

RobinHood: Collaborative Burst Mitigation Through in-Network Packet Deflection

*Original*

RobinHood: Collaborative Burst Mitigation Through in-Network Packet Deflection / Pantano, L., Zilli, C., Pappone, L., Sacco, A., Marchetto, G., Esposito, F.. - ELETTRONICO. - (2025), pp. 5890-5895. (2025 IEEE International Conference on Communications, ICC 2025 Montreal, QC (CAN) 08-12 June 2025) [10.1109/icc52391.2025.11160876].

*Availability:*

This version is available at: 11583/3007590 since: 2026-02-13T13:04:15Z

*Publisher:*

Institute of Electrical and Electronics Engineers

*Published*

DOI:10.1109/icc52391.2025.11160876

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

# RobinHood: Collaborative Burst Mitigation Through In-Network Packet Deflection

Lorenzo Pantano\*, Cristian Zilli\*, Lorenzo Pappone<sup>†</sup>, Alessio Sacco\*, Guido Marchetto\*, Flavio Esposito<sup>†</sup>

<sup>†</sup> *Department of Computer Science, Saint Louis University, USA*

\* *Department of Control and Computer Engineering, Politecnico di Torino, Italy*

**Abstract**—Microbursts - microsecond-scale congestion events - are a major cause of packet loss and performance degradation in modern datacenter networks. While packet deflection techniques can help manage microbursts, current implementations lead to excessive packet reordering, exacerbated congestion under high load, and head-of-line blocking in switch buffers. In this paper, we design and implement *RobinHood*, a novel in-network burst-tolerant protocol. At its core, the protocols mechanisms and policies are based on work-stealing, a technique originally designed to reduce job completion times in operating systems. Through extensive trace-driven simulations on leaf-spine and fat-tree topologies, we show that *RobinHood* improves flow completion times up to 22% over Equal-Cost Multi-Path (ECMP), and up to 7% over recent solutions, DIBS and Vertigo, under high load scenarios.

**Index Terms**—datacenter, microbursts, work-stealing, congestion

## I. INTRODUCTION

In modern datacenters, despite the increasing speeds of disaggregated resources like GPUs, memory, and storage, the network remains a critical performance bottleneck. While significant progress has been made toward building ultra-low-latency networks, microbursts - sudden traffic surges lasting only microseconds - continue to pose major challenges. These short-lived congestion events cause the majority of packet drops and performance degradation in datacenters [1], [2]. The dropped packets trigger retransmissions that impose significant latency overhead and degrade application performance [3], [4]. Indeed, host-based solutions for managing microbursts are inherently limited due to their short lifespan [5]. End-to-end congestion control solutions like DCTCP [3] or Swift [6] can only respond after at least one Round Trip Time (RTT), which is typically longer than the microburst duration. Host-based techniques often rely on congestion signals (e.g., ECN marks, increased RTTs) to detect bursts, but the complete control loop - from congestion detection to rate adjustment - requires multiple RTTs (200-800 $\mu$ s) [7]. Given that over 70% of microbursts in production datacenters last less than 100 $\mu$ s [1], host-based congestion control operates at too coarse a timescale to effectively prevent buffer overflow during microburst events. Given the extremely short lifespan of microbursts, reactive solutions like traditional congestion control are too slow to respond effectively. This has led to the emergence of packet deflection as a promising approach, where switches redirect packets arriving at congested buffers to neighboring switches instead of dropping them [8]. However, existing packet deflection

techniques face several key limitations: for example, Naive deflection can spread congestion and increase network load by making packets take longer paths [9]. Moreover, deflected packets experience higher latency and cause extensive packet reordering that impacts transport protocols [10]. Under high load, deflection can also worsen performance by delaying congestion signals to endpoints [9]. Although promising solutions to packet deflection have been proposed, they face significant deployment limitations in modern datacenters. Most existing techniques require modifications to end-host network stacks to add metadata fields to packets [9], which becomes impractical to coordinate across thousands of servers in large-scale datacenters. Moreover, even if host modifications were feasible, many solutions rely on selective deflection schemes, that cannot be implemented in commodity switches due to hardware constraints - for example, they require complex packet scheduling [11] or arbitrary packet extraction from queues [9], [10], [12] that programmable switches do not support. These practical limitations have hindered the adoption of packet deflection despite its theoretical benefits.

