EasyQuantile: Efficient Quantile Tracking in the Data Plane

Bo Wang[†], Rongqiang Chen[†], Lu Tang[†]
[†]School of Informatics, Xiamen University

ABSTRACT

Quantile tracking is an essential component of network measurement, where the tracked quantiles of the key performance metrics allow operators to better understand network performance. Given the high network speed and huge volume of traffic, the line-rate packet-processing performance and network visibility of programmable switches make it a trend to track quantiles in the programmable data plane. However, due to the rigorous resource constraints of programmable switches, quantile tracking is required to be both memory and computation efficient to be deployed in the data plane. In this paper, we present EasyQuantile, an efficient quantile tracking approach that has small constant memory usage and involves only hardware-friendly computations. EasyQuantile adopts an adjustable incremental update approach and calculates a pre-specified quantile with high accuracy entirely in the data plane. We implement EasyQuantile on Intel Tofino switches with small resource usage. Trace-driven experiments show that EasyQuantile achieves higher accuracy and lower complexities compared with state-of-the-art approaches.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Network monitoring**;

KEYWORDS

Quantile tracking, Programmable data plane, Performance measurement

1 INTRODUCTION

Quantile is an important statistic to be collected and monitored in network measurement [4, 10, 30]. By tracking the quantile of network performance metrics, such as packet latency [29], link utilization [13], and flow completion time [30], network operators can well understand the variations of network performance over time and across different flows. Based on the collected quantiles, operators can perform instant adjustments to network architecture, congestion control strategies [35], and resource allocation [10] to keep the network stable and reliable.

As an essential component of network measurement, tracking quantile in switch data plane becomes an inevitable trend. Quantile tracking should be able to process packets at line rate to prevent network performance from being impacted by measurement. However, the increasingly high speed network and huge volume of traffic incur unacceptable high cost on CPU-based platforms [8, 16]. On the other hand, tracking the quantile of some performance metrics like queue length requires network visibility that is unobtainable at the end [11, 27]. The emerging programmable switches [34] provide us viable opportunities to deploy quantile tracking entirely in the data plane to achieve both high performance and network visibility. The programmable data plane allows users to deploy customized packet-processing logics while maintaining a high processing rate of few terabits per second.

Although tracking quantiles in programmable switch data plane benefits a lot, it faces challenges due to the resource constraints of programmable hardware [12, 18]. First, the programmable switch has quite limited on-chip memory (e.g., tens of megabytes), which is shared among multiple network functions. Second, the computation capacity of programmable switches is limited, where only simple arithmetic computations (e.g., addition and subtraction) are allowed. The above resource constraints require quantile tracking to be performed in a one-pass manner with both high memory and computational efficiency.

Existing approaches of quantile tracking mainly focus on CPU-based platforms [1, 14, 20, 24–26] and have high computational complexity [3]. For example, SSA [3] involves power computation that is not supported in programmable data plane, while P2 [19] needs to traverse an array of intermediate results more than three times. Although these approaches have small constant memory usage, their high computational complexities prevent them from being deployed in programmable data plane. Some approaches [15, 33, 37]

Lu Tang is the corresponding author.

APNET 2023, June 29–30, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0782-7/23/06…$15.00
https://doi.org/10.1145/3600061.3600084

adopt incremental quantile tracking to achieve low complexities on both memory and computation, yet they incur large errors in quantile estimation and are hard to meet the accuracy requirements of network measurement applications.

We propose EasyQuantile (EZQ), an efficient quantile tracking approach that calculates a pre-specified quantile with high accuracy entirely in the data plane. EZQ adopts an adjustable incremental approach, where it updates the estimated quantile value incrementally using a step value derived from the quantile value's neighboring ranks. To achieve small errors across different quantile settings and traffic distributions, EZQ dynamically adjusts the calculation of the step value between two modes based on the quantile specified. When the quantile is small, EZQ adopts an *average* mode which updates the quantile value more smoothly with a small step. For large quantiles, it turns to a *max-min* mode that performs each update with a larger coarse-grained step. The adjustable incremental update design of EZQ contains only hardware-friendly computations that are well supported in the data plane, making EZQ be easily feasible on hardware.

We have implemented EZQ on Intel Tofino switches using P4 with small hardware resource usage. We evaluate EZQ with both synthetic traces and real-world network traces. The evaluation shows that EZQ achieves better or competitive accuracy (up to 23.58×) compared with state-of-the-art CPU-based approaches [15, 19, 20, 25, 37] and lower memory consumption (up to 71.11×).

