Identifying Frequent States of the System: The state of the system (s_t) is the union of states of all clients (s^c_t) 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.

5) 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

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

6) 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 is not calculated explicitly.

B. 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 IV. 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, as will be discussed in Section VII.

VI. Auction

As discussed in the Introduction, the model-based and model-free approaches require the self-reporting of states. We now consider an auction system, wherein agents place bids to determine their relative valuations for priority service. To determine its value, a client must solve (2) and obtain the optimal value function. The information required to compute this solution are the transition kernel and the bid (belief) distribution of agents. This belief distribution is obtained from the auction server (Controller), which collects the bids made over intervals of time and provides the empirical distribution back to all agents. Furthermore, the model-based approach immediately provides the transition kernel using the transition kernel of a client (Section V-A).

The auction is chosen as an $(N+1)^{th}$-price auction with N identical goods (i.e., the number of clients that may be admitted to the high-priority queue) in which each agent may obtain at most one unit of the good. It is straightforward to see that such an auction is a simple VCG auction [35], and so is incentive compatible (agents bid true values obtained from solving (2)). The Auction Agent receives the bids from all clients, conducts the $(N+1)^{th}$-price auction and performs the assignment on the basis of the result. This approach follows our earlier results on scheduling games, wherein we proved that such a scheme results in a mean field equilibrium in which the highest value clients are prioritized at each time [5], [6].

VII. Index Policy and Dynamic Number of Clients

The auction approach suggests that at equilibrium each client is associated with a value that only depends on the state of that client, and the transition kernel (4). The solution to (2) results in a *value* for each state s_t^c of the client. Then in the manner of the Whittle Index [7], we can order states in increasing order of value, and associate each state with an *index,* which is its position in the order. Then these indices can be used to directly decide which clients to prioritise, and we call this as an *index policy* that simply picks the clients with the highest indices to provide priority service.

Now, given the indices corresponding to a system with J clients, it would save computational effort if we could use the same indices for a system with $K < J$ clients, by simply setting indices of non-existent clients to 0. For example, would the indices for a system with 6 clients be consistent with one that has 3 clients?

We determined the values for different numbers of clients using the empirical model developed in Section V-A, and determined the ordering of states in each case. The comparison of the orderings for different client configurations (6, 5, 4, 3 clients) is shown in Figure 7, using the ordering for 6 clients as the base ordering. We have not shown several hundred states that are indistinguishable at minimal value, which are all assigned an order of 0. We find the values of the 350 states that turn out to have non-minimal value with 6 clients, and assign the label 350 to the highest value state, 349 to next highest value state and so on. Hence, when ordered by value, state 350 is also the 350^{th} in order and so on, which results in the 45° red line in Figure 7. Next, we find the values of each state for 5, 4 and 3 clients, and in each case show the ordering

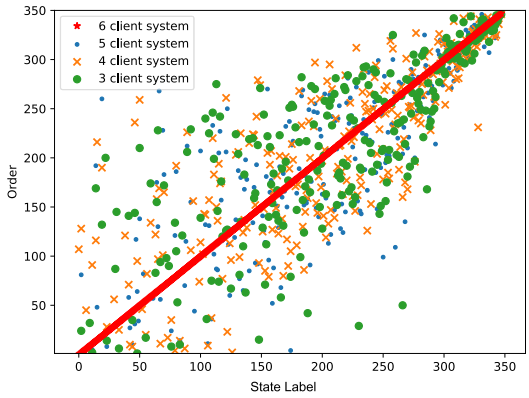

Fig. 7. Ordering of states in different client configurations.

of the states using the same state labels as we did in the case of 6 clients. We observe that the relative ordering of most of the high value states is consistent across configurations.

The above observation indicates that it is unnecessary to obtain policies tuned to the number of clients. Rather, simply training the system with a fixed number of clients and using the relative state priorities obtained from such training for an instance with a smaller number of clients is likely to perform well. In particular, a Whittle-index like policy developed for 6 clients is likely to prioritize the correct clients in a system with fewer clients. This is the approach that we use in the next section while considering experiments with a dynamically changing number of clients.

VIII. QFlow Implementation Details

We extend the OpenFlow protocol using experimenter messages [36]. 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 [37], 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.

