2023 59th Annual Allerton Conference on Communication, Control, and Computing (Allerton) | 979-8-3503-2814-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/Allerton58177.2023.10313418

Delay Performance Optimization with Packet Drop

Faraz Farahvash
Electrical and Computer Engineering
Cornell University
ff227@cornell.edu

Ao Tang
Electrical and Computer Engineering
Cornell University
atang@cornell.edu

Abstract—**In this paper, we aim to improve delay performance of an M/M/1 queue by dropping the right packets at the right time. First, for a given dropping budget, we explore the average delay metric and prove that dropping from the head of the queue is optimal. Next, we look into cases where packets have a deadline they need to make. We introduce the concept of gain and propose a policy called "Drop Positive Gain Policy (DPGP)" that increases the percentage of packets meeting their deadline. Finally, some numerical examples are provided to confirm that DPGP outperforms conventional policies.**

I. Introduction

Cloud computing has revolutionized our world by finally realizing the longstanding dream of computing as a utility. [1] However using clouds as a centralized server, increases the frequency of communication between users and cloud servers [2]. This results in several shortcomings including (a) High latency for real-time applications, (b) Security and privacy, and (c) Energy consumption [3].

With the recent advent of edge computing, the possibility of performing parts of computation on edge nodes (i.e. devices, routers, switches, and base stations) has emerged [4]. By doing so, much lower latency can be achieved. These advancements have made the need for better scheduling and queue management techniques more evident, since queueing delay becomes more notable with reduced propagation delay.

In this paper, we explore delay-minimization in M/M/1 queues with smart dropping policies. There have been numerous works on queue management techniques [5]. [6] characterizes the stability conditions, and queue metrics in an M/M/1 queue with packet drop and unlimited buffer. [7] extends the analysis to M/G/1 queues. There have also been various papers on queues with finite buffer sizes [8], [9], [10], [11]. However, the problem of optimal dropping policy with regard to delay has not been studied.

We will analyze two different delay metrics. The first metric we look into is the average delay. Initially, we restrict ourselves to policies that drop the packets from the tail of the queue. We show that the optimal policy is a widely used policy. Then, we will allow packets to be dropped from anywhere in the queue. In this formulation, a packet that enters the queue is not necessarily served. Implementing some of these policies results in a "leaky queue". In a leaky queue, some of the packets are dropped after entering the queue. Such queues don't have the simple delay distribution of the M/M/1 queue. [12] We argue that dropping from the head of the queue will optimize the average delay of the system. This result aligns with the algorithm proposed by [13].

The second metric analyzed is the percentage of packets meeting a deadline. In some scenarios, packets have a deadline that they should meet. The packet should be delivered by that deadline otherwise, it is useless [14], [15], [16]. In these situations, the percentage of packets meeting the deadline is a natural metric to be optimized. We will first focus on a special case which is an extension of the M/M/1/K queue. We will introduce the concept of *gain*. We show that dropping from the head maximizes the gain in that case. We will then propose a policy called the Drop Positive Gain Policy (DPGP) that increases the expected percentage of packets meeting the deadline in comparison with conventional policies.

The rest of the paper is organized as follows. In section II, we formally define the problem. Section III explores the average delay metric. In section IV the deadline metric is discussed. Section V presents some simulations. Section VI concludes the paper.

II. Problem Formulation

In this section, we will formally define different aspects of the problem. We will use conventional terminology to define the M/M/1 queue. We assume that the queue has a First Come, First Served policy. The queue has an arrival rate λ (a Poisson process), and service time has an exponential distribution with parameter μ. we also define $\rho = \lambda/\mu$.

Let ϵ be the dropping budget (meaning the probability of dropping a packet should be less than or equal to ϵ). Define D_ϵ to be the family of dropping policies where the dropping probability is less than or equal to ϵ.

This work was supported by National Science Foundation (NSF). Grant number: 2133481.