2 BACKGROUND AND RELATED WORK

2.1 Quantile tracking

We consider quantile tracking in data streams in this paper and formulate the problem as follows. Table 1 summarizes the notations used in the paper. Let $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ be an incoming data stream, where x_n is the n-th item observed in $\mathbf{X}$. For any $0 \le p \le 1$, the p-quantile of $\mathbf{X}$ is the item q in the stream such that at most $p|\mathbf{X}|$ items of $\mathbf{X}$ are smaller than q and at most $(1-p)|\mathbf{X}|$ elements of $\mathbf{X}$ are greater than q. Our aim is to approximately return the p-quantile of $\mathbf{X}$ as the items in the data stream arrive with small errors.

Quantile tracking is increasingly important in network performance measurement. More and more works for performance measurement utilize quantile as their performance indicator [2, 22, 31, 38], such as 95-percentile latency and flow completion time. PINT and DeltaINT measure the tail and median latency from each hop in the network to detect network events in real-time. HPCC [21] demonstrates the effectiveness of their algorithm by observing the 95-percentile queue length and flow completion time. Chen and Wu [10] use the quantile of the counters of the sketch algorithm to estimate the error of sketch, so that we can allocate the resources required by sketch more accurately.

Notation	Meaning
Defined in Section 2	
$\mathbf{X}$	a data stream
x_n	the n-th item observed in $\mathbf{X}$
p	the quantile of interest, where $0 \le p \le 1$
q	the p-quantile of $\mathbf{X}$
Defined in Section 3	
$\hat{q}_n$	estimated value of the p-quantile
c_n^-	the number of x_n less than $\hat{q}_n$
c_n^+	the number of x_n greater than $\hat{q}_n$
sum_n	sum of $\{x_1, x_2, \cdots, x_n\}$
avg_n	the average value of $\{x_1, x_2, \cdots, x_n\}$
λ_n	step length of each update
T	the threshold of toggling two modes
x_{max}	maximum item of $\{x_1, x_2, \cdots, x_n\}$
x_{min}	minimum item of $\{x_1, x_2, \cdots, x_n\}$

Table 1: Notations used in the paper.

2.2 Programmable data plane

The programmable data plane allows operators to customize their packet processing logic via domain-specific programming languages like P4 [5]. To keep high forwarding speed (e.g., 12.8Tbps [34]) while providing the programmability, the programmable data plane has quite stringent constraints on hardware resources. First, there is limited on-chip memory. For example, Tofino has only around 12 MB of SRAM in total. Second, the capacity of computing is limited, where only few tens of arithmetic-logic units (ALUs) are equipped and each of them supports simple arithmetic computations like addition and subtraction only. Third, the memory access model is rigid, where each memory block can be accessed only once for each packet processed.

The high packet processing rate and programmability of the programmable data plane have spurred a series of efforts works to offload network measurement [11, 18, 39] to the data plane, for high performance and network visibility. As an essential component of network measurement, quantile tracking entirely in the data plane is significantly necessary. However, due to the resource constraints of the programmable data plane, it is non-trivial to achieve that. The goal of this paper is to design an efficient quantile tracking approach that meets all the constraints of the programmable data plane and keeps high accuracy at the same time.

2.3 Existing approaches and limitations

Quantile Estimator Based on Stochastic Approximation (SA). Some quantile estimation methods are predominantly based on stochastic approximation (SA) [28], which is a powerful tool for estimating the parameters using stochastic rules. To improve the efficiency of SA-based methods, Tierney [32] proposes an incremental quantile estimator that involves density estimation, while Chen *et al.* [9] introduce exponentially weighted stochastic approximation (EWSA)

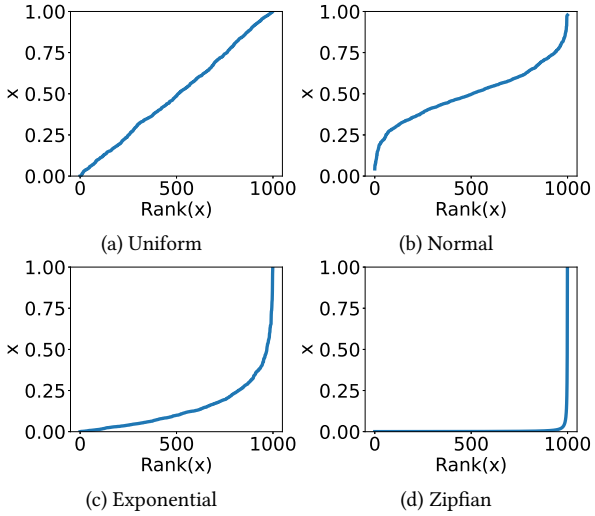
Technique	ADD	MUL	Power /Log	Variables	Parameters
P2 [19]	51	21	-	20	3
SSA [3]	7	15	3	9	4
DUMIQE [37]	6	2	-	5	1
QEWA [15]	15	11	-	8	2
EZQ	**5**	**3**	**-**	**9**	**1**

