Online Distributed Queue Length Estimation*

Aditya Bhaskara† Sreenivas Gollapudi ‡ Sungjin Im § Kostas Kollias †
Kamesh Munagala¶

Abstract. Queue length monitoring is a commonly arising problem in numerous applications such as queue management systems, scheduling, and traffic monitoring. Motivated by such applications, we formulate a queue monitoring problem, where there is a FIFO queue with arbitrary arrivals and departures, and a server needs to monitor the length of a queue by using *decentralized* pings from packets in the queue. Packets can send pings informing the server about the number of packets *ahead* of them in the queue. Via novel online policies and lower bounds, we tightly characterize the trade-off between the number of pings sent and the accuracy of the server's real time estimates. Our work studies the trade-off under various arrival and departure processes, including constant-rate, Poisson, and adversarial processes.

1 Introduction FIFO queues are a ubiquitous modeling tool in various applications such as job scheduling, packet routing, buffering, service request handling, etc. In this paper, we consider FIFO queues that consist of packets representing *distributed agents*. This is a natural way to model scenarios like cars moving on a congested road segment or waiting at a traffic light, customers lining up for their turn at a service counter, and so on. We aim to study the problem of monitoring the status of such a queue, in particular, keeping track of the length of the queue as the agents (who we will call *packets*) enter and leave the system. We consider the setting where the central server aims to maintain a real time estimate of the queue length, using *pings* received from the packets. Our goal is to understand the following questions: What are the tradeoffs between the number of pings sent to the server (i.e., total communication) and the accuracy of the estimates? What *policy* should individual packets follow in order to achieve the desired bounds?

Motivation. Queue length is commonly used to estimate a client's waiting time in the line for service. Tracking the length is highly important for various reasons. It can be used by the service provider for effective resource management and also by the clients to choose the best among multiple queues available to them. If the queue length can be monitored by a dedicated sensor, the problem is trivial. However, it becomes a significant challenge when the monitoring relies on packets that cannot observe the global queue length and are controlled by distributed and independent agents.

This situation arises particularly in traffic monitoring [20, 10, 16, 25, 8, 21]. Monitoring the congestion on various road segments in real time is a central component of routing systems (whose goal is to find the best route for a user). The monitoring server (which we will simply refer to as the server) receives information about the "congestion" on a segment from cars on the segment. These cars, in turn, use *local* information such as their speed, location, etc. to determine what to communicate to the server and when to do so. The natural measure of congestion is the total number of cars on the segment.

We need to consider both the *pinging policy* followed by the individual cars (or devices located in cars), as well as the server-side *reconstruction of the congestion*. We have two main design goals. The first one is the ping frequency. Pinging very frequently may allow the server to better estimate congestion, but it also drains the battery life on GPS devices, which are typically cellphones, and may compromise user privacy. The second goal is to have decentralized one-way pings. In other words, each device acts in isolation without coordinating pings with either the server or with other devices. This is motivated by issues of trust and privacy, as well as the cost of communication.

For designing policies achieving the goals above, we must first consider what information is available to the distributed devices. In the traffic setting, a device in a car has very little global information about the full

*The full version of the paper can be accessed at https://arxiv.org/abs/2504.18503

†School of Computing University of Utah (bhaskaraaditya@gmail.com). Supported by NSF awards CCF-2008688 and CCF-2047288.

‡Google Research, Mountain View, CA (sgollapu@google.com, kostaskollias@google.com).

§University of California, Santa Cruz, CA (sim9@ucsc.edu). Supported in part by NSF grants CCF-1844939, CCF-2121745, and CCF-2423106, and ONR grant N00014-22-1-2701.

¶Computer Science Department, Duke University (munagala@duke.edu). Supported by NSF grant CCF-2113798 and IIS-2402823.

road segment. It has access to the current position and speed of the car, but these are quantities that are only affected by the cars *ahead*, not cars behind. Detailed discussion on how the number of cars ahead can be inferred can be found in the related work for the Car Following Model (Section 1.3). Likewise, in the setting of people waiting at a service line, a person may know how many people are ahead, but not the total length of the line. Motivated by these goals and observations, we formulate the following *queue monitoring* problem.

Queue Monitoring Problem. Suppose we have a FIFO queue to which packets arrive and depart at discrete time steps according to arbitrary arrival and departure processes. A *server* needs to continuously monitor the queue length, which we use as a proxy for congestion in the queue. At any time instant, any packet can send a *ping* to the server notifying it about the number of packets *ahead* of it in the queue (along with any other information it sees fit). The system is decentralized and communication is one-way, so the server cannot send information to any specific packet, nor can the packets communicate with each other. The goal in this setting is to design a *pinging policy* for the packets that uses only locally available information, and a *recovery procedure* for the server that lets it use the pings received so far to maintain an estimate of the queue length at any point of time.

We assume that at any time, a packet in the queue knows how many packets are ahead of it, but has no further information. Thus, a pinging decision is made based on this information collected so far. On the packet side, the goal is to minimize the rate of pings. On the server side for recovery, our error measure is the average (over time) of the difference between the prediction and the true number of packets in the queue, i.e., the difference between the predicted and the true queue length.

The queue monitoring problem is one instance of a more general class of problems where we have distributed agents that each use myopic and local information to send information to a server that aims to keep track of a global state. Our algorithms and lower bounds can thus provide templates to reason about other such scenarios.

1.1 Our Results Our main contribution is the design and analysis of simple yet novel pinging policies for the queue monitoring problem. Our policies adaptively regulate their ping probability based on locally available information, and we show that the server can indeed accurately reconstruct the queue length. In our policies, a packet (at any time) sends a ping with a probability that is a function $g(h, w)$, where h is the number of packets *ahead* of it, which we term its height, and w is its waiting time, the time elapsed since it entered the queue. We further establish the optimality of the functional forms $g(h, w)$ that appear in our algorithms.

We develop two types of pinging policies: the first is termed Poa (Section 3), wherein a packet only pings upon arrival into the queue, and the second is termed Pico (Section 5) where the packet pings continuously, at an adapting rate. The former has the advantage that a packet sends at most one ping, and does not need to keep track of any other state until it exits the queue. However, Poa is insufficient for worst case arrival and departure processes, and we need continuous pinging. The details of our results are as follows.

Ping on Arrival. As described above, in Poa policies, packets send pings only on arrival. Due to the limitation on the information available to any packet, (distributed) Poa policies have a very simple structure: if a packet arrives and observes the queue to be of height h, it sends a ping with probability $f(h)$. Note that the observed height is equal to the queue length for Poa policies. We analyze Poa policies in Section 3. We show that despite seeming restrictive, Poa policies can yield non-trivial results. When departures occur at a constant rate while arrivals are adversarial, the optimal pinging policy belongs to this class. We then develop a Poa policy that achieves reconstruction error ϵ (defined formally in Section 2) times the average queue length, using $f(h) = \tilde{O}\left(\min\left(1, \frac{1}{\epsilon h}\right)\right)$, where the $\tilde{O}$ hides terms depending on $\ln \frac{1}{\epsilon}$. We prove the optimality by showing in Section 4 that any monotone pinging function $f(h)$ needs to be at least this value for Poa policies that achieve ϵ reconstruction error.

We further show that our upper bound for the Poa policy with constant-rate departures easily implies the same guarantee when departures follow a Poisson process ($G/M/1$ queueing model [11]), while arrivals can be arbitrary. We also consider the 'opposite' case, where arrivals are constant-rate and the departure process is adversarial, and extend this to Poisson arrivals. These extensions are deferred to the full version of this paper. For all these settings, we not only derive similar guarantees for the Poa policy, but also show matching lower bounds (in Section 4), thus justifying the functional forms of our Poa policies.

