μMon: Empowering Microsecond-level Network Monitoring with Wavelets

Hao Zheng, Chengyuan Huang, Xiangyu Han, Jiaqi Zheng, Xiaoliang Wang, Chen Tian, Wanchun Dou, Guihai Chen
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

ABSTRACT

Network monitoring is essential for network management and optimization. In modern data centers, fluctuations in flow rates and network congestion events (*e.g.*, microbursts) typically manifest on a microsecond timescale. However, the time granularity of network monitoring systems has not been refined correspondingly to efficiently capture these behaviors. Attaining the monitoring granularity at the microsecond scale can greatly facilitate network performance analysis and management, but poses considerable challenges regarding memory, bandwidth, and deployment costs. We propose μMon, a novel microsecond-level network monitoring system for data centers. The key of μMon is WaveSketch, an innovative algorithm that measures and compresses flow rate curves using in-dataplane wavelet transform. WaveSketch allows for more accurate characterization of application traffic patterns and aids in profiling transport algorithms. Furthermore, by combining the fine-grained flow rate measurements with network-collected congestion information, μMon can 'replay' congestion events to analyze their cause and impact. We evaluate μMon through testbed deployment and simulations at a granularity of 8.192 μs. The evaluation results demonstrate that μMon can achieve a 90% accuracy in microsecond-level rate measurements with an average of 5 Mbps bandwidth overhead per host. Additionally, it can capture 99% heavy congestion events with 31-82 Mbps bandwidth overhead per switch.

CCS CONCEPTS

• **Networks** → *Data center networks*; **Network monitoring**.

KEYWORDS

Network Monitoring, Sketching Algorithms

ACM Reference Format:
Hao Zheng, Chengyuan Huang, Xiangyu Han, Jiaqi Zheng, Xiaoliang Wang, Chen Tian, Wanchun Dou and Guihai Chen. 2024. μMon: Empowering Microsecond-level Network Monitoring with Wavelets. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3651890.3672236

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672236

1 INTRODUCTION

Network monitoring is essential for network management and optimization. In general, a network monitoring system serves two primary purposes. For one thing, it measures application traffic, providing insights into flow size distribution and flow behaviors such as average rate and rate variations. For another, it monitors network events, including the detection and analysis of congestions caused by microbursts [68], load imbalances [7], incast [34], among others.

In modern data centers, fluctuations in flow rates and network congestion events (*e.g.*, microbursts) typically manifest on a microsecond timescale. This is attributed to the deployment of ultra-low latency forwarding devices and network stacks that utilize technologies like kernel bypass (*e.g.*, user-space TCP [28, 36, 74]) and hardware offloading (*e.g.*, TOE [50], RDMA [21]). Regarding the flow behaviors, low latency in data centers enables congestion signals (*e.g.*, congestion notification packets (CNP)) to reach senders more quickly and significantly adjusts flow rates at the microsecond scale. Capturing these fine-grained rate variations is valuable for understanding network performance and debugging transport algorithms. As for the network events, flows can be generated at the microsecond scale with a high initial rate [75], converging on specific links and increasing the likelihood of microbursts. Detecting and analyzing these transient congestions is essential because they can significantly increase network latency and cause jitters in application performance [29].

However, the time granularity of network monitoring systems has not been refined correspondingly to efficiently capture the fine-grained network behaviors. Traditional monitoring systems (*e.g.*, Netflow [14], SNMP [16]) operate at the granularity of seconds to minutes. They can only obtain the average flow rate and struggle to capture the microsecond-level flow rate variations. Moreover, the coarse-grained monitoring is prone to missing transient congestion events in the network. Existing schemes [24, 26, 39, 53, 63, 71] on flow-level measurement achieve millisecond granularity, which is still three orders of magnitude coarser than the timescale of data center dynamics. Recently, some monitoring tools with microsecond-level precision have emerged. However, these tools generally concentrate on interface-level statistics [20] or detailed analysis of host stack performance [49, 66]. At present, few systems attempt to efficiently monitor fine-grained flow behaviors at scale.

Attaining microsecond-level monitoring in data centers poses significant challenges regarding memory, bandwidth, and deployment costs. As for microsecond-level flow rate measurements, a straw-man solution is to assign existing measurement schemes to each finer-grained time window directly. Specifically, if we refine the time granularity from 10 ms to 10 μs, the solution requires 1,000

times more counters in the worst case, resulting in an unacceptable bandwidth overhead. Regarding microsecond-level congestion detection, recent efforts [29, 54, 73] rely on programmable switches to obtain queue information directly in the data plane. Nevertheless, capturing congestion events on commodity switches also deserves investigation, as many data centers still use fixed-function switches.

We propose μMon, a novel microsecond-level network monitoring system for data centers. Our key idea is to develop a memory-efficient scheme for measuring microsecond-level flow rates. In this way, we can observe the microscopic characteristics of application flows. Furthermore, by combining the fine-grained flow rate measurements with network-collected congestion information, μMon can 'replay' congestion events to analyze their cause and impact. Our contributions can be summarized as follows:

First, we introduce WaveSketch, a novel measurement algorithm designed to accurately measure flow rates at the microsecond level. WaveSketch abstracts flow rate curves as waveforms and then compresses them using wavelet transform theory [67]. By leveraging the capability of wavelet transform for multi-resolution analysis, WaveSketch captures the most significant features of flow rate curves while discarding minor components, thus achieving a good balance between compression ratio and measurement accuracy. To integrate the wavelet transform into network measurement, we make targeted designs in the *mother wavelet*, coefficient selection, and streaming updates. We demonstrate that WaveSketch has a computational complexity of $O(1)$ and provide a prototype for hardware implementation.

Second, we propose a lightweight congestion event capture mechanism on commodity switches. We leverage the fact that packets are ECN-marked when congestion events occur [8, 75]. Inspired by Everflow [76], we obtain information about the flows involved in congestion events by matching and mirroring the congestion experienced (CE-marked) packets to an analyzer. Additionally, we mitigate the bandwidth overhead caused by duplicating packets of elephant flows by packet sampling. We realize the matching, packet sampling, and mirroring operations with common functions available in commodity switches [1, 2, 4, 13].

Third, we present how to perform network-wide synchronized analysis on an analyzer and give several use cases. The microsecond-level rate measurements can be utilized to perform fine-grained traffic analysis, such as in-depth analysis of transport process and debugging underutilization. The detected congestion events can be used to analyze the micro-scale load situation of the network and provide the distribution of congestion duration. More importantly, by pushing the rate curves and events together, network operators can replay congestion events by plotting the rate variation of the associated flows near the event occurrence.

We evaluate μMon through testbed deployment and simulations at a granularity of 8.192 μs. In a network under 15% load running the Hadoop workload [48], WaveSketch can achieve 3.5-57x higher accuracy (across four metrics) than baseline solutions in measuring microsecond-level flow rates, with an average bandwidth requirement of 5 Mbps per host. Besides, μMon can achieve a 99% recall for congestions exceeding ECN KMax threshold with 31-82 Mbps bandwidth per switch, with main flows captured. Operators can further decrease the overhead at the expense of reduced accuracy.

This work does not raise any ethical issues.

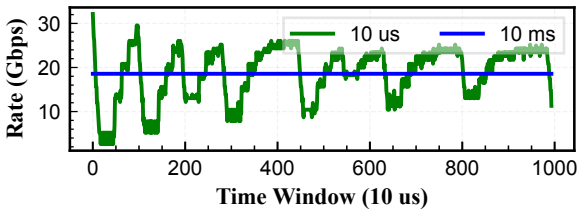


Figure 1: Changes in flow rate at different timescales. The flow is experiencing contention with background traffic and exhibits oscillatory behavior.

2 MICROSECOND-LEVEL NETWORK MONITORING

In this section, we begin by highlighting the importance of enabling microsecond-level network monitoring in data center networks (DCNs). Next, we outline our main objectives, the associated benefits, and practical challenges.

2.1 Background

Data center networks now boast ultra-low end-to-end latency measured in tens of microseconds [23, 37]. On the one hand, current switching hardware can complete packet forwarding within a range of hundreds of nanoseconds to a few microseconds [43]. On the other hand, advanced transport protocols like RDMA minimize latency on the host side by utilizing techniques such as kernel bypass and hardware offloading. As a result, the performance enhancements in data centers lead to *flow behaviors* and network *congestion events* occurring at the microsecond timescale.

As for *microsecond-level flow behaviors*, application traffic can be rapidly initiated at the microsecond scale, converging on specific network links and leading to congestion. Congestion signals such as congestion notification packets (CNP) are returned within tens of microseconds following the network's Round-Trip Time (RTT), leading to noticeable adjustments in flow rates. Figure 1 showcases a flow rate curve gathered from our RDMA testbed, where we execute the WebSearch workload [8] and induce traffic contention within a single bottleneck topology. In the 10-μs observing granularity, the flow rates reveal intricate patterns characterized by peaks, deep troughs, and recoveries. These patterns reflect the initial throughput of the flows, adjustments made in response to congestion, recovery of flow rate, and potential oscillations that may raise concerns. However, most of the existing flow measurement methods work at tens of millisecond granularity [26, 39, 53]. As shown in Figure 1, the 10-ms observing window yields only an average perspective, masking the subtle complexities and transient behaviors evident at the microsecond timescale.

As for *microsecond-level congestion events*, network congestion such as microbursts, packet loss, load imbalance, and PFC pauses frequently occur in data centers [75]. These network events increase the network latency and cause performance jitters [29]. With the rise in link bandwidth, the accumulated packets in the queue will be quickly emptied, making the capture timing of these congestion events fleeting. However, existing Simple Network Management Protocol (SNMP) systems and vendor-specific interfaces typically

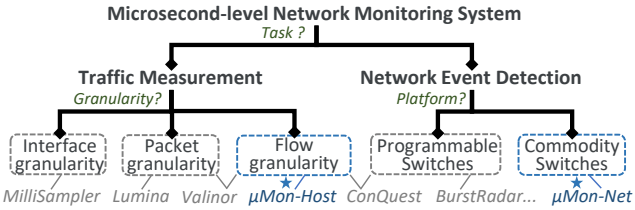


Figure 2: Taxonomy of microsecond-level network monitoring systems.

work at the millisecond to the minute level [47], making it impossible to accurately capture transient congestion events, analyze the root cause and take targeted solutions.

2.2 Objectives and Benefits

To capture fine-grained flow behaviors and congestion events, we urgently need to enhance the monitoring time granularity of the data center networks to the microsecond timescale. Recently, a group of works has achieved microsecond-level precision monitoring from various aspects. As shown in Figure 2, these efforts can be broadly classified into two two main categories: traffic measurement and network event detection.

For traffic measurement, Millisampler [20] is a data-center-scale monitoring system that measures microbursts at interface granularity (*e.g.*, the total received and transmitted bytes of a port or queue in a short time window). However, the per-flow measurements are not within their design scope. Valinor [49] and Lumina [66] focus on in-depth analysis of host/NIC stacks by tracking every packet or *sk_buff* arriving event. Nevertheless, reducing resource overhead to facilitate network-scale deployments is not their primary objective. For network event detection, state-of-the-art systems such as ConQuest [12] and BurstRadar [29] need to leverage emerging programmable switches. Moreover, ConQuest's support for flow granularity measurements focuses on immediate in-data-plane queries. It recycles expired data and thus cannot obtain complete rate curves. In this paper, we aim to achieve the following two new objectives:

- **Microsecond-level flow rate measurements at scale.** We aim to track rate variations for each flow throughout its lifecycle at the microsecond level, capturing both burst rate fluctuations and long-term trends in flow rate changes. The solution should be lightweight to facilitate network-wide deployments.
- **Congestion detection/analysis on commodity switches.** We aim to detect transient congestion events entirely using commodity switches, as many data centers still use legacy devices. More importantly, we must enable event context replay to analyze the causes and subsequent effects of the event.

