Bandwidth-efficient Microburst Measurement in Large-scale Datacenter Networks

Kaihui Gao
Tsinghua University
Tsinghua Shenzhen International Graduate School

Dan Li
Tsinghua University
Zhongguancun Laboratory

Shuai Wang
Tsinghua University
Zhongguancun Laboratory

ABSTRACT

Microburst measurement is essential for diagnosing and mitigating performance problems in datacenter networks. The key is to efficiently identify the flows that contribute the most to queue buildup. However, because the existing microburst measurement systems capture packet-level information, they incur significant bandwidth overhead. We present BurstScope, a bandwidth-efficient microburst measurement system that can profile the microburst characteristics and the contributing flows. BurstScope detects the microburst-involved packets in egress pipeline, then aggregates the measurement granularity from packet level to flow level by an invertible sketch. Finally, by carefully partitioning the measurement and statistic tasks between the data and control plane, we generate only one telemetry packet for each microburst. We have implemented BurstScope on Barefoot Tofino switches. Testbed-based evaluations show that BurstScope keeps low bandwidth overhead (<0.02%) and high identification accuracy (>97%). Compared with the state-of-the-art system, BurstScope can reduce 60X bandwidth overhead.

CCS CONCEPTS
- Networks → Data path algorithms; Network monitoring; Programmable networks.

KEYWORDS
Network Measurement, Microburst, P4, Sketch

ACM Reference Format:

1 INTRODUCTION

With the quickly rising of datacenter network scale [27] and bandwidth, the applications are increasingly rigorous to network latency. For instance, elastic block storage has sub-millisecond I/O latency requirements [1–3]. A small queuing (i.e., microburst) in the networks, e.g., ∼10μs queuing delay, could severely fluctuate application performance [11]. Besides, according to the fine-grained measurement data [33], microbursts are not rare in datacenter networks (DCN).

The detecting, profiling, and understanding of microbursts can improve the visibility and reliability of DCN [8, 26] and the performance of applications [16, 17]. The crucial task is to identify the contributing flows [15] which have the most contribution to a microburst. Accurate and efficient identification of the contributing flows is critical for many purposes [5, 11, 19, 28, 35] (§2.1).

Recent researches propose to utilize the programmable data plane to measure the microburst, which are divided into two categories. For microburst-agnostic solutions, because microburst has randomness [15, 33], INT-based (In-band Network Telemetry [18]) systems [13, 36] need to insert telemetry data into each packet and analyze the contributing flows offline, which incurs significant bandwidth and processing overheads. Conversely, microburst-aware solutions, e.g., BurstRadar [15], reduce the overhead by capturing the telemetry data for only the packets involved in microbursts. However, because constructing one telemetry packet for each involved packet, BurstRadar also suffers huge network-wide bandwidth overhead, especially in large-scale datacenters, e.g., up to 30Tbps in a production 3-tier DCN (§5.1).

The root reason existing solutions have such high bandwidth overhead is that they capture and report the redundant packet-level telemetry data. We argue that the microburst measurement requires only capturing flow-level statistical information [35] and reporting only one telemetry packet.

This paper proposes a bandwidth-efficient microburst measurement system called BurstScope, which aims to capture the microburst information at the flow level. Firstly, we formally define the microburst and the contributing flows, and
design a \textit{packet mark} algorithm. Thus BurstScope could detect the microburst-involved packets, even in multi-queues. To reduce the bandwidth overhead, instead of reporting the telemetry data for all these involved packets, BurstScope aggregates the measurement information from packet level to flow level by using an invertible sketch [31]. Finally, for each microburst, BurstScope generates only one telemetry packet to report the sketch.

However, it is challenging to realize BurstScope. Firstly, operations of deriving abundant information from the sketch are not \(O(1)\), such as the top-K contributing flows, which causes them cannot be efficiently implemented in the data plane. In response, BurstScope carefully partitions the measurement and statistic tasks between the data and control plane. The data plane is responsible for flow measuring using a sketch, and the control plane is responsible for calculating and reporting the top-K contributing flows and other information from the sketch. However, there is a non-negligible delay in the communication between the data and control plane, which may overwrite the sketch. We found that deploying two alternately sketches is enough to address this problem under the highest network load.