Ping Continuously. The second type of policies we consider (in Section 5) allow packets to ping at any time in which they are in the queue. We show that such policies are needed (and Poa policies are insufficient) when both departures and arrivals either follow more bursty stochastic processes than Poisson, or are entirely adversarial. In such a policy, that we term Pico, each packet pings every time step with probability $g(h, w)$,

where h is its height and w is its waiting time (the number of time steps elapsed) since its arrival. We show that for any adversarial arrival *and* departure processes, when $g(h,w) = O\left(\frac{\ln(1/\epsilon)}{\epsilon^2}\frac{1}{hw}\right)$, then the reconstruction error in queue length is at most ϵ. We finally show that the number of pings generated by this policy for a packet arriving at height h is $O\left(\frac{\log h}{\epsilon}\right)$ factor more than that of the Poa policy when either arrivals or departures occur at a constant rate, showing there is not much room for improvement in the functional form.

Beyond Monitoring a Single Queue. Although we focus on monitoring a single queue, we remark that this is not too restrictive. We note that our results yield guarantees for networks of queues via known results. In this case, the reconstruction error is the sum of the errors in the queue heights, and we can bound this error by ϵ times the sum of the queue lengths. From our results on the Poa policy for Poisson departures , the same guarantee holds for a Jackson network of queues where all departure processes are Poisson [11]. Similarly, the Pico policy in Section 5 extends to a network of queues with arbitrary arrival and departure processes.

1.2 Algorithmic Intuition and Technical Contributions

To build intuition, first assume that packets arrive, but do not depart. In this case it is natural to get the queue length from the latest arriving packet whenever the queue length increases by a certain factor. This can be simulated by the Poa policy that pings with a probability inverse to the height, h. However, it is unclear if this policy would continue to work when packets can depart the system, since the arrival and departure processes can interleave in a complicated way. We show that Poa policies do work when one of the arrival or departure processes is somewhat predictable (constant-rate or Poisson), while the other process is adversarial.

For further intuition behind our Poa policy, consider the setting where packets depart at a constant rate or according to a Poisson distribution. Since job departure is predictable, the only possible scenario for the server to be significantly outdated with the current queue height occurs when a large number of packets arrive within a short period of time. In this case, Poa is designed so that at least one packet sends a ping to the server with a good probability before the height changes by a factor of more than $1+\epsilon$.

However, the analysis is challenging. For each packet i, we measure the lag δ_i of it as the expected time it takes for the server to become aware of it by receiving a ping from it or its subsequent packet (packet i is counted in the height of a subsequent packet upon arrival if i is still present). We carefully charge $\mathbb{E}\sum_i \delta_i$ to an ϵ fraction of the total delay. Here, we combine ideas from competitive analysis, queueing theory, and random processes.

When the arrival and departure processes are both adversarial, we need an entirely different policy. With a Poa policy, if a large number of packets depart instantly and few new packets arrive, the server will remain clueless of this change. We argue that we have to allow packets to ping intermittently. The basic intuition behind our second algorithm Pico is to view each packet i's height h at each time t as a point (t,h). Suppose we are only interested in estimating the total delay of packets over all times. This delay is just the number of all points. We can now let this point (t,h) add weight $h \cdot w$ to the delay estimate with a probability $\frac{1}{hw}$, where w is the packet's waiting time. This will give an unbiased estimator of total delay.

However, the goal of the server is to accurately interpolate the true delay, meaning that we should estimate the number of packets present in the queue at *each* time. To do this, we consider the rectangle associated with the point, which has width w and height h. That is, this rectangle R has four vertices: $(t-w,h)$, (t,h), $(t-w,0)$, and $(t,0)$. Note that this rectangle lies below the height profile curve as a function of time because the number of packets ahead of packet i can only decrese in time. Thus, the union of such rectangles lies entirely below the profile curve. Further, the Pico ping rate is designed so that this union is close to the profile curve.

Then, the server projects each rectangle encoded by a ping forward in time and increases its height—the magnified rectangle has four vertices: $(t-w,h)$, $(t+3\epsilon w,(1+3\epsilon)h)$, $(t-w,0)$, and $(t+3\epsilon w,0)$. The server uses the union of these magnified rectangles as its estimate. The analysis shows that this union almost completely covers the profile curve while being only slightly larger than the union of the original rectangles.

1.3 Related Work Online Buffer Management.

There are several online models for queue management that are motivated by networking applications; this includes dynamic TCP acknowledgment [15, 6], buffer reordering [3, 14], and packet scheduling in switches [2]. However, in the TCP acknowledgment problem (and more generally buffer management), the entity sending acks is centralized and has complete information about arrivals so far. In contrast to this model, our setting is decentralized and each pinging vehicle (packet) has incomplete information about both the total delay and which other vehicles are simultaneously sending pings. Therefore, the decentralized aspect in our problem makes the typical benchmark of a

centralized offline optimum used in competitive analysis too strong. Part of our innovation therefore is in coming up with the right analytic framework to show the optimality of our policies.

Stochastic Queue Management. Similarly, several recent works in queueing theory [28, 22] have considered the problem of stochastic load balancing of multiple queues when the assigner gets limited feedback from the queues about their load. The focus of this work is to derive policies for sending feedback under which the scheduling policy is *stable*. In contrast with this work, our motivation comes from routing applications, where giving the end-user accurate feedback about delay on their route is crucial. This makes our goal different: we seek to *monitor* the length (or height) of the queue via feedback, and this is interesting even in single-queue setting where load-balancing is a trivial problem. Further, at a technical level, the performance of our policies crucially depend on the *nature* of the arrival and departure processes, and merely assuming ergodicity is insufficient. (See Section 4 for lower bounds for general ergodic processes.)

Energy-efficient Functional Monitoring. The general problem of *distributed functional monitoring* [7, 24, 19] has received considerable attention in the context of energy efficiency in sensor networks. In this setting, a set of distributed low-power sensor nodes receive an arbitrary sequence of items over time. These nodes need to communicate as few bits of information as possible to the server (to save energy), so that the server can maintain a good running approximation to some statistic over all the items, such as frequency moments, heavy hitters, etc. This model generalizes the classical streaming model [1]. In the distributed monitoring model, nearly tight bounds on the trade-off between communication and approximation are known for several basic statistics, even when item insertions are adversarial. However, when deletions are allowed, the problem becomes hard. Our problem can be viewed as functional monitoring, where the function is the delay in the queue. The key difference is that the vehicles are sensors, and they arrive and depart the system instead of being fixed. Further, these observations are correlated with each other and across time via traffic dynamics. This makes our problem technically different.

Delay Estimation for Traffic Networks. We briefly review relevant work in traffic monitoring. Existing work in this area largely focuses on using sparse and noisy GPS traces to accurately estimate historical traffic, real-time traffic or future traffic [13, 12, 9]. Solutions for estimating route-specific ETAs have been proposed using ML models based on physical world features such as weather conditions, vehicle type, current network congestion, etc. [26, 23]. The work most closely related to ours is [27]. This work assume a uniformly random subset of the vehicles are "probe" vehicles that continuously update the server about queue lengths ahead of them (*i.e.*, their stop positions at intersections). They develop various statistical estimators for the actual queue lengths based on the positions of these probe vehicles. In contrast, we assume vehicles cannot continuously update the server, and this leads to the natural question of adaptively minimizing the number of probes needed to generate good estimates of the delays.

Car Following Model. Since we assume that a packet can infer the number of packets ahead of it in our model, we explicitly discuss why this is a reasonable assumption in the contexts of traffic congestion. A vehicle can infer the number of vehicles (or congestion) ahead of it as follows. If the vehicle is equipped with technology that enables it to calculate following distance (say via radar) and its exact location, it can set the local estimate of congestion as ℓ/δ, where ℓ is the distance to the head of the segment, and δ is the following distance. If the quantity δ is not directly available, this can be inferred with a simple and well-known model called the *simplified car following* model of Newell *et al.* [17]. In this model, the segment is assumed to have a uniform speed v, and given this speed, the vehicles have an average "following distance" s given by $s = d + \tau v$, where the parameters d and τ can be learnt from historical data. These two parameters are average intrinsic properties of drivers, for instance τ represents the time needed to safely brake if the previous car suddenly stops. Suppose that a vehicle knows its speed v and its location ℓ relative to the head of the segment, then it can estimate the number of vehicles ahead of it as $h = \frac{\ell}{d+\tau v}$. Alternatively, the mapping from speed v to h can be learnt (say per segment) from historical data and the resulting ML model can be used by the vehicle. Note that since a car only knows its location and speed (and not about future arrivals into the segment), it can only infer the number of vehicles ahead of it in the segment, motivating our assumption.