A. *Delay Metrics*

Various delay metrics can be of interest including but not limited to:

(a) Average delay: One of the natural metrics to optimize is the average delay. In the most general case, the problem can be formulated as follows. Let T_d be the Random variable of the packet delay under dropping policy d. The general optimization problem will be:

$$\begin{aligned} &\min \mathbb{E}[T_d] \\ &\text{s.t } d \in D_\epsilon \end{aligned} \tag{1}$$

(b) Percentage of packets making the deadline: In some scenarios, packets have a deadline that they should meet. The packet should be delivered by that deadline otherwise, it is not useful. In these situations, the percentage of packets meeting the deadline is a better metric than the average delay alone.
In the simplest case, we assume that all packets have the same hard deadline D. We define $F_{T_d}(x)$ to be the Cumulative Distribution Function (CDF) of the packet delay. The optimization problem then becomes:

$$\begin{aligned} \max_{d} \quad & F_{T_d}(D) \\ \text{s.t.} \quad & d \in D_\epsilon \end{aligned} \tag{2}$$

(c) Tail latency: An important metric to assess network quality is the tail of the delay distribution. We define k^{th} percentile delay as s_d^k under dropping policy d, where k percent of packets experience a lower delay than s_d^k.

$$Pr[T_d < s_d^k] = k/100 \tag{3}$$

Lower k^{th} percentile delay is preferred. We can write our optimization problem as:

$$\begin{aligned} \min_{d} \quad & s_d^k \\ \text{s.t.} \quad & d \in D_\epsilon \end{aligned} \tag{4}$$

B. *Policy families*

Note that the optimization problems in equations 1, 2, and 4 are too general. We will consider some special families of policies throughout this paper. First of all, we restrict ourselves to dropping policies that are triggered by the arrival of a new packet (a packet can only be dropped after the arrival of a new packet). Furthermore, we can have further restrictions on the policies as follows:

- Drop from the tail policies. In these policies, packets are only dropped from the tail. Most conventional dropping policies (for example M/M/1/K queues) are from this family.
- Drop from the head policies. These policies are similar to drop-the-tail policies but packets are only dropped from the head of the queue instead of the tail.
- Drop from anywhere policies. In this family of policies, we are allowed to drop from anywhere in the queue.

We can also further categorize the policies into deterministic and probabilistic policies.

C. *State definition*

We can define the state of the queue in different ways. We will look into two different definitions.

(a) Queue length: Traditionally, the state of the queue is defined as its length. Thus, we can show the state as N where N is the length of the queue.

(b) Extensive states: In conventional methods, we assume that the only information that we have about the current state of the queue is its length. Including more information about the queue's state, can lead to better queue management policies.
A simple extension to the state could be storing the time each packet has spent in the queue. Let N be the length of the queue and $T = (T_1, ..., T_N)$ denote the time each packet has spent in the queue until now. (let T_1 be the head of the queue and T_N the tail of the queue.)
As packets are served in an FCFS manner, the packets that are closer to the head have spent longer times in the queue. Thus, $T_1 \geq T_2 \geq ... \geq T_N$.

As we are only looking into policies where we drop after an arrival, state transitions only happen after an arrival or departure. In the case of extensive states, we have:

1) Service: After a packet service, the new state will be $T' = (T_2 + \tau, T_3 + \tau, ..., T_N + \tau)$, where τ is the time it took from the last state to serve a packet.

Lemma 1. *Assume X and Y are independent random variables with parameters μ, λ. Then*

$$p(X < Y) = \frac{\mu}{\mu + \lambda} \tag{5}$$

and

$$f(x|X < Y) = (\lambda + \mu)e^{-(\lambda+\mu)x} \tag{6}$$

Using the lemma above, we get that τ comes from an exponential distribution with parameter $\mu + \lambda$. Furthermore, the probability of a packet service triggering the state transition is $\frac{\mu}{\mu+\lambda}$.

2) Arrival: After a packet arrival, the new state will be $T' = (T_1 + \tau, T_2 + \tau, ..., T_N + \tau, 0)$, where τ is the time it took from the last state to serve a packet. Now we have two possibilities:

- $N > 0$: If the queue is not empty, an arrival happens if the service time is longer than the next arrival time. Thus, τ has the distribution of $f(arrival|service > arrival)$. Using lemma 1, we get that τ comes from an exponential distribution with parameter $\mu + \lambda$. Furthermore, the probability of a packet arrival triggering the state transition is $\frac{\lambda}{\mu+\lambda}$.

- $N = 0$: If the queue is empty, a packet arrival will trigger the state transition. Also, τ comes from an exponential distribution with parameter λ.