A. 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

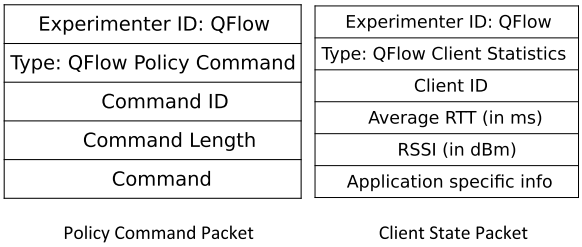

Fig. 8. Packet formats in QFlow.

the format of the policy experimenter messages as shown in Figure 8 (left). The Controller packs a policy command, and sends it to the Access Point using 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.

1) 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.

B. Statistics Commands

We define Statistics commands to collect queue and client information 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), signal strength (RSSI) and Application specific statistics like buffer state, stall information and video bitrate gathered by client middleware from the browser playing the YouTube session. 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 8 (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.

IX. Evaluation

An off-the-shelf WiFi router with QFlow is used as the Access Point and three Intel NUCs are used to instantiate up to 9 YouTube sessions as clients for our experiments. Note that each such session will generate multiple TCP flows, and we treat all the flows associated with a particular YouTube session identically. Relevant session information such as ports used by an application, play/load progress, bitrate and stall information for YouTube sessions is collected by middleware every second and written to the database.

We use the platform to study several scenarios involving one or more "bins", each containing two downlink queues, one with a higher bandwidth allocation (resulting in better QoS) than the other using token bucket queueing via Ubuntu Traffic Controller (TC). In Section IX-D, we ensure that devices with similar signal strengths are made to occupy the same bin, and hence do not adversely affect the performance of clients with better signal strengths. An example with two bins corresponding to "High" and "Low" signal strengths and four queues is shown in Figure 1. A default queue is used for any background traffic. Two clients may be allowed into each high priority queue. For the no differentiation case, we set up a single queue with the same total throughput limit as the sum of all queues in the previous scenarios. Our control problem is to determine which sessions to assign to which queues. The first two policies are based on fair network resource allocation, while the others account for application state as well.

A. Policies

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

1) Vanilla: This is the base case with a single queue that is allocated the full bandwidth, and with no differentiation between clients. This scheme attains the random access fairness that is native to CSMA.

2) Round Robin: We assign clients to the high priority queue in turn. Since we first bin clients based on channel quality, this is an approximation of channel-weighted proportional fairness. Although it is computationally inexpensive, work-conserving and prevents starvation, it might lead to the clients who have no hope of significantly increasing their QoE being promoted to high-quality service.

3) 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.

4) Greedy Buffer: This approach promotes the clients with the lowest buffered video to the high priority queue to reduce their stall probabilities. It can thus be seen as an approximation of scheduling clients that have the highest deficit in terms of service obtained thus far [3]. This policy might promote the 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.

B. Static Network Configuration

In our static configuration, we have just one bin, and each NUC hosts two YouTube sessions to simulate a total of 6 clients. The QoE performance comparison of the different

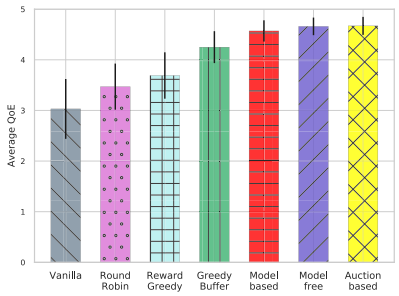

Fig. 9. Comparison of average QoE.

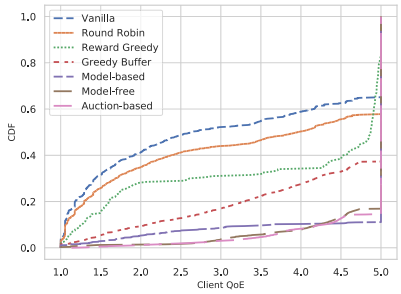

Fig. 10. Comparison of client QoE CDF.

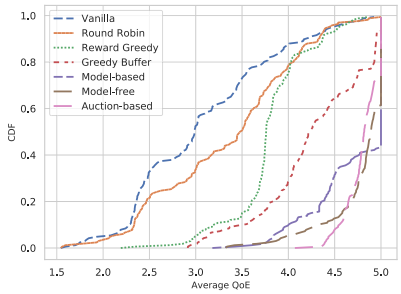

Fig. 11. Comparison of average QoE CDF.