Enabling the two objectives will bring significant **benefits (B#)** in following aspects:

B1. Analyzing performance issues of applications and transport algorithms: Fine-grained flow rate information can accurately reflect the network behavior of applications. For instance, we can access the working state of the transport-layer algorithms by observing the flow rates at the round-trip time (RTT) scale. This analysis helps determine whether the congestion control protocol operates correctly (*e.g.*, convergence, fairness) and whether the related parameters are appropriately set. Besides, the information

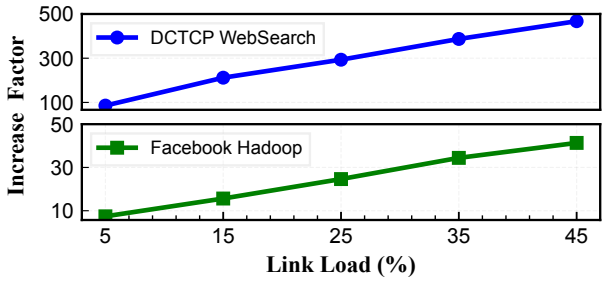


Figure 3: The amplification factor of measurement data volume introduced by 10-us window measurement.

can also help us infer the reasons for the low throughput. For example, multiple gaps in a flow rate curve indicate that the insufficient throughput results from inadequate application data.

B2. Understanding the micro-scale load status of the network: Capturing congestion events can provide a fine-grained understanding of the load conditions of links in the network. By combining it with microsecond-level flow rate information, we can replay the causes and consequences of the events. For example, when a microburst occurs, we can query the relevant flow rate near the corresponding time to distinguish the root cause and the event's subsequent impact on victim flows. As a result, we can identify the links most prone to congestion and the main contributors and victims of the bottlenecks, which enables us to develop targeted strategies to prevent future congestion.

B3. Modeling microscopic traffic behavior and guiding network specifications: Existing traffic models are idealized as they do not consider the operational mode of the upper-layer application or the interference caused by the hardware and software operating environment. With the microsecond-level measurements, operators can model microscopic traffic behavior that better fits real network workloads. Additionally, information about peak rates and duration has significant implications for optimizing chip parameters, such as buffer size, ECN marking, and meters.

2.3 Challenges

Enabling microsecond-level network monitoring will greatly benefit network management, analysis, and optimization. However, there is no such thing as a free lunch. There are three major **challenges (C#)** to achieving the two objectives:

C1. How to reduce significant memory and bandwidth overhead? In flow rate measurements at a time granularity δ, the number of required counters for a flow f is $n(f,\delta) = \frac{t_f}{\delta}$, where t_f denotes the flow's active time. The total expected number of counters in time granularity δ for a specific traffic workload is $N(\delta) = \sum_f n(f,\delta)$. We evaluate the counter increase factor $\frac{N(10\mu s)}{N(10ms)}$ (*i.e.*, refining the time granularity from 10 ms to 10 μs) under two popular workloads. As shown in Figure 3, the counters increased by 34.4x in Facebook Hadoop [48] and 387x in DCTCP WebSearch [8] when the link load is over 35%. Suppose each measurement window uploads 20 KB of measurement data every 10 ms. Recklessly expanding to a 10 μs granularity will result in a bandwidth consumption of up to 6.34 Gbps in the WebSearch workload when the link load is over 35%, and the situation worsens when the link load becomes heavier. The data volume mentioned refers to that of a single device. When

multiplied by tens of thousands of network devices, the resulting data volume becomes overwhelming.

State-of-the-art measurement systems [24, 26, 39, 53, 63, 71] work at tens of milliseconds of granularity. Directly refining their measurement window to the microsecond level results in unacceptable bandwidth overhead. Using traditional compression methods (*e.g.*, Huffman coding [38], LZW [42]) in the data plane is impractical due to their complex computational requirements, such as handling variable bit widths, executing floating-point operations, and dynamic memory allocation. If we deploy them using CPUs, we first need to transfer the extensive counters from the data plane to the control plane, which will significantly strain device PCIe interfaces and computation resources. Persistent sketches [60] are a class of multi-version data structures from database communities. They use the piecewise-linear approximations (PLA) method [45], which requires complex calculations involving the half-plane intersection of two polygons.

C2. How to capture transient congestion events in the network? The current network observation methods in data centers typically include polling hardware counters through control plane interfaces, which operate at the millisecond to the minute level. However, they may fail to capture congestion events that occur between two polling intervals. Additionally, the counters can only provide information about the queue length without further analysis of event details. For instance, they cannot differentiate which applications contribute to congestion and how it impacts the network. To address these limitations, recent efforts [29, 54, 73] have leveraged programmable switches to achieve microsecond-level event monitoring by directly accessing the queue length in the data plane. Nevertheless, we must take commodity switches into account in our design, as many data centers still use fixed-function switches.

C3. How to distribute measurement functions among network devices? To achieve microsecond-level traffic measurement and congestion event detection, deploying dedicated measurement modules in network devices is imperative. However, simultaneously deploying the two functionalities across all network devices would be overkill since it will result in additional bandwidth overhead (*e.g.*, redundant traffic statistics) and necessitate the programmability of all devices. Designing a reasonable microsecond-level monitoring architecture and coordinating analysis from distributed measurements remains an unexplored area of research.

3 µMON OVERVIEW

This section gives a high-level introduction of µMon. As shown in Figure 4, µMon has three functional components deployed at different data center locations.

µFlow measurement at hosts. We perform microsecond-level flow (µFlow) measurements at end hosts. To address **C1**, we design WaveSketch, which leverages wavelet transform theory to compress flow rate counters efficiently. The wavelet transform is a powerful mathematical tool used for signal analysis. It can capture the most significant features of a signal at multiple scales, guiding subsequent compression. In §4, we introduce the fundamental principles of the wavelet transform and present how WaveSketch applies this technology to network measurement, including basic structure, update and query procedures, and hardware implementation.

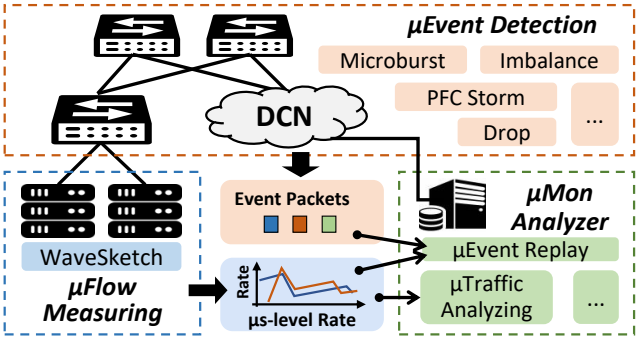


Figure 4: µMon system overview

µEvent detection at switches. We detect transient congestion events (µEvent) in network switches. To address **C.2**, we implement an event capture solution based on Access Control Layer (ACL) tables and remote mirroring, which are common functions in commercial switches. In particular, we mirror the packets involved in congestion events to the analyzer and reduce the bandwidth overhead through sampling. In §5, we introduce how to capture transient congestion events in the network efficiently.

Network-wide analysis at µMon analyzer. The µFlow measurements and detected µEvents are sent to a unified analysis platform for network-wide analysis. Operators can utilize µMon analyzer to address numerous pain points that were previously unsolvable by monitoring systems with coarse time granularity. In §6, we introduce how to conduct the network-wide synchronized analysis and introduce several typical use cases of µMon, such as fine-grained application performance analysis and congestion event replay.

The above architecture solves **C.3** by considering the characteristics of different network locations. Specifically, we only perform microsecond-level flow measurements at end hosts, where we can cover all application traffic without introducing repeated statistics. Moreover, using technologies such as eBPF [57] and smartNICs [17, 32, 56], we can easily deploy customized measurement algorithms at hosts with a relatively acceptable cost. As for congestion events, since they can only be captured on the network side, we use common functions available in commodity switches, making it feasible to implement in existing data centers.

4 µFLOW MEASURING AT HOSTS

This section provides a detailed description of µFlow measurements at end hosts. We first introduce the wavelet transform and explain why we adopt wavelet methods to compress flow rates and the challenges of applying them to network measurements (§4.1). Next, we delve into the design of WaveSketch and elucidate how it utilizes the wavelet approach for network measurements (§4.2). Finally, we give the hardware implementation of WaveSketch (§4.3).

4.1 Preliminary on Wavelets Transform

The wavelet transform [67] is a powerful mathematical tool for analyzing time series signals. In flow rate measurement, the concept of 'signal' can be regarded as the variation of a flow's packet or

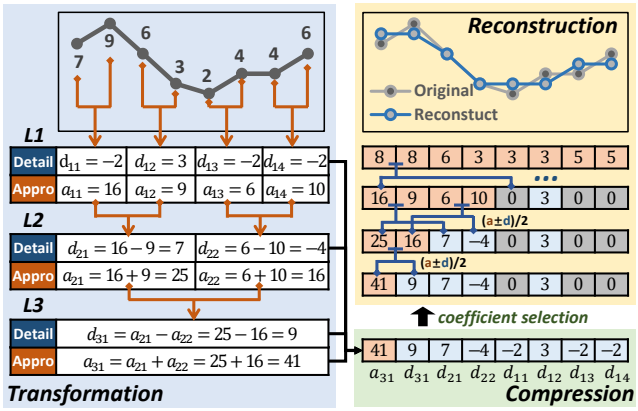


Figure 5: An example of wavelet-based counter series compression in μMon.

byte counters within consecutive time windows. In the subsequent paper, we will interchange 'signal' and 'sequence of count values' for ease of expression.

Discrete Wavelet Transform (DWT [51]) is the discrete version of the wavelet transform. As shown in Figure 5, a DWT-based signal compression involves three stages: transformation, compression, and reconstruction. Given a signal f of length n, the transformation stage hierarchically decomposes it into a set of basis functions known as wavelets. The decomposition process, performed over L levels, yields two categories of coefficients: approximation coefficients and detail coefficients. The coefficients can be understood as the weight of a particular wavelet on the original signal. At each level, a low-pass filter generates the approximation coefficients, representing the signal's coarse or average characteristics, while a high-pass filter produces the detail coefficients, encapsulating the signal's fine or rapidly changing details. The wavelets in the high-pass filter are generated from a single *mother wavelet* $\psi(x)$ through scaling and translation, and their coefficients W_f on the signal are given by:

$$W_f[s,t] = \frac{1}{\sqrt{s}} \sum_{i=0}^{\infty} f[i]\psi\left(\frac{i-t}{s}\right) \tag{1}$$

where s is the scale parameter and t is the translation parameter. Finally, the last-level approximation wavelets and the detail wavelets at all L levels form a set of orthogonal bases with the dimension n. Provided that no coefficients are discarded, the reconstruction phase of the DWT can accurately restore the original signal from these coefficients.

The compression process occurs between the transformation process and the reconstruction process. It removes some insignificant coefficients by treating them as zeros. If some coefficients are already zero, lossless compression can be directly applied. Removing non-zero coefficients can achieve further compression while producing some reconstruction errors. In the example shown in Figure 5, we remove three smallest detail coefficients d_{11}, d_{13}, d_{14}. The reconstructed signal still preserves the fundamental waveform. Note that the operations in Figure 5 is a customized version of wavelet transform, which will be described in detail in § 4.2.

The most advantageous point of the wavelet transform lies in its ability to analyze signal characteristics at different scales, also called multi-resolution analysis. It can automatically capture the strength of the signal changes at various time points and time scales. Putting this feature in the flow measurement scenario, it can analyze varying degrees of rate change, including short-term jitters (*i.e.*, shallow-level detail coefficients) and long-term ups and downs (*i.e.*, deep-level detail coefficients). Therefore, when performing compression, it can capture the components that best characterize the signal from different scales and discard those minor components, thus achieving a good balance between compression ratio and measurement accuracy.