3) Packet drop: After an arrival event, the dropping policy could decide to drop packets from the queue. If the dropping policy decides to drop the i^{th} packet, the next state will be $(T_1, ..., T_{i-1}, T_{i+1}, ..., T_N)$.

In this paper, we explore two different metrics. First, we explore the average delay where states are defined as their queue length. Then, we explore the deadline case with the extensive states definition. Further exploration of different metrics, policy families, and states can be interesting for future works.

III. Average Delay

This section will look into policies that try to minimize the average delay. We will first consider policies that drop packets from the tail of the queue. After that, we will allow packets to be dropped from anywhere in the queue.

A. Drop from tail

The conventional method of packet-dropping policies involves dropping packets from the end of the queue, meaning that packets are discarded before they enter the queue.

We can show these policies by $d = (d_0, d_1, ..., d_i, ...)$ where d_i is the probability of dropping a newly arrived packet when the system is at state i (i.e., the queue length is i).

These policies result in the queue having this nice property: "Any packet added to the queue will be served."

We denote $p = (p_0, p_1, .., p_i, ...)$ to be the probability of the system being at state i.

Lemma 2. *Given a dropping policy d, if the queue is stable, p can be computed as [6]:*

$$p_j = \frac{\rho^j \prod_{k=0}^{j-1}(1-d_k)}{1+\sum_{t=1}^{\infty}\rho^t \prod_{k=0}^{t-1}(1-d_k)} \tag{7}$$

Proof. The Markov Chain associated with this queue is a Birth-death process where $\lambda_i = \lambda(1-d_i)$ and $\mu_i = \mu$. Thus, using the formula for the stationary distribution of such processes, we get:

$$p_0 = \frac{1}{1+\sum_{t=1}^{\infty}\rho^t \prod_{k=0}^{t-1}(1-d_k)} \tag{8}$$

and,

$$p_j = \left(\prod_{k=0}^{j-1}\frac{\lambda(1-d_k)}{\mu}\right)p_0 = \frac{\rho^j \prod_{k=0}^{j-1}(1-d_k)}{1+\sum_{t=1}^{\infty}\rho^t \prod_{k=0}^{t-1}(1-d_k)} \tag{9}$$

□

As every packet entering the queue is served, the average delay is equal to the average number of packets in the system at each time (expected length of the queue).

Therefore, we can formulate the optimization problem as follows:

$$\begin{aligned} \min_{d} \quad & \sum_{j=0}^{\infty} j p_j \\ \text{s.t.} \quad & \sum_{j=0}^{\infty} d_j p_j \leq \epsilon \\ & 0 \leq d_j \leq 1, \quad \forall j = 0, 1, ... \end{aligned} \tag{10}$$

Where the objective function is the expected length of the queue and the constraint is the probability of packet drop (It is computed using the law of total probability by conditioning on the length of the queue). The following theorem provides the optimal policy from this family.

Theorem 1. *The optimal d should have one of these two forms:*

(i) d_i is equal to 0 for $i < K$, and $d_K = 1$. (M/M/1/K queue)
(ii) d_i is equal to 0 for $i < K$, $d_K \in (0, 1)$, and $d_{K+1} = 1$. (A policy between M/M/1/K and M/M/1/(K+1))

Note that K here is the minimum number that doesn't violate the dropping constraint.

Proof. First, we derive the Lagrange multiplier for this optimization problem:

$$\begin{aligned} L(d, \mu, \alpha, \beta) = & \sum_{j=0}^{\infty} j p_j + \mu\left(\sum_{j=0}^{\infty} d_j p_j - \epsilon\right) \\ & - d^T \alpha - (\mathbf{1} - d)^T \beta \end{aligned} \tag{11}$$

Let d^* be a locally optimal policy, the KKT conditions would be:

$$\frac{\partial L(d^*, \mu, \alpha, \beta)}{\partial d_i} = 0, \qquad \forall i \tag{12}$$

$$\sum_{j=0}^{\infty} d_j^* p_j^* \leq \epsilon \tag{13}$$

$$0 \leq d^* \leq 1 \tag{14}$$

$$\alpha, \beta, \mu \geq 0 \tag{15}$$

$$\alpha_i d_i^* = 0, \qquad \forall i \tag{16}$$

$$\beta_i (d_i^* - 1) = 0, \qquad \forall i \tag{17}$$

before continuing with the proof, we first compute $\frac{\partial p_j}{\partial d_i}$. For simplicity, define $S_i = \sum_{t=0}^{i} p_t$. We have two conditions:

- $j \leq i$:

$$\begin{aligned} \frac{\partial p_j}{\partial d_i} &= \frac{(\rho^j \prod_{k=0}^{j-1}(1-d_k))(\sum_{t=i+1}^{\infty}\rho^t \prod_{k=0}^{t-1}(1-d_k))}{(1-d_i)(1+\sum_{t=1}^{\infty}\rho^t \prod_{k=0}^{t-1}(1-d_k))^2} \\ &= \frac{1}{1-d_i} p_j \sum_{t=i+1}^{\infty} p_t = \frac{p_j(1-S_i)}{(1-d_i)} \end{aligned} \tag{18}$$

- $j > i$:

$$\frac{\partial p_j}{\partial d_i} = -\frac{p_j S_i}{1-d_i} \tag{19}$$

Using the equations above we have:

$$\sum_j j\frac{\partial p_j}{\partial d_i} = \sum_{j=0}^{i} j\frac{p_j(1-S_i)}{1-d_i} - \sum_{j=i+1}^{\infty} j\frac{p_j S_i}{1-d_i}$$
$$= \frac{1}{1-d_i}(\sum_{j=0}^{i} jp_j - S_i\sum_{j=0}^{\infty} jp_j) \qquad (20)$$

And:

$$\sum_j d_j\frac{\partial p_j}{\partial d_i} = \sum_{j=0}^{i} d_j\frac{p_j(1-S_i)}{1-d_i} - \sum_{j=i+1}^{\infty} d_j\frac{p_j S_i}{1-d_i}$$
$$= \frac{1}{1-d_i}(\sum_{j=0}^{i} d_j p_j - S_i\sum_{j=0}^{\infty} d_j p_j) \qquad (21)$$

Now to compute $\frac{\partial L(d^*,\mu,\alpha,\beta)}{\partial d_i}$:

$$\frac{\partial L(d^*,\mu,\alpha,\beta)}{\partial d_i} = \frac{\partial}{\partial d_i}\sum_j jp_j^* + \mu\frac{\partial}{\partial d_i}\sum_j d_j^* p_j^* - \alpha_i + \beta_i = 0 \qquad (22)$$

We have:

$$\alpha_i - \beta_i = \frac{1}{1-d_i^*}(\sum_{j=0}^{i} jp_j^* - S_i^*\sum_{j=0}^{\infty} jp_j^*)$$
$$+\mu(p_i^* + \frac{1}{1-d_i^*}(\sum_{j=0}^{i} d_j^* p_j^* - S_i^*\sum_{j=0}^{\infty} d_j^* p_j^*)) \qquad (23)$$

Now, we define a few new variables:

$$\gamma_i^* = \sum_{j=0}^{i} jp_j^*, \gamma^* = \gamma_\infty^*, \epsilon_i^* = \sum_{j=0}^{i} d_j^* p_j^*, \epsilon^* = \epsilon_\infty^* \qquad (24)$$

We can rewrite equation 23 as follows:

$$\mu(p_i^*(1-d_i^*) + \epsilon_i^* - S_i^*\epsilon^*) = (1-d_i^*)(\alpha_i - \beta_i) + S_i^*\gamma^* - \gamma_i^* \qquad (25)$$

Define $A_i = \mu(p_i^*(1-d_i^*) + \epsilon_i^* - S_i^*\epsilon^*)$ and $B_i = S_i^*\gamma^* - \gamma_i^*$. We have the following properties.

- $A_{i+1} - A_i \leq 0$.
- $B_{i+1} - B_i = p_{i+1}^*(\gamma^* - i - 1)$ which is non-decreasing at first and non-increasing after.
- $A_0 \geq B_0 \geq 0$ and $A_\infty = B_\infty = 0$.
- If $1 > d_i^* > 0$, then $B_i = A_i$, and if $d_i^* = 0$ then $B_i \leq A_i$.

Using the properties above, it is easy to see, that d^* can only be of the forms described in theorem 1. □

Thus, the optimal policy is pretty simple and widely used.