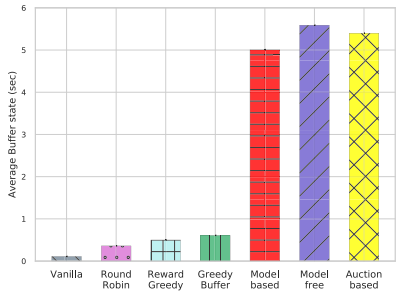

Fig. 12. Comparison of average buffer.

policies is shown in Figures 9, 10 and 11. We first compare the average QoE of the various policies in the first figure. It is clear that the model-based, model-free and auction-based 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 10 and 11. For example, we can observe from Figure 10 that the Model-based, Model-free and Auction-based policies are able to provide a QoE of 5 for almost 90, 85 and 87% 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 11 that the Model-based, Model-free and Auction-based policies are able to achieve an average QoE of 4.5 for

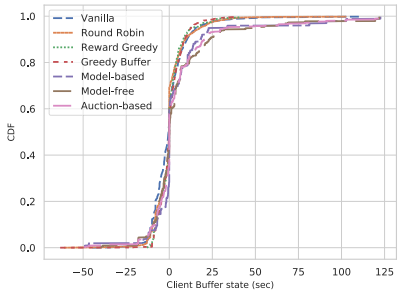

Fig. 13. Comparison of client buffer CDF.

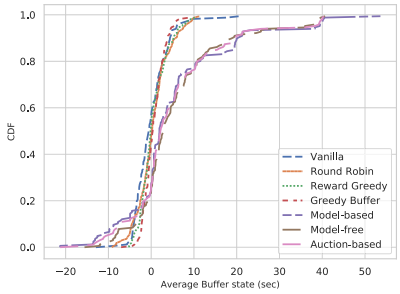

Fig. 14. Comparison of average buffer CDF.

all participating clients in the system about 70, 85 and 90% respectively. The value for the next best policy is about 35%.

Interestingly, the auction outperforms the model-based policy. We believe that this is due to the coarse quantization of the state space. System-wide identification of value is worse affected by such coarse quantization due to the fact that 6 clients together are considered in the sate, whereas in Auction only 1 client is part of the (marginal) state. Hence, we believe that relative value identification (indexing of client states; see Section VII) is more accurate in the Auction case.

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 12 to 17. Again, it is evident from the figures that the Model-based, Model-free and Auction-based policies ensure better buffer state and lower stall durations (both individual and average) than the other policies under consideration.

We also compared the bid distributions of the clients in the Auction-based policy for two different client configurations. The first had 6 clients whereas the second had 3, with the total bandwidth being the same. The comparison of the two distributions is shown in Figure 18. When there are more clients participating, resources are scarce and valuable, and clients tend to bid higher in order to get into the high priority queue and experience better QoE as seen in Figure 18 (bottom). Current cellular data rates are in the range of 0.5 to 1 cent per MB in the US, and 1080P video consumes around 1 MB per second, while 4k consumes about 2 MB per second. Hence, bids of about $[0, 5]$ cents at each auction round of 10 seconds gives us an average payment in the auction that is comparable to current cellular billing rates.

C. Dynamic Number of Clients

We next study the performance of the policies in a scenario with a varying number of clients. We still maintain a single bin, but choose the number of active clients in the system to

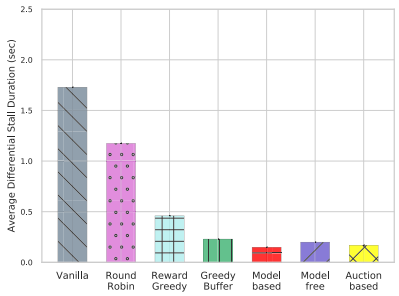

Fig. 15. Comparison of average stall duration.

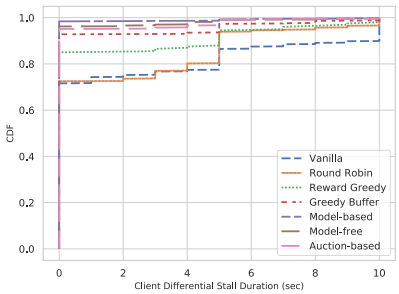

Fig. 16. Comparison of stall duration CDF.