Although the wavelet transform is quite suitable for flow rate measurement, applying the wavelet transform in network measurement still requires addressing three problems. First, we need to simplify the calculation of coefficients in the wavelet transform, which involves complex multiplication and floating-point operations, as shown in Equation (1). Second, an effective strategy for coefficient selection is required to balance the compression and measurement accuracy. Last, the wavelet transform described above is performed offline in a complete sequence. However, to accommodate the limitations of memory and enable compression without waiting for all windows to finish, we need to adapt the wavelet transform to an online process.

4.2 WaveSketch DESIGN

We introduce WaveSketch, which applies the wavelet-based compression to network measurement and solves the above three problems. We present the two versions of WaveSketch in turn: a basic version and a full version.

Structure of the basic WaveSketch. As shown in Figure 6, the basic WaveSketch is built upon a Count-Min sketch [15] consisting of $d \cdot w$ buckets. In WaveSketch, we incorporate an internal time dimension to refine each bucket to a microsecond-level window sequence with length n. Specifically, we introduce several variables in each bucket, including an initial window id w_0, current window offset i, current window count c, approximation coefficients set $\mathcal{A}$ and detail coefficients set $\mathcal{D}$. The first packet accessing a bucket will initialize w_0, marking the beginning of the window sequence. i denotes the current window offsets to w_0 while c denotes the current window's packet/byte count value. During a measurement period T (*e.g.*, 10 ms), each bucket generates a sequence of window counters with a maximum length of n, denoted as $c_1, c_2, \ldots, c_n$. Whenever a window counter c_i finishes counting, we immediately perform wavelet transformation for this counter. For the generated wavelet coefficients, we perform a compression process, which filters out unimportant coefficients and saves important coefficients in sets $\mathcal{A}$ and $\mathcal{D}$, where $|\mathcal{A}| + |\mathcal{D}| \ll n$.

Update of the basic WaveSketch. The update of WaveSketch is similar to that of the Count-Min Sketch. In both cases, a set of counter buckets are selected based on d pairwise independent hash values applied to flow identifiers, such as 5-tuple. However, WaveSketch introduces three internal stages for each bucket update:

- **Counting.** When a new packet arrives at w_j with value v, if the packet still belongs to the current counting window w_i (*i.e.*, $i == j$), c is incremented by v, and the update is terminated. Otherwise,

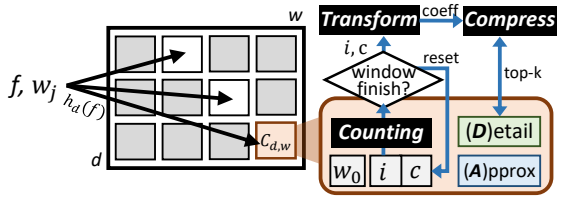


Figure 6: WaveSketch basic version

i and c are passed to the transformation and compression stages. After that, i and c are set to $w_j - w_0$ and v, respectively, for counting the packets in the next microsecond-level window.

- **Transformation.** In this stage, wavelet coefficients are calculated in an online manner every time a window counter c_i is finished. We adopt a variant of the Haar wavelet [52] as the *mother wavelet*, which is defined as:

$$\psi(i) = \begin{cases} 1 & \text{if } 0 \le i < 0.5, \\ -1 & \text{if } 0.5 \le i < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

  As a step function, the Haar wavelet is adept at encapsulating significant local variations within a signal. This feature helps us capture rapid flow rate changes. More importantly, it is highly suitable for data-plane implementation because it can convert the multiplication operations into addition and subtraction operations when calculating the coefficients. For instance, for a wavelet with scale $s = 2$ and translation $t = 0$, its coefficient can be simplified to $\frac{1}{\sqrt{2}}(f[1] - f[0])$, according to Equation (1). The multiply $\frac{1}{\sqrt{2}}$ operation is used to maintain energy conservation. We eliminate it in WaveSketch while the entire transformation maintains its reversibility. As shown in Figure 5, during the transformation, an important observation is that each window counter operates on a specific last-level approximation coefficient and the latest detail coefficient within each level. We use this feature to directly calculate the last-level approximate coefficients and implement the online transformation of the detail coefficients at each level. As shown in Algorithm 1, we use window offset i to determine the position of the last-level approximation coefficient (lines 14-15) and how the counter contributes to each-layer latest detail coefficients (lines 22-27).
- **Compression.** The transformation incrementally produces a small set of last-level approximation coefficients with size $\frac{n}{2^L}$. We retain all of them to accurately reconstruct the total size of a flow. For each layer, one detail coefficient is generated for every 2^{l+1} counter. When a detail coefficient is generated, the compression stage selectively retains the top-k most significant coefficients with a weight $1/\sqrt{2}^l$. This approach minimizes the Euclidean Distance between the reconstructed and original counter series (please see the proof in Appendix A). Unfortunately, the ideal algorithm is feasible for CPUs but needs to be simplified to implement on an ASIC data plane. We approximate the weighted and top-k operations for a practical hardware implementation, which will be introduced in § 4.3.

Computational complexity analysis. The pseudo-code of the WaveSketch update is shown in Algorithm 1. For most packets, the update process only requires a counter accumulation with a time complexity of $O(1)$. When a packet triggers a new counting window, the old counter must be transformed and compressed (lines 18-21), introducing additional computational overhead. Assume that there are m packets within the measurement period T, divided into n microsecond level windows. Using amortized analysis, we demonstrate that the average update cost per packet is $O(1+\epsilon(L+logK))$, where $\epsilon = \frac{n}{m} \le 1$ (please see Appendix B for details). When the network load is heavy, $\epsilon = \frac{n}{m} \to 0$, and $L + logK$ is a small constant based on the desired compression ratio. Therefore, the average update cost per packet is still $O(1)$. Although a higher complexity is required for the worst-case transformation and compression, one key feature is that WaveSketch can operate independently on each level's coefficient, as shown in Algorithm 1 (lines 16-28). The feature enables an efficient hardware implementation (see § 4.3).

Compression ratio analysis. Assuming the number of window counters is denoted as n. For each bucket, the algorithm requires storing $\frac{n}{2^L}$ approximation coefficients in $\mathcal{A}$, K detail coefficients in $\mathcal{D}$, and L temporary detail coefficients (referred to as variable *_details* in Algorithm 1). When transmitting data to the analyzer, it only needs to send w_0, $\mathcal{A}$ and $\mathcal{D}$, resulting in bandwidth usage of $O(\frac{n}{2^L} + K)$. Therefore, the compression ratio is given by

Algorithm 1 WaveSketch Counter Update

```
1:  Initialize w_0, i, c ← 0
2:  Initialize A ← array(size = n/2^L)
3:  Initialize D ← priority_queue(size = K)
4:  Initialize _details ← array(size = L)
5:  procedure COUNTING(w_j, v)
6:      if w_0 == 0 then
7:          w_0 ← w_j
8:      end if
9:      if w_j == w_0 + i then
10:         c ← c + v
11:     else
12:         Transformation(i, c)
13:         c ← v,   i ← w_j − w_0
14:     end if
15: end procedure
16: procedure TRANSFORMATION(i, c)
17:     pos_a ← i >> L
18:     A[pos_a] ← A[pos_a] + c
19:     for l ← 0, ..., L − 1 do
20:         pos_d ← i >> (l + 1)
21:         if pos_d > _details[l].idx then
22:             Compression(l, _details[l])
23:             _details[l] = {pos_d, 0}
24:         end if
25:         sign_d ← (i >> l)&1
26:         if sign_d == 0 then
27:             _details[l].val ← _details[l].val + c
28:         else
29:             _details[l].val ← _details[l].val − c
30:         end if
31:     end for
32: end procedure
33: procedure COMPRESSION(l, detail)
34:     Retain the top-K detail coefficients in D by comparing
        (1/√2^l)|detail.val|
35: end procedure
```

$(\frac{n}{2^L} + \alpha \cdot K)/n$, where $\alpha > 1$ represents the additional metadata of reserved detail coefficients, such as the detail coefficients' level and index. For instance, if we set L to 8 and K to 32, assuming α is 1.5, and considering a window sequence with a length of $n = 2000$, the expected compression rate is 0.028. This implies that when measuring a flow of 20 ms at a granularity of 10 us, WaveSketch can achieve the 0.028 compression rate. Increasing L or decreasing K can improve the compression rate, and the compression effect will be more obvious for a longer measurement period T with larger n. As a trade-off, the increase of L will consume more computing resources while reducing K would lose details of flow rate changes. We discuss the parameter settings in detail in § 7.1.

Query of the basic WaveSketch. The query of WaveSketch is similar to Count-Min Sketch, which selects d buckets by d hash functions and returns the one with the smallest query value among the d buckets. The difference is that a reconstruction process is required for each bucket. The reconstruction is to recover the original signal from the detail and approximation coefficients, which should be performed in the μMon analyzer. As shown in Figure 5, the reconstruction begins at the deepest level. The approximation coefficients are upsampled with the corresponding detail coefficients to reconstruct the approximation coefficients at the shallower level. In particular, for the approximation coefficient $a_{l,i}$ and detail coefficient $d_{l,i}$ of layer l, we can obtain two approximation coefficients of layer $l-1$, which are $a_{l-1,2i-1} = \frac{a_{l,i}+d_{l,i}}{2}$ and $a_{l-1,2i} = \frac{a_{l,i}-d_{l,i}}{2}$ respectively, where $i = 1, \ldots, \frac{n}{2^l}$. For the detail coefficients that are not preserved, we treat them as 0. This process is repeated, progressively using shallower-level detail coefficients until the original signal is fully reconstructed. Due to space constraints, the detailed reconstruction algorithm is presented in Appendix C.

The full version of WaveSketch. To realize the objectives of application traffic analysis, it is necessary to have explicit knowledge of the fine-grained rate information of heavy flows. The full WaveSketch consists of a *heavy* part and a *light* part, serving heavy flows and mice flow, respectively. The heavy part is a hash table with a flow key, a counter bucket, and a vote in each row. It elects heavy flows and directly applies wavelet-based compression for each flow. The light part is a basic version WaveSketch used to measure all mice flows but introduce hash collisions. Due to the existence of the temporal dimension, hash collisions do not necessarily result in counting errors, as they can act in different time windows. Therefore, the width w of WaveSketch is usually smaller than traditional sketches, as it is set according to the number of concurrent flows in a microsecond-level window rather than the total number of flows during the entire measurement period. Moreover, the hash process can effectively aggregate those mice flows into a small set of elongated flows, which can be considered background flows outside the heavy flows. It has a positive effect on wavelet-based compression since it is easier to achieve a better compression rate on a long sequence.

We use majority vote [33] to elect heavy flows from mice flows, an algorithm widely applied to existing sketches [55, 63] for heavy-hitter detection. When a heavy candidate flow is evicted, a new challenge is that the $\frac{n}{2^L} + K$ wavelet coefficients in the heavy part are not easily evicted to the light part in WaveSketch. To address this, we update the heavy and light parts simultaneously when heavy flow packets arrive. In this way, when a heavy candidate is evicted, we only need to cancel the heavy part directly since it has been completely counted in the light part. When querying, the value of the light part may be overestimated due to the existence of heavy flows, so we need to subtract the value of the heavy part flows when reconstructing the light part.

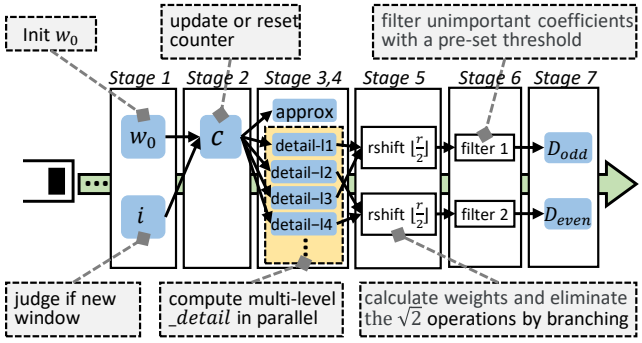


Figure 7: WaveSketch implementation in PISA architectures. Blue boxes denote buckets in registers.