Drawing inspiration from the task scheduling domain, where work-stealing has proven highly effective for microsecond-scale tasks [13], we propose *RobinHood*, a novel packet deflection approach based on work-stealing principles. We design a distributed work-stealing protocol that enables switches to systematically offload packets to underutilized neighbors. Moreover, we implement a flow-aware deflection mechanism that maintains packet ordering by ensuring all packets from the same flow are deflected to the same neighbor switch. *RobinHood* includes a simple yet effective coordination protocol that allows switches to exchange buffer occupancy information with minimal overhead, using only standard packet header fields. These contributions make *RobinHood* practical to implement in commodity programmable switches without complex hardware modifications. We evaluate *RobinHood* through extensive simulations comparing it against state-of-the-art approaches on both leaf-spine and fat-tree topologies. Our results show that *RobinHood* improves average flow completion time by 22% compared to ECMP while reducing tail latency by up to 3x compared to existing deflection schemes under high-load scenarios.

## II. RELATED WORK

Micro-bursts are increasingly recognized as a significant cause of transient congestion in data center networks. These

short-lived but impactful phenomena are difficult to detect using traditional tools like SNMP [14] or NetFlow [15] because they often do not appear in low-resolution measurements or are missed by sampling-based telemetry [1], [16]. To address this, researchers have proposed high-resolution mechanisms for detecting micro-bursts. For instance, exposing data plane states, such as switch queue states, via packet metadata [17] or INT in P4 environments [18], [19] can provide fine-grained insights, albeit with added overhead. Techniques like BurstRadar [20] and Snappy [21] mitigate this overhead by using snapshotting to minimize data collection.

Micro-burst mitigation can be broadly classified into host-based and in-network solutions. Host-based approaches often utilize Explicit Congestion Notification (ECN) to synchronize actions between network nodes and end-hosts [22]. Solutions such as MATCP [23], CEDM [24], and MBECN+ [25] build on ECN to dynamically control sender behavior and minimize congestion while preventing overreaction to fleeting events. QA-ECN [26] adapts ECN thresholds dynamically to absorb micro-bursts without excessive packet marking.

In-network and hybrid solutions have been proposed leveraging packet deflection, another approach that mitigates micro-bursts. PABO [8] and Vertigo [9] redirect packets to upstream switches or available buffers to avoid packet drops, effectively absorbing burst traffic without host intervention. Vertigo also allows packet prioritization based on flow size, reducing the impact on smaller, latency-sensitive flows. FastPass [27] offers a centralized, zero-queue approach to mitigate micro-bursts by scheduling packet transmission in a congestion-free manner. However, these solutions heavily rely on the modification of the host network stack [9] or on a centralized entity [27]. Furthermore, selective deflection – i.e., deflecting prioritized packets – has yet to find a feasible implementation in commodity network devices. [12].

In-network burst-tolerant approaches address micro-bursts locally at the switch level, allowing better absorption without requiring end-host modifications. LossPass [28] evicts packets from large flows in favor of smaller flows, aiming to minimize latency impacts. DIBS [10] takes a probabilistic approach by redirecting packets to alternative ports when buffers are full, thus preventing immediate drops.

In our proposed solution, RobinHood, we adapt the concept of work stealing from scheduling algorithms to create an in-network packet deflection mechanism that effectively addresses micro-bursts while preserving packet order within flows.

### III. ROBINHOOD: OVERVIEW AND DESIGN

We hereby present the design of RobinHood: an efficient approach based on work-stealing protocols aiming to provide micro-burst tolerance in data center networks. We define burstiness as the ratio of the peak rate of packet arrival to the average arrival rate during a period of observation. Observe that, the peak rate is defined as the ratio of the size of a churn (number of newly arriving packets) to the length of the interval over which the churn occurs, and that the average arrival rate

is defined as the ratio of the total number of packets to the total considered time.

