BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks

Paper #27

ABSTRACT
Microbursts can degrade application performance in datacenters by causing increased latency, jitter and packet loss. The detection of microbursts and identification of the contributing flows is the first step towards mitigating this problem. Unfortunately, microbursts are unpredictable and typically last for 10’s or 100’s of µs. The high line rates (> 10 Gbps) in modern datacenter networks further exacerbate the problem. In this paper, we show that modern programmable switching ASICs have made it practical to detect and characterize microbursts at multi-gigabit line rates. Our system, called BurstRadar, operates in the dataplane and monitors microbursts by capturing the telemetry information for only the packets involved in microbursts. We have implemented a prototype of BurstRadar on a Barefoot Tofino switch using the P4 programming language. Our evaluation on a multi-gigabit testbed using microburst traffic distributions from Facebook’s production network shows that BurstRadar incurs 10 times less data collection and processing overhead than existing solutions. Furthermore, BurstRadar can handle simultaneous microburst traffic on multiple egress ports while consuming very few resources in the switching ASIC.

1 INTRODUCTION
Over the last decade, the performance of datacenter networks has improved significantly [28]. However, availability and service-level guarantees still continue to be challenging problems as datacenter bandwidths increase and applications become more sophisticated [30]. To achieve more 9’s in availability and service-level guarantees, we need greater visibility into the network. Modern datacenter networks operate at high-speeds (> 10 Gbps) and have ultra-low end-to-end latency (~10’s of µs) [10]. As a result, even small amounts of queuing, called microbursts, that occur for short periods of time can have a significant impact on application performance and thereby revenue [1].

Microbursts are events of intermittent congestion that last for 10’s or 100’s of µs. Microbursts can cause intermittent increase in latency, network jitter and packet loss in datacenter networks [27]. Common causes include TCP Incast [1] scenarios, bursty UDP traffic from an offending flow, TCP segment offloading or application-level batching [18]. The performance degradation arising from microbursts is becoming more common today because link speeds are moving beyond 10 Gbps while switch buffers have remained shallow. Traditionally, the impact of microbursts has been greatest for high frequency trading (HFT) applications with reported profit differentials of $100 million per year due to latency advantage of just 1 ms [20]. However, today with low end-to-end latency in datacenters and high SLA requirements by applications, the impact of microbursts is no longer limited to such niche applications. Popular webservises like LinkedIn are reported to have experienced high application latency due to microbursts [17]. To address this problem, we first need to be able to accurately detect the occurrence of these microbursts and identify the contributing flows.

The extremely low timescales make it impossible for traditional sampling-based techniques such as Netflow [7] and sFlow [25] to even detect the occurrence of microbursts. Some existing commercial solutions [6, 16, 23] are able to detect microbursts, but provide no information about the cause. Recent advances in programmable dataplanes [4] and dataplane telemetry have led to proposals for In-Band Telemetry (INT) [12, 19] that embed telemetry information into each packet and enable debugging for several network issues including microbursts. However, since microbursts are unpredictable [31], it is wasteful to use INT to monitor them as it would require the telemetry information for every single packet in the network to be captured and processed, while only a small number of packets contribute to microbursts.

In this paper, we demonstrate that programmable dataplanes can be used to detect microbursts more efficiently by capturing the telemetry information of only the packets involved in microbursts. Our system, called BurstRadar, builds on the out-of-band approach for exporting telemetry information [15]. Our key insight is that microbursts are localized to a port’s egress queue. This makes all the information required for detecting and characterizing a microburst available on a single switch. We introduce a Snapshot algorithm (§3.1) to capture the snapshot of packets involved in a microburst and use egress packet cloning (§3.2) to generate courier packets on demand.

While our approach is relatively straightforward given existing programmable dataplane architectures, we made three observations from our implementation. First, we need a strategy to temporarily store telemetry information before it can be transferred to the courier packets. Second, there is a sizable delay in the generation of courier packets and it depends on packet size among other factors. And third, it is...
possible that if there are multiple simultaneous microbursts on different egress ports, telemetry information for some packets involved might be lost. BurstRadar provisions the temporal storage by implementing a ring buffer using the transactional stateful memory available in the dataplane (§3.3). We then handle the issues of courier packet delays and multiple simultaneous microbursts by sizing the ring buffer appropriately (§3.3).