4.3 WaveSketch Implementation

There are two main challenges in implementing a WaveSketch. The first is the long logic of computing the multi-level wavelet coefficients. The second is the selection of weighted top-k coefficients during the compression stage.

CPU Implementation. The first challenge does not affect CPUs since they do not impose a limit on the logical length. As for selecting the top-k coefficients, we can efficiently accomplish this task using min-heaps. In addition, the multi-row sketch updating and the multi-level wavelet coefficient calculations can be further accelerated by using Single Instruction Multiple Data (SIMD) instructions [55, 69].

Hardware Implementation. While the two challenges can be easily addressed on CPU-based platforms, they pose difficulties for hardware implementation based on ASIC chips with a pipelined architecture, such as Protocol-Independent Switch Architecture (PISA [10]). Specifically, the resource utilization of a pipelined architecture is always determined by the longest logic of an algorithm. Although the transformation and compression stages occur only on a small number of packets (*i.e.*, the last packet of a microsecond-level window), we must pre-allocate sufficient computational resources for the worst-case logic in the hardware pipeline. In addition, the weight calculation with $\frac{1}{\sqrt{2}}$ (*i.e.*, in the weight $\frac{1}{\sqrt{2}^l}$) and the top-k election are also unrealistic in PISA hardware chips.

We first leverage the property that the computation of the detail coefficients at different levels in WaveSketch is mutually independent. As shown in Figure 7, we can compute these coefficients in parallel (Stage 3, 4). This dramatically reduces the logic length of the algorithm. For the weighted top-k election, we approximate the process using a branching and thresholding method. Specifically, we observe that as the level increases, the weights are as follows: $\frac{1}{\sqrt{2}}$, $\frac{1}{2}$, $\frac{1}{2\sqrt{2}}$, $\frac{1}{4}$, ... We perform the weighted comparisons based on odd and even levels separately using two priority queues. This way, the weight calculation between coefficients with the same parity will

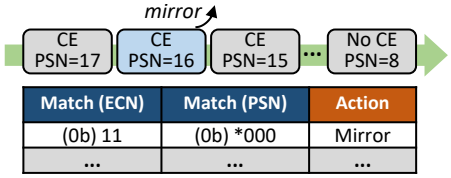


Figure 8: Match, Sample (ratio = $\frac{1}{8}$), and Mirror

become exponential powers of 2, which can be achieved through right shifting. After the coefficient weighting, we use a threshold method to approximate the top-k selection. However, selecting the threshold is a challenge. We sample flow traces from actual scenarios in advance and measure them using an ideal WaveSketch based on the CPU. We treat the median value of minimum values in priority queues as a threshold reference, which is then applied to the hardware version. Our experimental results in § 7.1 demonstrate that the accuracy of the hardware approximate implementation is close to the accuracy of an ideal WaveSketch.

5 µEVENT DETECTION AT SWITCHES

Beyond microsecond-level traffic measurements, network operators are also concerned about microsecond-level network events. This section focuses on solving transient congestion events caused by network traffic, which is widespread and challenging to observe directly. We name these events as µEvent, including microbursts, PFC storms, load imbalances, and packet loss. To detect these events on commodity switches, our approach is to match and mirror packets that exhibit event-specific characteristics while reducing bandwidth overhead through packet sampling:

Event-specific characteristics in packets. All the µEvents mentioned above are queue-related events. Currently, both the DCQCN [75] algorithm and the DCTCP [8] rely on network devices to mark ECN (Explicit Congestion Notification) to sense congestion in the network. Hence, a common characteristic of these events is that packets exceeding the ECN marking threshold are marked with the CE (Congestion Experienced) field. In µMon, we identify packets with the CE field marked as event packets and mirror them accordingly. For packet loss, CE packets are generated prior to the tail drop, and some advanced switches support features like deflect-on-drop [73] to handle the loss packets directly.

Match and mirror the event packets. Nowadays, commodity switches commonly support packet mirroring triggered by access control list (ACL) tables [2]. For example, Everflow [76] matches flags such as TCP SYN, FIN, and RST and mirror related packets to track TCP connections. Similarly, we can achieve the desired logic by adding an ACL rule that matches the location and length of the CE field and associating it with the mirroring action. When defining the mirroring action, we utilize the remote mirroring function to transmit the event packets to µMon analyzer. µEvents on different ports are distinguished by attaching different VLAN tags.

Reduce bandwidth overhead by sampling. The above approach mirrors all packets marked with CE. However, it will result in significant bandwidth overhead when an elephant flow is mirrored. In µMon, we introduce an indirect packet deduplication method using sampling. Specifically, we leverage the characteristic that adjacent packets in a flow have different sequence numbers (SN) [1], such as sequence number in TCP and packet sequence number (PSN) in RoCEv2 [58]. As shown in Figure 8, we can match the lowest w bits of the SN with ACL rules, achieving a sampling probability of $\frac{1}{2^w}$. When analyzing events, our focus is usually on understanding the behavior of heavy flows, while for mice flows, we only need to know their quantity and overall distribution [62]. Therefore, the sampling does not lead to significant information loss because it is highly probable that the sampler will capture at least one packet for each heavy flow.

Moreover, introducing programmable switches would significantly enhance the µEvent detection capabilities. Namely, programmable switches are capable of customized observation of data-plane events. ConQuest [12], BurstRadar [29], and Snappy [11] have all leveraged them for advanced event detection and measurement directly in the data plane. When programmable switches are available, µMon can adopt these designs for higher accuracy and recall rate, and the solutions can also be cooperatively used with WaveSketch for event replay. Besides, we can directly achieve effective de-duplication of event packets and enable batch reporting [73], promoting efficiency considerably.

6 µMON ANALYZER AND USE CASES

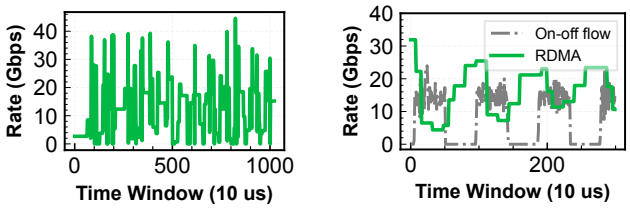
This section first introduces how µMon analyzer performs network-wide synchronized analysis (§6.1). We then present several use cases for using the microsecond-level statistics on the analyzer (§6.2).

6.1 Network-wide Synchronized Analysis

The microsecond-level flow rate measurements and captured network events are sent to the µMon analyzer for comprehensive analysis. However, when we need to perform network-wide traffic analysis and replay congestion events, the time of different nodes must be aligned. For example, to enable the congestion events replay, we need to ensure that the time of the captured event corresponds to a specific time window at the end hosts, allowing for precise querying of the associated flow rate when the event occurs.

When collecting WaveSketchs and µEvent packets, measurement data also carry the corresponding time information to the µMon analyzer. In particular, WaveSketch carries time information (*i.e.*, w_0) in each counter bucket, and switches can configure the mirroring port to add a local timestamp to each mirrored packet [13]. More importantly, time synchronization is essential among the hosts and switches. To achieve the microsecond-level network monitoring, data centers must deploy time synchronization technology operating at the nanosecond level. The conventional synchronization protocol Network Time Protocol (NTP [6]) cannot achieve satisfactory accuracy as it introduces millisecond-level errors. Fortunately, Precision Time Protocol (PTP [59]) is widely deployed, along with some advanced time synchronization methods [18, 30], which can achieve stable nanosecond-level time synchronization. The errors of these nanosecond-level synchronization methods do not extend beyond two microsecond-level windows, which are considered sufficient for µMon. When querying the flow rate, the rate of several

[1] Some scenarios also contain non-negligible traffic that is not TCP or RDMA protocols. The key idea of the sampling is to construct a uniform probability filter based on specific protocol fields. Therefore, a more general method is to match timestamps, a random number, or checksum that varies per packet.



(a) TCP flow (b) RDMA flow

Figure 9: Different flow behaviors evident at μs level.

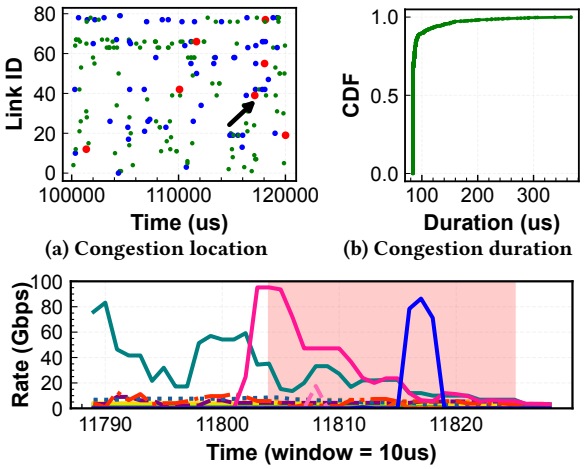
windows before and after the event can be queried rather than focusing solely on the specific window in which the event occurred.

6.2 Use Cases

As a foundational capability, the microsecond-level measurements can be used in many practical analysis and optimization tasks. We list several typical applications here:

Analyzing application performance and transport algorithms. The fine-grained rate measurements provided by WaveSketchs can assist users in analyzing the patterns of their applications at the microsecond timescale. For example, the microsecond-level rate curve aids users in analyzing the underlying causes of link underutilization. Figure 9a presents a low-throughput TCP flow captured in our testbed using WaveSketch. Through the microsecond-level analysis, we find the rate curve is intermittent, which indicates that the host cannot provide sufficient data to send over the network continuously. That is, the under-throughput is caused by the host (*e.g.*, the high latency of the TCP stack), which leads to gaps when transmitting data. In another case, we can observe whether the congestion control protocol is working correctly with the microsecond-level flow rates. We generate RDMA flows with an on-off background flow to compete with the RDMA flows. Figure 9b shows the rate reaction of an RDMA flow when receiving disturbance. We can evaluate the performance of the congestion control algorithm in terms of key metrics such as convergence and fairness. In the past, the above microsecond-level analysis could only be observed in a simulation environment or a small-scale testbed. With μMon, users can observe the fine-grained behavior of network-wide flows in real-time with an acceptable bandwidth overhead.

Monitoring microscope network loads and replay congestion events. By capturing ECN-marked packets, we can perceive the location and timing of congestion in the network. As shown in Figure 10a and Figure 10b, μMon can present a time-location map of congestion events and the distribution of congestion duration. Furthermore, suppose network operators are interested in a long-lasting congestion event, indicated by an arrow in in Figure 10a. As shown in Figure 10c, they can replay the congestion by plotting the rate variation of the associated flows near the event occurrence by querying WaveSketchs. Through the event replay, we learn that the leading cause of the congestion is the contention of a bursty (pink) flow with an existing (green) flow. Moreover, we can assess the event's impact on these flows, that the two flows converge to a lower rate after around 100 μs, and a new burst flow causes a further rate reduction. The detailed experimental settings of the above case are introduced in §7.



