Reinforcement Learning Based Active Queue Management for Priority Queues

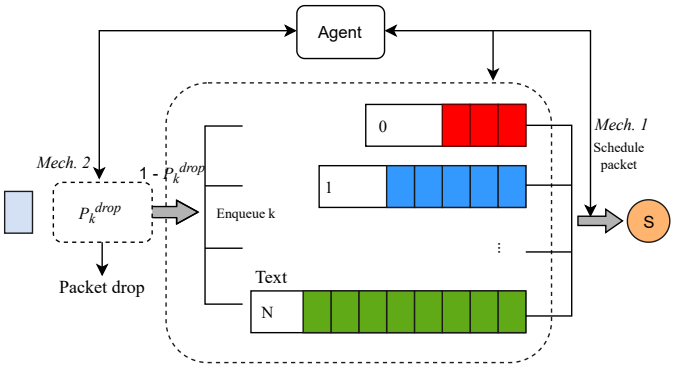

Fig. 1. System model.

***Abstract*—We study the joint user association and resource allocation problem in both uplink (UL) and downlink (DL) for full-duplex heterogeneous cellular networks...**

***Index Terms*—UAV communication, User association, Network deployment, Cache placement, MDS-coded caching.**

I. Introduction

II. System Model

In this section, we introduce the design of the proposed TD-AQM algorithm. An overview of our system model is shown in Fig. 2, where we consider N priority queues. Following a similar design rationale in conventional AQM algorithms [1], TD-AQM aims to achieve low latency, high link utilization, and guaranteed stability by dropping or marking the ECN bit in packets early at bottlenecks, especially for TCP traffics. An early congestion signal causes reduction in end user's sending rate to alleviate network congestion. On the other hand, high drop rate may cause link under-utilization. Therefore, a trade-off exists between high link utilization and low latency. Apart from that, unlike conventional AQM algorithms, where a single-buffer scenario is normally considered, we aims at achieving an overall optimal utilization and delay-guarantee for multiple priority queues.

To guarantee the queue delay in high priority queues while maximizing the overall throughput, TD-AQM works based on the weighted-round-robin (WRR) scheduling scheme, which is adopted in multiple queues scenarios [2] in case not to starve the lower priority queues. Two mechanisms are designed in the proposed TD-AQM as shown in Fig. 2. For simplicity, in our system model, we assume each time-sensitive packet carries a delay budget which indicates the queue delay upper bound at each node.

We further classify the priority queues into two categories, namely *guaranteed-service queue* (GS) and *best-effort queue* (BE). The GS queues are assigned with fixed number of tokens for scheduling, while the BE queues are scheduled only when there are remaining tokens in each round. In our model, we consider one BE queue, and the rest are GS queues.

In $Mech.$ 1, the scheduler schedules the priority queues in a predefined order, e.g., in terms of priority. In each round, the scheduler uses the full link rate C for forwarding at each queue, and moves to the next queue either because the current queue is empty or it has used up all the tokens. It is also worth noting that in $Mech.$ 1, the scheduler inspects the delay budget of the packet before dequeuing. If the sojourn time of the packet x exceeds its delay budget Th, the packet is dropped before being served, i.e.,

$$D(\tau) = \begin{cases} 1, & \text{if } x \geq Th, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In $Mech.$ 2, the smart agent dynamically computes the packet drop probability P of the priority queues based on the observations of the queue status. At the end of each round, the agent observes the queue length of each priority queue and computes the drop probability respectively by maximizing the reward function defined in Eq. 2.

$$\max_{P_k} \left\{ \sum_{k=1}^{N} \gamma_k (Delay_k^{ref} - Delay_k^{now}), \sum_{k=1}^{N} \gamma_k Enq_k \right\}, \tag{2}$$

where $\gamma_k, Delay_k^{ref}, Delay_k^{now}, Enq_k$ are the reward gain, reference queue delay, current queue delay, and enqueue rate of each priority queue, respectively. The details of the reward function are explained in Sec. III.

