Straggler-aware Observability for Flow Scheduling

Muhamad Rizka Maulana
Eindhoven University of Technology

Habib Mostafaei
Eindhoven University of Technology

Nirvana Meratnia
Eindhoven University of Technology

Abstract

Modern datacenter networks use flow scheduling to reduce flow completion time (FCT). However, schedulers without precise flow size information, like QClimb (NSDI 2024) and PIAS (NSDI 2015), can create stragglers—packets that suffer high queueing delays–undermining scheduling benefits. Current network monitoring tools find root causes of these issues but miss detecting affected "victims." In our recent work [11], we introduce StragFlow, a data plane-based system that detects stragglers in realtime at line rate and reports them to the control plane for analysis. Experiments with real-world traffic demonstrate StragFlow's effectiveness across multiple scheduling algorithms.

CCS Concepts

• **Networks** → **Network measurement**; **Network performance analysis**; **Programmable networks**; **Network monitoring**.

Keywords

Network Measurement; Flow scheduling observability;

ACM Reference Format:
Muhamad Rizka Maulana, Habib Mostafaei, and Nirvana Meratnia. 2025. Straggler-aware Observability for Flow Scheduling. In *Applied Networking Research Workshop (ANRW '25), July 22, 2025, Madrid, Spain.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3744200.3744782

1 Background

Stragglers have been widely studied in the systems community [7, 8], where they refer to tasks in a job that take significantly longer to complete than others [3]. Causes include hardware heterogeneity, resource contention, and device faults. Stragglers can delay the entire job, degrading performance, especially for small or interactive workloads [3]. Early detection enables mitigation strategies like blacklisting, speculative execution [7], or task cloning [3], which help maintain predictable job completion times.

ANRW '25, Madrid, Spain

ACM ISBN 979-8-4007-2009-3/2025/07
https://doi.org/10.1145/3744200.3744782

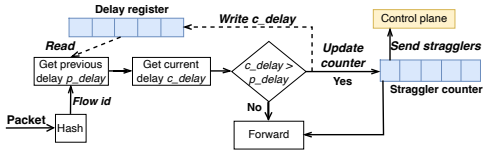


Figure 1: StragFlow's detection mechanism.

Definition: We define a straggler as a packet that suffers higher queueing delay than most other packets in its flow, thereby increasing flow completion time (FCT). Similar to task-level stragglers caused by job scheduling, flow-level stragglers can arise from flow scheduling mechanisms [2, 4, 12], which demote flows to lower-priority queues. These queues can result in longer delays, causing some packets to become stragglers and degrading overall flow performance. Schedulers like PIAS [4] and QClimb [9], which dynamically adjust priorities based on observed flow behavior without prior knowledge of exact flow sizes, can inadvertently introduce these stragglers while aiming to reduce flow completion time (FCT). Thus, we design StragFlow to better analyze the scheduling behavior of these schedulers.

2 Our Approach

The core of StragFlow's straggler detection capability is based on the use of the `deq_timedelta` metadata provided by programmable switch hardware. This field records the time (in nanoseconds) that each packet spends waiting in its queue. By monitoring this queueing delay, StragFlow can identify packets that experience unusually high delays–referred to as stragglers–and flags them for further analysis.

StragFlow operates entirely at switch egress due to availability of `deq_timedelta` in the egress pipeline. For each packet, it hashes the flow ID to index into a register storing the previous per-flow delay (*p_delay*). The current delay (*c_delay*), from `deq_timedelta`, is compared to *p_delay*. If *c_delay* exceeds *p_delay*, the packet is marked as a straggler, the delay register is updated, and a per-flow counter is incremented. While this method is lightweight and adaptive, it may flag many packets as stragglers during natural delay fluctuations. To improve accuracy, StragFlow can periodically reset or decay *p_delay*, ensuring it reflects recent network conditions. Straggler events are pushed to the control plane, providing realtime visibility into emerging queueing behaviors at packet-level granularity.

Detecting stragglers. Defining when a packet becomes a straggler is non-trivial due to fluctuating network conditions and diverse queueing behaviors across priority queues.

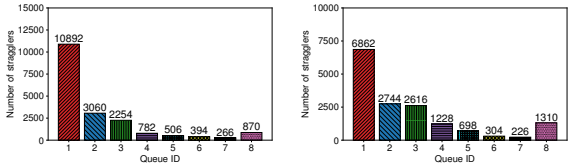
We consider three practical approaches: (i) *Threshold-based:* The simplest approach uses a static delay threshold. Packets exceeding this are labeled as stragglers. However, a single threshold may not generalize across priority queues, which can have different baseline delays. (ii) *Moving average:* This approach tracks delay trends per flow using low-pass filters (LPF). A packet is flagged if its delay exceeds the current average. While adaptive, it can be insensitive to sudden spikes or drift under gradually increasing delays, causing late or missed detection. (iii) *Previous delay comparison:* We compare the current packet's delay to a per-flow reference delay, maintained as the maximum observed delay within a recent time window or epoch. If the current delay exceeds this reference, the packet is marked as a straggler, and the record is updated. We adopt this method for its simplicity, adaptiveness, and responsiveness to sudden delay increases without requiring global thresholds or smoothing. To improve adaptivity, the reference delay can be periodically reset or decayed, allowing the system to adjust to changing network conditions and avoid desensitization due to outdated congestion episodes.