(a) Congestion location (b) Congestion duration

(c) Event replay by WaveSketch

Figure 10: Congestion event detection and replay

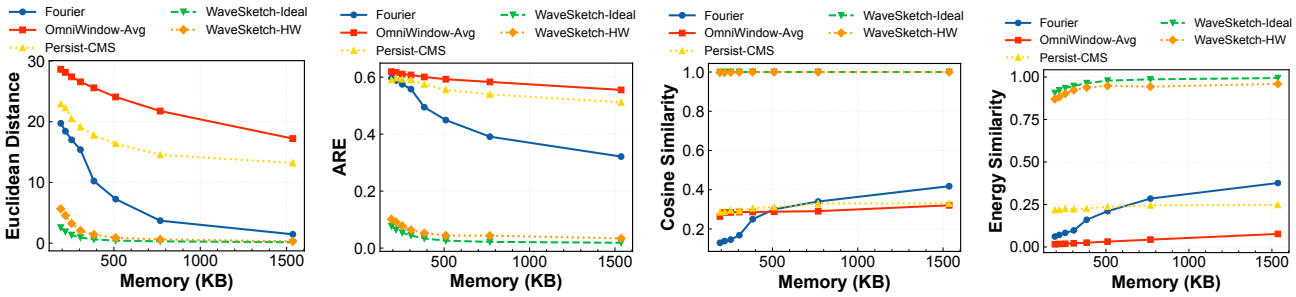
Guide fine-grained traffic scheduling and simulations. Existing technologies support microsecond-level traffic scheduling [46, 61]. With the fine-grained flow rates and congestion information, μMon can guide the scheduling process, improving system resource utilization efficiency. Besides, the microsecond-level measurements provide a more accurate understanding of traffic patterns, allowing us to create simulation models that closely mirror real-world scenarios.

7 EVALUATION

We evaluate μMon in a testbed and simulation environment and conduct extensive experiments to assess its performance and resource overhead. We summarize the experimental results based on the two main functions of μMon:

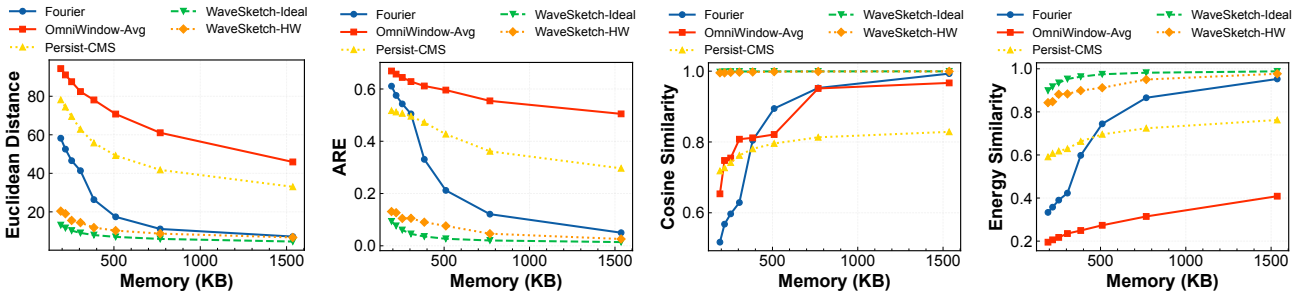
- **μFlow measurements at hosts:** In a typical scenario, WaveSketch can achieve 3.5-57x better accuracy than baseline solutions across four metrics under a window granularity of 8.192 μs. Even with 1/8 of the memory usage, WaveSketch still maintains higher accuracy. Each host requires around 5 Mbps bandwidth to achieve less than 10% ARE and more than 90% energy similarity in flow rate measurements. The hardware version of WaveSketch achieves close accuracy and can be implemented on PISA architecture chips with moderate overhead.
- **μEvent detection at switches:** For congestion events exceeding the ECN KMax threshold, using a sampling rate of 1/64, μMon can achieve a 99% recall ratio with 31-82 Mbps bandwidth per switch, with main flows captured. Increasing the sampling rate can further reduce bandwidth overhead but at the expense of decreasing the recall of congestion below the ECN KMax threshold. The ACL-based mirroring and sampling methods can be implemented on commodity switches.

Setup. In the testbed, we evaluate μMon on an Arista DCS-7060CX switch, a Tofino2 switch, and two Intel Xeon E5-2650 servers. They are connected with 40 Gbps links. We implement WaveSketch on CPUs and ASICs platforms using C++ and P4 language, respectively. We also prototype μMon in a fat-tree topology (k=4) in NS-3, with a 100 Gbps link bandwidth and 1 μs per-hop latency. In the simulation,



(a) Euclidean Distance (b) ARE (c) Cosine Similarity (d) Energy Similarity

Figure 11: Accuracy evaluation on the 15%-load Hadoop workload. The window size is 8.192 µs.



(a) Euclidean Distance (b) ARE (c) Cosine Similarity (d) Energy Similarity

Figure 12: Accuracy evaluation on the 25%-load WebSearch workload. The window size is 8.192 µs.

we can obtain the ground truth for all traffic characteristics and network events, which are utilized to evaluate the performance of µMon under network-wide deployments.

Workloads. We use two types of workloads. The first workload is real RDMA flows collected from our testbed. We utilize the perftest tool [5] to generate RDMA flows configured with a default DCQCN algorithm. The second type workload is collected in the simulation running 20-ms WebSearch [8] and Facebook Hadoop [48] workloads under 15%, 25%, and 35% link load, containing 356-11773 flows. Detailed workload information (*e.g.*, level of burstiness, flow arrival times) is introduced in Appendix D.

7.1 Evaluation on µFlow Measurements

Baseline. We compare WaveSketch against algorithms that incorporate compression techniques on flow counter sequences, including Persist-CMS [60], OmniWindow [53], and Fourier transform scheme [44]. For OmniWindow, we allocate m sub-windows based on a given memory size and the bit-width of counters. Due to limited memory space, each sub-window is coarser than the microsecond-level window. We consider the rate of each sub-window to be the rate of all microsecond-level windows within the sub-window and named the solution OmniWindow-Avg. Note that only WaveSketch and OmniWindow-Avg are suitable for data-plane implementation.

Parameter Setting. In the accuracy evaluation, we use an 8.192 µs observation window. The reason is that it can easily get the window ID from the nanosecond-level hardware timestamp by right-shifting 13 bits. For a general workload, D is usually set to 3-5 for sketching algorithms [55, 70], and W depends on the number of concurrent flows in a window. The wavelet parameters L and

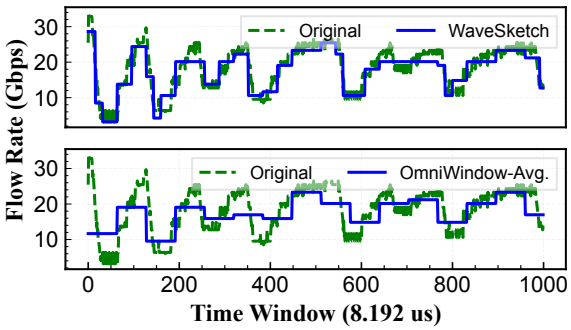


Figure 13: Reconstruction with the same memory.

K are set according to the flow sequence length and the desired compression rate. In this paper, we set WaveSketch with $D = 3$, $W = 256$. Since most traffic is finished in the tens of milliseconds range, this results in a counter sequence with a maximum length n of around 500-10000. We set $L = 8$ as a trade-off between compression rate and resource cost, and set, while K is set based on the given memory size (*e.g.*, 32-256). Longer flows are handled in multiple reporting periods of WaveSketch.

Metrics: We adopt multiple metrics, including Euclidean Distance, Cosine Similarity, Energy Similarity, and Average Relative Error (ARE), to evaluate the similarity between original and estimated flow rate curves. The formulas of the metrics are introduced in Appendix E. For a workload containing many flows, we use the above metrics for each flow and calculate the average value as the metric of the workload.

Table 1: Resource usage of a full version WaveSketch with a heavy part (h=256, L=8, K=64) and a light part (w=256, L=8, K=64, D=1).

Resource	Usage	Percentage
Exact Match Input xbar	248	12.11%
Hash Bit	752	11.3%
Gateway	29	11.33%
SRAM	134	10.31%
Map RAM	98	12.5%
VLIW Instr	75	14.65%
Stateful ALU	49	76.56%

Accuracy on the network-wide simulation workloads. As shown in Figure 11 and Figure 12, WaveSketch performs better on all four metrics. The advantage is more evident with a smaller memory. In the 15%-load Hadoop workload, using 200 KB memory, WaveSketch improves accuracy on ARE by 7.69-8.07x, energy similarity by 4.17-57.02x, and cosine similarity by 3.5-7.73x, compared to the baseline solutions. Besides, the accuracy of hardware-version WaveSketch (WaveSketch-HW) is close to the ideal version. More accuracy results can be found in Appendix F.

Measurement fidelity on testbed flow behaviors. As shown in Figure 13, we compare the reconstruction performance of WaveSketch and OmniWindow-Avg on a single RDMA flow. We set K to 32 for WaveSketch and allocated the same amount of memory for OmniWindow-Avg. We find WaveSketch can focus on the most dramatic part of rate changes, while OmniWindow-Avg easily loses some key information, such as peaks and sharp drops.

Bandwidth usage. The bandwidth overhead is around 80 Mbps in our simulation environment with 16 hosts when uploading 200-KB WaveSketch every 20 ms. On average, it is about 5 Mbps per host. If the topology scale does not significantly affect a single host's traffic scale, this result applies to larger-scale topologies. Using the same workload, we compare the bandwidth cost of WaveSketch with Valinor [49] and Lumina [66]. With head-only mirroring of 64B per packet, their average bandwidth is around 1.98 Gbps per host. As for the task on measuring microsecond-level flow rate curves, WaveSketch achieves less than 10% average relative error and more than 99% cosine similarity while utilizing 0.253% of the bandwidth required by the solutions above.

Hardware resource occupancy. We implement WaveSketch-HW on a Tofino2 chip, which verifies the WaveSketch-HW can measure traffic at line rate on a programmable NIC with a similar architecture. Table 1 presents the hardware resource usage of a full version WaveSketch. Overall, the WaveSketch hardware version utilizes a moderate amount of resources. Notably, the Stateful ALU (SALU) is the most consumed component, accounting for 76.56% of the usage. This is because each variable in a bucket requires a separate SALU. Fortunately, increasing the number of buckets (W) and retained coefficients (K) does not result in an increased SALU usage.

7.2 Evaluation on μEvent Detection

Settings. We run RoCEv2 traffic with the DCQCN congestion control algorithm enabled. The parameters of the DCQCN algorithm remain consistent with the original paper [75]. We enable ECN marking on switches with the KMin threshold set to 20 KiB, the KMax threshold set to 200 KiB, and the maximum marking probability set to 0.01.

Recall of transient congestion events. We evaluated event capture with different sampling rates. As shown in Figure 14a-14c, the more severe the congestion (*i.e.*, larger maximum queue lengths), the higher the probability of the event being captured. When queue congestion exceeds the KMax threshold, even with a sampling probability of 1/64, the recall rate can still reach 99.2%.

Coverage of event participants. As shown in Figure 14d-Figure 14f, we can collect more flows as congestion increases. In 35%-load WebSearch workload, for congestions exceeding the ECN KMax threshold, we collect an average of 11.1 flows under 1/64 sampling rate, which covers 73% of flows compared to no sampling, sufficient for events replay as presented in § 6.2. Reducing the sampling rate does not significantly reduce the number of captured heavy flows because they usually have more packets. However, the increase in the proportion of small flows will reduce the recall rate of flows. As shown in Figure 14e and Figure 14f, for a Hadoop workload with a relatively large number of small flows, the number of flows captured at a 1/64 sampling rate is less than 50% of the actual total number of flows.

Bandwidth overhead. As shown in Figure 15, we find the bandwidth gradually decreases to 31-82 Mbps per switch when using a 1/64 sampling rate. The Hadoop workload occupies more bandwidth because its average flow length is small. Under the same load, this workload generates more flows, thereby increasing the probability of congestion.

8 DISCUSSION

Can WaveSketch be implemented on other platforms? Currently, we implement WaveSketch on CPUs and P4. Implementation for other platforms is in progress. For instance, the CPU version can be adapted to ARM-based programmable NICs [3] with some optimization efforts. Moreover, the hardware version of WaveSketch can be applied to programmable platforms like FPGA [31].