Table 2: Comparison in the number of operations, variables and parameters.

as an alternative to SA. Another approach based on smooth stochastic approximation that utilizes a gaussian kernel is proposed by Amiri and Thiam [3]. These variants of SA all employ exponential decay for past estimates, enabling the effective tracking of quantiles for non-stationary data stream. However, these methods require a high number of complex calculations, such as exponentiation, which can render them infeasible in the data plane.

Incremental Quantile Estimator. To minimize resource usage, recent researches focus on incremental quantile estimator. Jain and Chlamtac [19] introduce a heuristic algorithm with only five markers, which estimates the quantile by moving the markers and fitting the piecewise-parabolic curve. Frugal [23], a quantile tracking algorithm that uses only one or two units of memory, is groundbreaking in memory requirement. Another work from Yazidi and Hammer [37] suggests deterministic updates where the step size is adjusted in a subtle manner, which is different from [23]. Hammer *et al.* [15] introduce a lightweight quantile estimator QEWA using a generalized form of the exponentially weighted average method, where the update size is proportional to the difference between current observation and the current quantile estimate. While these incremental quantile tracking algorithms offer reduced resource consumption, their implementation introduce significant overhead. Moreover, certain algorithms require parameter adjustments, which lead them infeasible for deployment in complex and dynamic network environments. To illustrate this, we compare the number of operations, variables, and parameters in our proposed approach with those of other algorithms, as shown in Table 2. Compared with these approaches, EZQ implements quantile tracking through less complex operations and parameters.

Quantile Sketch. Quantile sketch approaches aim to provide high accuracy for all quantiles through the structure of sketch. In addition to accuracy and size, quantile sketches need to ensure mergeability. KLL [20] provides ϵ rank accuracy using $O((1/\epsilon)loglog(1/\delta))$ space (where δ is the probability of failure). In order to reduce error for larger quantile on skewed distribution, DDSketch [25] has made great progress on heavy-tailed data. QPipe [18], the first quantiles sketching algorithm that can be implemented entirely in the data plane, is based on SweepKLL [17] algorithm. Although QPipe implements most of the functions of quantile sketch, its resource consumption is a major bottleneck in the implementation of the data plane, which occupies 12 stages (almost all the stages) resource. As the most common quantiles of network measurement are the median, 99th percentile, and other special quantiles, quantile sketches cause a great waste of resources. Different from other approaches, EZQ only tracks specific quantile to save the resource usage, which is hardware-friendly for programmable data plane.



(a) Uniform (b) Normal (c) Exponential (d) Zipfian

Figure 1: Rank Functions.

3 DESIGN

3.1 Basic Idea

Our basic idea is to incrementally update the estimated quantile value based on the count of observations. We selectively update depending on whether the current rank of the estimate deviates from the real rank. To reduce the error at a finer granularity, we respectively adopt two modes for small quantile and large quantile. The step is smooth based on the average of observations for small quantiles. When the quantile is large, we adjust to a large-scale step according to the maximum value and minimum value to suit the various environments. The adjustable incremental approach allows us to track quantile with high accuracy and constant memory.

3.2 Algorithm Detail

The aim of our algorithm is to obtain the estimated value $\hat{q}_n$ and make $\hat{q}_n$ close to groundtruth q_n when each stream data arrives. We split our algorithm into three phases to achieve memory-efficiency and high accuracy. We provide the details of each phase and explain its rationale.

- **Phase 1:** Toggle modes between *average* and *max-min*.
- **Phase 2:** Gather information to trigger update logic.
- **Phase 3:** Execute update logic.