III. TD-AQM netowrk model analysis

A. Network model

For simplicity, we consider the problem in discrete time slots. Let $\mathcal{K} = \{1, \ldots, N\}$ denotes the set of priority queues that are deployed in a network node. Let $A_k(t) \in \{0, 1\}$ denotes the packet arrival index. We assume that every packet has the same size and at most one packet arrives randomly at each queue during one time slot, i.e., $\sum_k A_k(t) \leq N$. Thus, the packet arrival rate of the system follows a Poisson distribution $\pi(\lambda)$ with an average rate of λ.

We define $\mathbf{Q}(t) = \{Q_1(t), \cdots, Q_N(t)\}$ as the set of packets currently backlogged for each queue. $x_{p,k}$ denotes the sojourn time of packet p in queue k. Besides, the queuing delay budget of the packet is denoted by Th_p. According to $Mech.$ 1 that packet p will be dropped if its sojourn time is larger than its queue delay budget, consequently, a discrete binary variable $\{D_k(t), k \in \mathcal{K}\}$ is introduced to indicate the queuing status

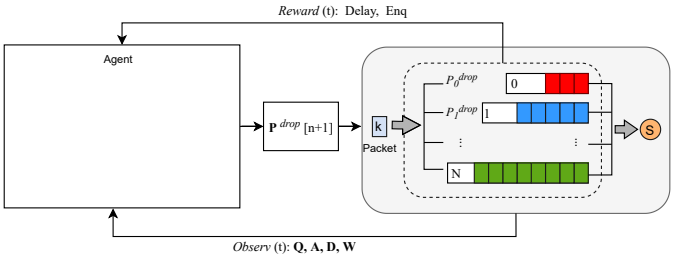

Fig. 2. System model.

of each packet. Then the following queuing constraints need to be satisfied

$$D_k(t) = \begin{cases} 1, & \text{if } x_{p,k} \geq \text{Th}_p, \\ 0, & \text{if } x_{p,k} < \text{Th}_p, \end{cases} \quad \forall k \in \mathcal{K}. \tag{3}$$

Since the WRR scheduler is considered in our system, the service time for each priority queue is guaranteed. We define T as the total number of time slots within the time period of one round, T_k as the assigned tokens to each priority queue. Let w_k denote the tokens being used at queue k, then $w_k \in [0, T_k]$ is determined by the current queue length.

To sum up, the dynamics of the GS queues within one round time period can be formulated as follows:

$$\begin{aligned} Q_k(t+T) = &\max\left(* - T_k - D_k, 0\right) \\ &+ \sum_{\tau=w_k}^{T-w_k} A_k(\tau)(1 - P_k(t)), \forall k \in \mathcal{N}, \end{aligned} \tag{4}$$

where

$$* = Q_k(t) + \sum_{\tau=0}^{T_k} A_k(\tau)(1 - P_k(t)), \tag{5}$$

$$w_k = \begin{cases} T_k, & \text{if } * \geq T_k, \\ *, & \text{otherwise}, \end{cases} \quad \forall k \in \mathcal{N}, \tag{6}$$

and $D_k = \sum_{\tau=0}^{w_k} D(\tau)$ is the total number of packet being dropped at queue k due to time out. Similarly, the dynamics of the BE queues within one time period can be formulated

$$Q(t+T) = Q(t) - (T - \sum_{k=1}^{N-1} w_k). \tag{7}$$

B. TD-AQM model analysis

Based on the analysis of queue dynamics of the system, our AQM model to decide can be further formulated as a Markov decision process (MDP) problem with finite state space $\mathcal{S}$ and action space $\mathcal{A}$. The state space of the system at time t can be written as $s_t = \{\mathbf{Q}(t)\}, \forall s_t \in \mathcal{S}$, and the action space $a_t = \{\mathbf{P}(t)\}, \forall a_t \in \mathcal{A}$. At the beginning of each time period, the agent observes and gives an action a_t based on a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Then at the end of the period, the agent receives the reward of that action $r(s_t, a_t)$, where $r : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}$. The reward function $r(\cdot)$ in our model is formulated based on the current delay $Delay_k^{now}$, enqueue rate Enq_k, and output gain γ_k of each packet. The justification of reward function is explained in the following section. An overview of the TD-AQM framework is shown in Fig. 2.