B. *Drop from anywhere*

Now, we look into policies that allow packets to be dropped after entering the queue. In this formulation, a packet that enters the queue is not necessarily served. Implementing some of these policies results in a "leaky queue". In a leaky queue, some of the packets are dropped after entering the queue. Such queues don't have the simple delay distribution of the M/M/1 queue. [12]. These policies potentially could have a lower average delay than drop-the-tail policies. As a simple example, consider an M/M/1/K queue where we drop packets from the head instead of the tail. A simple analysis shows that this policy has a lower average delay than the regular M/M/1/K queue.

We can define these policies by $d = ((d_{0,1}), (d_{1,1}, d_{1,2}), ..., (d_{i,0}, ..., d_{i,i+1}), ..)$, where $d_{i,j}$ is the probability of dropping j^{th} packet in the queue when the system is at state i (i.e., the queue length is i) and a new packet arrives ($d_{i,i+1}$ denote the probability of dropping the newly arrived packet).

Remark. *Drop-the-tail policies can be described as*

$$\{d : d_{i,j} = 0, j \leq i\} \qquad (26)$$

Theorem 2. *Dropping from the head is always the best dropping policy (it minimizes the average delay).*

Proof. Let d be an optimal dropping policy. We define d' to be the dropping policy that drops packets exactly like d, except for one drop:

"When d decides to drop a packet from anywhere other than the head of the queue for the first time, d' drops the packet from the head of the queue."

As the system is memoryless, both dropping policies have the same dropping probability (If $d \in D_\epsilon$, then we would also have $d' \in D_\epsilon$).

Define G to be the time all packets spend in the queue and H as the time dropped packets spend in the queue. Also, let N be the number of packets served.

$$\mathbf{E}[T_d] = \frac{\mathbf{E}[G_d] - \mathbf{E}[H_d]}{N} \qquad (27)$$

As the system is memoryless (the decision of which packet to drop doesn't affect the next state of the system), $E[G_d] = E[G_{d'}]$.

As for the dropped packets, the only difference is the packet dropped differently. As the time spent in the queue is non-increasing from the head to the tail, $E[H_d] \leq E[H_{d'}]$. Therefore, $E[T_d'] \leq E[T_d]$.

Repeating the following procedure will lead to an optimal policy that only drops from the head of the queue. □

Finally, we present this conjecture without proof.

Conjecture 1. *The optimal d should have one of these two forms:*

(i) $d_{i,j}$ *is equal to 0 for* $i < K$, *and* $d_{K,1} = 1$. *(M/M/1/K queue drop from the head)*

(ii) $d_{i,j}$ *is equal to 0 for* $i < K$, $d_{K,1} \in (0,1)$, *and* $d_{K+1,1} = 1$. *(A policy between M/M/1/K and M/M/1/(K+1) drop from the head)*

IV. Percentage of Packages Meeting the Deadline

In this section, we explore the deadline problem. We assume packets have a deadline that they should meet. In this problem, we don't have a limited dropping budget, but the percentage of packets meeting the deadline is computed from all of the packets arriving at the queue (not just the ones served.) The packet should be delivered by that deadline otherwise, it is not useful. Furthermore, we assume that all packets have the same hard deadline of D. When deciding to drop a packet, two factors should be considered:

(a) The probability of the packet making the deadline: In the case of M/M/1 queue, if the packet is at the i^{th} position in the queue, the probability of the packet making the deadline is $F_{Erlang}(D - T_i; i, \mu)$.

(b) The effect of the packet dropping on other packets' chance to make the deadline: By dropping a packet, the packets arriving after the packet dropped will have a better chance of meeting the deadline.

In the next part, we will discuss a special case to further explore the trade-off between these factors.

A. Special case: M/M/1/K queue

We consider a special case of the problem. We have an M/M/1/K queue and we want to decide which packet to drop when the queue is full. In other words, deciding to drop a packet has two components.

- When to drop a packet
- Which packet to drop

In this case, the first component is enforced by the physical constraints of the system and the policy can only affect the second component.