2 The Queue Monitoring Problem

We consider a FIFO queue into which packets arrive and depart. Packets arrive at the tail of the queue and depart at the head. Let a_i denote the arrival time of packet i. Departures can be viewed as follows: There is a stream of "departure tokens"; when a departure token is generated, if there is at least one packet in the queue, the packet at the head of the queue departs. Let d_i denote the departure time of packet i. The *delay* experienced by this packet (i.e., the time spent in the queue) is thus

$d_i - a_i$. Let $\mathcal{A}$ denote the set of arrival times $\{a_1, a_2, \ldots\}$ and $\mathcal{D}$ denote the set of departure times $\{d_1, d_2, \ldots, \}$. We will assume throughout that time is discrete, and thus a_i, d_i are all integers. There is a horizon of T steps at the end of which all packets have departed.

At time t, we denote the number of packets in the queue as the "height" h_t. We have the following equality between the time-averaged delay and average height (which we call OPT).

$$\text{OPT} := \frac{1}{T}\sum_{i \in S}(d_i - a_i) = \frac{1}{T}\sum_{t=0}^{T} h_t. \tag{2.1}$$

This holds because the sum on the LHS measures the number of time steps packet i is in the queue, and at each such step, it contributes a value of 1 to the height, which is the RHS sum.

Queue monitoring policy. Packets in the queue send pings to a central monitoring server (henceforth called the server) using a pinging algorithm $\mathcal{P}$, and the goal of the server is to keep track of h_t at every time step t, using an estimation algorithm $\mathcal{E}$. Thus we define queue monitoring policy as the pair $(\mathcal{P}, \mathcal{E})$. We assume that the algorithm $\mathcal{P}$ is fully decentralized, and there is no communication between packets.

Next, we have the main twist that we motivated earlier: each packet in the queue only knows the number of packets ahead of it in the queue. The packet does not know of the existence of later arrivals into the queue, and in particular, it need not know the current height h_t of the queue. We let h_{it} denote the "position" of packet i in the queue (i.e., number of packets ahead of it including itself) at time t, for $t \in [a_i, d_i]$. Our key assumption is that the pinging algorithm $\mathcal{P}$ (which decides the probability of sending a ping and the information to include in each ping) depends only on i's *information set* at time t, which is the set of $h_{it'}$ values at all times $a_i \le t' \le t$.[1]

On the server side, the estimation algorithm $\mathcal{E}$ must use the pings received from the packets to maintain a real-time estimate of the queue height. We denote the estimate at time t by e_t. Note that the server does not have direct access to the queue and estimates only using the pings.

We denote by ALG the time-average error in the server's estimate of queue height. Since the algorithms $\mathcal{P}$ and $\mathcal{E}$ can be randomized, we formally define it as an expected value:

$$\text{ALG} = \frac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T} |h_t - e_t|\right]. \tag{2.2}$$

Design goals. Our aim is to design a pinging policy such that (i) the total number of pings is small, and (ii) the average error ALG in the server's estimate of queue height is small. Formally, let $\epsilon \in (0, 1)$ be a small constant that is given as an error parameter. Our goal is to design a pinging policy with ϵ reconstruction error, which we define as:

$$\text{ALG} \le \epsilon \cdot \text{OPT} + c \tag{2.3}$$

for some absolute constant c (recall that OPT was defined in (2.1)). For this guarantee to be meaningful, we assume that either the arrival or departure process is sufficiently variable, so that $\text{OPT} = \omega\left(\frac{1}{\epsilon}\right)$. The question we ask is: *how many pings are necessary and sufficient for obtaining such a guarantee?*

Roadmap. We will first focus on a very simple class of policies where each packet only pings on arrival (if at all). We term these POA policies, as discussed earlier. Section 3 studies POA policy when the departure is constant-rate while the arrival is adversarial. As mentioned before, the full version of this paper shows extension of this result to the Poisson arrival and to the opposite case where the arrival process is constant-rate or Poisson. In Section 4 we show that POA policies are essentially optimal under these restrictions on departures or arrivals, while they are insufficient for general arrival and departure processes. For the latter setting, we present a continuous pinging policy PICO and analyze its performance in Section 5.

3 POA Policy for the Constant-rate Departure Case

We first consider the constant-rate departure setting, where the arrival process is completely arbitrary but the departure process generates one token per step.[2] Therefore, in each time step, if the queue is non-empty, one packet departs the system. An arbitrary number of packets can arrive at any time step; all these packets will have the same arrival time, but we will assume an arbitrary but fixed ordering among these packets. For our results, we can view the arrivals as being determined by an adversary, albeit one who is oblivious to the randomness of the pinging policy (i.e., does not know which packets have pinged so far).

[1] While this might sound restrictive, most natural policies, e.g., pinging when the 'speed' changes can be captured here (e.g., by looking at the rate of change of the h values).

[2] This is without loss of generality by scaling the arrival and departure rates by the same factor. Further, the total number of packets generated by POA remains unchanged regardless of the departure rate. As mentioned before, we only assume $\text{OPT} = \omega(1/\epsilon)$ to keep the setting interesting.

3.1 Optimality of POA Policies for Constant-rate Departures We begin with the observation that for constant-rate departures, if packet i sends a ping at time t with its current height h_{it}, the server knows that its height at all times $t' \geq t$ is $\max\{h_{it} - (t' - t), 0\}$. In fact, the server also knows the packet's height at previous time steps, assuming the packet has entered the queue. Furthermore, since a packet can only look ahead into the queue, the information set of this packet (defined in Section 2) is the same at time $t = a_i$ (arrival time) and at all subsequent times. This immediately yields the following structural property.

LEMMA 3.1. *For any pinging policy for constant-rate departures, an equivalent policy (in terms of the expected number of pings and the reconstruction error) has packets only sending pings on arrival.*

More formally, for any pinging algorithm $\mathcal{P}$, we can construct an algorithm $\mathcal{P}'$ in which when packet i arrives, we compute the total probability that $\mathcal{P}$ will send a ping (in the future; we can do this since the packet's information set at all times is known), and send a ping at arrival with that probability. The server will have at least the same amount of information as in the case of $\mathcal{P}$.

The lemma implies that any policy can be characterized by a single function $f : \mathbb{N} \mapsto [0, 1]$. If a packet arrives and finds the height of the queue is h, it sends a ping to the server with probability $f(h)$, else it never sends a ping. This shows that the optimal policy belongs to the class of POA policies.

3.2 The POA Policy and its Analysis We begin with a simple POA policy and analyze its performance. The policy is as follows.

- Ping algorithm $\mathcal{P}$ (packet side; run independently for each packet): Let h be the queue height after the packet arrives. Then it sends a ping with probability $f(h) := \min\left(1, \frac{2\ln(1/\epsilon)}{\epsilon h}\right)$. The ping consists of the arrival time and the height h.
- Estimation algorithm $\mathcal{E}$ (server side; run at each time step): At time t, if the last packet received was at time t' with height h, the server's estimate $e_t = \max(0, h-(t'-t))$. In other words, the server's estimation assumes there were no arrivals since the last ping.

In the rest of this section, we will show the following theorem. It turns out to achieve an essentially optimal number of pings; we show a matching lower bound in Section 4.

THEOREM 3.2. *Consider the queue monitoring problem over a time interval $[0, T]$ where packet arrivals $\mathcal{A}$ are arbitrary and departures are contant-rate at a unit rate. Assuming the queue heights at times 0 and T are zero, the POA policy described above achieves the bound:* $\mathbb{E}[\text{ALG}] \leq 37\epsilon \cdot \text{OPT}$.