1) State space: The agent periodically observes the states of the system at the beginning of each period. The observations essentially shows the results of the action taken in the previous round, which include current queue length **Q**, total number of packets arrived **A**, number of tokens being used **W**, and the total number of dropped packets due to time out **D**. Each state is a vector of size $N \times 1$, where N is the number of priority queues. Thus, the state space of each round can be formulated as follows:

$$\mathcal{S}_t = \{\mathbf{Q}_t, \mathbf{A}_t, \mathbf{W}_t, \mathbf{D}_t\}. \tag{8}$$

2) Action space: The agent gives an action $\mathcal{A}_t$ for the upcoming period based on the observations $\mathcal{S}_t$. The agent first computes the packet drop probability for each queue $P_k, k \in N$, then makes the drop decision according to a deterministic policy π. Thus, the action space is a function of drop probability written as follows:

$$\mathcal{A}_t = \{\pi_\mu(\mathbf{P})\}. \tag{9}$$

A policy μ is said to be optimal if π_μ yields the maximum reward of the system.

3) Reward function design: The reward function plays an important role in reinforcement learning as it prompts the agent to give the optimal action. After executing the action, i.e., computing the drop probability, the agent obtains the reward at the end of each period. The design principle of reward function is to penalize action that results in longer delay and more time-out drops, meanwhile, encourage higher enqueue rate. In our system, two reward functions are designed representing two important properties.

- *Delay function.* In TD-AQM, each priority queue holds a reference delay value $Delay^{ref}$ which reflects the delay requirement. Then the delay function for queue k can be defined

$$R_k^d = \gamma_k(Delay_k^{ref} - Delay_k^{now})/Delay_k^{ref}, \tag{10}$$

  where γ_k is the scaling factor that reflects the reward gain for different priority queues. $Delay_k^{now}$ measures the current queue delay after one round of scheduling, i.e., $Delay_k^{now} = \frac{Q(t+T)}{C_k}$. $C_k = \frac{T_k}{T}C$ is the proportional link rate assigned to queue k.
- *Enqueue rate function.* Maximizing the delay function alone will lead to high drop probability and consequently, harm the throughput of the system. To balance the trade-off between drop probability and throughput, the enqueue rate function is defined

$$R_k^e = \gamma_k Enq_k = \gamma_k \frac{\sum_{\tau=0}^{T} A_k(\tau)(1 - P_k) - D_k}{\sum_{\tau=0}^{T} A(\tau)}, \tag{11}$$

  where $\sum_{\tau=0}^{T} A_k(\tau)(1 - P_k) - D_k$ reflects the effective number of packets enqueued to the system.

Above all, the solution to the MDP problem is to find the optimal drop probability **P** that maximize the total reward

function of the system, i.e.,

$$\max_{P_k} \left\{ \sum_{k=1}^{N} \gamma_k (1 - \frac{Delay_k^{now}}{Delay_k^{ref}}), \sum_{k=1}^{N} \gamma_k Enq_k \right\} \tag{12a}$$

$$\text{s.t.} \qquad 0 \leq P_k \leq 1, \ \forall k \in N, \tag{12b}$$

where

$$\begin{cases} Delay_k^{now} & = \frac{Q(t+T)}{C_k}, \\ C_k & = \frac{T_k}{T} C, \end{cases} \quad \forall k \in \mathcal{N}, \tag{13}$$

$$Enq_k = \frac{\sum_{\tau=0}^{T} A_k(\tau)(1 - P_k) - D_k}{\sum_{\tau=0}^{T} A(\tau)}. \tag{14}$$

The multi-objective optimization problem (MOOP) can be simplified using the weighted sum method. The objective function can be formulated as follows:

$$\alpha \cdot \sum_{k=1}^{N} \gamma_k (1 - \frac{Delay_k^{now}}{Delay_k^{ref}}) + \beta \cdot \sum_{k=1}^{N} \gamma_k Enq_k, \tag{15}$$

where α, β are two weights which reflect to what extent the reward function impact the total reward.