Algorithm 1 Incremental Update for EZQ

```
1:  function UPDATE(p, x_n)
2:      if p ≤ T then
3:          sum_n ← sum_{n−1} + x_n, λ_n ← 2 · sum_n/n
4:      else
5:          λ_n ← x_max − x_min
6:      λ_n ← λ_n/(n − 1)
7:      if x_n < q̂_n then
8:          if c⁻ + 1 > n · p then
9:              q̂_{n+1} ← q̂_n − λ_n, c⁺ ← c⁺ + 1
10:         else
11:             c⁻ ← c⁻ + 1
12:     else if x_n > q̂_n then
13:         if c⁺ + 1 > n · (1 − p) then
14:             q̂_{n+1} ← q̂_n + λ_n, c⁻ ← c⁻ + 1
15:         else
16:             c⁺ ← c⁺ + 1
```

Phase 1: Toggle Update Mode (line 2-6). In phase 1, according to the quantile p, we mainly choose the update mode of each update between *average* mode and *max-min* mode. We propose our approach according to the concept of a rank function. The x-axis and the y-axis of the rank function is respectively the rank of all stream data and the value. A simple approach to get the rank function is sorting the stream data in ascending order. We list the rank functions for the uniform, normal, exponential, zipfian distributions, which are shown as Figure 1. The stream data of four distributions is ranged from zero to one which is randomly generated one thousand times. We define the rank function as $g(x) : rank(x) \rightarrow x$, whose independent variable and dependent variable are defined as rank and value of the stream data. We observe that the upward trend of $g(x)$ varies with the distribution. Let λ_n be the step of each update. As a result, λ_n should be $g((n+1)\cdot p) - g(n \cdot p)$.

In our memory-efficient algorithm, accurately obtaining the corresponding rank values is infeasible. To overcome this challenge, we propose a method that fits by concatenating samples of stream data, which approximates $g(x)$ as an arithmetic progression. Specifically, we introduce two modes, namely *average* mode and *max-min* mode, that can be implemented on the programmable switch.

Our *average* mode is based on the average of stream data. Let x_{min} and avg_n be the minimum item and the average value of stream data, respectively. Based on the principle of arithmetic sequence, we approximate λ_n by $2 \cdot (\sum_i x_i/n - x_{min})/(n-1)$. To exclude the influence of outlier data, we remove x_{min} in the above formula. This simplified formula can be expressed as $\lambda_n = 2 \cdot avg_n/(n-1)$. Since the overall data is considered, the update of the *average* mode is smooth. However, for some skewed distributions, too much smoothness leads to slow convergence, particularly at high quantiles. In such cases, we adopt the *max-min* mode.

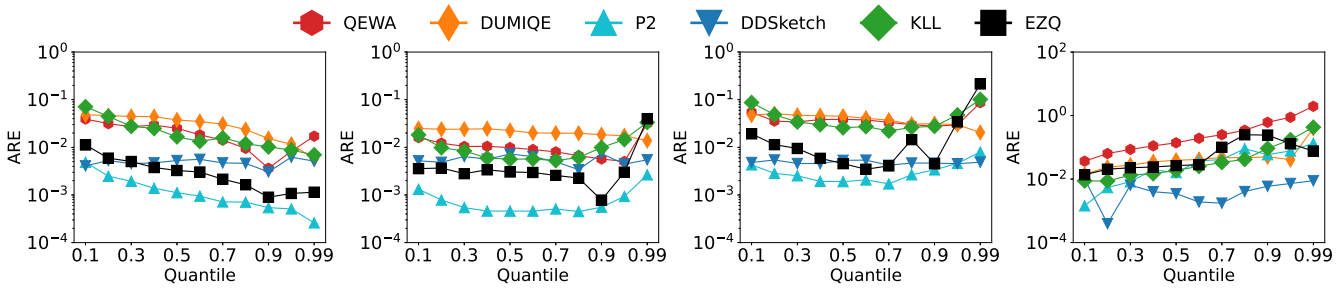
In the *max-min* mode, we estimate the slope of $g(x)$ based on the maximum and minimum item of stream data. Let x_{max} be the maximum item of stream data. We approximate λ_n by $(x_{max} - x_{min})/(n-1)$ through linear regression of rank function. However, when the quantile is small, such as 10-percentile, the appearance of outliers results in a large update step size, which affects the accuracy.

Since the two modes have different advantages under different p, we use the *average* mode for low quantiles and the *max-min* mode for high quantiles. We suggest setting the toggle threshold T to 0.8, as this setting provided the best results after multiple sets of experiments.

Phase 2: Gather Stream Information (line 8, 11, 12, 16). In this phase, we collect the information of stream data to prepare for phase 3. Let c_n^- and c_n^+ respectively be the counters that record the number of x_n less or greater than $\hat{q}_n$. Whenever a stream data x_n is input to the algorithm, we first compare the current item x_n with the value of $\hat{q}_n$. If x_n is less than $\hat{q}_n$, then c_n^- needs to add 1; if x_n is greater than $\hat{q}_n$, then c_n^+ needs to add 1. A fact is that c_n^- and c_n^+ are $n \cdot p$ and $n \cdot (1-p)$ after n stream data if $\hat{q}_n$ is exactly the groundtruth of the p-quantile. We upperbound c_n^- and c_n^+ by $n \cdot p$ and $n \cdot (1-p)$ which are the thresholds whether $\hat{q}_n$ updates. When c_n^- and c_n^+ exceed the threshold, we execute our update logic where c_n^-, c_n^+ and $\hat{q}_n$ update with an approximate approach.