We begin the proof by setting up notation. Assume that the arriving packets $\mathcal{A}$ are numbered $1, 2, \ldots, N$, and that the arrival times a_i are in non-decreasing order. (For convenience, even if packets arrive in the same slot, assume that the order is respected, as is natural in an application like traffic.) We denote the height of the queue just *after* the arrival of packet i but before the arrival of packet $i + 1$ by $h(a_i)$. As before, let T denote the time horizon at which all packets have departed.

Because of the unit departure rate assumption, the packet stays in the queue for exactly $h(a_i)$ steps. Therefore, the quantity OPT (defined as the time average of the height) satisfies:

$$\text{OPT} = \frac{1}{T}\sum_{t=1}^{T} h_t = \frac{1}{T}\sum_{i\in\mathcal{A}} h(a_i).$$

Since we have a unit departure rate, the update satisfies the property that $e_t \leq h_t$ at all points of time. Thus the error term of interest is therefore

$$\text{ALG} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[(h_t - e_t)]. \tag{3.1}$$

Accounting for ALG via packet lags. Define the *lag* of packet i, denoted δ_i, as the additional expected time it takes for the server to become "aware" of packet i. Formally, if packet i arrives at time t and j is the first packet $\geq i$ to send a ping to the server, then δ_i would be $\min(h(a_i), a_j - a_i)$.[3] Note that if i itself sends a ping, this ensures $\delta_i = 0$.

With this notation, we now observe that a packet's contribution to the numerator of ALG (Eq (3.1)) is exactly equal to δ_i. This is because the quantity $(h_t - e_t)$ can be viewed as the number of packets that the server is unaware of at time t. Combining these formulas, we therefore have

$$\frac{\text{ALG}}{\text{OPT}} = \frac{\sum_{i\in\mathcal{A}} \delta_i}{\sum_{i\in\mathcal{A}} h(a_i)}. \tag{3.2}$$

Note that the denominator is deterministic and depends solely on the packet arrival and departures,

[3] To be precise, the minimum must also include the term $(T - a_i)$ for handling the packets arriving in the last few time steps. This is a mild technicality that we will ignore here since T is be assumed to be sufficiently large.

while the numerator is an expectation that depends on the pinging algorithm. The main theorem is now the following, which when combined with Eq (3.2) implies Theorem 3.2.

THEOREM 3.3. *The pinging strategy given by f satisfies $\sum_i \delta_i \leq 37\epsilon \cdot \sum_i h(a_i)$.*

3.2.1 Proof of Theorem 3.3 The rest of this section is devoted to proving Theorem 3.3. The idea of the proof is to consider the expression for δ_i and show that it is small if sufficiently many packets arrive "soon after" packet i. There can be packets for which this may not happen, and for these packets, we show how to charge the δ_i values to the sum of the heights (or the area under the height curve) in a certain time interval, which is the contribution of this interval to the delay OPT.

Let us focus on packet i, arriving at time a_i. We assume that $h(a_i) > \frac{2\log(1/\epsilon)}{\epsilon}$, as otherwise the packet sends a ping on arrival with probability 1, and thus $\delta_i = 0$. Let us now write an expression for δ_i. Define $p(i,j)$ to be the probability that $(i+j)$ is the first packet $\geq i$ to send a ping. Then by definition, we have

$$\delta_i = \sum_{j \geq 0} p(i,j) \min\{(a_{i+j} - a_i), h(a_i)\}. \tag{3.3}$$

(The sum thus runs over all the packets in the interval $[1,T]$ that arrive after i.) Next, from the choice of our ping probability, the intuition is that one of the packets in the set $\{i, i+1, \ldots, i+\epsilon h(a_i)\}$ will send a ping. Let $S_i = \{0, 1, \ldots, \epsilon h(a_i)\}$ capture the relevant indices. Now consider the prefix of δ_i as $\gamma_i = \sum_{j \in S_i} p(i,j) \min\{(a_{i+j} - a_i), h(a_i)\}$.

LEMMA 3.4. *For any packet i, we have $\delta_i \leq \gamma_i + \epsilon\, h(a_i)$.*

Proof. Consider the packets $i+j$, for $j \in S_i$. For all these packets, we have that the ping probability (no matter when they arrive) is at least $\frac{2\log(1/\epsilon)}{\epsilon(1+\epsilon)h} > \frac{\log(1/\epsilon)}{\epsilon h}$, where $h = h(a_i)$. This is because $h(a_{i+j}) \leq (1+\epsilon)h$ for these packets.

Since each packet $i+j$ for $j \in S_i$ sends a ping with probability at least $\frac{\log(1/\epsilon)}{\epsilon h}$, we have:

$$\sum_{j > \epsilon h} p(i,j) \leq \left(1 - \frac{\log(1/\epsilon)}{\epsilon h}\right)^{\epsilon h} \leq \epsilon$$

Therefore $\sum_{j>\epsilon h} p(i,j) \min\{(a_{i+j} - a_i), h(a_i)\} \leq \epsilon h(a_i)$. This completes the proof. ∎

Combining with Equation (3.2), we have

$$\frac{\text{ALG}}{\text{OPT}} = \frac{\sum_i \delta_i}{\sum_i h(a_i)} \leq \frac{\sum_i \gamma_i}{\sum_i h(a_i)} + \epsilon. \tag{3.4}$$

Therefore, to show the theorem, we need to bound $\sum_i \gamma_i$. We do this as follows. We make a sweep over the packets. At every step, we maintain a "leftover" (or unaccounted) γ mass, which is the sum of γ_j values for a certain range of $j < i$. We then charge the sum of this and the γ_j values for all packets that arrive in the interval $[a_i, t_i]$, where t_i is an appropriately chosen time. For interval $[t_1, t_2]$, denote the area under the height curve (or total delay) as: $A[t_1, t_2] = \sum_{t=t_1}^{t_2} h_t$. We will charge the total leftover γ to $A[a_i, t_i]$.

Formally, U denotes the unaccounted γ mass so far. Define the following:

DEFINITION 3.1.

$$t_i = \min\left\{\frac{h(a_i)}{3}, a_{i+\epsilon h(a_i)} - a_i\right\},$$

and let $r_i \in S_i$ denote the largest index s.t. $a_{i+r_i} \leq a_i + t_i$.

Algorithm 3.1 The CHARGING Analysis

```
Set i ← 0; U ← 0; k ← 0.
while i ≤ N do
  Compute t_i and r_i as in Definition 3.1.
  if γ_j ≤ 3γ_i for all j ∈ [i, i + r_i] then
    U ← U + Σ_{j=i}^{i+r_i} γ_j; and Ã ← A[a_i, t_i].
    Charge U to Ã.
    Set U ← 0; and i ← i + r_i + 1.
  else
    ℓ ← Smallest index with γ_ℓ > 3γ_i and ℓ ≤ i + r_i.
    U ← U + Σ_{j=i}^{ℓ−1} γ_j; and i ← ℓ.
  end if
end while
```

Our charging scheme is presented in Algorithm 3.1. Here, N is the total number of packets. Our key lemma is the following:

LEMMA 3.5. *Whenever U is charged to $\tilde{A}$ in the above scheme, we have $U \leq 36\epsilon\tilde{A}$.*

Proof. Consider any iteration of the while loop of Algorithm 3.1, and let i be the index of the packet at the start of the loop. We will show that the following invariant is always maintained at the start of the loop:

Invariant. $U \leq 3\epsilon h(a_i)\gamma_i$.

Denote $h = h(a_i)$ and $r = r_i$. First observe that $t_i \geq \gamma_i/3$. To see this, denote $\tilde{t}_i = a_{i+\epsilon h} - a_i$. Note that $\gamma_i \leq \min(h, \tilde{t}_i)$. If $\tilde{t}_i \leq h/3$, then $t_i = \tilde{t}_i \geq \gamma_i$. Otherwise, $t_i = h/3 \geq 3\gamma_i$.

We now consider the two cases dictated by the if statement in Algorithm 3.1.