Limitations on flow rate compression. WaveSketch can achieve an effective compression ratio under the microsecond-level time granularity between 1 to 100 μs for a 100 Gbps level network. However, a time granularity that is either too coarse or too fine can diminish the effectiveness of the compression. For a too-coarse granularity (*e.g.*, 100's ms), there is not a sufficient sequence length of flow rate for compression. For a nanosecond-level granularity less than the packet sending interval, the flow rate curve will appear as discrete points, causing no waveform regularity to be captured by the wavelet transform.

Limitations on network congestion detection. μMon's congestion event capture and replay rely on collecting CE-marked packets. This method focuses more on the behavior of heavy flows during severe congestion. Due to its threshold-based and sampling strategy, μMon may omit some information regarding minor congestions and small flows. Besides, μMon replays congestion events by conducting offline collaborative analysis with the fine-grained flow rate curves measured in end hosts. It cannot precisely query the contribution of flows to queues in real-time and then take immediate control actions like ConQuest [12]. In contrast, μMon's strengths lie in performing cause/effect analysis of events over an extended time range, as presented in Figure 10c.

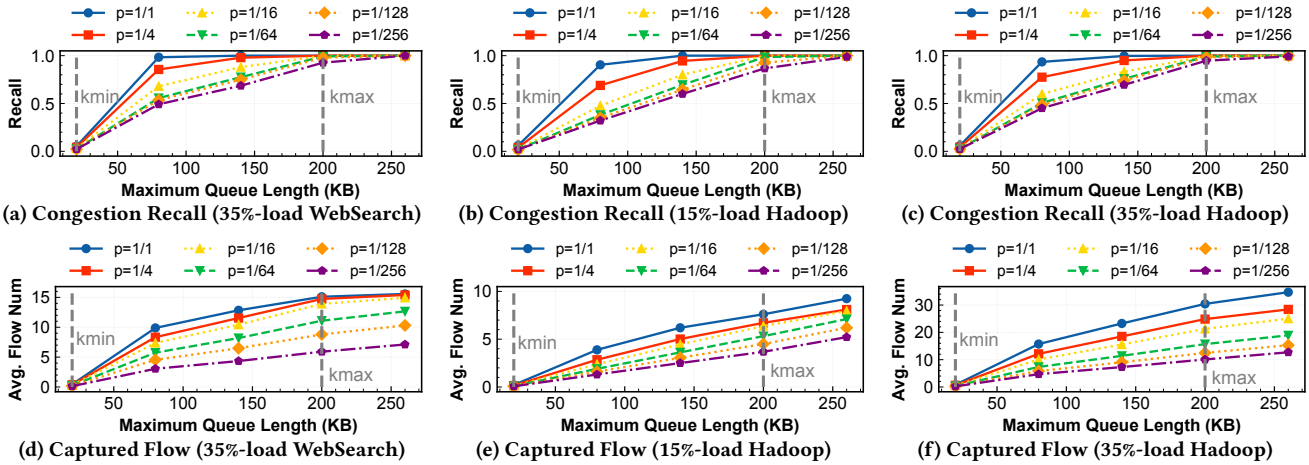


Figure 14: Recall Rate of Congestion Events and the Number of Captured Flows

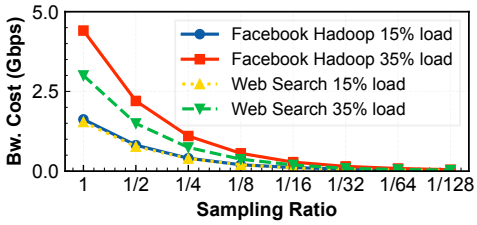


Figure 15: Max bandwidth cost per switch

Future work. We try to further reduce the resource footprint of the WaveSketch hardware version with technologies like SketchLib [40] and accelerate the CPU version with techniques like Agg-Evict [72] and SIMD instructions. Besides, we will apply μMon to capture microsecond-level characteristics in production environments.

9 RELATED WORK

Application traffic measurement. State-of-the-art measurement systems, such as Trumpet [39], OmniWindow [53], UnivMon [35], and sketching algorithms [24, 26, 27, 39, 53, 63, 71] focus on improving the measurement system's versatility, accuracy, or performance. LightGuardian [70] adopts network-wide sketch deployments and collects sketchlets in an in-band fashion to achieve the full-visibility and lightweight criteria. However, due to the lack of a time-compressing mechanism, refining their measurement window to the microsecond level results in a proportional increase in memory and bandwidth overhead. NZE sketches [25] cleverly regard sketching measurement as a compressive sensing [9] process to improve sketch accuracy in a single measurement window. However, they do not delve into multi-window temporal information and perform an actual compressing process. Persistent sketches [60] introduces the piecewise-linear approximations (PLA) method involving the half-plane intersection of two polygons, which also poses challenges for its implementation in the data plane. Valinor [49] and Lumina [66] analyze host-stack burstness by tracing all packets or *sk_buff* arriving events. Millisampler [20] captures aggregate information such as total transmitted and received bytes on a port or queue instead of tracking per-flow rate variations within μMon. The wavelet-based compression has the potential to reduce its memory usage. [64] is an orthogonal work that adjusts the sampling frequency of measurement epochs based on the Nyquist-Shannon theorem to reduce data waste. In case continuous monitoring is non-compulsory, μMon can use the sampling method to activate microsecond-level monitoring with a specific frequency.

Network event detection. Commodity switches like Cisco's Nexus 5600 and Arista's 7150S support congestion monitoring. [68] measures fine-grained network congestion through poll switch statistics on the control plane. However, they lack detailed information about the event's cause. The advent of programmable switches has enabled the monitoring of network events by directly observing queues and deploying algorithms in the data plane. INT [54]) can get per-packet queue information but introduces significant bandwidth overhead. SIMON [19] accurately senses network status at hosts. BurstRadar [29], ConQuest [12], Snappy [11] can effectively capture information about flows when congestion occurs. Mantis [65] is a framework for implementing fine-grained reactive behavior on today's programmable switches. Marple [41] and Sonata [22] compile data flow operators into programmable switches to perform telemetry tasks. However, they struggle to provide the microsecond-level rate changes, making it challenging to analyze these events' causes and effects in depth.

10 CONCLUSION

We propose μMon, a novel microsecond-level network monitoring system for data centers. μMon accurately measures microsecond-level flow rates with WaveSketch and captures transient congestion events on commodity hardware switches. The experimental results show that the proposed approaches allow us to analyze the micro-characteristics of network traffic, providing micro-scale insights into network management, both with acceptable overhead.

ACKNOWLEDGMENTS

We thank our shepherd, Soudeh Ghorbani, and the anonymous reviewers for their constructive feedback. This research is supported by the National Natural Science Foundation of China under Grant Numbers 62325205, 62072228, and 62172204.

REFERENCES

[1] [n. d.]. Configuring a User-Defined ACL - S1720, S2700, S5700, and S6720 V200R011C10 Configuration Guide - Security - Huawei. https://support.huawei.com/enterprise/en/doc/EDOC1000178177/fea6d191/configuring-a-user-defined-acl

[2] [n. d.]. Example for Configuring ACL-based Remote Traffic Mirroring - S1720, S2700, S5700, and S6720 V200R011C10 Configuration Guide - Network Management and Monitoring - Huawei. https://support.huawei.com/enterprise/en/doc/EDOC1000178174/cbfae336/example-for-configuring-acl-based-remote-traffic-mirroring

[3] [n. d.]. Intel® Infrastructure Processing Unit. https://www.intel.com/content/www/us/en/products/details/network-io/ipu/e2000-asic.html

[4] [n. d.]. Interface and Hardware Component Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 7.7.x - Configuring Traffic Mirroring [Cisco Network Convergence System 5500 Series]. https://www.cisco.com/c/en/us/td/docs/iosxr/ncs5500/interfaces/77x/b-interfaces-hardware-component-cg-ncs5500-77x/configuring-traffic-mirroring.html

[5] [n. d.]. Perftest Package. https://enterprise-support.nvidia.com/s/article/perftest-package

[6] 1985. *Network Time Protocol (NTP).* Request for Comments RFC 958. Internet Engineering Task Force. https://doi.org/10.17487/RFC0958 Num Pages: 14.

[7] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM (SIGCOMM '14).* Association for Computing Machinery, New York, NY, USA, 503–514. https://doi.org/10.1145/2619239.2626316

[8] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 conference (SIGCOMM '10).* Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[9] Richard G. Baraniuk. 2007. Compressive Sensing [Lecture Notes]. *IEEE Signal Processing Magazine* 24, 4 (July 2007), 118–121. https://doi.org/10.1109/MSP.2007.4286571 Conference Name: IEEE Signal Processing Magazine.

[10] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[11] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. 2018. Catching the Microburst Culprits with Snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks (SelfDN 2018).* Association for Computing Machinery, New York, NY, USA, 22–28. https://doi.org/10.1145/3229584.3229586

[12] 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 *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19).* Association for Computing Machinery, New York, NY, USA, 15–29. https://doi.org/10.1145/3359989.3365408

[13] Cisco. [n. d.]. Cisco Nexus 3550-F Fusion User Guide - mirrorMirror and Timestamping - Fusion [Cisco Nexus 3550-F High Precision Timestamping Switch]. https://www.cisco.com/c/en/us/td/docs/dcn/nexus3550/3550-f/sw/user-guide/cisco-nexus-3550-f-fusion-user-guide/mirror.html

[14] Benoît Claise. 2004. *Cisco Systems NetFlow Services Export Version 9.* Request for Comments RFC 3954. Internet Engineering Task Force. https://doi.org/10.17487/RFC3954 Num Pages: 33.

[15] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (April 2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[16] Mark Fedor, Martin Lee Schoffstall, Jeff D. Case, and James R. Davin. 1989. *Simple Network Management Protocol (SNMP).* Request for Comments RFC 1098. Internet Engineering Task Force. https://doi.org/10.17487/RFC1098 Num Pages: 34.

[17] Yixiao Feng, Sourav Panda, Sameer G Kulkarni, K. K. Ramakrishnan, and Nick Duffield. 2020. A SmartNIC-Accelerated Monitoring Platform for In-band Network Telemetry. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN.* 1–6. https://doi.org/10.1109/LANMAN49260.2020.9153279 ISSN: 1944-0375.

[18] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2018. Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization. 81–94. https://www.usenix.org/conference/nsdi18/presentation/geng&lang=en

[19] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. {SIMON}: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks. 549–564. https://www.usenix.org/conference/nsdi19/presentation/geng

[20] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22).* Association for Computing Machinery, New York, NY, USA, 567–580. https://doi.org/10.1145/3517745.3561430

[21] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16).* Association for Computing Machinery, New York, NY, USA, 202–215. https://doi.org/10.1145/2934872.2934908

[22] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18).* Association for Computing Machinery, New York, NY, USA, 357–371. https://doi.org/10.1145/3230543.3230555

[23] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17).* Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[24] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. SketchVisor: Robust Network Measurement for Software Packet Processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17).* Association for Computing Machinery, New York, NY, USA, 113–126. https://doi.org/10.1145/3098822.3098831

[25] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. 2021. Toward {Nearly-Zero-Error} Sketching via Compressive Sensing. 1027–1044. https://www.usenix.org/conference/nsdi21/presentation/huang

[26] Qun Huang, Haifeng Sun, Patrick P. C. Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. OmniMon: Re-architecting Network Telemetry with Resource Efficiency and Full Accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20).* Association for Computing Machinery, New York, NY, USA, 404–421. https://doi.org/10.1145/3387514.3405877

[27] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. 2019. QPipe: quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19).* Association for Computing Machinery, New York, NY, USA, 285–291. https://doi.org/10.1145/3359989.3365433

[28] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. {mTCP}: a Highly Scalable User-level {TCP} Stack for Multicore Systems. 489–502. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong

[29] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems (APSys '18).* Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3265723.3265731

[30] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. 2019. Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs. In *Proceedings of the 2019 ACM Symposium on SDN Research (SOSR '19).* Association for Computing Machinery, New York, NY, USA, 8–20. https://doi.org/10.1145/3314148.3314353

[31] Ian Kuon, Russell Tessier, and Jonathan Rose. 2008. FPGA Architecture: Survey and Challenges. *Foundations and Trends® in Electronic Design Automation* 2, 2 (April 2008), 135–253. https://doi.org/10.1561/1000000005 Publisher: Now Publishers, Inc..