We have implemented BurstScope on Barefoot Tofino switches and evaluated it on a 3-tier hardware testbed using real-world workloads. Evaluations show that BurstScope can keep low bandwidth overhead (<0.02%) and high identification accuracy (>97%) simultaneously. Compared with BurstRadar, BurstScope can reduce 60X bandwidth overhead.

2 BACKGROUND AND MOTIVATION

2.1 Microburst Measurement

Microburst is a surge of queuing, and the buffered packets may be blocked for a period of time, which exceeds the operator-specified threshold. This queue buildup can cause network jitter and performance degradation of applications [26]. The detecting, profiling and understanding of microbursts are critical for a wide range of purposes. \textbf{Diagnosing the root cause of congestion}. Congestion can be caused by various reasons, such as priority contention, TCP incast, and bursty UDP. Profiling the detailed flow-level information in microbursts can help diagnose the actual root cause for the congested packets, further enhancing the reliability and explainability of the DCN performance [11]. \textbf{Avoiding conflicting workloads}. Public cloud share by mutually uncooperative tenants and applications. For instance, bandwidth-hungry storage applications can cause sudden queue buildup, leading to high tail latency for latency-sensitive applications [5]. Identifying the responsible applications for microburst can help better load balance, and VM placement in datacenters.

Figure 1: Bandwidth overhead of packet-level microburst-aware measurement. \textbf{Defensing congestion-related attacks}. During a public cloud’s Shrew attack [19], a few bursty flows cause a significant transient queue buildup and impact the experience of other tenants. Quickly identifying the microburst and the contributing flows enables rapid mitigation of these attacks.

2.2 Limitations of Existing Solutions

The traditional network measurement systems [9, 23, 24] cannot detect the microburst or capture the detailed causes. Recent researches propose utilizing the programmable data planes to identify the contributing flows, which can be divided into two categories, microburst-agnostic or -aware.

(i) \textit{Microburst-agnostic measurement}. INT [18] is a network telemetry tool that is built on programmable data planes. INT-based solutions [13, 36] add telemetry metadata about switch port and queue into the packets or the mirrored packets, then analyse the microburst and the contributing flows offline. However, due to the randomness of occurrence of microbursts, they need to measure all packets to detect microbursts accurately. They consume >10% additional network bandwidth [15, 35]. Furthermore, the massive telemetry information requires complex data correlation, which costs expensive processing resources.

(ii) \textit{Microburst-aware measurement}. BurstRadar [15], the state-of-the-art system in microburst-aware measurement, captures the packet-level telemetry data for only the packets involved in microbursts. However, because it generates one telemetry packet for each involved packet, BurstRadar also suffers sizable network-wide bandwidth overhead (BWO), especially in large-scale datacenters. We can theoretically analyze its BWO according to microburst’s duration and interval, approximating to

\[
\text{BW} = \frac{\text{duration} \cdot C \cdot \theta}{\text{duration} + \text{interval}}
\]

\[
\text{BWO}(\%) = \frac{\text{BW}'}{\text{BW}} = \frac{\theta}{\text{duration} + \text{interval}}
\]

where \(\theta\) represents the proportion of the telemetry packet size in original packet size and \(C\) is the link rate. \(\text{BW}\) is the required bandwidth in a microburst period. We take \(\theta=64/1500=0.04\) as example, and show the distribution of BurstRadar’s BWO in Figure 1, based on the distributions of microburst duration and interval measured by Meta networks [33]. The median of BurstRadar’s BWO is >1%, which means that the measurement requires >25 Tbps for a network with 400 switches (6.4 Tbps).
3 DESIGN

3.1 Problem Formulation

Microburst threshold. We need the operators to specify a queuing delay threshold. When the queuing delay of a packet exceeds the threshold, it indicates that the packet experiences a microburst and the application will be affected. For example, if the network’s no-queuing RTT is 24 µs, the threshold can be 50% of the RTT, i.e., 12 µs.

Formal definition. Like the definition of microburst in many researches [8, 15], we restate the definition in terms of detection-friendly. We define the binary states (including microburst and normal) of switch ports to define the contributing flows. Assume that we could get the snapshot and an array of lengths (deqQlen[]), of the multi-queues when packet dequeuing.