Remark. *The probability of the i^{th} packet making the deadline is equal to $F_{Erlang}(D - T_i; i, \mu)$, Where F is the CDF function.*

Proof. Define X_j to be the service time of packet j. We know that $X_1, ..., X_i$ are i.i.d exponential random variables with parameter μ. Let, $S_i = \sum_{j=1}^{i} X_j$. Then S_i is an Erlang random variable with parameters i and μ. Also, the time that packet i spends in the queue is equal to $S_i + T_i$. Thus, the probability of packet i making the deadline is equal to $P(S_i + T_i < D)$. We have:

$$P(S_i < D - T_i) = F_{Erlang}(D - T_i; i, \mu) \tag{28}$$

□

To decide which packet to drop, we will define the gain of dropping the i^{th} packet in state s as:

$$gain_i^s(\mu) = \sum_{j=i+1}^{K+1} (F_{Erl}(D - T_j; j-1, \mu) - F_{Erl}(D - T_j; j, \mu)) \\ -F_{Erl}(D - T_i; i, \mu) \tag{29}$$

The first part computes the increase in the probability of the packets further back in the queue making the deadline after dropping packet i. The second part is the probability of packet i making the deadline (which is lost when packet i is dropped).

Remark. *Note that gain is a myopic concept as it is only concerned with the packets that are already in the system and doesn't take into consideration the effect of future arrivals or future packet drops.*

We propose a policy that will drop the packet with maximum gain. The following theorem helps formulate such a policy.

Theorem 3. *$gain_1^s(\mu)$ is the maximum gain.*

Proof. We will compute $gain_i^s(\mu) - gain_{i+1}^s(\mu)$:

$$\begin{aligned} & gain_i^s(\mu) - gain_{i+1}^s(\mu) \\ &= F_{Erl}(D - T_{i+1}; i, \mu) - F_{Erl}(D - T_{i+1}; i, \mu) \\ &+ F_{Erl}(D - T_{i+1}; i+1, \mu) - F_{Erl}(D - T_i; i, \mu) \\ &= F_{Erl}(D - T_{i+1}; i, \mu) - F_{Erl}(D - T_i; i, \mu) \end{aligned} \tag{30}$$

Now as $T_{i+1} \leq T_i$ and CDF is a non-decreasing function. We have $gain_i^s(\mu) - gain_{i+1}^s(\mu) \geq 0$. Thus, we get that $gain_1^s(\mu)$ is the maximum.

□

Thus, the policy that will drop the packet with the maximum gain is the same as the policy that drops from the head of the queue every time.

Using the theorem above, we will propose a new policy.

B. Drop Positive Gain Policy (DPGP)

Now, note that theorem 3 is true for the general M/M/1 queue. Assume we don't have a constraint on the dropping budget, i.e. we only care about the number of packets making the deadline. We propose the following policy:

DPGP: "Drop packets from the head of the queue if and only if $gain_1^s(\mu)$ is positive."

Results in section V show that DPGP outperforms conventional queue management techniques.

In the next part, we will extend DPGP to some different scenarios.

C. Extensions of DPGP

This approach (DPGP) can handle different scenarios by changing the definition of $gain_i^s$ to fit the scenario. We present some of those possible scenarios here.

1) Different deadlines: Assume that packets have different deadlines (we show the deadline of the i^{th} packet in the queue by D_i.) Now, the gain function can be changed to:

$$gain_i^s(\mu) = \sum_{j=i+1}^{K+1} (F_{Erl}(D_j - T_j; j-1, \mu) - F_{Erl}(D_j - T_j; j, \mu)) \\ -F_{Erl}(D_i - T_i; i, \mu) \tag{31}$$

2) Different rewards: Assume that packets have different rewards (we define the reward of the i^{th} packet in the queue by r_i.) and we want to maximize the accumulative reward of the packets meeting the deadline. We derive the gain function as:

$$gain_i^s(\mu) = \sum_{j=i+1}^{K+1} r_j(F_{Erl}(D - T_j; j-1, \mu) - F_{Erl}(D - T_j; j, \mu)) \\ -r_i F_{Erl}(D_i - T_i; i, \mu) \quad (32)$$

Remark. *This gain function computes the change in the expected reward of packets meeting the deadline at each time by dropping packet i.*

3) Delay-based utility function: Suppose that instead of a hard deadline, there exists a utility function u that maps the delay of a packet to a reward. We want to maximize the expected utility of packets.