[32] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M. Swift, and T. V. Lakshman. 2017. UNO: uniflying host and smart NIC offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17).* Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3127479.3132252

[33] N. Littlestone and M. K. Warmuth. 1994. The Weighted Majority Algorithm. *Information and Computation* 108, 2 (Feb. 1994), 212–261. https://doi.org/10.1006/inco.1994.1009

[34] Kexin Liu, Chen Tian, Qingyue Wang, Hao Zheng, Peiwen Yu, Wenhao Sun, Yonghui Xu, Ke Meng, Lei Han, Jie Fu, Wanchun Dou, and Guihai Chen. 2021. Floodgate: taming incast in datacenter networks. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21).* Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/3485983.3494854

[35] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16).* Association for Computing Machinery, New York, NY, USA,

101–114. https://doi.org/10.1145/2934872.2934906
[36] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 399–413. https://doi.org/10.1145/3341301.3359657
[37] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM Computer Communication Review* 45, 4 (Aug. 2015), 537–550. https://doi.org/10.1145/2829988.2787510
[38] Alistair Moffat. 2019. Huffman Coding. *Comput. Surveys* 52, 4 (Aug. 2019), 85:1–85:35. https://doi.org/10.1145/3342555
[39] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and Precise Triggers in Data Centers. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 129–143. https://doi.org/10.1145/2934872.2934879
[40] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2022. {SketchLib}: Enabling Efficient Sketch-based Monitoring on Programmable Switches. 743–759. https://www.usenix.org/conference/nsdi22/presentation/namkung
[41] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/3098822.3098829
[42] Mark R. Nelson. 1989. LZW data compression. *Dr. Dobb's Journal* 14, 10 (Oct. 1989), 29–36.
[43] Arista Networks. [n. d.]. Arista Platforms 400GbE - 100GbE - 40GbE - 25GbE - 10GbE - Arista - Arista. https://www.arista.com/en/products/platforms
[44] Henri J. Nussbaumer. 1982. The Fast Fourier Transform. In *Fast Fourier Transform and Convolution Algorithms*, Henri J. Nussbaumer (Ed.). Springer, Berlin, Heidelberg, 80–111. https://doi.org/10.1007/978-3-642-81897-4_4
[45] Joseph O'Rourke. 1981. An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM* 24, 9 (Sept. 1981), 574–578. https://doi.org/10.1145/358746.358758
[46] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3544216.3544226
[47] Matthew Roughan. 2010. A case study of the accuracy of SNMP measurements. *Journal of Electrical and Computer Engineering* 2010 (Jan. 2010), 33:1–33:7. https://doi.org/10.1155/2010/812979
[48] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/2785956.2787472
[49] Erfan Sharafzadeh, Sepehr Abdous, and Soudeh Ghorbani. 2023. Understanding the impact of host networking elements on traffic bursts. 237–254. https://www.usenix.org/conference/nsdi23/presentation/sharafzadeh
[50] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. {FlexTOE}: Flexible {TCP} Offload with {Fine-Grained} Parallelism. 87–102. https://www.usenix.org/conference/nsdi22/presentation/shashidhara
[51] M.J. Shensa. 1992. The discrete wavelet transform: wedding the a trous and Mallat algorithms. *IEEE Transactions on Signal Processing* 40, 10 (Oct. 1992), 2464–2482. https://doi.org/10.1109/78.157290
[52] Radomir S. Stanković and Bogdan J. Falkowski. 2003. The Haar wavelet transform: its status and achievements. *Computers & Electrical Engineering* 29, 1 (Jan. 2003), 25–44. https://doi.org/10.1016/S0045-7906(01)00011-8
[53] Haifeng Sun, Jiaheng Li, Jintao He, Jie Gui, and Qun Huang. 2023. OmniWindow: A General and Efficient Window Mechanism Framework for Network Telemetry. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 867–880. https://doi.org/10.1145/3603269.3604847
[54] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. 2021. In-band Network Telemetry: A Survey. *Computer Networks* 186 (Feb. 2021), 107763. https://doi.org/10.1016/j.comnet.2020.107763
[55] Lu Tang, Qun Huang, and Patrick P. C. Lee. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, Paris, France, 2026–2034. https://doi.org/10.1109/INFOCOM.2019.8737499
[56] Maroun Tork, Lina Maudlej, and Mark Silberstein. 2020. Lynx: A SmartNIC-driven Accelerator-centric Architecture for Network Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 117–131. https://doi.org/10.1145/3373376.3378528
[57] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, and Luiz F. M. Vieira. 2020. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications. *Comput. Surveys* 53, 1 (Feb. 2020), 16:1–16:36. https://doi.org/10.1145/3371038
[58] Yunlan Wang, Ming Lan, Tianhai Zhao, Zhaokui Gao, and Min Xie. 2020. Combining RTT and ECN for RoCEv2 Protocol. In *Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence (HPCCT & BDAI '20)*. Association for Computing Machinery, New York, NY, USA, 158–164. https://doi.org/10.1145/3409501.3409509
[59] Steve T. Watt, Shankar Achanta, Hamza Abubakari, Eric Sagen, Zafer Korkmaz, and Husam Ahmed. 2015. Understanding and applying precision time protocol. In *2015 Saudi Arabia Smart Grid (SASG)*. 1–7. https://doi.org/10.1109/SASG.2015.7449285
[60] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. 2015. Persistent Data Sketching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 795–810. https://doi.org/10.1145/2723372.2749443
[61] David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. 2023. Improving Network Availability with Protective ReRoute. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 684–695. https://doi.org/10.1145/3603269.3604867
[62] Kaicheng Yang, Yuhan Wu, Ruijie Miao, Tong Yang, Zirui Liu, Zicang Xu, Rui Qiu, Yikai Zhao, Hanglong Lv, Zhigang Ji, and Gaogang Xie. 2023. ChameleMon: Shifting Measurement Attention as Network State Changes. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 881–903. https://doi.org/10.1145/3603269.3604850
[63] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 561–575. https://doi.org/10.1145/3230543.3230544
[64] Nofel Yaseen, Behnaz Arzani, Krishna Chintalapudi, Vaishnavi Ranganathan, Felipe Frujeri, Kevin Hsieh, Daniel S. Berger, Vincent Liu, and Srikanth Kandula. 2021. Towards a Cost vs. Quality Sweet Spot for Monitoring Networks. In *Proceedings of the 20th ACM Workshop on Hot Topics in Networks (HotNets '21)*. Association for Computing Machinery, New York, NY, USA, 38–44. https://doi.org/10.1145/3484266.3487390
[65] Liangcheng Yu, John Sonchack, and Vincent Liu. 2020. Mantis: Reactive Programmable Switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 296–309. https://doi.org/10.1145/3387514.3405870
[66] Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, and Xin Jin. 2023. Understanding the Micro-Behaviors of Hardware Offloaded Network Stacks with Lumina. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 1074–1087. https://doi.org/10.1145/3603269.3604837
[67] Dengsheng Zhang. 2019. Wavelet Transform. In *Fundamentals of Image Data Mining: Analysis, Features, Classification and Retrieval*, Dengsheng Zhang (Ed.). Springer International Publishing, Cham, 35–44. https://doi.org/10.1007/978-3-030-17989-2_3
[68] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. Association for Computing Machinery, New York, NY, USA, 78–85. https://doi.org/10.1145/3131365.3131375
[69] Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. 2020. On-off sketch: a fast and accurate sketch on persistence. *Proceedings of the VLDB Endowment* 14, 2 (Oct. 2020), 128–140. https://doi.org/10.14778/3425879.3425884
[70] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. 2021. {LightGuardian}: A {Full-Visibility}, Lightweight, In-band Telemetry System Using Sketchlets. 991–1010. https://www.usenix.org/conference/nsdi21/presentation/zhao
[71] 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 *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 486–502. https://doi.org/10.1145/3544216.3544239

[72] Yang Zhou, Omid Alipourfard, Minlan Yu, and Tong Yang. 2018. Accelerating network measurement in software. *ACM SIGCOMM Computer Communication Review* 48, 3 (Sept. 2018), 2–12. https://doi.org/10.1145/3276799.3276800

[73] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, Pengcheng Zhang, Dennis Cai, Ming Zhang, and Mingwei Xu. 2020. Flow Event Telemetry on Programmable Data Plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 76–89. https://doi.org/10.1145/3387514.3406214

[74] Lingjun Zhu, Yifan Shen, Erci Xu, Bo Shi, Ting Fu, Shu Ma, Shuguang Chen, Zhongyu Wang, Haonan Wu, Xingyu Liao, Zhendan Yang, Zhongqing Chen, Wei Lin, Yijun Hou, Rong Liu, Chao Shi, Jiaji Zhu, and Jiesheng Wu. 2023. Deploying User-space {TCP} at Cloud Scale with {LUNA}. 673–687. https://www.usenix.org/conference/atc23/presentation/zhu-lingjun

[75] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (Aug. 2015), 523–536. https://doi.org/10.1145/2829988.2787484

[76] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 479–491. https://doi.org/10.1145/2785956.2787483

APPENDIX

Appendices are supporting material that has not been peer reviewed

A PROOF OF COEFFICIENT SELECTION METHOD

The discrete Haar wavelet transform has the following properties:

Lemma A.1. *Under one level of the inverse transform, the L2 distance from the results to the actual values equals the L2 distance from the coefficients to the actual coefficients.*

Proof. For each pair of approximation coefficient and detail coefficient, let the actual values be (α^*, δ^*) respectively. Therefore, the actual values after the inverse transform should be

$$\frac{\sqrt{2}}{2}\left(\alpha^* \pm \delta^*\right)$$

Assume the process received (α, δ) instead of the actual coefficients. The squared L2 distance between the coefficients is

$$\left(\alpha^* - \alpha\right)^2 + \left(\delta^* - \delta\right)^2$$

Since the results after the inverse transform are

$$\frac{\sqrt{2}}{2}\left(\alpha \pm \delta\right)$$

The squared L2 distance will be

$$\left(\frac{\sqrt{2}}{2}\left(\alpha^* + \delta^*\right) - \frac{\sqrt{2}}{2}\left(\alpha + \delta\right)\right)^2 + \left(\frac{\sqrt{2}}{2}\left(\alpha^* - \delta^*\right) - \frac{\sqrt{2}}{2}\left(\alpha - \delta\right)\right)^2$$

after the inverse transform. Obviously the latter can be reduced to the former, so the equality holds for all coefficients and the results. □

Corollary A.2. *In inverse transform, the L2 distance from the final results to the actual values equals the L2 distance from the coefficients to the actual coefficients.*

Proof. By Lemma A.1, the L2 distance remains the same in one level. Apply Lemma A.1 to all levels in order can prove this corollary. □

Theorem A.3. *In wavelet compression, if k coefficients are recorded while the rest are set to 0, retaining coefficients with the largest absolute value yields the smallest L2 distance between the result and the ground truth.*

Proof. For any coefficient c, there are 2 cases:

(1) c is retained. The squared L2 distance is 0 in this case.
(2) c is set to 0. The squared L2 distance is c^2 in this case.