The core idea behind our solution is to share buffer capacity among adjacent switches so that over-loaded devices can offload packets to their under-loaded neighbours. In order to realize this idea, switches need to be aware of the state of the neighbours; for this reason, we also introduce also a basic communication protocol that allows switches to exchange their load status information. In a RobinHood-enabled network, switches will broadcast their status to neighbors periodically. A careful selection of the period is of the utmost importance as it directly affects the efficiency of the mechanism: increasing its value would reduce the frequency of status updates provided by switches, thus diminishing the overhead for packet processing, but at the same time it would lead to higher response lag when congestions occur. For an optimal solution that does not introduce overhead and maximizes the number of updates, state messages are introduced in the Type of Service (ToS) field of the IP header.

Each switch maintains in its memory a list of all the neighbours eligible to accept offloaded packets in case of congestion; this list is periodically updated by the status messages. The status of a switch is determined by setting a threshold value for its buffers: the moment buffer occupancy for one of the ports exceeds this set threshold, RobinHood will activate its deflection mechanism. This makes for a more preemptive solution compared to Vertigo and DIBS, which react to congestion when packets arrive at full buffers. The value of the threshold is critical: decreasing it will activate the packet deflection mechanism of RobinHood more frequently, potentially when switches are not actually congested. This occurrence consequently leads to a deterioration of network performance, and in particular to longer flow completion times, since detoured packets are likely not to follow the shortest paths. Once a switch reaches an overloaded state, it starts deflecting packets towards one of the neighboring switches in its list; in order to avoid the issue of reordering, the deflection is performed at a flow-level, meaning that all the packets belonging to a flow are redirected towards the same switch. Additionally, other incoming flows are also deflected until the switch no longer detects congestion in the buffer.

#### A. Work Stealing Problem

In this section, we formalize the work-stealing problem in our burst-tolerant datacenter protocol. For each switch  $s$ , we define its neighborhood  $N(s)$  as the set of adjacent switches that are directly connected to  $s$  through links in  $E$ .

Let  $F_s(t)$  be the set of active flows traversing switch  $s$  at time  $t$ . When a switch becomes overloaded, it must select which flows to deflect to its neighbors based solely on locally available information. We define a deflection function  $\delta : F_s(t) \times V \rightarrow V$  that maps a flow and its current switch to a neighboring switch that will receive the deflected flow.

The work-stealing based congestion management problem can then be formulated as an optimization problem:

$$\begin{aligned}
& \text{minimize}_{\delta} \quad \sum_{f \in F} \text{FCT}(f; \delta) & (1) \\
& \text{subject to:} \quad b_s(t) \leq B_s, \quad \forall s \in V, \forall t & (2) \\
& \quad \delta(f, s) \in N(s), \quad \forall f \in F_s(t), \forall s \in V & (3) \\
& \quad \delta(f, s) = \delta(f, s'), \quad \forall p, p' \in f & (4) \\
& \quad \sum_{v \in N(s)} y_{f,s,v} = 1, \quad \forall f \in F_s(t), \forall s \in V & (5) \\
& \quad y_{f,s,v} \in \{0, 1\}, \quad \forall f \in F_s(t), \forall s \in V, \forall v \in N(s) & (6)
\end{aligned}$$

where  $\text{FCT}(f; \delta)$  represents the flow completion time of flow  $f$  as a function of the deflection decisions  $\delta$ . Constraint (2) ensures that buffer occupancy never exceeds capacity. Constraint (3) ensures that flows can only be deflected to neighboring switches. Constraint (4) enforces that all packets  $p$  belonging to the same flow  $f$  must be deflected to the same switch to maintain packet ordering. Constraint (5) ensures that each flow is either not deflected or assigned to only one neighboring switch, and Constraint (6) defines the deflection decisions as binary.