Phase 3: Execute Update Logic (line 9, 14). In phase 3, we update $\hat{q}_n$, c_n^- and c_n^+ when the threshold of phase 2 is exceeded. Our solution is based on that c_n^- and c_n^+ should move accordingly if estimated value $\hat{q}_n$ is updated. If c_n^- exceeds the threshold $n \cdot p$, it means that the estimated value $\hat{q}_n$ is greater than groundtruth q_n. We update $\hat{q}_n$ with the step length $-\lambda_n$ from phase 1. As $\hat{q}_{n+1}$ is less than $\hat{q}_n$, we increment c_n^+ by one, which means that we treat $\hat{q}_n$ as a stream data that has gone through. On the contrary, if c_n^+ exceeds the threshold $n \cdot (1-p)$, we update $\hat{q}_n$ with λ_n and subtract 1 from c_n^- following the same principle.

Discussion. According to the above details, we integrate three phases as Algorithm 1. We observe that no matter which control logic it is, one and only one of c^+ and c^- performs the auto-increment operation. For simplicity, we merge the auto-increment operation of phase 2 and phase 3. We collect stream information and update $\hat{q}_n$ in separate operations. The adjustable incremental update design of EZQ achieves small errors across different quantile settings and traffic distributions. We implement specific quantile tracking with constant memory (c_n^+, c_n^-, sum_n, avg_n) and low computational operations, which are the basis for deployment on programmable switches. In §3.3, we discuss our detail on data plane implementation.



(a) Uniform Distribution (b) Normal Distribution (c) Exponential Distribution (d) Zipfian Distribution

Figure 2: Accuracy on synthetic traces.

3.3 Data Plane Implementation

We implement EZQ in the data plane with 800 lines of P4 code. Our implementation handles the following key points in the programmable data plane. (1) Multiple reads and writes between variables; (2) The implementation of the division part in calculation of λ_n. We respectively introduce our implementation of two key points.

Note that we use two registers to store the state variables. c^+ and c^- are implemented through a data structure *pair* in one register. As we know, registers can only be read or written once in a pipeline. However, the comparison and the update of q_n cannot be implemented in a stage. We leverage packet recirculation to update q_n to work around this limitation. Packet recirculation allows our algorithm to approximately achieve multiple reads and writes in one register.

As we cannot implement the division in the data plane, we leverage lookup tables to approximately compute the λ_n. In particular, we store combinations of *max-min* (or *average*) and n in lookup tables. When the packet matches the corresponding entry in the table, we set λ_n to the value specified in advance. Since maintaining all results consumes a lot of TCAM, we aggregate results into coarser-grained entries to reduce TCAM consumption. We find in experiments that the approximate calculation of λ_n will affect the convergence speed of EZQ. However, as long as a sufficient number of stream data are used to make EZQ converge, the impact of the approximate calculation of λ_n on the accuracy is limited.

What's more, parameters (e.g., p and update mode) of EZQ can be reconfigured at switch runtime through Southbound API, which allows us to switch the quantile to be measured and achieve higher accuracy.

4 EVALUATION

In this section, we persent a performance comparison of EZQ with other state-of-the-art solutions on both synthetic and real-world traces through simulation in §4.1 and §4.2. Furthermore, we evaluate the utilization of hardware resource on Barefoot Tofino switches in §4.3.

Environment: While we implement EZQ in P4 on a Barefoot Tofino switch, the experiments are conducted as simulations in Python on a server with 10-core CPU (Intel Xeon Silver 4210R 2.4GHz). We confirm that the outcomes of both the hardware implementation and simulations are identical.

Metrics	**EZQ**	QEWA	DUMIQE	P2	KLL	DDSketch
Runtime (ms)	**11.04**	11.94	6.55	41.74	21.26	86.17
Memory (bytes)	**36**	32	24	96	1024	2560

Table 3: Runtime and memory comparison of EZQ and other approaches.