By Corollary A.2, Minimizing the L2 distance of the result requires minimizing the L2 distance of the coefficients. For any coefficients c and c' satisfying $|c| > |c'|$, if c is set to 0 instead of c', the squared L2 distance should increase by $c^2 - c'^2$. Therefore, retaining the coefficients with the largest absolute value yields optimal results. □

B PROOF OF COMPUTATIONAL COMPLEXITY

Here, we give an amortized analysis proof of computational overhead. We are given a total of m packets, divided into n windows, each containing an average of m/n packets. In each window, the update cost for $m/n - 1$ packets is $O(1)$, while the update cost of the last packet is $O(L)$ detail coefficient computation and overhead on coefficient compression. The compression overhead depends on the number of generated detail coefficients in this window. For the l-th layer, one wavelet coefficient is generated every 2^l window. Therefore, the total number of detail coefficients d is

$$d = \frac{n}{2^1} + \frac{n}{2^2} + \cdots + \frac{n}{2^L} = n(1 - \frac{1}{2^L}) < n$$

Since each coefficient needs to go through a $log(K)$ coefficient compression process, the total cost of transformation and compression is $L \cdot n + n \cdot log(K)$. For m packets, the total update overhead is $O(m - n + L \cdot n + n \cdot log(K))$. On average, the cost of each packet update is up to:

$$O(\frac{m-n}{m} + \frac{n(L + log(K))}{m})$$

Since $n < m$, we can get the computational complexity as:

$$O(1 + \frac{n(L + log(K))}{m}) = O(1 + \epsilon(L + log(K)))$$

where $\epsilon = \frac{n}{m}$. When the network load is heavy, this means that $n \ll m$, the update overhead of the algorithm is:

$$O(1 + \frac{n(L + log(K))}{m}) \approx O(1)$$

This completes our amortized analysis.

C RECONSTRUCTION ALGORITHM

Here, we give the pseudocode of our reconstruction stage in Algorithm 2. We first process the remaining un-transformed counters, then align the entire sequence to the exponential times of 2 by padding 0 and completing the corresponding number of compressions. The reconstruction process initiates at the deepest level. The corresponding detail coefficients are upsampled to reconstruct the approximation coefficients at shallower levels. Specifically, for the

Table 2: Simulation Workloads

Workload	Websearch			Facebook Hadoop		
	15% Load	25% Load	35% Load	15% Load	25% Load	35% Load
Packets	994480	1661240	2067850	943419	1544687	2132097
Flows	367	625	815	4966	8366	11773

approximation coefficient $a_{l,i}$ and detail coefficient $d_{l,i}$ of layer l, we can derive two approximation coefficients of layer $l-1$: $a_{l-1,2i-1} = \frac{a_{l,i}+d_{l,i}}{2}$ and $a_{l-1,2i} = \frac{a_{l,i}-d_{l,i}}{2}$, where $i = 1, \ldots, \frac{n}{2^l}$. Any detail coefficients that are not preserved are considered as 0. This iterative process continues, utilizing progressively shallower-level detail coefficients until the original signal is completely reconstructed.

Algorithm 2 WaveSketch Bucket Reconstruction

```
procedure RECONSTRUCTION(w_0, A, D)
    if w_0 == 0 then
        return [0, 0, ··· , 0]                          ▷ no packets
    end if
    Transform(i, c)                                     ▷ transform the last counter
    length ← i + 1
    max_level ← ⌊log_2(length − 1)⌋
    for j ← i + 1, ..., 2^(max_level+1) do
        Transform(j, 0)                                 ▷ padding the sequence to 2^m
    end for
    for l ← 0, ..., L − 1 do
        Compress(l, _detail[l])                         ▷ compress rest coeffs
    end for
    iterate_num ← min(max_level, L − 1)
    for l ← 0, ..., iterate_num do
        _approxs ← []
        n = ⌈n/2^(l+1)⌉
        for k ← 0, ··· , n − 1 do
            a = A.pop(0)
            for each detail d in D with level l do
                _approxs.append((a+d)/2)
                _approxs.append((a−d)/2)
            end for
            if no detail is retained in this level then
                _approxs.append(a/2)
                _approxs.append(a/2)                    ▷ consider detail as zero
            end if
        end for
        A ← _approxs
    end for
    return A
end procedure
```

D WORKLOAD COLLECTION METHOD

For testbed workloads, we use a method similar to Lumina to observe the flows by time-stamping and mirroring all packets with a Tofino switch. Additionally, we generate an on-off background flow to compete with the RDMA flows to induce flow rate changes.

In the simulation workloads, we deploy applications conforming to the WebSearch and Facebook Hadoop flow size distribution in a 100 Gbpps fat-tree topology (k=4) deployed on an NS-3 simulator. The flow size distribution of the above workloads is shown in Figure 16a. We set the link load caused by the workload to 15%, 25%, and 35%, with each running for 20 ms, and define the number of flows accordingly. We then randomly distribute the flows to the hosts in the network. Then, we collect 20-ms traces of the simulation, including application packets, packets marked with ECN, and all timestamp information. Table 2 shows the number of packets and flows of each workload. In simulations, flows can lead to congestion on network links, depending on the inter-arrival time of the flows and the duration of each flow. To better understand the workloads, we collect the statistics at the interface (or port) level. Figure 16b shows the distribution of flows' inter-arrival time in a TOR switch port. The overall arrival interval of Hadoop flows is relatively short, with 20% of the flows taking less than 20 microseconds to arrive. In contrast, the overall flow arrival intervals in the WebSearch workload are longer. This is due to the fact that, at the same link load, the average flow size of the WebSearch workload is greater. In Figure 16c, these workloads result in significant congestion. Despite Hadoop flows having longer arrival intervals, their extended duration increases the likelihood of congestion. Among these workloads, 35%-load Hadoop experiences the most congestion, with the queue length exceeding 200KB 6.6% of the time.

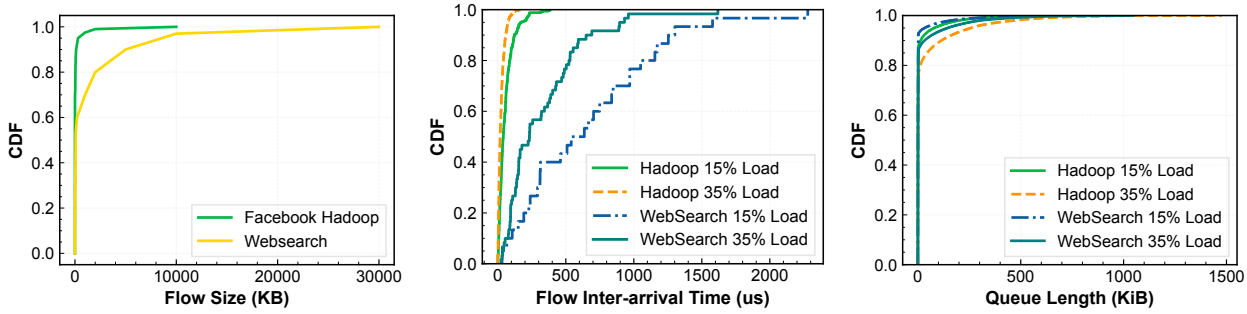
E ACCURACY METRICS

Let $f(t)$ and $\hat{f}(t)$ be the true and estimated flow rate curves for a given flow, where t ranges from 1 to n.

- *Euclidean Distance*: $\sqrt{\Sigma_{t=1}^{n}(f(t)-\hat{f}(t))^2}$. Euclidean distance is used to measure the straight-line distance between the true and estimated flow rate curves. The smaller the value of this metric, the better.
- *Cosine Similarity*: $\frac{\Sigma_{t=1}^{n} f(t)\cdot\hat{f}(t)}{\sqrt{\Sigma_{t=1}^{n} f(t)^2}\cdot\sqrt{\Sigma_{t=1}^{n}\hat{f}(t)^2}}$. Cosine similarity is a measure of similarity between the true and estimated flow rate curves, which treats them as vectors and measures the cosine of the angle between them. The closer the value of this metric is to 1, the better.
- *Energy Similarity*:

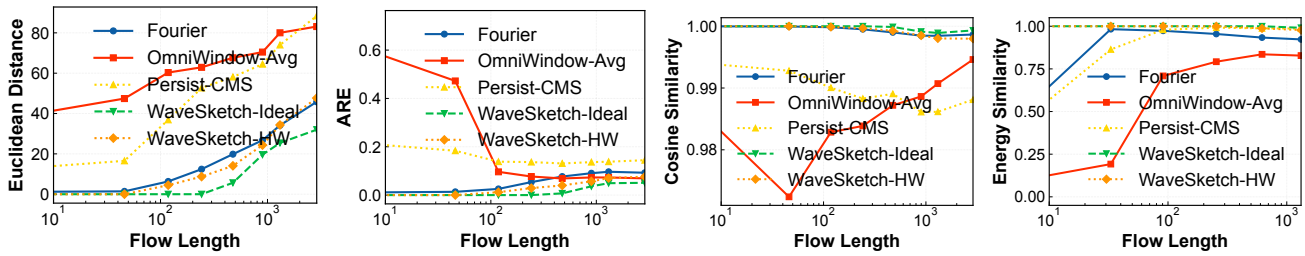
$$C(f,\hat{f}) = \begin{cases} \frac{\sqrt{\Sigma_{t=1}^{n} f(t)^2}}{\sqrt{\Sigma_{t=1}^{n}\hat{f}(t)^2}} & \text{if } \Sigma_{t=1}^{n} f(t)^2 \le \Sigma_{t=1}^{n}\hat{f}(t)^2 \\ \frac{\sqrt{\Sigma_{t=1}^{n}\hat{f}(t)^2}}{\sqrt{\Sigma_{t=1}^{n} f(t)^2}} & \text{if } \Sigma_{t=1}^{n} f(t)^2 > \Sigma_{t=1}^{n}\hat{f}(t)^2 \end{cases}$$

  Energy similarity is used to measure the energy difference between the estimated and true flow rate curves over. The closer the value of this metric is to 1, the better.
- *ARE (Average Relative Error)*: $\frac{1}{n}\Sigma_{t=1}^{n}\frac{|f(t)-\hat{f}(t)|}{f(t)}$. The closer the value of this metric is to 0, the better.



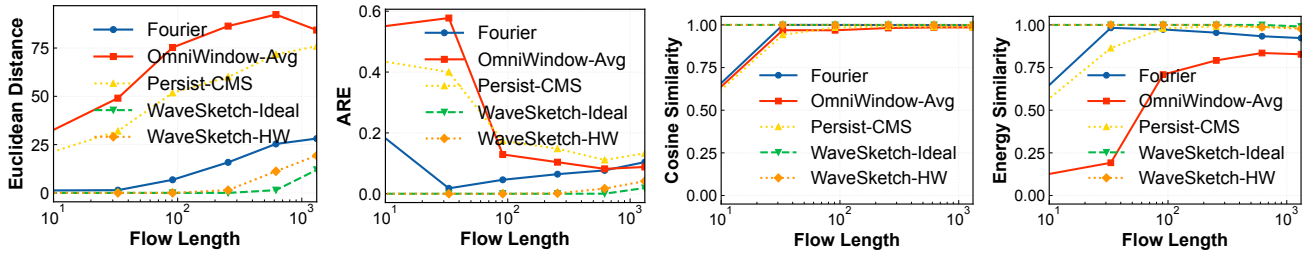
(a) Flow size distribution (b) Flow arrival time (port-level) (c) Burst level (port-level)

Figure 16: Workloads information.



(a) Euclidean Distance (b) ARE (c) Cosine Similarity (d) Energy Similarity

Figure 17: Accuracy evaluation on different flow size (WebSearch 25% load, Window Size 8.192 μs).



(a) Euclidean Distance (b) ARE (c) Cosine Similarity (d) Energy Similarity

Figure 18: Accuracy evaluation on different flow size (Hadoop 15% load, Window Size 8.192 μs).

F ACCURACY RESULTS

Figure 17 and Figure 18 shows the accuracy of WaveSketch for flows of different sizes.