We have implemented BurstRadar on a Barefoot Tofino [24] switch and evaluated it on a multi-gigabit hardware testbed using utilization burst distributions from Facebook’s production network [31]. Our results show that even with microbursts occurring as frequently as every 200 µs, BurstRadar processes 10 times less telemetry information compared to INT [12, 19], while providing all information to fully characterize microbursts and identify contributing flows. BurstRadar captures telemetry information for all packs contributing to microbursts, even with bursts occurring simultaneously on multiple egress ports. Further, it achieves real-time detection of microbursts within tens of µs from the start of a burst.

2 RELATED WORK

Commercial solutions such as Cisco’s Nexus 5600 and 6000 series switches as well as Arista’s 7150S series switches can detect the occurrence of microbursts but provide no details about the cause [6, 23]. Learning the cause requires traffic mirroring and data correlation across different monitoring data streams [6]. In contrast, BurstRadar provides a full snapshot of telemetry information about the packets involved in a microburst. With this information, we can identify the contributing flow(s) without the significant costs associated with data correlation and traffic mirroring.

In-band Telemetry (INT) [12, 19] is a network debugging tool that is built on programmable dataplanes. INT adds network telemetry information into an additional packet header (INT header) to identify and characterize microbursts. However, since microbursts are unpredictable [31], INT would need to be enabled for all flows in order to detect and characterize microbursts reliably. INT would then need to process telemetry information for every single packet in the network, even though only a small number of these packets contribute to the microbursts. Expensive data correlation would then be required to reconstruct a microburst event. Furthermore, enabling INT on all flows would consume 10% additional bandwidth\(^1\) in the entire network due to the extra INT header. BurstRadar, on the other hand, processes telemetry information only for the packets involved in microbursts, does not require expensive data correlation and is non-intrusive to production traffic since it operates out-of-band. Marple [22] is another network monitoring system that proposes augmenting dataplane programmability with a custom key-value store hardware primitive. It presents a microburst detection case study in which microbursts are assumed to occur at regular intervals. The proposed approach will not work in practical networks because microbursts do not occur at regular intervals [31]. In any case, the hardware primitives required by Marple are not available on today’s programmable switching ASICs. BurstRadar does not make any assumption about the arrival pattern of microbursts and can be implemented on programmable switching ASICs available today.

There have been systems proposed for monitoring a different class of microbursts, called link utilization microbursts [29, 31]. Link utilization microbursts are intervals where the utilization of a link exceeds a certain threshold, and unlike queuing microbursts, might not result in queuing. These systems [29, 31] can only detect link utilization microbursts at the time scale of tens of microseconds. BurstRadar instead monitors queuing microbursts at a sub-microsecond resolution. It remains a future work to extend BurstRadar to monitor link utilization microbursts.

Mirroring or packet cloning has been previously used to generate courier packets to export telemetry information [13]. However, unlike [13], BurstRadar generates courier packets only on demand.

3 BURSTRADAR

Our key idea is to first detect a microburst in the dataplane and capture a snapshot of telemetry information of all the involved packets. This information allows queue composition analysis to identify the culprit flow(s) and burst profiling to know burst characteristics such as duration, queue build-up/drain rates, etc. Detecting a microburst is relatively easy with queuing telemetry information provided by modern programmable switching ASICs [24]. However, taking a snapshot of all the packets involved in the microburst and further exporting this information in an out-of-band manner is non-trivial for several reasons. First, the switching ASIC’s “Buffer and Queuing Engine” (BQE) does not provide any support to peek into the contents of any queue, and so the snapshot needs to be captured from outside the BQE. Second, any logic in the programmable pipelines outside the BQE can only execute on a per-packet basis. And finally, exporting the snapshot information requires on-demand generation of new courier packets in the dataplane and transferring of snapshot information to these courier packets.

Figure 1 shows the general architecture of a programmable switching ASIC. BurstRadar runs in the egress pipeline of each switch in the network. It consists of three functional

\(^1\)For a 5-hop diameter network, INT requires extra 54 bytes per packet [15] which is 10% extra for a median packet size of 500 bytes [2].
While BurstRadar can monitor all the queuing microburst events, small queuing events that cause negligible increase in application latency are generally not of interest to the operator. Therefore, BurstRadar allows the operator to specify a latency-increase threshold (specified as a percentage). For example, if the network’s no-queuing RTT is 50 µs, then the operator may specify a threshold of 30% which translates to a minimum latency increase of 15 µs. BurstRadar would then ignore any microbursts that incur less than 15 µs of delay.