The deflection function  $\delta$  must be designed to achieve two competing objectives: (i) balancing load across switches to minimize congestion and (ii) minimizing the additional latency introduced by deviating from the original path. We implement  $\delta$  using a combination of threshold-based triggering and randomized neighbor selection. When a switch  $s$  becomes overloaded, it selects a flow from its current buffer and deflects it to a randomly chosen underloaded neighbor from  $N(s)$ . This in-network approach requires no knowledge of flow sizes or end-host coordination.

To maintain an up-to-date view of neighbor states, switches exchange their load status through a simple signaling protocol consisting of Steal Request (SR) and Steal Cancel (SC) messages. We define the set of eligible neighbors for deflection  $E_s(t) \subseteq N(s)$  as:

$$E_s(t) = \{v \in N(s) \mid b_v(t) < \theta_v\} \quad (7)$$

### B. State Exchange Protocol

RobinHood's work-stealing mechanism relies on switches exchanging state information through a lightweight signaling protocol that uses two types of control messages: Steal Request (SR) and Steal Cancel (SC). SR messages are periodically broadcast by underloaded switches to signal their availability to absorb additional traffic, while SC messages indicate a switch can no longer accept deflected packets. To minimize overhead, RobinHood integrates these messages into the standard IPv4 header by using the last bit of the ToS field: 0 for SR and 1 for SC messages. When the network initializes, each switch begins broadcasting SR messages to its neighbors if none of its buffers exceed a configured threshold. These messages populate a Candidate Switches table at each receiving switch, which tracks potential deflection targets. Each entry in this table is associated with an expiration timeout, which

is reset whenever a new SR message is received from the corresponding switch. If no SR is received before the timeout expires, that neighbor is removed from the candidate list. When a switch becomes overloaded (i.e., its buffer occupancy exceeds the threshold), it broadcasts an SC message to its neighbors and initiates the deflection process. Upon receiving an SC message, switches immediately remove the sender from their candidate lists, regardless of timeout status. While SC messages may seem redundant with the timeout mechanism, they serve a crucial role: they allow for longer timeout values, which reduces SR message frequency, helping RobinHood maintain an accurate network state with minimal control overhead. To maintain flow-level packet ordering, RobinHood performs deflection on a per-flow basis using a Deflection Table. This table tracks mappings between flows and their assigned deflection targets, ensuring all packets from the same flow are consistently deflected to the same neighbor switch. When selecting deflection targets, switches randomly choose from their available candidates in the Candidate Switches table. This procedure allows RobinHood to operate as a purely in-network work-stealing protocol that addresses microburst events through coordinated buffer management across neighboring switches.

Formally, let  $G = (V, E)$  represent our network, where  $V$  denotes the set of switches and  $E$  represents the links between switches. Each switch  $s \in V$  has a finite buffer capacity  $B_s$  and maintains a current buffer occupancy  $b_s(t)$  at time  $t$ . To enable proactive congestion management, we define a buffer occupancy threshold  $\theta_s$ , where  $0 < \theta_s < B_s$ , that triggers the work-stealing mechanism.

The state of a switch  $s$  at time  $t$  can be expressed as:

$$\sigma_s(t) = \begin{cases} \text{underloaded,} & \text{if } b_s(t) < \theta_s \\ \text{overloaded,} & \text{if } b_s(t) \geq \theta_s \end{cases} \quad (8)$$

For each switch  $s$ , we define its neighborhood  $N(s)$  as the set of adjacent switches directly connected through links in  $E$ . To maintain an up-to-date view of neighbor states, switches exchange their load status through SR and SC messages defined as follows:

- **Steal Request (SR):** Periodically broadcast by underloaded switches to signal their availability to receive deflected flows.
- **Steal Cancel (SC):** Broadcast when a switch becomes overloaded to prevent receiving additional flows.