Setup: In the experiments, we evaluate EZQ's performance using two types of traces. One is the synthetic stationay data corresponding to Uniform, Gaussian, Exponential, and Zipfian distributions, as described below. (1) Uniform: $f(x) = 1/(b-a)$, where $a = 0, b = 1$; (2) Gaussian: $f(x) = (1/(\sqrt{2\pi}\sigma))$ $exp(-(x-\mu)^2/(2\sigma^2))$, where $\mu = 0.5$, $\sigma = 0.125$; (3) Exponential: $f(x) = \mu\ exp(-\mu x)$, where $\mu = 1/6$; (4) Zipfian: $f(x) = 1/(k^s\ H_s)$, where $H_s = \sum_{n=1}^{\infty} 1/n^s$ and $s = 1.1$. For real world traces, we collect DNS round-trip time (RTT) and flow size information from CAIDA [6, 7].

Baselines and metrics: We compare EZQ to the popular quantile tracking solutions P2 [19], DUMIQE [37], QEWA [15], KLL [20], DDSketch [25]. The measurements are performed with Python implementations of all approaches. We consider the following performance metrics:

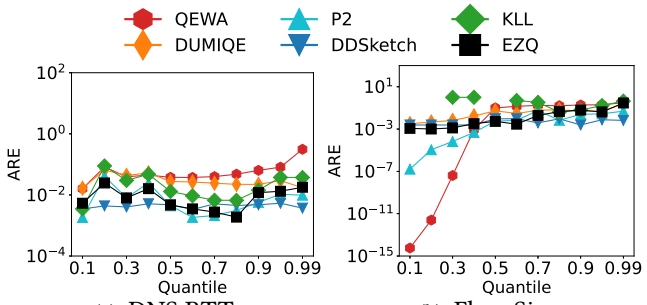
- *Accuracy:* We report *relative error* = $\frac{|\hat{q}-q|}{q}$. For each traces and quantile points, we run 100 times independently and obtain *average relative error (ARE)* = $\frac{\sum relative\ error}{n}$, where n is the number of independent experiments.
- *Runtime:* The average time (milliseconds) per update and query opreation.
- *Memory:* Resource usage of variables and sketch items (bytes).

Parameters: We select $T = 0.8$ in EZQ. For other approaches, we select the parameters corresponding to the best performance for different distributions and traces. Then we fix the parameters on all experiments. (a) P2: no parameters to set. (b) DUMIQE: we set $\lambda = 0.01$. (c) QEWA: we set $\lambda = 0.01, \gamma = 0.01$. (d) KLL: we choose 256 items of sketch. (e) DDSketch: we set $\alpha = 0.01$ by default.

4.1 Synthetic Traces

We show the performance of EZQ and baselines on synthetic stationay data in this experiment. The test data mainly contains uniform, normal, exponential, zipfian distributions as above described. The sample-by-sample quantile estimates are obtained for quantile p ranging from 0.1 to 0.9, with additional high quantiles of 0.95 and 0.99. We obain ARE within 100 random generated sequences. We show the accuracy under the different distributions in Figure 2.

Figure 2 shows that EZQ achieves lower ARE than DUMIQE and QEWA (up to 23.58×) in all settings. EZQ achieves AREs that are similar to DDSketch and KLL. Although EZQ is slightly inferior to P2, KLL and DDSketch on specific distributions, EZQ outperforms P2, KLL and DDSketch on runtime up to 3.78×, 1.93× and 7.81× (see Table 3), respectively. The resource usage of KLL and DDSketch grows with the number of observations. EZQ consumes two orders of magnitude less memory than KLL, P2 and DDSketch (see Table 3).



(a) DNS RTT (b) Flow Size

Figure 3: Accuracy on real-world traces.

4.2 Real-World Traces

We evaluate EZQ with real-world traces [6, 7], which include DNS Round-Trip Time (DNS RTT) and aggregated flow size. As shown in Figure 3, we observe that EZQ outperforms DUMIQE, QEWA, P2, KLL in almost all settings for esimating quantiles of RTT. For estimating the quantile of flow size, the ARE achieved by EZQ is less than 1% and lower than DUMIQE, QEWA, DDSketch in most cases. Meanwhile, the zero-ARE point of KLL is not shown in Figure 3. We observe that KLL has some outliers ARE points at $p = 0.3, 0.4, 0.6, 0.7$, which is not stable on skewed data. Although EZQ is slightly inferior to P2 and DDSketch, EZQ saves much runtime and resource usage as we discuss in §4.1.