Case 1: For all $j \in [i, i+r]$, $\gamma_j \leq 3\gamma_i$. In this case,

noting that $r \le \epsilon h$, we have

$$\sum_{j \in [i,i+r]} \gamma_j \le r(3\gamma_i) \le 3\epsilon h \gamma_i.$$

Since the U at the start of the loop is $3\epsilon h\gamma_i$ by the invariant, the final U is at most $6\epsilon h\gamma_i$. The area $\tilde{A} = A[a_i, t_i]$ is at least $\frac{1}{2}ht_i$. Since $\gamma_i \le 3t_i$, the area is always at least $\frac{1}{6}h\gamma_i$. Therefore, $U \le 36\epsilon\tilde{A}$. The invariant is trivially satisfied for the next iteration.

Case 2: There exists $\ell \in [i, i+r]$ such that $\gamma_\ell > 3\gamma_i$.

In this case, we only need to prove that our invariant is maintained when the index moves to ℓ. To see this, note that the initial U is at most $3\epsilon h\gamma_i$ by the invariant. Therefore, final U satisfies

$$U \le 3\epsilon h\gamma_i + \sum_{j \in [i,\ell)} \gamma_i \le 6\epsilon h\gamma_i,$$

where we use the fact that $\ell - i \le r < \epsilon h$ and that ℓ is the first index with $\gamma_\ell > 3\gamma_i$. Thus the total unaccounted mass U at index ℓ is less than $6\epsilon h\gamma_i$. We argue that this $U \le 3\epsilon\gamma_\ell h(a_\ell)$. Using the fact that $\gamma_\ell > 3\gamma_i$, it suffices to prove that $h(a_\ell) > \frac{2}{3}h$. Since $a_{i+r} - a_i \le h/3$, there are at most $h/3$ departures during this period (by the constant-rate departure assumption), so that $h(a_\ell) \ge \frac{2}{3}h$. This proves the invariant holds at the next iteration, and completes the proof of the lemma. ∎

Since the $\tilde{A}$ in different iterations correspond to areas for disjoint time intervals and since the total U is the same as $\sum_{i \in \mathcal{A}} \gamma_i$, we have: $\sum_{i \in \mathcal{A}} \gamma_i \le 36\epsilon \sum_t h_t$. Since the RHS is equal to $\sum_i h(a_i)$, we combine with Eq (3.4) to complete the proof of Theorem 3.3.

4 Optimality Results and Lower Bounds for Poa Policies

We will now present almost matching lower bounds on the number of pings needed to achieve ϵ reconstruction error (Eq (2.3)) for the constant-rate departure and constant-rate arrival settings, showing the optimality of the Poa class of policies. We will also show that Poa policies stop being optimal when we further generalize the arrival or departure processes, motivating the Pico class of policies considered in Section 5. All proofs are deferred to the full version of this paper.

Constant-rate Departures. We first consider the setting from Section 3. We know from Lemma 3.1 that the optimal policy is a Poa policy. We have the following lemma whose bound almost matches the ping probability of the Poa policy we analyzed in Section 3.

Lemma 4.1. *For $\epsilon < \frac{1}{8}$, let f be a ping probability function in a* Poa *policy that achieves $O(\epsilon)$ reconstruction error. Then for any $h = \Omega\left(\frac{1}{\epsilon}\right)$, we must have $\sum_{j=h}^{(1+8\epsilon)h} f(j) \ge \frac{1}{6}$. Therefore, for $\epsilon < \frac{1}{8}$ and any monotone f that achieves ϵ reconstruction error, we have $f(h) = \Omega\left(\min\left(1, \frac{1}{\epsilon h}\right)\right)$.*

Constant-rate Arrivals. In this setting, the optimal policy need not be a Poa policy. Nevertheless, in Lemma 4.2, we show for certain types of canonical instances where the queue height is almost surely close to a given value h, no policy can do much better than the Poa policy, that is, the expected number of pings cannot be improved by more than a constant factor for these instances.

Given any height h, for any instance, define $N_h = \{i \mid h_{ia_i} \in [h(1-8\epsilon), h]\}$ as the number of arrivals (or time steps) with queue size in $[h(1-8\epsilon), h]$. We show a lower bound for the number of pings required by *any* pinging policy below.

Lemma 4.2. *Fix any $\epsilon \in (0, 1/16)$. For any given value $h = \Omega\left(\frac{1}{\epsilon}\right)$, there exist instances with $N_h = N - o(N)$ (where N is the total number of arrivals), such that any pinging policy that achieves $O(\epsilon)$ reconstruction error sends at least $\frac{1}{48\epsilon h}$ pings per packet on average.*

The above lemma shows that no policy (not just Poa policies) can improve on the expected number of pings when the queue height stays close to any value h. As a corollary, it shows that any Poa policy that achieves ϵ reconstruction error requires $f(h) = \Omega\left(\frac{1}{\epsilon h}\right)$ on the instances from Lemma 4.2. The last bound matches the upper bound of the Poa policy.

Insufficiency of Poa for General Stochastic Arrivals or Departures. As our final lower bound, we show that when we consider stochastic processes that are more bursty than Poisson, Poa policies are insufficient to achieve ϵ-reconstruction error. This motivates the Pico class of policies in Section 5.

Lemma 4.3. *There exist instances with i.i.d. arrival and departure processes and a constant $\epsilon > 0$ such that no* Poa *policy can achieve ϵ-reconstruction error.*

5 General Arrivals and Departures: The Pico Policy

In this section, we consider the most general case when both the arrivals and departures are arbitrary processes, either stochastic or adversarial. Lemma 4.3 shows that Poa policies are insufficient for this case, and we need a novel set of policies. Below, we present more nuanced examples that show that even pinging on both arrival and departures is insufficient. These examples make a case for policies that ping *continuously*, albeit at a rate that is continuously adjusted based on their current queue height and waiting time.

Justification for Continuous Pinging Policies. One might wonder about policies that ping on departure

in addition to arrival. However, this can either lead to large reconstruction error or a large number of pings.

EXAMPLE 5.1. *Initially, an unknown number h of packets arrive into the system. Now consider two scenarios: In the first scenario, all packets depart in one step, while in the latter scenario, all but one packets depart, and the remaining packet stays for h steps.*

It is an easy exercise to show that the server cannot achieve ϵ-reconstruction error without distinguishing the two scenarios. We will show that if packets ping only on arrival or departure, then all packets must ping on arrival. First note that only pings from the last packet can help the server with distinguishing the two scenarios. Further, a ping on departure is useless, since the server needs to maintain an estimate for the h steps till this packet departs (in the second scenario), and will hence incur a large error by the time this ping is received. Thus, the last packet must ping on arrival if pings are made only on arrival or departure. Finally, it remains the case that a packet must ping even if it is not the last packet, as it does not see packets arriving after it. Therefore, we cannot achieve ϵ-reconstruction error unless every packet pings on arrival.

The next example goes a step further and makes a case for continuous pinging.

EXAMPLE 5.2. *Initially, an unknown number h packets arrive into the system and the i-th arriving packet departs in either 2^i or 2^{i+1} time steps.*

Suppose the i-th packet is the last packet. Then, the server must know its existence because otherwise it doesn't know if the queue is empty or not for at least 2^i steps after the first $i-1$ packets depart. The resulting error of 2^i is unacceptable as the total time the first $i-1$ packets stay in the queue is also $\Theta(2^i)$. In this case the server can't learn about the i-th packet without a ping from it, and just as in the previous example, this ping cannot be on departure. Finally, it remains the case that it must ping even if it is not the last, as it doesn't see packets arriving after it. This motivates the need for each packet to ping continuously.

5.1 The PICO Policy We now present our *ping continuously* or PICO policy. Let $w_{it} := t - a_i$ denote packet i's waiting time in the queue at time t. We define h_{it} to be the number of packets ahead of packet i at time t. We make the mild assumption that the minimum waiting time is 1 unit (i.e., a packet cannot arrive and leave the same time instant).