Definition 1 (Microburst). When a packet with deqQlen[] and snapshot dequeuing, the port enters the microburst state if delay(deqQlen[]) > th, until all the packets in the snapshot are dequeued. (delay() converts queuing length to queuing delay based on the corresponding multi-queue scheduling policy.)

Example. Egress queues in Figure 2 is the snapshot when packet#1 dequeuing, and the port enters the microbursting state. If no new packet enters the queues during the period, the port ends the microburst state when packet#13 dequeuing, which is the last packet in the snapshot. All of packet#1-#13 are involved in the microburst.

Definition 2 (Contributing flow). Given a complete microburst: [T_s, T_e], the involved packets have dequeue time T_d ∈ [T_s, T_e]. If the number of these packets belong to flow f is C(f), then arg max C(f) are the contributing flows.

3.2 Overview

Basic idea. Our basic idea is first to detect microburst-involved packets and measure the flow-level information by a sketch [10, 14, 31] in the data plane. The information aggregating from packet level to flow level can significantly reduce bandwidth overhead and capture the necessary information for network performance diagnosing simultaneously [35].

Challenges. There are two challenges to realize our basic idea. First, identifying the microburst-involved packets in switch queues, including multi-queues, is non-trivial. Egress Queues cannot peek and mark the packets in any queue, and so the involved packets need to be marked in the egress pipelines. Second, after the packets have been marked and counted, how to reversely derive the flow IDs and statistical information in the sketch? Performing these operations is not yet supported or would hurt the performance of high-speed commodity switching ASICs [8].

Architecture. In Figure 2, we present BurstScope’s architecture which addresses the challenges. We design a packet

3.3 Microburst Packet Detection

Following the definition, we can easily obtain the microburst-involved packets if we have deqQlen[] and snapshot for each packet. However, We cannot get the snapshot. We must identify the involved packet when it enters the egress pipeline. We follow the identification algorithm in BurstRadar, and extend it to support multi-queues, which are presented in Algorithm 1. First, we could get the deqQid when each packet leaves the queues. Then, we use a pktRemaining[] array to track the remaining packets in the snapshot. We always update the pktRemaining[] when a packet’s delay(deqQlen[]) > th (lines 1-3). When a packet’s delay <= th and the corresponding pktRemaining[deqQid] > 0, the packet will also be marked as it belongs to the tail snapshot (lines 5-7). Note that the switching ASICs buffer packets using fixed-size memory buckets, or segments, i.e., pktRemaining[] is in segments, thus we transfer the packet size from bytes to the number of segments (lines 6).

3.4 Flow Measurement

After microburst packets identification, collecting them with low overhead is challenging. A naive solution is to upload these packets to the control plane, which performs preprocessing and sends the significantly reduced volume of telemetry data to the remote collector. However, because the bandwidth and processing capability of the control plane are limited [30], packets would flood the control plane. We urgently need to reduce the amount of data at the source.

To this end, BurstScope collects flow counters in the data plane. The sketch technology [10, 14] widely used in network
Algorithm 1 Packet Mark Algorithm

Require: pkt, depQlen[], depQid, th
1: if delay(depQlen[]) > th then
2:      pktsRemaining[] = depQlen[]
3:      mark(pkt)
4: else
5:     if pktsRemaining[depQid] > 0 then
6:         pktsRemaining[depQid] -= segments(size(pkt))
7:         mark(pkt)
8:     procedure delay(depQlen[])
9:     \[ t = 0; \]
10:    for Qlen in depQlen[] do \( \triangleright \) from high priority to low
11:        \( t += \text{Qlen} \times \text{SIZE}/\text{LinkRate}; \) \( \triangleright \) strict priority schedule
12:    return \( t; \)

measurement presented a promising capability to measure the flow counters with low overhead and bounded errors. Since BurstScope needs to recover the IDs of all contributing flows from the sketch, we deploy an invertible sketch, e.g., MV-Sketch [31], to implement the flow measurement. Invertibility, high accuracy, small and static memory are the goals of MV-Sketch. It is initialized as a two-dimensional array of buckets, and each bucket tracks the counter of the flows that are hashed to itself, the key (i.e., flow 5-tuple) of the candidate heavy flow and an indicator counter. By applying the majority vote algorithm (MJRRTY), MV-Sketch could track the heavy flow in an online streaming fashion. MJRRTY ensures that the actual heavy flow must be the candidate heavy flow stored by MJRRTY at the end of the stream. Consequently, using MV-Sketch, BurstScope could derive the flow-level microburst information in the data plane.