### C. Deflection Event

RobinHood operates through a distributed work-stealing mechanism that coordinates packet deflection across switches. Figure 1 shows an example of a deflection event occurring in RobinHood. When a sender initiates a flow ①, packets traverse the network until reaching a switch experiencing buffer overflow (i.e., when the buffer occupancy is greater than a predefined threshold) ②. The switch maintains a Candidate Switches table ③ populated by previous Steal Request messages from underutilized neighbors, enabling quick

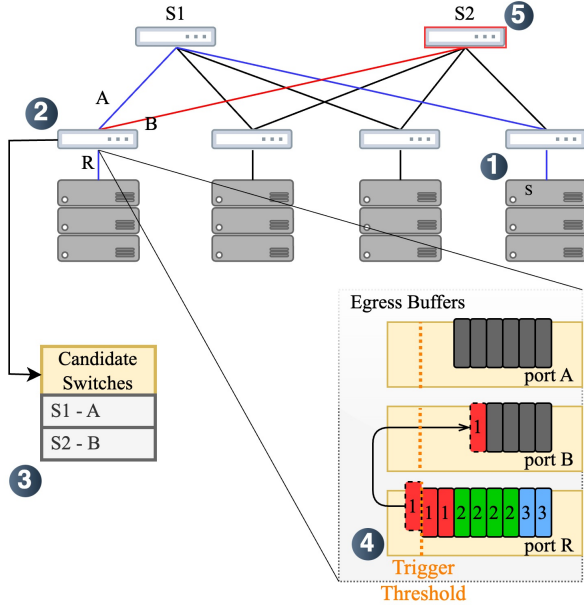


Fig. 1: Illustration of RobinHood’s work-stealing mechanism: when buffer occupancy exceeds a configurable threshold, packets from selected flows are systematically deflected to eligible neighboring switches, preventing buffer overflow while maintaining flow-level ordering.

selection of deflection targets. When the buffer occupancy exceeds the predefined threshold (4), rather than waiting for complete buffer overflow like previous approaches, RobinHood proactively initiates deflection and runs the procedure reported in Algorithm 1. A neighboring switch is randomly selected from the candidate list, and all subsequent packets from the same flow are deflected through this switch to maintain flow ordering. Specifically, in Figure 1, the switch S2 is selected, and the packet from flow 1 is deflected to output port B. All packets belonging to flow 1 will be sent to the same output port. Each candidate switch (e.g., S1 and S2 via paths A and B) already has signaled its availability to absorb additional traffic through the Steal Request protocol, ensuring the deflected packets (5) can be accommodated. RobinHood’s proactive, flow-aware approach helps prevent buffer overflow while minimizing packet reordering that commonly plagues existing deflection schemes.

#### IV. PERFORMANCE EVALUATION

We evaluate RobinHood using a custom built network simulation framework, using both a synthetic traffic generation and trace-driven scenario. Summarizing our findings:

- RobinHood achieves better throughput under various degrees of load, comparing to Vertigo, DIBS and ECMP.
- The flow completion times are improved by 22% compared to ECMP, 6% and 7% compared to Vertigo and DIBS, respectively.

##### A. Experimental Settings

We generate datacenter network topologies using the BRITE [29] topology generator, and Waxman and Barabasi-

#### Algorithm 1 RobinHood - Packet Deflection and Switch Selection

- 1:  $BS$ : Buffer Size of the switch
- 2:  $BT$ : Buffer Threshold defined by the algorithm
- 3:  $\{S_1, S_2, \dots, S_n\}$ : Set of candidate neighbors for deflection
- 4:  $DefT$ : Deflection table for tracking flow deflections
- 5: activation: Procedure starts when  $BS > BT$
- 6: **while**  $BS > BT$  **do**
- 7:    $P_i$ : Incoming packet to be processed
- 8:   **if**  $P_i$  is a Steal Cancel packet **then**
- 9:      $S_x$ : Sender of the Steal Cancel message
- 10:     remove entry for  $S_x$  from  $DefT$
- 11:   **end if**
- 12:   **if**  $P_i.FlowId$  exists in  $DefT$  **then**
- 13:     deflect  $P_i$  to the neighbor specified in  $DefT$
- 14:   **else**
- 15:      $N_{selected} \leftarrow \text{random}(\{S_1, S_2, \dots, S_n\})$
- 16:     add entry ( $P_i.FlowId, N_{selected}$ ) to  $DefT$
- 17:     deflect  $P_i$  to  $N_{selected}$
- 18:   **end if**
- 19: **end while**