Remark. *The standard hard deadline case can be expressed as follows:*

$$u(T) = \begin{cases} 1, & \text{if } T < D \\ 0, & \text{otherwise} \end{cases} \quad (33)$$

The gain function, in this case, will be:

$$gain_i^s(\mu) = -\mathbf{E}_{S_i \sim Erl(i,\mu)}[u(T_i + S_i)] \quad (34)$$
$$+ \sum_{j=i+1}^{K+1} \Big(\mathbf{E}_{S_j \sim Erl(j-1,\mu)}[u(T_j + S_j)] - \mathbf{E}_{S_j \sim Erl(j,\mu)}[u(T_j + S_j)] \Big)$$

The DPGP is still the same policy but we look at all i and drop them if their gain (with a new definition of gain for each situation) is positive.

We have no theoretical results for this extension of policies but they perform well in experiments as seen in section V.

V. Numerical results and experiments

In this section, we will present some numerical results and simulation results.

A. Simulations for section IV-A

We have an M/M/1/K queue and we want to decide which packet to drop when the queue is full. We consider 4 different policies:

1) Drop from the tail: This is the standard M/M/1/K queue where packets are dropped before entering the queue.
2) Drop from the head: This policy drops packets from the head of the queue.
3) Drop the least probable: This policy drops the packet with the least probability of making the deadline.
4) Drop the biggest gain: This policy drops the packet with the largest gain.

Figure 1 shows the percentage of packets making the deadline across different policies for different deadlines. (Also $\lambda = 0.99$, $\mu = 1$, and $K = 5$)

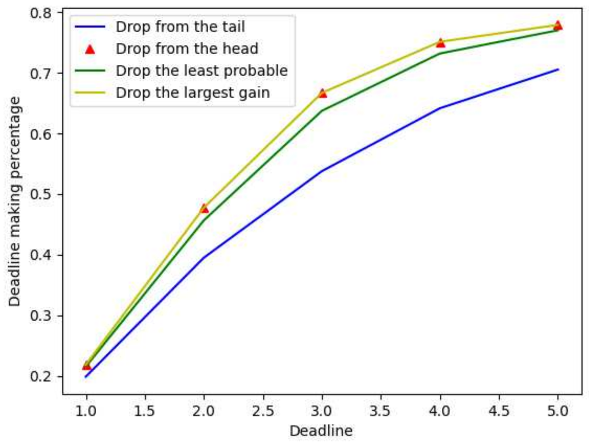

Fig. 1. performance of different policies for the M/M/1/K problem

Remark. *Drop First and Drop best perform exactly the same which aligns with theorem 3. Also, for all deadlines, they have the best percentage of packets making the deadline.*

B. Simulations for DPGP

Here, we compare the DPGP with 3 other policies:

- M/M/1/3 queue that drops from the head of the queue when full.
- M/M/1/5 queue that drops from the head of the queue when full.
- M/M/1/10 queue that drops from the head of the queue when full.

In figure 2, we see that the DPGP outperforms all other policies. ($\lambda = 0.99$, and $\mu = 1$)

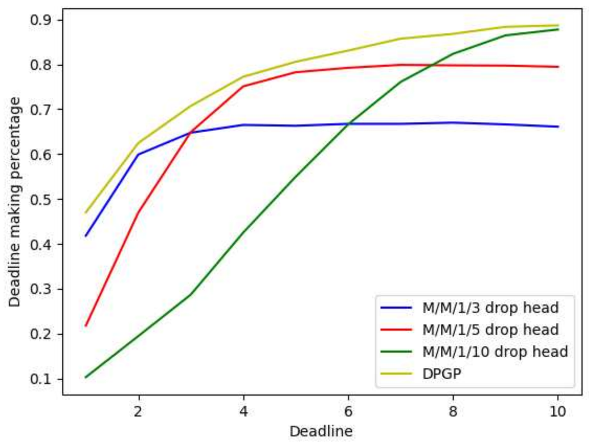

Fig. 2. DPGP performance compared to M/M/1/K with different K

C. Extensions of DPGP

We will consider a case where each packet has a reward that comes from a uniform distribution from 0 to 2. We implement the extension of DPGP for different rewards cases and compare it to the 3 policies described in the previous part. We also, compare with M/M/1/K queues where the packet with minimum reward is dropped. Figure 3 shows that the extension of DPGP outperforms all other policies. ($\lambda = 0.99$, and $\mu = 1$)