### 3.5 Result Computation & Report

The sketch stores multiple flows and counters. Analyzing the contributing flows and other useful information about microbursts needs complex computations, which are not supported in high-speed switching ASIC [15]. We install the computation into the control plane of the local switch. When the port state transforms from microburst to normal, BurstScope reports the sketch snapshot to the control plane via PCIe communications. When the control plane receives a new sketch snapshot, the CPU calculates the flow-level statistical information, including the contributing flows, duration, microburst length, traffic proportion, etc. Finally, the control plane generates a telemetry packet to a remote server.

However, there is a non-negligible delay in the reset and report of the sketch. For example, it takes milliseconds to transmit 1MB of data from the data plane to the control plane [35]. If the frequency of microbursts is greater than the rate of sketch reporting and resetting, sketch overwriting will occur, leading to the error of the contributing flows identify. BurstScope deploys two alternately sketches in egress pipelines to address this problem. When the port state transforms from microburst to normal, BurstScope reports and resets one sketch, meanwhile taking the other sketch as the ready sketch to store the next microburst.

### 4 IMPLEMENTATION

**Switch ASIC.** The Packet Mark algorithm and the invertible sketches are implemented using a sequence of match-action tables. The program first marks the involved packets in a metadata header and does not modify the original packet. Note that we use a register to track the state of the port. When the state switches, we report and reset the old sketch, meanwhile updating the pointer to the other sketch. Besides, we record the timestamp in registers to get the microburst duration. Then, since MV-Sketch needs to access some variables multiple times for each packet, we use packet recirculation. Finally, we removed the metadata header in packets before they departed this port. The sketch data structure’s size is a design choice. Using more memory can reduce collisions and improve accuracy. We can achieve over 97% accuracy using the sketch, which uses 64 KB.

**Switch CPU.** The switch control plane receives the sketch snapshot from the data plane via PCIe. Transferring 64 KB data takes about 50 μs. Since most inter-microburst times are greater than 100 μs [33], two alternately sketches are enough to handle a high network load. The CPU finds the flows with the largest counter and other burst characteristics. If multiple flows have the same largest counter, they will all be carried in the telemetry packet. Finally, we use TCP to deliver the telemetry packet to the remote server (e.g., gRPC).

**Telemetry Packet Format.** We record the following necessary information in each flow-level telemetry packet.
- Flow counters (65B): \( <5\)-tuple, counter>. Flow IDs and counters are recorded in the invertible sketch. We record the top-5 by default.
- Location (2B): \( <\text{egress port}>\). We could get the egress port from the data plane.
- Characteristics (4B): \( <\text{duration, queue length}>\).

**Discussion.** This paper provides a solution to obtain rich information from microbursts, of which specific information can be configured by operators. Note that if operators just want to get one ID of the contributing flow, this operation can be done directly on the data plane in line rate.

### 5 EVALUATION

The evaluation needs to answer the following questions. 1) Bandwidth overhead: What’s the BWO of BurstScope to deliver telemetry packets to remote server? Can it scale with the increasing datacenter size and bandwidth? 2) Accuracy: What’s the measurement accuracy of BurstScope for the contributing flows? 3) Resource utilization: What is the hardware resources required by BurstScope in the switching ASIC?
We quantify the overhead of microburst measurement in terms of the ratio of the amount of required bandwidth to the total network capacity. We run BurstScope and baselines separately to count their bandwidth overhead (BWO).