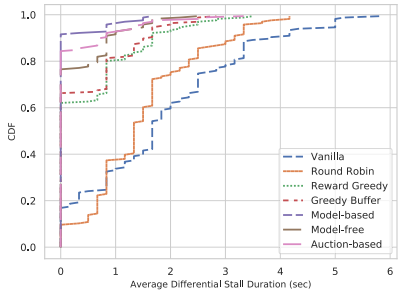

Fig. 17. Comparison of average stall duration CDF.

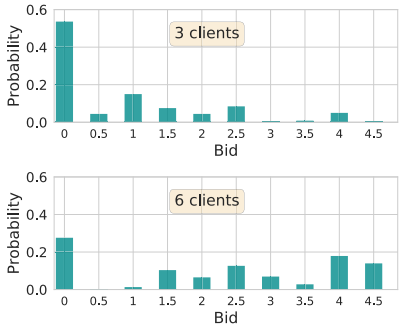

Fig. 18. Bid distribution for 6 and 3 client configurations.

vary between 4 and 6, while keeping the bandwidth allocation same as that of the static configuration. We study three policies, namely (i) Model-free: This is obtained by retraining Q-Learning for 4, 5 and 6 clients, (ii) Model-based: This is obtained by retraining the model-based approach for 4, 5 and 6 clients, (iii) Auction-based (index policy): This is obtained by training via the auction for 6 clients, and using the ordering of values so obtained as state indices.

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

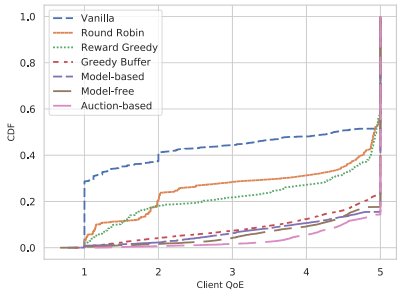

Fig. 19. Comparison of client QoE CDF for dynamic clients.

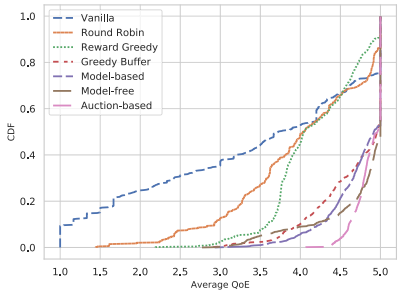

Fig. 20. Comparison of average QoE CDF for dynamic clients.

introduce two more clients in the system. It is observed that Model-based, Model-free and Auction-based (index) 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. Also, the index approach also implies that we only need train once, and not for each possible number of clients separately. Hence, it has a lower complexity as compared to the other two approaches.

Observe that 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 19 and 20, where the CDF curves of the other policies are closer to those of model-based, model-free and auction-based policies. Even so, these three policies exhibit the best performance, which reinforces their superiority in both static and dynamic client scenarios.

D. Time Varying Channel Conditions

Wireless clients could have time varying signal strengths, and consequently face different link-level throughputs, latencies, and loss rates. We need to ensure that clients having lower signal strengths do not adversely affect the performance of clients with better signal strengths by occupying the channel longer for each packet transmission [38]. Hence, we create two bins of downlink queues, each containing a high priority and a low priority queue as shown in Figure 1. We then have a *Good* bin for clients with high signal strengths, and a *Bad* bin for those who have low signal strengths.

In order to ensure repeatably of experiments across different polices as clients experience good and bad channels, we emulate a bad channel by reducing the throughput, and increasing the latency and loss rates of the queues in the *Bad* bin as compared to those in the *Good* bin using Ubuntu Network Emulator (NetEm). We then create a sequence of good and bad (emulated) channel conditions over time for each client that we repeat for each policy. In order to determine a realistic emulation of what "bad" might mean for video streaming, we

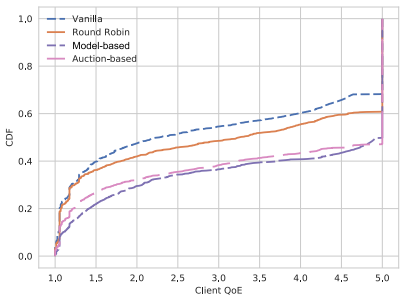

Fig. 21. Comparison of QoE CDF for bad channel.