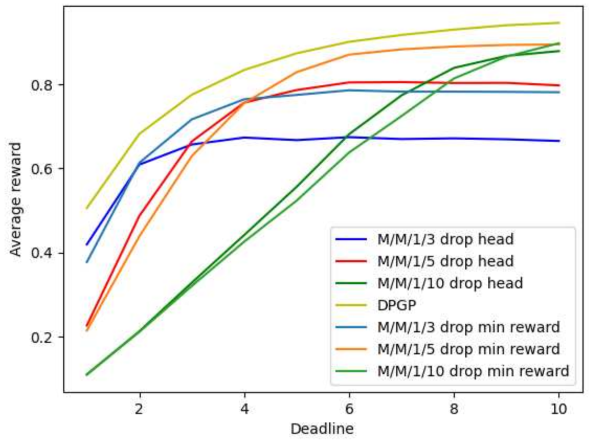

Fig. 3. Expected reward of different policies

VI. Conclusion

In this paper, we looked into M/M/1 queue with packet drop and we tried to optimize delay metrics by finding smart dropping policies. First, our analysis of the average delay metric confirmed that dropping packets from the head of the queue is the optimal strategy. This approach minimizes the delays experienced by subsequent packets, thereby improving the overall efficiency of the system.

Furthermore, we addressed the challenge of meeting packet deadlines by introducing the concept of gain. We proposed the Drop Positive Gain Policy (DPGP). Our evaluation demonstrates that DPGP increases the percentage of packets meeting their deadlines, leading to enhanced system performance. The numerical examples provided in this paper validates the effectiveness of the DPGP policy. In comparison to conventional policies, DPGP consistently outperforms in terms of meeting packet deadlines and improving overall system performance, independent of the deadline.

One line of future work includes finding a farsighted policy for the deadline metric. Also, further exploration of different metrics, policy families, and states can be of interest for different applications.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” *Commun. ACM*, vol. 53, p. 50–58, apr 2010.

[2] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. Nikolopoulos, “Challenges and opportunities in edge computing,” 11 2016.

[3] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An overview on edge computing research,” *IEEE Access*, vol. 8, pp. 85714–85728, 2020.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] R. Adams, “Active queue management: A survey,” *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013.

[6] A. Chydzinski, M. Barczyk, and D. Samociuk, “The single-server queue with the dropping function and infinite buffer,” *Mathematical Problems in Engineering*, 2018.

[7] A. Chydzinski and P. Mrozowski, “Queues with dropping functions and general arrival processes,” *PLOS ONE*, vol. 11, pp. 1–23, 03 2016.

[8] T. Bonald, M. May, and J.-C. Bolot, “Analytic evaluation of red performance,” in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, pp. 1415–1424 vol.3, 2000.

[9] W. Hao and Y. Wei, *An Extended GIX/M/1/N Queueing Model for Evaluating the Performance of AQM Algorithms with Aggregate Traffic*, vol. 3619, pp. 395–404. 09 2005.

[10] W. M. Kempa, “On main characteristics of the $m/m/1/n$ queue with single and batch arrivals and the queue size controlled by aqm algorithms,” *Kybernetika*, vol. 47, no. 6, pp. 930–943, 2011.

[11] A. Chydziński and Łukasz Chróst, “Analysis of aqm queues with queue size based packet dropping,” *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 3, pp. 567–577, 2011.

[12] A. Tang, J. Wang, and S. Low, “Understanding choke: throughput and spatial characteristics,” *IEEE/ACM Transactions on Networking*, vol. 12, no. 4, pp. 694–707, 2004.

[13] S. Abbasloo and H. J. Chao, “Bounding queue delay in cellular networks to support ultra-low latency applications,” *ArXiv*, vol. abs/1908.00953, 2019.

[14] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, “Better never than late: Meeting deadlines in datacenter networks,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM ’11, (New York, NY, USA), p. 50–61, Association for Computing Machinery, 2011.

[15] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM ’10, (New York, NY, USA), p. 63–74, Association for Computing Machinery, 2010.

[16] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter tcp (d2tcp),” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM ’12, (New York, NY, USA), p. 115–126, Association for Computing Machinery, 2012.