4.3 Hardware Resource Utilization

As one of the few quantile estimators that can be deployed in the data plane, we show that the resource utilization of EZQ is limited. Table 4 shows the hardware resource overhead of EZQ on Tofino. We compare EZQ with existing quantile estimator soltuion QPipe [18] which can be implemented in the programmable data plane. We observe that EZQ consumes only 7 stages and a small amount of resources due to the lower computational operations. Since QPipe does not provide the P4 code based on the Tofino, we try our best to estimate the resource consumption of QPipe on the Tofino. Note that our resource estimation about QPipe adopts a quite conservative way. For example, for SRAM, we only count the SRAM consumption of QPipe for registers. Table 4 shows that QPipe consumes all of stages (12 stages) and more Stateful ALUs (4.41× higher than EZQ). For resources that we cannot estimate, we do not show them in the table. Based on the comparison of hardware overhead, we argue that QPipe is not scalable and cannot deploy other network telemetry systems in programmable data plane. As an incremental quantile tracking approach, lightweight EZQ can be widely deployed for various environment.

Solutions	Stages	SRAM	TCAM	sALUs	Hash Bits	VLIW
EZQ	**7**	**3.1%**	**6.3%**	**12.5%**	**2.4%**	**7.6%**
QPipe	12	5.2%	\	50.0%	\	\

Table 4: Switch resource usage.

5 APPLICATION AND FUTURE WORK

EZQ's hardware-friendly design and memory efficiency make it suitable for various network telemetry applications. By integrating it into PINT [4] and DeltaINT [30], we address the existing gaps in quantile tracking approaches in the data plane. In scenarios with skewed workloads, dynamic resource reallocation is crucial for effective network telemetry tasks. Chen *et al.* [10] proposed a tighter error bound based on the quantile of the counter. Leveraging EZQ in the data plane enables us to dynamically reallocate resources with high throughput and accuracy, as supported by the theoretical foundations [10]. Furthermore, EZQ, when used to track the 95th-percentile queue length and flow completion time, proves effective in capturing network congestion, as demonstrated by HPCC [21].

In the future, we aim to provide theoretical guarantees for EZQ, demonstrating its convergence within a specific range. Additionally, we plan to investigate a unified solution by integrating the *average* and *max-min* modes into our algorithm. Finally, we intend to apply our approach to a broader range of network telemetry systems [36, 39] within the programmable data plane.

6 CONCLUSION

We present EZQ, an adjustable incremental approach designed to effectively track quantiles. Implemented in the data plane, EZQ leverages constant memory and hardware-friendly operations. Through comprehensive evaluation, we showcase the remarkable capabilities of EZQ, emphasizing its ability to achieve both resource efficiency and high accuracy across diverse distributions and quantile levels. Additionally, EZQ enables aggregated network performance monitoring and supports adaptive resource allocation applications that depend on efficient quantile tracking within the programmable data plane.

REFERENCES

[1] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *ACM Trans. Database Syst.* 38, 4 (2013), 1–28.

[2] V Altukhov and E Chemeritskiy. 2014. On real-time delay monitoring in software-defined networks. In *Proc. of IEEE MoNeTeC*. IEEE, 1–6.

[3] Aboubacar Amiri and Baba Thiam. 2014. A smoothing stochastic algorithm for quantile estimation. *Statistics & Probability Letters* 93 (2014), 116–125.

[4] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *Proc. of ACM SIGCOMM*. 662–680.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (2014), 87–95.

[6] CAIDA. 2014. IPv4 Routed /24 DNS Names Dataset. https://www.caida.org/data/active/ipv4_dnsnames_dataset.xml. (2014).

[7] CAIDA. 2019. The CAIDA UCSD Anonymized Internet Traces. https://www.caida.org/catalog/datasets/passive_dataset. (2019).

[8] Jin Cao, Li Erran Li, Aiyou Chen, and Tian Bu. 2010. Tracking quantiles of network data streams with dynamic operations. In *Proc. of IEEE INFOCOM*. IEEE, 1–5.

[9] Fei Chen, Diane Lambert, and José C Pinheiro. 2000. Incremental quantile estimation for massive tracking. In *Proc. of ACM SIGKDD*. 516–522.

[10] Peiqing Chen, Yuhan Wu, Tong Yang, Junchen Jiang, and Zaoxing Liu. 2021. Precise error estimation for sketch-based flow measurement. In *Proc. of ACM IMC*. 113–121.

[11] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. 2019. Fine-grained queue measurement in the data plane. In *Proc. of ACM CoNEXT*. 15–29.

[12] Xiang Chen, Qun Huang, Dong Zhang, Haifeng Zhou, and Chunming Wu. 2020. Approsync: approximate state synchronization for programmable networks. In *Proc. of IEEE ICNP*. IEEE, 1–12.