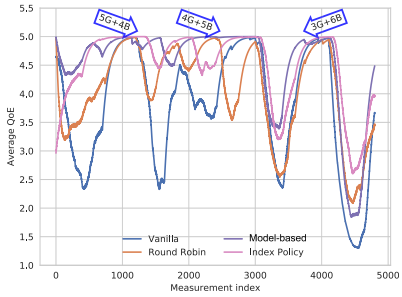

Fig. 22. Evolution of QoE: Dynamic clients with variable channels.

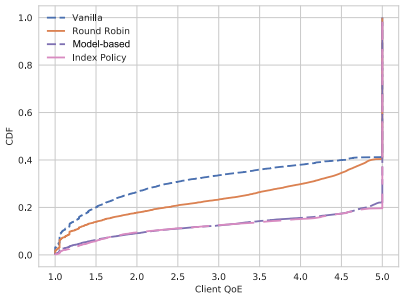

Fig. 23. Comparison of QoE CDF for dynamic clients with variable channels.

ran several hours of experiments with clients having low signal strengths (via antenna attenuators) to determine appropriate emulator settings. Thus, we are able to mimic varying network conditions by dynamically assigning the sessions hosted on the NUCs to either the *Good* or the *Bad* bin.

We consider four policies: (i) Vanilla and (ii) Round Robin, and two advanced polices that show good performance, namely (iii) Model-based RL and (iv) Auction-based (which yields an ordering for the index policy). We first illustrate the difference in achieved performance for the different policies with a static 6 clients under *Good* vs. *Bad* channel conditions by comparing Figures 10 and 21. We observe that the gap between the baseline and advanced policies decreases in the *Bad* channel scenario, but Model-based RL and the Auction-based policies still achieve higher QoE for the clients.

We next fix a sequence of client configurations (number of active clients) under each channel condition for the evaluation of all policies. The first configuration consists of 6 clients under *Good* channel conditions and 3 under *Bad* channel conditions. We decrease the number of clients in the *Good* channel by 1 and increase those in the *Bad* channel by 1 for the next three intervals. The evolution of the average QoE for each of the policies for the above sequence is shown in Figure 22. The Model-based RL and Auction-based (index) policies exhibit a high average QoE in most of the configurations except for the last where it is not possible to achieve a high QoE for the 6 clients in the *Bad* channel. Even in such a scenario, the drop in QoE is less severe than the other policies.

Finally, we show the overall CDF of client QoEs taken over the whole experiment interval in Figure 23. While the QoE improvement from the baseline using the learning-based policies is not as striking as it is in Figure 10, the QoE samples with the learning policies have a perfect QoE score in about 80% of the samples as compared to he baseline policies that only manage this in about 60% of the samples.

E. Discussion of Limitations

We fix the video resolution at 1080p, which means that our learning algorithms do not account for variable video bitrate. While this can be included by joint selection of priority and bitrate, that would require APIs for coordination across the content provider (YouTube) and the RL-controller. Another issue is that on the one hand, the model-based approach is based on offline empirical data and so is limited to exactly that environment. On the other hand, the model-fee approach has inaccuracies due to its being simulation-trained, but does undergo some fine-tuning during experiments. This appears to be sufficient to match model-based algorithm performance, but we do not explicitly use Sim2Real methods to add robustness to simulation inaccuracies.

X. Conclusion

We considered the design, development and evaluation of QFlow, a platform for learning based edge network configuration, applicable to small cell architectures such as 5G. Working with off-the-shelf hardware and open source operating systems and protocols, we showed how to couple queueing, learning, and markets to develop a system that is able to reconfigure itself to best suit the needs of video streaming applications.

We instantiated a variety of learning-based policies on the platform, and showed that they significantly outperform purely network resource fairness-based or application state-based policies. The RL approach is successful in learning complex multi-layer interactions across network control (such as TCP) and application performance. We also showed that using an auction framework is able to a elicit truthful proxy for state in terms of the bid made for prioritized service, and discovered an ordering of state values that can be applied directly as a simple index policy.