Albert models for node placement. We test our model on a Fat-Tree topology with 4 1-level switches and 8 2-level switches, as in [9]. We also validate our results against a two-tier Leaf-Spine architecture with 4 core and 8 aggregate switches. We test different network loads by regulating packet and flow rates relative to switch buffer capacity. At full load, hosts send packets periodically, and the interval is randomized for each simulation run to validate results across seeds. Alternatively, the packet arrival rate is modified by adjusting inter-arrival times. In trace-driven scenarios, hosts replicate traffic patterns from [3]. We set the background traffic to 20% to maintain node activity, with variable load ranging from 30% to 80%, resulting in an aggregate full load of up to 100%.

##### B. Evaluation over Synthetic Traffic

As illustrated in Figure 2a, RobinHood delivers robust performance across different network loads. At lower network loads (around 30%), Vertigo and DIBS exhibit a roughly 25% lower average FCT compared to RobinHood. Nevertheless, RobinHood’s superior adaptability becomes evident as the network load increases, with its performance progressively surpassing the other solutions. This indicates the algorithm’s capacity to handle heavier traffic efficiently, leveraging its dynamic thresholding and congestion-aware mechanisms.

Regarding packet loss, as shown in Figure 2b, RobinHood manages to keep loss rates lower than ECMP on average and remains only slightly less effective than DIBS. Vertigo, however, achieves the lowest packet loss, particularly at higher loads. Despite this, RobinHood’s loss control remains highly effective, showcasing its capacity to limit congestion-induced packet loss while prioritizing overall network performance. We believe that the higher packet drop rate primarily stems from

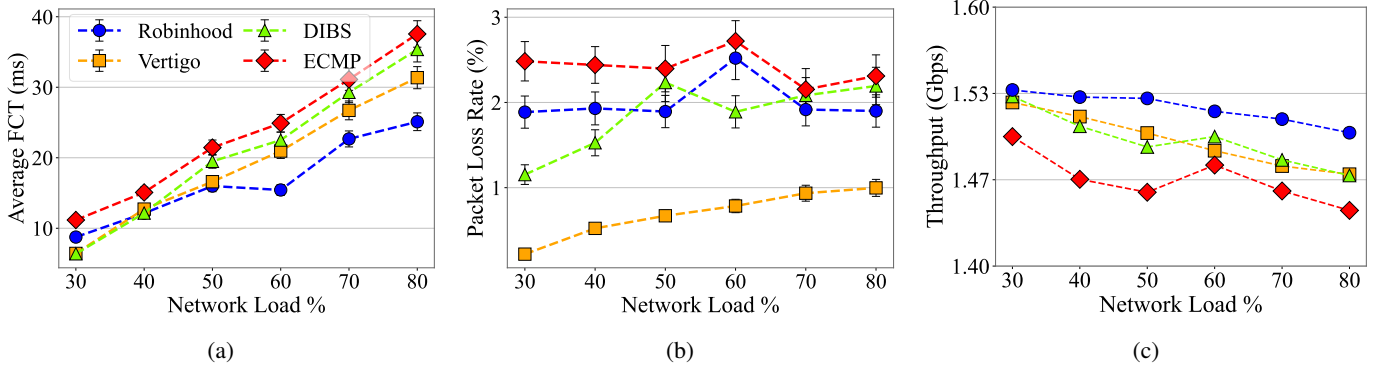


Fig. 2: Comparison between RobinHood and other state-of-the-art solutions in terms of (a) Flow Completion Time, (b) Packet Loss Rate and (c) Throughput with Synthetic traffic.