We define a queue snapshot to be the set of packets present in the queue when the queue-induced delay is above the operator-specified threshold. Figure 2 shows the evolution of a toy queuing microburst at different instants in time. At time instant t=1, the queue length exceeds the threshold (dotted line). Thus the snapshot of the queue at this instant consists of packets [2,3,4]. Similarly at t=2, the queue snapshot consists of packets [3,4,5,6]. At t=3, the queue starts to drain, but we still have a queue snapshot consisting of packets [4,5,6]. At t=4 and beyond, the queue length falls below the threshold and we stop taking snapshots. In other words, a single queuing microburst event consists of multiple overlapping queue snapshots. Also notice that at t=5, an additional packet #7 enters the queue but is not a part of any of the queue snapshots.

Since egress port queues are a part of the BQE (refer Figure 1), queue snapshots are easiest to capture inside the BQE. However, the BQE in today’s programmable switching ASICs doesn’t provide any such functionality and instead provide only the queuing telemetry information (enqQdepth and deqQdepth) for each packet leaving the BQE. Our Snapshot algorithm uses this telemetry information and runs outside the BQE in the egress pipeline.

Since the egress pipeline follows a per-packet execution model, the Snapshot algorithm (Algorithm 1) needs to decide (mark) whether a packet entering the egress pipeline belongs to any queue snapshot or not. To decide if a packet should be marked, we consider the queue length when a packet is dequeued (deqQdepth). There are two possible cases: (i) The deqQdepth is greater than the threshold, or (ii) The deqQdepth is less than or equal to the threshold. In the former, it is clear that the packet belongs to at least one of the queue snapshots. For example in Figure 2, packet #2’s deqQdepth is greater than the threshold and thus packet #2 would be marked. In the latter case, our Snapshot algorithm decides if a packet is to be marked by tracking the number of bytes remaining in the queue (bytesRemaining) each time a packet reports the deqQdepth to be greater than the threshold (line 3). When the reported deqQdepth is less than or equal to the threshold, only the packets equivalent of bytesRemaining would be marked (lines 6-8). In the example, packets #5 and #6 would be marked due to

```
Algorithm 1: Queue Snapshot Algorithm

Input: threshold
Initialization: bytesRemaining = 0;

foreach pkt in egressPipeline do
    bytesRemaining = deqQdepth - size(pkt);
    if bytesRemaining > 0 then
        bytesRemaining = bytesRemaining - size(pkt);
        mark(pkt);
    end
end
```

---

2The marking is done via a metadata header and does not modify the packet in any way.
bytesRemaining set by packet #4; but packet #7 would not be marked.

3.2 Courier Packet Generation
BurstRadar generates a courier packet for each marked packet on demand. To do this, BurstRadar uses the clone egress to egress or clone_e2e primitive provided by programmable switching ASICs [9]. The clone_e2e primitive makes a copy of the exiting regular packet and places it in the egress queue of the mirror port (see Figure 1). The courier packet is also appropriately truncated to remove the original payload.

3.3 Ring Buffer
A ring buffer is designed using the transactional stateful memory available in the egress pipeline and exposed by the P4 programming language [3] as register arrays. The ring buffer acts as temporary storage for the telemetry information of marked packets until it can be copied into the courier packets.

Ring Buffer Sizing. We found that the first read from the ring buffer by a courier packet happens only after a cloning delay. During a microburst, the interval between the first and second writes to the ring buffer is mainly determined by the serialization delay of the first packet. If the serialization delay of the first packet is smaller than the cloning delay, the second packet in the microburst will write to the ring buffer before the first courier packet performs the first read. Therefore, the ring buffer needs to be large enough to store the information for the marked packets passing through the pipeline before the first read by the courier packets.