- Ping algorithm $\mathcal{P}$ (packet side): At each time $t \in (a_i, d_i]$, packet i sends a ping with probability
$$g(h_{it}, w_{it}) = \frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{w_{it}h_{it}}. \tag{5.1}$$
The ping consists of its waiting time w_{it} and its current height h_{it}.
- Estimation algorithm $\mathcal{E}$ (server side): At each time t, the server does the following step:
  - Let P_t denote the set of packets paired with the times when they sent a ping so far. That is, $(i, t') \in P_t$ means that packet i sent a ping at time $t' \le t$.
  - For each $(i, t') \in P_t$ the server sets its estimated height $e_{tit'}$ as:
$$e_{tit'} = \begin{cases} h_{it'}(1+3\epsilon) & \text{if } t \le t' + 3\epsilon w_{it'} \\ 0 & \text{otherwise} \end{cases}$$
  - The server's estimate of height is $e_t = \max_{(i,t')\in P_t} e_{tit'}$.

Roughly speaking, if the server receives a ping from a packet with height h and waiting time w, it guesses that the height is $\ge h$ for $3\epsilon w$ steps in the future. If the queue suddenly empties, the error will not be too large as it can be 'charged' to the previous waiting cost using the fact that the packet waited for w steps and had a height $\ge h$ all that time. We also illustrate the policy via an example:

EXAMPLE 5.3. *Suppose h packets arrive in the system at time 0, and they all depart at time 1. Suppose at time 0, a subset of the packets sends a ping. Let h' denote the largest height of any packet that sends a ping. Note that the waiting time is 1. Then the server maintains an estimate of $h'(1+3\epsilon)$ till time $1+3\epsilon$, and subsequently sets the estimate to zero.*

5.2 Algorithmic Intuitions and Analysis Overview Before jumping into the analysis, we give more intuitions behind the policy and how they are naturally tied to the analysis. Recall that our goal is to approximately reconstruct the time-varying height profile $\{h_t\}$, $t \in [T]$ from pings. We can visualize it by having a stack of h_t points at time t where yth point from the bottom represents the yth oldest one among packets alive at time t. Let (t, y) denote the point and $j(t, y)$ the corresponding packet. For brevity let $i = j(t, y)$. We know that the packet i has waited for w_{it} steps since its arrival and it had at least h_{it} packets (including itself) ahead of it all the time. It means that the rectangle of width w_{it} and height h_{it} having (t, y) as its upper and right corner must lie inside the profile. Let $\mathcal{L}_{it}$ denote the rectangle.

Rectangles $\mathcal{L}_{it}$ have several important properties. First, rectangle $\mathcal{L}_{it}$ only depends on packet $j(t, y)$'s waiting time and height at time t. Thus, the packet

can communicate the rectangle to the server without consulting with other packets and this is what a ping encodes conceptually. Second, the union of all rectangles $\mathcal{L}_{it}$ is exactly equal to the time-varying height profile.

But, there are are some issues. Communicating one rectangle means one ping. Thus, to ping parsimoniously, a rectangle is communicated with probability inversely proportional to its area. Another issue is that each rectangle $\mathcal{L}_{it}$ only contains past information. Thus, the server projects the rectangle into the future by stretching it by a small factor. The crux of the analysis lies in showing the height profile is approximately sandwiched by the union of communicated rectangles $\mathcal{L}_{it}$ and the union of their future projections. See Figure 5.1 for visualization of this discussion.

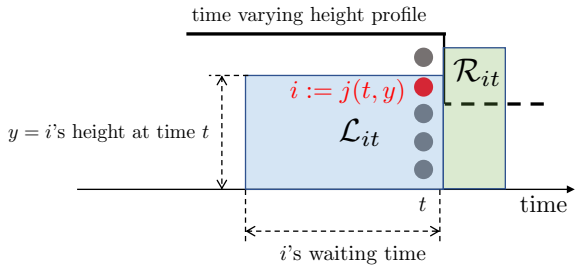

Figure 5.1: Visualization of PICO. Suppose the packet $i = j(t, y)$ of height y pings at current time t. The packet has had height at least y since it arrival and therefore the rectangle $\mathcal{L}_{it}$, colored blue, of height h and width equal to i's waiting time at time t fits under the time varying height profile. The server projects $\mathcal{L}_{it}$ into the future. The projection $\mathcal{R}_{it}$, colored green, is a vertical and horizontal extension of $\mathcal{L}_{it}$. The server's estimation is the union of the extended rectangles.

5.3 Analysis Our main result about the PICO policy is the following.

THEOREM 5.1. *Consider the queue monitoring problem over a time interval $[0, T]$ where packet arrivals and departures are arbitrary. Assuming the queue heights at the initial and final time steps are zero, for any $\epsilon \in (0, 1/5)$, the* PICO *policy described above achieves the bound* $\mathbb{E}[\text{ALG}] \le 10\epsilon \cdot \text{OPT}$.

Recall the definition of ALG from Eq (2.2). Now consider the following geometric view of the pinging process. The x-axis represents time, and the y-axis represents queue height. If the height of the queue at integer time t is h_t, any point (t', y) for $y \le h_t$ and $t' \in (t-1, t]$ is marked as full. The full points – denote them $\mathcal{H}$ – in this 2-D space capture the height profile of the queue, and we call this the *height diagram*. We let $j(t, y)$ denote the packet corresponding to point $(t, y) \in \mathcal{H}$, which is the packet that has height y at time t.

Now, when the server gets a ping from packet $j(t, y)$ at time t with height y and waiting time $w(t, y) := w_{j(t,y),t}$, it creates a rectangle $\mathcal{R}_{ty} := [t, t+3\epsilon \cdot w(t, y)] \times [0, (1+3\epsilon)y]$. For brevity, we may say (t, y) has waiting time $w(t, y)$ and height y, instead of saying $j(t, y)$ does. Similarly, we may say (t, y) pings when $j(p, y)$ does. Let

$$\mathcal{R}_t = \bigcup_{y, t' \le t} \mathcal{R}_{t'y}$$

denote the union of rectangles the server has created until time t. The server uses the height of $\mathcal{R}_t$ at the current time t as the queue height estimate e_t at the time. Note that since any rectangle constructed at time t only covers current and future time steps, $\mathcal{R}_t$ is not affected by pings, arrivals, or departures at future time steps.

The analysis consists of two parts, bounding the under-estimation error and the over-estimation error. Formally we will show $\mathbb{E}[\text{AREA}(\mathcal{H} \setminus \mathcal{R}_T)] \le \epsilon \text{AREA}(\mathcal{H})$ (Corollary 5.3) and $\text{AREA}(\mathcal{R}_T \setminus \mathcal{H}) \le 9\epsilon \text{AREA}(\mathcal{H})$ (Lemma 5.9), which will complete the proof of Theorem 5.1. Here, $\text{AREA}(\cdot)$ means the area of what is inside in the parentheses; thus, the total delay is $\text{AREA}(\mathcal{H})$.

5.3.1 Bounding Under-estimation Error We first show the under-estimation error is low. The following key lemma claims that our estimation covers any point in height diagram with large probability.

LEMMA 5.2. *Consider any point (t, y) in $\mathcal{H}$. Then,* $\Pr[(t, y) \notin \mathcal{R}_t] \le \epsilon$.

COROLLARY 5.3. $\mathbb{E}[\text{AREA}(\mathcal{H} \setminus \mathcal{R}_T)] \le \epsilon \text{AREA}(\mathcal{H})$.

The following lemma articulates our high-level proof strategy to show Lemma 5.2.

LEMMA 5.4. *For any $(t, y) \in \mathcal{H}$, there exists a set A of points in $\mathcal{H}$ such that i) at least one point $(t', y') \in A$ pings with probability $1 - \epsilon$ and ii) $(t, y) \in \mathcal{R}_{t'y'}$ for all $(t', y') \in A$.*

In words, if a point in A pings, it adds a rectangle to our estimation that covers (t, y) and at least one point in A pings with large probability. It is easy to see Lemma 5.4 immediately implies Lemma 5.2, and therefore, Corollary 5.3. Thus, the remainder of the section is devoted to proving Lemma 5.4. We fix a point $(t, h) \in \mathcal{H}$ throughout the section.