Counting stragglers. To meet line-rate processing and memory constraints in the data plane, we use compact data structures. While advanced sketches like ElasticSketch [13] and Universal Sketch [10] offer specialized features, such as traffic adaptability, hierarchical accuracy, or general-purpose measurement, we opt for the Count-Min Sketch (CMS) [6] due to its simplicity and suitability for our use case. We use CMS for the `Straggler counter` in Figure 1. When a packet is identified as a straggler, its flow ID is hashed to locate and update the corresponding counters. Non-straggler packets are not counted. This approach enables efficient tracking of straggler occurrences per flow, using minimal resources, while ensuring all packets are forwarded normally based on existing rules.

Reporting stragglers. Straggler reports can use push or pull mechanisms. In our approach, we use a push-based reporting method, where the data plane sends digests to the control plane. To prevent overload, we implement threshold-based reporting: straggler information is aggregated by flow and reported only when a specified count threshold is reached. This method effectively balances message overhead, flow coverage, and granularity of straggler detection.

3 Evaluation Results

We implemented StragFlow in P4 (740 LOC) and evaluated it on a Netberg Aurora 710 switch with Intel Tofino. We use FastClick [5] to replay CAIDA backbone traces (30M IPv4, 1.2M IPv6 packets) [1] and assessed straggler detection. In our evaluation, we set the straggler reporting threshold to two packets, enabling detection of stragglers from both large and small flows, even those with only a few packets. StragFlow uses 65K 16-bit counters in its `Straggler counter` structure with two hash functions, balancing memory usage and accuracy. For delay tracking, it allocates 65K 32-bit entries in the `Delay register`, capturing queueing delays up to 4.29 seconds. We evaluate StragFlow under two conditions: an uncongested link (1.2 Gbps egress, 1 Gbps traffic) and a balanced link (1 Gbps in both directions), measuring responsiveness to queueing dynamics.



(a) Balanced **(b) Uncongested**

Figure 2: Straggler counts observed under PIAS scheduling with eight priority queues.

We compared the scheduling behavior of FIFO and PIAS under both balanced and uncongested scenarios. In the balanced setting, PIAS resulted in 19,024 stragglers, while FIFO produced 30,034, indicating a reduction of approximately 37%. In the uncongested scenario, PIAS yielded 15,988 stragglers compared to 29,046 for FIFO. Overall, PIAS consistently reduces the number of stragglers compared to FIFO, particularly in the balanced scenario. This improvement can be attributed to PIAS's use of strict priority queues, which allow shorter flows to complete more quickly and avoid excessive queuing delays.

We now analyze PIAS behavior in more detail. In Figures 2a–2b, queue 1 consistently shows the highest number of stragglers. This is expected, as PIAS assigns all flows to the highest priority queue initially. With a 100-packet demotion threshold, queue 1 retains flows smaller than this size—roughly 97% of all flows—making it the busiest and most prone to straggler events. Straggler counts decrease across lower-priority queues since only large flows are demoted. Queues 4 through 7 see minimal straggler activity. Furthermore, we observe more stragglers in the balanced link setting than the uncongested one, due to reduced egress capacity. These results highlight that even in well-provisioned networks, queueing delays can impact small flows, which StragFlow can help detect at packet granularity.

4 Future Work

Our current design focuses on per-flow straggler detection for network flows in a single switch. As future work, we plan to extend StragFlow to support multi-hop straggler correlation, enabling end-to-end visibility across multiple switches. Furthermore, integrating StragFlow with dynamic scheduling or congestion control mechanisms could enable real-time mitigation strategies triggered by observed straggler patterns.

References

[1] 2016. The CAIDA UCSD Anonymized Internet Traces [February 2016]. https://www.caida.org/catalog/datasets/passive_dataset/

[2] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: minimal near-optimal datacenter transport *(SIGCOMM '13)*.

[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective Straggler Mitigation: Attack of the Clones. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 185–198.

[4] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA, 455–468.

[5] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 5–16.

[6] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* (April 2005), 58–75.

[7] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*.

[8] Sukhpal Singh Gill, Xue Ouyang, and Peter Garraghan. 2020. Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres. *J. Supercomput.* 76, 12 (Dec. 2020), 10050–10089.

[9] Wenxin Li, Xin He, Yuan Liu, Keqiu Li, Kai Chen, Zhao Ge, Zewei Guan, Heng Qi, Song Zhang, and Guyue Liu. 2024. Flow Scheduling with Imprecise Knowledge. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*.

[10] 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* (Florianopolis, Brazil) *(SIGCOMM '16)*. 101–114.

[11] Riz Maulana, Habib Mostafaei, and Nirvana Meratnia. 2025. Detecting Stragglers in Programmable Data Plane. In *2025 IFIP Networking Conference*. IEEE Press, 1–9.

[12] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable Packet Scheduling at Line Rate. In *SIGCOMM '16*. 44–57.

[13] 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 *(SIGCOMM '18)*. 561–575.