Acknowledgment

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

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

[2] A. Eryilmaz, R. Srikant, and J. R. Perkins, "Stable scheduling policies for fading wireless channels," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 411–424, Apr. 2005.

[3] I.-H. Hou, V. Borkar, and P. R. Kumar, "A theory of QoS for wireless," in *Proc. IEEE INFOCOM 28th Conf. Comput. Commun.*, Apr. 2009, pp. 1–9.

[4] Ericsson. (2015). *Ericsson Mobility Report: On the Pulse of the Networked Society*. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2015/er%icsson-mobility-report-june-2015.pdf

[5] M. Manjrekar, V. Ramaswamy, and S. Shakkottai, "A mean field game approach to scheduling in cellular systems," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 1554–1562.

[6] M. Manjrekar, V. Ramaswamy, V. Reddyvari Raja, and S. Shakkottai, "A mean field game approach to scheduling in cellular systems," *IEEE Trans. Control Netw. Syst.*, vol. 7, no. 2, pp. 568–578, Jun. 2020.

[7] P. Whittle, "Restless bandits: Activity allocation in a changing world," *J. Appl. Probab.*, vol. 25, pp. 287–298, Jan. 1988.

[8] R. Singh and P. R. Kumar, "Optimizing quality of experience of dynamic video streaming over fading wireless networks," in *Proc. 54th IEEE Conf. Decis. Control (CDC)*, Dec. 2015, pp. 7195–7200.

[9] R. Bhattacharyya *et al.*, "QFlow: A reinforcement learning approach to high QoE video streaming over wireless networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jul. 2019, pp. 251–260.

[10] 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 *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 87–92.

[11] H. Nam, K.-H. Kim, J. Y. Kim, and H. Schulzrinne, "Towards QoE-aware video streaming using SDN," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 1317–1322.

[12] S. Ramakrishnan, X. Zhu, F. Chan, and K. Kambhatla, "SDN based QoE optimization for HTTP-based adaptive video streaming," in *Proc. IEEE Int. Symp. Multimedia (ISM)*, Dec. 2015, pp. 120–123.

[13] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proc. ACM SIGCOMM Workshop Future Hum.-Centric Multimedia Netw.*, Aug. 2013, pp. 15–20.

[14] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 197–210.

[15] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "QARC: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 1208–1216.

[16] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li, "DRL360: 360-degree video streaming with deep reinforcement learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1252–1260.

[17] G. Xiao, M. Wu, Q. Shi, Z. Zhou, and X. Chen, "DeepVR: Deep reinforcement learning for predictive panoramic video streaming," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 4, pp. 1167–1177, Dec. 2019.

[18] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 325–338.

[19] Y. Qin *et al.*, "A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2505–2519, Nov. 2020.

[20] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[21] J. Sun, E. Modiano, and L. Zheng, "Wireless channel allocation using an auction algorithm," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 5, pp. 1085–1096, May 2006.

[22] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang, "TUBE: Time-dependent pricing for mobile data," in *Proc. ACM SIGCOMM*, 2012, pp. 247–258.

[23] P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde, and A. Sprintson, "CrossFlow: A cross-layer architecture for SDR using SDN principles," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 37–39.

[24] P. Shome, J. Modares, N. Mastronarde, and A. Sprintson, "Enabling dynamic reconfigurability of SDRs using SDN principles," in *Proc. Ad Hoc Netw.*, 2017, pp. 369–381.

[25] M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton, "ÆtherFlow: Principled wireless support in SDN," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 432–437.

[26] J. Schulz-Zander, N. Sarrar, and S. Schmid, "AeroFlux: A near-sighted controller architecture for software-defined wireless networks," in *Proc. USENIX ONS*, 2014, pp. 1–2.

[27] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic control over home and enterprise WiFi," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, pp. 1–12.

[28] H. Yeganeh, R. Kordasiewicz, M. Gallant, D. Ghadiyaram, and A. C. Bovik, "Delivery quality score model for internet video," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 2007–2011.

[29] N. Eswara *et al.*, "A continuous QoE evaluation framework for video streaming over HTTP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 11, pp. 3236–3250, Nov. 2018.