Finding a desirable A is not straightforward. Note that i) and ii) are somewhat in conflict. To cover more points along with the fixed point (t, y) by rectangles created in the past, intuitively they should be large. But as discussed before, the rectangles are created with

probability inversely proportional to their respective area. Thus, if we want to use large rectangles to cover the point, we will have to use more to guarantee a large success probability. Note that the obvious approach of setting $A := \{(t', y') \mid t' \in [t - \epsilon w(t, y)], y' \in [y - \epsilon y, y]\}$ doesn't work because some points in A may create too large rectangles (particularly if they have a large waiting time) and therefore they are created with too small probabilities.

To find a proper A, we find a special point for each $y' \in [(1-\epsilon)y, y]$.

LEMMA 5.5. *There exists a point $q_{y'} := (\tau_{y'}, y')$ for each $y' \in [(1-\epsilon)y, y]$ such that every point (t', y') with $t' \in [\tau_{y'} - \epsilon w_{y'}, \tau_{y'}]$ has waiting time between $(1-\epsilon)w_{y'}$ and $4w_{y'}$, and $t - \tau_{y'} \leq \frac{1}{3}\epsilon w_{y'}$, where $w_{y'} := w(\tau_{y'}, y')$ denotes the waiting time of $q_{y'}$.*

Proof. Fix a $y' \in [(1-\epsilon)y, y]$. Let $t_1 := t$. To find a desired $q_{y'}$, we will iteratively find a sequence of $p_1 := (t_1 := t, y'), p_2 := (t_2, y'), ..., p_{K-1} := (t_{K-1}, y'), q_{y'} := p_K := (t_K := \tau_{y'}, y')$. Let $w_k := w(t_k, y')$. Let p_{k+1} be any point in $\{(t', y') \mid t' \in [t_k - \epsilon w_k, t_k]\}$ such that $w_{k+1} > 4w_k$, if any; if there's no such point, then we set $K = k$. Clearly, this process must terminate for fixed t, since w_k increases exponentially in k and $w_k \leq t$. Note that $w_k < \frac{1}{4}w_{k+1} < \frac{1}{4^2}w_{k+2} < \ldots < \frac{1}{4^{K-k}}w_K$.

By definition of p_K, any point (t', y') with $t' \in [t_K - \epsilon w_K, t_K]$ has waiting time at most $4w_K$. Further, the packet $j(t_K, y')$ must have height at least y' before time t_K. This implies the packet $j(t', y')$ arrives no later than $j(t_K, y')$. Thus, (t', y') must have waiting time at least $w_K - \epsilon w_K$ as $t' \in [t_K - \epsilon t_K, t_K]$. Knowing $w_{y'} = w_K$ by definition, we have the first claim.

The second claim follows from an easy algebra:

$$\begin{aligned} t - \tau_{y'} &= t_1 - t_K \\ &= \sum_{k=1}^{K-1} (t_k - t_{k+1}) \\ &\leq \epsilon \sum_{k=1}^{K-1} w_k \\ &\leq \epsilon \sum_{k=1}^{K-1} \frac{1}{4^{K-k}} w_K \\ &\leq \frac{1}{3}\epsilon w_K = \frac{1}{3}\epsilon w_{y'}. \end{aligned}$$

∎

Intuitively, $q_{y'}$ is a point that creates a rectangle that covers (t, y) if it pings. Further, all points $\{(t', y') \mid t' \in [\tau_{y'} - \epsilon w_{y'}, \tau_{y'}]\}$ have similar waiting times. Thus, they create rectangles of similar areas that cover (t, y). We include those points in A.

LEMMA 5.6. *Let $A_{y'} := \{(t', y') \mid t' \in [\tau_{y'} - \epsilon w_{y'}, \tau_{y'}]\}$ and $A := \bigcup_{y' \in [(1-\epsilon)y, y]} A_{y'}$. Then, A satisfies the properties stated in Lemma 5.4.*

Proof. Recall that each point $(t', y') \in A_{y'}$ has waiting time at most $4w_{y'}$ (Lemma 5.5). and therefore pings with probability at least $\frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{4w_{y'}y'}$ (Eqn. (5.1)). Thus, the probability that no point in A pings is at most,

$$\begin{aligned} &\prod_{y' \in [(1-\epsilon)y, y]} \prod_{t' \in [\tau_{y'} - \epsilon w_{y'}, \tau_{y'}]} (1 - \frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{4w_{y'}y'}) \\ &\leq \prod_{y' \in [(1-\epsilon)y, y]} \prod_{t' \in [\tau_{y'} - \epsilon w_{y'}, \tau_{y'}]} \exp(-\frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{4w_{y'}y'}) \\ &\leq \prod_{y' \in [(1-\epsilon)y, y]} \exp(-\frac{1.25\ln(1/\epsilon)}{\epsilon}\frac{1}{y}) \\ &= \prod_{y' \in [(1-\epsilon)y, y]} \exp(-\frac{1.25\ln(1/\epsilon)}{\epsilon}\frac{\epsilon y}{y}) \leq \epsilon, \end{aligned}$$

which shows property i).

To show property ii), consider any $(t', y') \in A$. Our goal is to show that (t', y') creates a rectangle covering (t, y) if it pings, i.e., $(t, y) \in \mathcal{R}_{t'y'}$. Note $\mathcal{R}_{t'y'} = [t', t' + 3\epsilon w(t', y')] \times [0, (1 + 3\epsilon)y']$. Clearly we have $t' \leq t$.

$$\begin{aligned} &t' + 3\epsilon w(t', y') - t \\ &\geq \tau_{y'} - \epsilon w_{y'} + 3\epsilon w(t', y') - t \quad \text{[Definition of } A\text{]} \\ &\geq \tau_{y'} - \epsilon w_{y'} + 3\epsilon(1-\epsilon)w_{y'} - t \quad \text{[Lemma 5.5]} \\ &\geq t - \frac{1}{3}\epsilon w_{y'} - \epsilon w_{y'} + 3\epsilon(1-\epsilon)w_{y'} - t \quad \text{[Lemma 5.5]} \\ &\geq 0 \quad \text{[When } \epsilon \in (0, 5/9]\text{]} \end{aligned}$$

Thus, we also have $t \leq t' + 3\epsilon w(t', y')$. Finally, since $y' \in [(1-\epsilon)y, y]$, we have $0 \leq y \leq (1+3\epsilon)y'$. Therefore, we have shown $(t, y) \in \mathcal{R}_{t'y'}$. ∎

5.3.2 Bounding Over-estimation Error We now switch to bounding the over-estimation error. Any $(t, h) \in \mathcal{H}$ corresponds to a packet i that exists in the queue at time t. We therefore overload terminology and denote this as $(i, t) \in \mathcal{H}$. To measure the over-estimation error of $\mathcal{H}$ by $\mathcal{R}_T$, we view $\mathcal{H}$ as the union of $\mathcal{L}_{it} := [t - w_{it}, t] \times [0, h_{it}]$ for $(i, t) \in \mathcal{H}$. The proof of the next lemma immediately follows from the observation that the height of the packet corresponding to (i, t) can only decrease over time.

LEMMA 5.7. *For every $(i, t) \in \mathcal{H}$, we have $\mathcal{L}_{it} \subseteq \mathcal{H}$. Therefore, we have $\bigcup_{(i,t) \in \mathcal{H}} \mathcal{L}_{it} \subseteq \mathcal{R}_T$.*

Now for the sake of analysis, we consider the union of $\mathcal{U}_{it} := [t - w_{it}, t + 3\epsilon w_{it}] \times [0, (1+3\epsilon)h_{it}]$, which is an extension of $\mathcal{L}_{it}$. Note that $\mathcal{U}_{it}$ fully contains both $\mathcal{L}_{it}$ and $\mathcal{R}_{it}$. Therefore, their union is a super-set of $\mathcal{R}_T$, which implies the next lemma.

LEMMA 5.8. $\mathcal{R}_T \subseteq \bigcup_{(i,t)\in\mathcal{H}} \mathcal{U}_{it}$.

We finally bound the over-estimation error by bounding the area of the regions $\mathcal{U}$.