C. Proposed algorithm

Since the traffic arrives each priority queue at random, it is hard to give accurate prediction on the queue length. Consequently, other parameters such as used tokens w_k, time-out drops D_k cannot be determined. To simplify the problem, the temporal difference (TD) method is adopted. The state values of two successive time periods are used for queue length estimation. The complete TD-AQM algorithm is shown in Alg. 1. Initially, the system is empty and all states values are zero. The agent records the estimated queue length Q^* by Eq. 16 and the drop probability of n th period $\mathbf{P}^*[n]$ based on the observations $\mathcal{S}_t[n-1]$ of $(n-1)$ th period. After executing the action with $\mathbf{P}^*[n]$, the agent observes the system state values $\mathcal{S}_t[n]$ including the true queue length $\mathbf{Q}[n]$. Then the optimal value of drop probability at n th period $\mathbf{P}[n]$ can be calculated by solving Eq. 12 with the posterior knowledge. The differences between the estimated values $\mathbf{P}^*[n], \mathbf{Q}^*[n]$ and true values $\mathbf{P}[n], \mathbf{Q}[n]$ are used for adjusting estimations respectively using Eq. 16

$$\begin{cases} Q_k^*[n+1] = Q_k^*[n+1] + \eta_1 (Q_k[n] - Q_k^*[n]) \\ P_k^*[n+1] = P_k^*[n+1] + \eta_2 (P_k[n] - P_k^*[n]) \end{cases} \quad \forall k \in \mathcal{N}, \tag{16}$$

where the primary estimations $Q_k^*[n+1], P_k^*[n+1]$ on the RHS are calculated by Eq. 4 and Eq. 12, respectively.

Algorithm 1 Temporal-Difference Active Queue Management

Initialize state values: $\mathcal{S}_t[0] = \{\mathbf{Q}, \mathbf{A}, \mathbf{D}, \mathbf{W}.\}$, and control parameters: $\{\alpha, \beta, \eta_1, \eta_2, \mathbf{T}_k\}$.
for time slot $\tau \in T$ **do**
  if $\tau = 0$ **then**
    for all queues $k \in \mathcal{N}$ **do**
      Estimate $Q_k^*[1]$ based on $\mathcal{S}_t[0]$ by Eq. 4.
      Update $Q_k^*[1]$ by Eq. 16.
      Obtain $P_k^*[1]$ by solving Eq. 12 based on $Q_k^*[1]$ and $\mathcal{S}_t[0]$.
      Update $P_k^*[1]$ by Eq. 16
      Set $P_k^*[1]$ as drop probability for queue k.
    end for
  end if
  if $\tau = T$ **then**
    for all queues $k \in \mathcal{N}$ **do**
      Observe current state values $\mathcal{S}_t[1]$.
      Find optimal $P_k[1]$ by solving Eq. 12 based on $Q_k[1]$ and $\mathcal{S}_t[1]$.
      Update $\mathcal{S}_t[0] = \mathcal{S}_t[1]$, $P_k^*[0] = P_k^*[1]$, and $P_k[0] = P_k[1]$.
    end for
  end if
end for

IV. Simulation and performance analysis

In the simulation experiment, we consider a

The result concerning the convergence of Algorithm 1 is investigated

A. Analysis of Performance

In this subsection, we first compare our proposed

V. Conclusions

In this paper, we solve the problem of joint optimization

References

[1] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "Pie: A lightweight control scheme to address the bufferbloat problem," in *2013 IEEE 14th international conference on high performance switching and routing (HPSR)*. IEEE, 2013, pp. 148–155.

[2] T.-G. Kwon, S.-H. Lee, and J.-K. Rho, "Scheduling algorithm for real-time burst traffic using dynamic weighted round robin," in *ISCAS'98. Proceedings of the 1998 IEEE International Symposium on Circuits and Systems (Cat. No. 98CH36187)*, vol. 6. IEEE, 1998, pp. 506–509.