Figure 3 compares the cloning delay to the serialization delay (at 10 Gbps link speed) for packets of different sizes. For 64 byte packets, the cloning delay (270 ns) is more than five times the serialization delay (51.20 ns). This means that in the worst case of having all 64 byte packets in a microburst, more than five writes would be made to the ring buffer before the first read happens. For our ASIC implementation, we found that factors other than the packet size also affect the cloning delay. Accordingly, we found (by measurement) the required minimum ring buffer sizes for link speeds of 10 Gbps and 25 Gbps to be 26 entries (754 bytes) and 32 entries (928 bytes), respectively. These are very small requirements given the SRAM memory sizes (upto 100 MB) in today’s ASICs [21].

Concurrent Microbursts. Since the egress pipeline is shared among the ports, it serves the egress port queues in a round-robin manner. Therefore, if multiple egress ports simultaneously experience microbursts, in one scheduling round of the egress pipeline, there would be multiple writes to the ring buffer while only a single read due to a single

3.4 Implementation
BurstRadar can be implemented with small modifications to fixed function switching ASICs or programmed on modern programmable switching ASICs [5, 14, 24]. We implemented BurstRadar on a Barefoot Tofino switch [24] in about 550 lines of P4 [3] code. The operator-specified latency-increase threshold is stored in a register in the dataplane and can be dynamically configured by the control plane. The Snapshot algorithm and the ring buffer are implemented using a sequence of exact match-action tables. Arithmetic operations are facilitated by stateful ALUs.

Switching ASICs (fixed or programmable) provision memory for buffered packets using fixed size memory buckets or segments [23]. Therefore, the reported deqQdepth is in terms of number of segments. Snapshot algorithm converts the deqQdepth from segments to bytes to compute bytesRemaining (line 3 in Algorithm 1). This conversion results in excess bytesRemaining than the actual remaining bytes, causing BurstRadar to mark extra packets towards the tail-end (lines 6-8 in Algorithm 1) of a microburst.

4 EVALUATION
The evaluation of our BurstRadar prototype is centered around answering three questions. First, how efficient is BurstRadar, given that it selectively snapshots microburst queues? Second, how well does BurstRadar handle multiple simultaneous microbursts? And finally, what is the cost of BurstRadar in terms of hardware resources required in the switching ASIC?

Footnotes:
1 Investigating all factors is a matter of a separate measurement study.
2 If the segment size is 160 bytes, a single 161 byte packet in the queue reports the deqQdepth as two segments (converted to 320 bytes).
We compare BurstRadar to INT [12, 19] and to an offline Oracle algorithm. The Oracle algorithm has access to the telemetry information of all the packets and is thus able to capture queue snapshots as if they were captured by the BQE (c.f. §3.1). It represents the optimal solution.

### 4.1 Efficiency

We quantify the overhead of continuous microbursts monitoring in terms of the fraction of total number of packets that are required to be processed by the monitoring system. We compare the overhead incurred by BurstRadar, INT and the Oracle algorithm for the cache and web traffic in Figure 5. We observe in Figure 5(a) that even for a low latency-increase threshold of 5% RTT, BurstRadar is 10 times more efficient than INT. Since the RTT is approximately 25 μs in our testbed, this threshold translates to 1.25 μs of queuing delay, or 1562.5 bytes worth of queuing at 10 Gbps. We verified with our experiments that this threshold only filters out packets that are not involved in microbursts. In practice, latency sensitive applications might not require such a low threshold and therefore the overhead for BurstRadar would be even lower. Note that at a latency-increase threshold of 0% RTT, BurstRadar would be equivalent to INT as telemetry information for every single packet is processed. The efficiency result for web traffic is similar to cache traffic as shown in Figure 5(b).

**Overhead of Extra Packets.** While BurstRadar is more efficient than INT in terms of the number of packets processed, it does process a few extra packets due to the segments to bytes conversion of deqQdepth (see §3.4). The number of extra packets identified by BurstRadar depends on the packet size distribution and the segment size of the ASIC’s packet buffer. The worst case occurs when every packet is exactly one byte larger than the segment size, thereby causing each packet to occupy two segments. The best case occurs when the size of all packets is an integer multiple of the segment size.
In Figure 6, we plot the number of extra packets identified by BurstRadar compared to the Oracle algorithm for the cache workload with different packet size distributions – worst, real-world and best. For real-world, we use the cache workload’s original packet size distribution [31]. We note that while the worst case shows about 21% extra packets compared to the Oracle, the number of extra packets is typically only about 6% as shown by the real-world case. In the best case, there are no extra packets. Figure 6 also shows that a larger latency-increase threshold leads to a higher proportion of extra packets. This is because for a given packet-size distribution, larger the latency-increase threshold, larger the number of packets in the remaining queue below the threshold, with each packet contributing extra bytes to bytesRemaining. The real-world overhead for web traffic is lower than the cache traffic due to larger packet sizes [31] and is omitted because of space constraints.