LEMMA 5.9. *For any* $\epsilon \in (0, 1/3)$, $\textsc{Area}(\mathcal{R}_T \setminus \mathcal{H}) \leq 9\epsilon\textsc{Area}(\mathcal{H})$.

Proof. From Lemmas 5.7 and 5.8, it suffices to show that

$$\textsc{Area}\Big(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{U}_{it}\Big) \leq (1+3\epsilon)^2 \textsc{Area}\Big(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{L}_{it}\Big).$$

Let $\mathcal{U}'_{it} := [t - w_{it}, t + 3\epsilon w_{it}] \times [0, h_{it}]$. Since $\mathcal{U}_{it}$ stretches $\mathcal{U}'_{it}$ vertically by a $(1 + 3\epsilon)$ factor keeping the rectangle abutting the x-axis, we clearly have $\textsc{Area}(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{U}_{it}) = (1+3\epsilon)\textsc{Area}(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{U}'_{it})$.

Thus, we only need to show that $\textsc{Area}(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{U}'_{it}) \leq (1 + 3\epsilon)\textsc{Area}(\bigcup_{(i,t)\in\mathcal{H}} \mathcal{L}_{it})$. Note that $\mathcal{U}'_{it}$ is equivalent to a rectangle that results from extending $\mathcal{L}_{it}$ by $(1 + 3\epsilon)$ factor to the right. The claim then follows by considering each height $h' \in [0, h]$: if we stretch the interval at height h' in each $\mathcal{L}_{it}$ by a uniform factor, and the length of their union stretches by the same factor or less. ∎

As discussed, this lemma, combined with Corollary 5.3, proves Theorem 5.1.

5.4 Comparison with POA Policies We finally consider how much worse is the number of pings of this policy compared to the POA policy. We show that the the number of pings is $\tilde{O}\left(\frac{\log h}{\epsilon}\right)$ factor larger for constant-rate arrivals or departures.

LEMMA 5.10. *For constant-rate arrivals or departures, for packets arriving when the queue height is* h*, the number of pings of the* PICO *policy is within* $O\left(\frac{\ln(1/\epsilon)}{\epsilon} \cdot \ln h\right)$ *of the* POA *policy.*

Proof. First consider the constant-rate departure case. Here, a packet arriving at height h sends a ping after t steps with probability $\frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{t(h-t+1)}$. Therefore, the total number of pings sent by this packet (in expectation) is

$$\sum_{t=0}^{h} \frac{5\ln(1/\epsilon)}{\epsilon^2}\frac{1}{t(h-t+1)} = O\left(\frac{\ln(1/\epsilon)}{\epsilon^2}\frac{\ln h}{h}\right) \tag{5.2}$$

This is a factor $O\left(\frac{\ln h}{\epsilon}\right)$ worse than the POA policy that would have pinged with probability $O\left(\frac{\ln(1/\epsilon)}{\epsilon h}\right)$ for this packet. Similarly, for the constant-rate arrival case, suppose the current height of the queue is h. Then a packet with height j arrived at least $h - j$ steps ago, so that its waiting time is at least $h - j + 1$. Therefore, the expected number of pings at this step is again at most the quantity from Eq (5.2). Therefore, the overhead is again a factor of $O\left(\frac{\ln h \ln(1/\epsilon)}{\epsilon}\right)$ more than the POA policy, which would have pinged with probability $O\left(\frac{1}{\epsilon h}\right)$ at this step. ∎

6 Conclusion In this paper, we presented the queue monitoring problem as an abstraction for tracking congestion in traffic applications. We showed decentralized online algorithms that achieve near-optimal trade-offs between the number of pings and the error in queue length. Our work shows that the question of congestion monitoring is non-trivial even for a single queue.

The most important direction for future work is to combine congestion monitoring with algorithms for optimal routing in a network to minimize delay based on this congestion information, much like low-complexity methods for load balancing [28, 22, 4, 5, 18] that have been widely studied in networking and queueing theory. Such algorithms do not follow in any obvious way from our current work, and we leave addressing this as an interesting open question.

Another interesting direction is the following: the POA policy provides an unbiased estimator and is therefore likely robust. In contrast, the PICO policy is not unbiased as it is, and it would be very interesting to develop another policy that is unbiased. Finally, it would be interesting to study the effect of noise in our problem.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137 – 147, 1999.

[2] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for qos switches. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, page 761–770, 2003.

[3] N. Avigdor-Elgrabli and Y. Rabani. An improved competitive algorithm for reordering buffer management. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, page 1–10, USA, 2013. IEEE Computer Society.

[4] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommod-

ity flow. In *Proc. IEEE FOCS*, pages 459–468, 1993.

[5] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. ACM STOC*, page 487–496, 1994.

[6] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2–3):93–263, Feb. 2009.

[7] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. *ACM Trans. Algorithms*, 7(2), 2011.

[8] C. de Fabritiis, R. Ragona, and G. Valenti. Traffic estimation and prediction based on real time floating car data. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pages 197–203, 2008.

[9] B. Deng, S. Denman, V. Zachariadis, and Y. Jin. Estimating traffic delays and network speeds from lowfrequency gps taxis traces for urban transport modelling. *European Journal of Transport and Infrastructure Research*, 15(4), 2015.

[10] A. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2007.

[11] G. Grimmett and D. Stirzaker. *Probability and random processes*. Oxford University Press, 2001.

[12] R. Herring, A. Hofleitner, P. Abbeel, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 929–936, 2010.

[13] T. Hunter, R. Herring, P. Abbeel, and A. Bayen. Path and travel time inference from gps probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs*, 12(1):2, 2009.

[14] S. Im and B. Moseley. New approximations for reordering buffer management. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, page 1093–1111, USA, 2014. Society for Industrial and Applied Mathematics.

[15] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, page 502–509, New York, NY, USA, 2001. Association for Computing Machinery.

[16] W. Min and L. Wynter. Real-time road traffic prediction with spatio-temporal correlations. *Transportation Research Part C: Emerging Technologies*, 19(4):606–616, 2011.

[17] G. F. Newell. A simplified car-following theory: a lower order model. *Transportation Research Part B: Methodological*, 36(3):195–205, 2002.

[18] A. Sankar and Z. Liu. Maximum lifetime routing in wireless ad-hoc networks. In *Proceedings IEEE INFOCOM 2004*, pages 1089–1097. IEEE, 2004.

[19] A. S. Silberstein, R. Braynard, C. Ellis, K. Munagala, and Jun Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 68–68, 2006.

[20] E. Strano, A. Giometto, S. Shai, E. Bertuzzo, P. Mucha, and A. Rinaldo. The scaling structure of the global road network. *R Soc Open Sci.*, 4(10), 2017.

[21] T. T. Tchrakian, B. Basu, and M. O'Mahony. Realtime traffic flow forecasting using spectral analysis. *IEEE Trans. Intell. Transp. Syst.*, 13(2):519–526, 2012.

[22] M. van der Boor, S. C. Borst, J. S. H. van Leeuwaarden, and D. Mukherjee. Scalable load balancing in networked systems: A survey of recent advances, 2018.

[23] Z. Wang, K. Fu, and J. Ye. Learning to estimate the travel time. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 858 – 866, 2018.

[24] D. P. Woodruff and Q. Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 941–960, New York, NY, USA, 2012. Association for Computing Machinery.

[25] F. Yang, Z. Yin, H. Liu, and B. Ran. Online recursive algorithm for short-term traffic prediction. *Transportation Research Record*, 1879(1):1–8, 2004.

[26] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316 – 324, 2011.

[27] Y. Zhao, J. Zheng, W. Wong, X. Wang, Y. Meng, and H. X. Liu. Estimation of queue lengths, probe vehicle penetration rates, and traffic volumes at signalized intersections using probe vehicle trajectories. *Transportation Research Record*, 2673(11):660 – 670, 2019.

[28] X. Zhou, J. Tan, and N. Shroff. Heavy-traffic delay optimality in pull-based load balancing systems: Necessary and sufficient conditions. In *SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, page 5–6, 2019.