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

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

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

[33] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[34] M. Schaarschmidt, A. Kuhnle, and K. Fricke. (2017). *Tensorforce: A Tensorflow Library for Applied Reinforcement Learning*. [Online]. Available: https://github.com/reinforceio/tensorforce

[35] V. Krishna, *Auction Theory*. New York, NY, USA: Academic, 2009.

[36] *Open Networking Foundation STD*, document ONF TS-023, OpenFlow Switch Specification, Rev., vol. 1, no. 5, 2015, p. 3.

[37] CPqD. (2015). *OpenFlow Software Switch*. [Online]. Available: http://cpqd.github.io/ofsoftswitch13/

[38] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11 b," in *Proc. IEEE INFOCOM 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, vol. 2, Mar. 2003, pp. 836–843.

Rajarshi Bhattacharyya received the Ph.D. degree in electrical and computer engineering from Texas A&M University in 2019. He currently works as a Software Designer at Aruba Networks, the wireless networking subsidiary of Hewlett Packard Enterprise. His research interest is in the area of wireless communication networks, with a focus on providing guarantees of quality of service and quality of experience at the wireless edge.

Archana Bura is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University. Her research interests are reinforcement learning, optimization, and their applications to wireless networks.

Desik Rengarajan is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University. His research interests include reinforcement learning and game theory, with a focus on their application to the real world.

Mason Rumuly received the M.S. degree in electrical engineering from Texas A&M University in 2019. He currently works as a Software Engineer at Arista Networks. His research interests include cryptography, machine learning, and reinforcement learning.

Bainan Xia received the Ph.D. degree in computer engineering from Texas A&M University in 2019. He currently works as a Research Software Lead at Breakthrough Energy, the clean energy innovation subsidiary of Gates Ventures. His research interests include optimization, game theory, reinforcement learning, and network economics, with a focus on large scale power system optimization and mechanism design in electricity market.

Srinivas Shakkottai (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 2007.

He was a Post-Doctoral Scholar in management science and engineering at Stanford University in 2007. In 2008, he joined Texas A&M University, where he is currently a Professor of computer engineering with the Department of Electrical and Computer Engineering. His research interests include caching and content distribution, wireless networks, multi-agent learning and game theory, and network data collection and analytics. He was a recipient of the Defense Threat Reduction Agency Young Investigator Award in 2009, the NSF CAREER Award in 2012, and the research awards from Cisco in 2008 and Google in 2010. He received an Outstanding Professor Award in 2013, the Select Young Faculty Fellowship in 2014, and the Engineering Genesis Award at Texas A&M University in 2019.

Dileep Kalathil (Senior Member, IEEE) received the Ph.D. degree from the University of Southern California (USC) in 2014. From 2014 to 2017, he was a Post-Doctoral Researcher with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research is in the area of reinforcement learning, with applications in communication networks, power systems, and intelligent transportation systems. He was a recipient of the NSF CAREER Award in 2021, the NSF CRII Award in 2019, the Best Ph.D. Dissertation Award from the Department of Electrical Engineering, USC, in 2014–2015, and the Best Academic Performance Award from the EE Department, IIT Madras, in 2008.

Ricky K. P. Mok (Member, IEEE) received the B.Eng. degree in computer engineering from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, in 2009, and the Ph.D. degree in computer science from the Department of Computing, The Hong Kong Polytechnic University, in 2016. He is currently an Assistant Research Scientist at CAIDA/University of California San Diego. His research interest includes network performance measurements, network data analysis, HTTP streaming systems, quality of experience (QoE) measurement, and reliable crowd-sourcing-based QoE assessment methods.

Amogh Dhamdhere received the Ph.D. degree in computer science from Georgia Institute of Technology, Atlanta, GA, USA, in 2009. From 2009 to 2019, he was at the Center for Applied Internet Data Analysis (CAIDA), University of California, San Diego, first as a Post-Doctoral Scholar (2009–2011) and then as a Research Scientist (2011–2019). In 2019, he joined Amazon Web Services as a Principal Research Scientist. His research interests revolve around measurement and modeling of internet topology, traffic, economics, and protocols. His recent work has focused on building tools and systems to measure internet availability and performance. He received the ACM SIGCOMM Best Paper Award in 2018 for the paper "Inferring Persistent Interdomain Congestion."