[13] Ulrich Fiedler and Bernhard Plattner. 2003. Using latency quantiles to engineer qos guarantees for web services. In *Proc. of ACM IWQoS*. Springer, 345–362.

[14] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2, 58–66.

[15] Hugo Lewi Hammer, Anis Yazidi, and Håvard Rue. 2019. A new quantile tracking algorithm using a generalized exponentially weighted average of observations. *Appl. Intell.* 49 (2019), 1406–1420.

[16] Zaobo He, Zhipeng Cai, Siyao Cheng, and Xiaoming Wang. 2015. Approximate aggregation for tracking quantiles and range countings in wireless sensor networks. *Theoretical Computer Science* 607 (2015), 381–390.

[17] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. 2022. Streaming quantiles algorithms with small space and update time. *Sensors* 22, 24 (2022), 9612.

[18] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. 2019. Qpipe: Quantiles sketch fully in the data plane. In *Proc. of ACM CoNEXT*. 285–291.

[19] Raj Jain and Imrich Chlamtac. 1985. The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Commun. ACM* 28, 10 (1985), 1076–1085.

[20] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *Proc. of IEEE FOCS*. IEEE, 71–78.

[21] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proc. of ACM SIGCOMM*. 44–58.

[22] Lingxia Liao, Victor CM Leung, and Min Chen. 2018. An efficient and accurate link latency monitoring method for low-latency software-defined networks. *IEEE Trans. Instrum. Meas.* 68, 2 (2018), 377–391.

[23] Qiang Ma, Shanmugavelayutham Muthukrishnan, and Mark Sandler. 2013. Frugal streaming for estimating quantiles. *Space-Efficient Data Structures, Streams, and Algorithms* (2013), 77–96.

[24] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1999. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record* 28, 2, 251–262.

[25] Charles Masson, Jee E Rim, and Homin K Lee. 2019. DDSketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *arXiv preprint arXiv:1908.10693* (2019).

[26] Byung-Hoon Park, George Ostrouchov, Nagiza F Samatova, and Al Geist. 2004. Reservoir-based random sampling with replacement from data stream. In *SIAM*. SIAM, 492–496.

[27] Sandhya Rathee, Shubham Tiwari, K Haribabu, and Ashutosh Bhatia. 2022. qMon: A method to monitor queueing delay in OpenFlow networks. *Journal of Communications and Networks* 24, 4 (2022), 463–474.

[28] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics* (1951), 400–407.

[29] Rana Shahout, Roy Friedman, and Ran Ben Basat. 2022. SQUAD: Combining sketching and sampling is better than either for per-item quantile estimation. *arXiv preprint arXiv:2201.01958* (2022).

[30] Siyuan Sheng, Qun Huang, and Patrick PC Lee. 2021. DeltaINT: Toward general in-band network telemetry with extremely low bandwidth overhead. In *Proc. of IEEE ICNP*. IEEE, 1–11.

[31] Debanshu Sinha, K Haribabu, and Sundar Balasubramaniam. 2015. Real-time monitoring of network latency in software defined networks. In *Proc. of IEEE ANTS*. IEEE, 1–3.

[32] Luke Tierney. 1983. A space-efficient recursive procedure for estimating a quantile of an unknown distribution. *SIAM J. Sci. Statist. Comput.* 4, 4 (1983), 706–711.

[33] Nitya Tiwari and Prem C Pandey. 2019. A technique with low memory and computational requirements for dynamic tracking of quantiles. *J. Signal Process. Syst.* 91 (2019), 411–422.

[34] Tofino2. 2023. https://www.intel.com. (2023).

[35] Milan Vojnovic, J-Y Le Boudec, and Catherine Boutremans. 2000. Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times. In *Proc. of IEEE INFOCOM*, Vol. 3. IEEE, 1303–1312.

[36] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and TS Eugene Ng. 2022. Closed-loop network performance monitoring and diagnosis with {SpiderMon}. In *Proc. of USENIX NSDI*. USENIX Association, 267–285.

[37] Anis Yazidi and Hugo Hammer. 2017. Multiplicative update methods for incremental quantile estimation. *IEEE Trans. Cybern.* 49, 3 (2017), 746–756.

[38] Xinchang Zhang, Yinglong Wang, Jianwei Zhang, Lu Wang, and Yanling Zhao. 2019. RINGLM: A link-level packet loss monitoring solution for software-defined networks. *IEEE J. Sel. Areas Commun.* 37, 8 (2019), 1703–1720.

[39] Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, and Guihai Chen. 2022. FlyMon: enabling on-the-fly task reconfiguration for network measurement. In *Proc. of ACM SIGCOMM*. 486–502.