**Threshold sensitivity.** Since the frequency of microburst is related to the operator-specific queuing delay threshold, we show the BWO under different threshold in Figure 3. NetSight suffers from ~15% BWO at any threshold, because it performs per-packet telemetry. For BurstRadar, the smaller the threshold, the more telemetry data it generates, thus the more bandwidth it consumes. For instance, BurstRadar incurs ~1.2% BWO for a threshold of 5% RTT (i.e., 1.2 μs); even for a large threshold of 10% RTT, it also has 0.7% BWO. Even though the microburst frequency is smaller under a large threshold, more involved packets need to report.

By contrast, BurstScope generates only one telemetry packet (135B) for each microburst. Lower the microburst frequency, lower the BWO. For a threshold of 5% RTT, BurstScope incurs only <0.02% BWO, which is 60X less than BurstRadar. For a 6.4Tbps switch, the BWO is at most 1.3Gbps or ~1.5M microbursts per second (mps). 1.5Mmps means that a microburst occurs every ~43us on a switch with 64 ports, which represents the extreme network load.

**Processing overhead.** To further understand the scalability of BurstScope in production, we calculate the telemetry traffic as well as the processing overhead of BurstScope according to the configuration of a production datacenters.

For a 3-tier datacenter, connecting 10,000 servers requires approximately 400 switches (6.4Tbps), which produce a maximum of 400 × 6.4Tbps × 0.02% = 512Gbps telemetry traffic at most. Processing such traffic requires 6 servers with 100Gbps NICs, which implies a 0.06% processing overhead. However, if the BWO is 1.2%, like BurstRadar, it requires 307 servers, which implies a 3.07% processing overhead.

**Only one contributing flow.** We evaluate BurstScope if the operators only cares about one (i.e., top-1) contributing flow. BurstScope generates one telemetry packet (80B) directly in data plane to carry the ID and the location of the microburst queue, which consumes only ~0.01% BWO.

### 5.2 Accuracy

Due to hash collision, the contributing flows identified by BurstScope may be wrong. We evaluate the identification accuracy of BurstScope and baselines in this experiment. In order to get the ground truths for BurstScope and BurstRadar, we deployed NetSight while testing them, which captures all actual contributing flows. We define the accuracy as the proportion of times the contribution flow is correctly identified. To this end, we calculate their accuracy by comparing the identifying results of BurstScope and BurstRadar and the identifying results of NetSight. We show the accuracy under the different thresholds in Figure 4.

For BurstRadar, it tracks queuing bytes to mark packets, while ASICs only provision memory for buffered packets using fixed-size segments. For example, if the segment size is 160B and the queue consists of a single 161B packet, it will mark multiple packets with a total length of 320B. So
## Table 1: Hardware resource consumption.

<table>
<thead>
<tr>
<th>Resource</th>
<th>switch.p4</th>
<th>BurstRadar</th>
<th>BurstScope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Bits</td>
<td>32.3%</td>
<td>37.2%</td>
<td>40.5%</td>
</tr>
<tr>
<td>SRAM</td>
<td>29.8%</td>
<td>33.9%</td>
<td>34.9%</td>
</tr>
<tr>
<td>TCAM</td>
<td>28.4%</td>
<td>29.2%</td>
<td>28.9%</td>
</tr>
<tr>
<td>VLIW Actions</td>
<td>34.6%</td>
<td>39.3%</td>
<td>37.2%</td>
</tr>
<tr>
<td>Stateful ALUs</td>
<td>15.6%</td>
<td>28.2%</td>
<td>25.2%</td>
</tr>
</tbody>
</table>

Figure 5: Capacity of switch PCIe and CPU.

BurstRadar may mark extra packets towards the tail-end of a microburst. Besides, a larger microburst threshold leads to more extra packets. This is because the larger the threshold, the larger the number of packets involved in the microburst, while each packet may contribute error. Thus, the larger the threshold, the less accuracy BurstRadar has. It is 94.2% when the threshold is at 5% RTT, while it is 93.4% for 30% RTT.

On the contrary, BurstScope eliminates this error by tracking the number of queuing segments. For a threshold of 5% RTT, BurstScope’s accuracy is up to 97.1%, where the error comes from the hash collisions of the invertible sketch. For example, 6 flows hashed to a bucket. Firstly, the actual heavy flow A appeared 10 times, and then five mice flows appeared three times, respectively. So that the number of votes of A did not exceed half. Finally, the heavy flow ID in the bucket was not flow A. However, the error is bounded, and the probability of such an error decreases as the number of involved packets increases [31]. Thus, the larger the threshold, the higher accurate BurstScope is. It’s 97.4% for 30% RTT.