4.2 Handling Concurrent Microbursts

As discussed in §3.3, multiple concurrent microbursts can result in a higher rate of writes to the ring buffer than the rate of reads. If the ring buffer size is not sufficiently large, this may result in ring buffer overwrites leading to the loss of telemetry information for some of the marked packets. Currently, no data is available on how often we should expect concurrent microbursts at different egress ports for a switch. Therefore, we simulate\(^6\) microburst traffic on multiple ports of a switch using 10-second long traces of cache and web traffic from [31]. In Figure 7, we plot the fraction of microburst packets missed by BurstRadar for different ring buffer sizes (10 to 1k entries) when microbursts occur on 2, 6 and 10 ports concurrently. With just 300 entries, BurstRadar is expected to handle 10 simultaneous microbursts (cache traffic) with a packet miss rate lower than 1%. About 1000 entries are required to reduce the miss rate to absolute 0%. This suggests that BurstRadar is resilient and can handle simultaneous microbursts with a modestly-sized ring buffer.

\(^6\)This experiment is currently not supported by our testbed due to lack of equipment to generate multiple concurrent microburst traffic.

Table 1: Hardware resource consumption of BurstRadar (ring buffer size of 1k entries) compared to the baseline switch.p4

<table>
<thead>
<tr>
<th>Resource</th>
<th>switch.p4</th>
<th>BurstRadar</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match Crossbar</td>
<td>50.13%</td>
<td>3.39%</td>
<td>53.52%</td>
</tr>
<tr>
<td>Hash Bits</td>
<td>32.35%</td>
<td>4.83%</td>
<td>37.18%</td>
</tr>
<tr>
<td>SRAM</td>
<td>29.79%</td>
<td>4.06%</td>
<td>33.85%</td>
</tr>
<tr>
<td>TCAM</td>
<td>28.47%</td>
<td>0.69%</td>
<td>29.16%</td>
</tr>
<tr>
<td>VLIW Actions</td>
<td>34.64%</td>
<td>4.69%</td>
<td>39.33%</td>
</tr>
<tr>
<td>Stateful ALUs</td>
<td>15.63%</td>
<td>12.5%</td>
<td>28.13%</td>
</tr>
</tbody>
</table>

4.3 Resource Utilization

In Table 1, we compare the hardware resources required by our BurstRadar prototype (with a ring buffer of 1k entries) to that required by the baseline switch.p4 [8]. The switch.p4 is a baseline P4 program that implements various networking features (L2/L3 forwarding, VLAN, QoS, ACL, etc.) for a typical datacenter ToR switch. We note that BurstRadar’s overall resource consumption is low for various hardware resources. BurstRadar consumes a relatively larger proportion (12.5%) of stateful ALUs as they are used for the computations in our Snapshot algorithm and for managing the ring buffer pointers. The SRAM is used for the exact match-action tables and for implementing the ring buffer. Despite the ring buffer size of 1k entries, BurstRadar’s SRAM requirements remain low. Also, the combined usage of all resources by switch.p4 and BurstRadar is well below 100%. This means that BurstRadar can easily fit on top of switch.p4.

5 CONCLUSION

Detecting microbursts in a datacenter network and identifying the contributing flows is difficult because microbursts are unpredictable and last for 10’s or 100’s of µs. BurstRadar leverages programmable switching ASICs to implement continuous and efficient monitoring of microbursts by capturing the telemetry information of only the packets involved in microbursts. Our testbed evaluation using production network traces demonstrates that BurstRadar can detect microbursts with 10 times less overhead compared to existing approaches and is resilient to simultaneous microbursts. This paper describes our BurstRadar prototype and the design decisions and considerations in dataplane packet cloning and ring buffer sizing. Our work demonstrates that modern programmable ASICs have made it practical to detect and characterize microbursts at multi-gigabit line rates in high-speed datacenter networks. BurstRadar is a work in progress and we are extending BurstRadar to detect link utilization microbursts.
REFERENCES