### 5.3 Resource Utilization

In Table 1, we compare the hardware resources required by BurstScope, BurstRadar and a production version of switch.p4. Since the computations in our packet mark are implemented in the ALUs, BurstScope consumes a relatively larger proportion (9.6%) of stateful ALUs. The SRAM is used for the exact match-action tables and two invertible sketches. BurstScope’s consumption is less than BurstRadar’s for most respects. In conclusion, the usage of all resources by BurstScope is well below 100%, which means it can easily fit on top of switch.p4.

### Module capacity

Measuring data will pass several modules, including data plane, PCIe, and switch CPU. To evaluate the capacity of the PCIe channel between pipeline and switch CPU, we fix the sketch size (64 KB) and vary #CPU cores for data processing. Figure 5a shows that the PCIe channel works at 16 Gbps or 31 Knps (a thousand microbursts per second) with 1 CPU core, and 31 Gbps or 63 Knps with 2 CPU cores, which is enough for the maximum possible microburst volume (~1.5 Mmps) of a 6.4 Tbps switch.

Switch CPU is responsible for performing the result computation and reporting. As shown in Figure 5b, one or two cores can process microbursts at 28 Knps or 60 Knps, respectively. Thus, it can handle the largest load in a switch.

### 6 RELATED WORK

The microburst measurement has been an enduring research topic. BurstScope can accurately identify the contributing flows with very little bandwidth overhead.

**The closest work:** BurstRadar [15] is the SOTA microburst monitoring system, which operates in the dataplane and captures the telemetry information for only the microburst-involved packets. We found that its per-packet monitoring approach can consume a lot of bandwidth in large-scale datacenters. Besides, if its ring buffer size is inappropriate, there is a risk of overwriting. Conversely, BurstScope aggregates telemetry data from the packet level to flow level and eliminates the risk of overwriting.

**Queue Measurement:** In recent years there has been much work on measuring queue buildups in datacenters [7, 11, 20, 28, 29, 32, 33, 35]. ConQuest [8] provides fine-grained online queue measurement entirely in the data plane. When a packet is congested, it queries the contribution for the queue buildup of the packet’s flow. If the contribution exceeds a threshold, the flow is considered a contributing flow. Obviously, its definition of contributing flow is different from that in this paper, causing that it cannot solve the problem. Snappy [7] estimates the contents of a microburst and identifies the culprit flows based on probability, which requires a large number of stages for achieving a decent recall.

**Network Monitoring:** Many INT-based network monitoring tools [13, 22, 34–36] could capture abnormal events in networks, such as congestion, loss and etc. However, they are either unable to capture the cause of the microburst (i.e., the contributing flows) [22, 35] or are expensive [13, 34, 36].

**Sketch:** Many of invertible sketches [14, 21] incur substantial memory access overhead that leads to degrade processing performance. MV-Sketch [31] is designed for fast and accurate heavy flow detection with lightweight memory access.

### 7 CONCLUSION

This paper shows that existing microburst measurement systems can incur significant overhead in large-scale datacenter networks. We propose BurstScope, a bandwidth-efficient and flow-level microburst measurement system. It detects the microburst-involved packets and counts them in an invertible sketch. Then, the control plane computes and reports the flow-level microburst information. Finally, BurstScope generates only one telemetry packet for each microburst.
ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their constructive comments. Dan Li is the corresponding author. This work is supported by the National Key Research and Development Program of China under Grant 2018YFB1800600, the National Natural Science Foundation of China (U21B2022), the Kuaishou Research Collaboration Program, Tsinghua University-China Mobile Communications Group Co.,Ltd. Joint Institute, and Tsinghua University (Department of Computer Science and Technology)-Siemens Ltd.

REFERENCES


[17] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be general and fast. In USENIX NSDI.


[29] Lu Tang, Qun Huang, and Patrick PC Lee. 2019. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In IEEE INFOCOM.