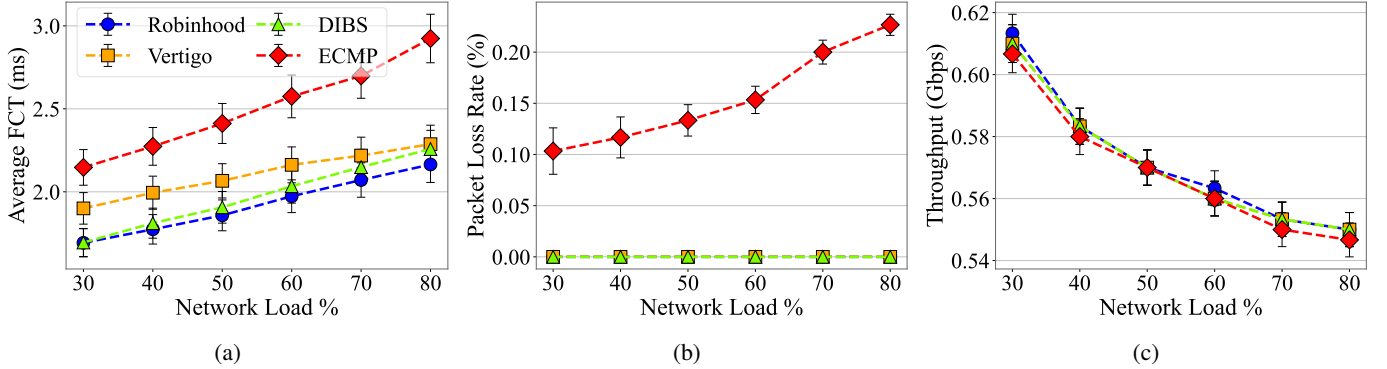


Fig. 3: Comparison between RobinHood and other state-of-the-art solutions in terms of (a) Flow Completion Time, (b) Packet Loss Rate and (c) Throughput with Trace-driven traffic.

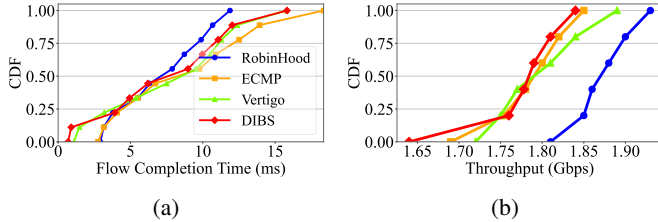


Fig. 4: RobinHood improves Flow Completion Time (FCT) and Throughput of Vertigo, ECMP and DIBS under 80% load in a fat-tree topology over trace-driven traffic.

using a fixed buffer threshold. As future work, we plan to explore adaptive thresholding techniques that can dynamically respond to varying traffic patterns, helping to prevent excessive buffer overflows and improve overall performance.

Throughput performance (Figure 2c) reveals that RobinHood outperforms the baselines, with throughput slightly degrading with increasing network loads. RobinHood shows to be competitive with state-of-the-art algorithms, proving its balanced approach to maximizing data transfer rates without compromising its core objectives of FCT reduction and efficient congestion management.

### C. Evaluation over Trace-Driven Traffic

RobinHood demonstrates robust and adaptive performance in managing realistic, trace-driven network traffic across vary-

ing load conditions, as illustrated in Figure 3. When evaluating flow completion time (FCT), RobinHood consistently outperforms other algorithms, achieving reductions in FCT of approximately 1.2%, 11%, and 22% compared to DIBS, Vertigo, and ECMP, respectively (Figure 3a). This improvement highlights RobinHood's effective load balancing and efficient congestion management strategies, particularly under high network loads. In terms of packet loss rate (Figure 3b), RobinHood maintains a loss rate comparable to DIBS and Vertigo, both of which are known for their congestion control mechanisms. The packet loss rate remains negligible across increasing network loads for these algorithms, reflecting the effectiveness of their congestion mitigation strategies. Throughput analysis in Figure 3c reveals that RobinHood delivers throughput levels that are on par with DIBS, Vertigo, and ECMP across all network load conditions. Despite its focus on minimizing FCT and packet loss, RobinHood does not compromise on throughput performance, demonstrating its balanced design that manages to maintain high efficiency.

We report a complementary analysis of cumulative distributions, highlighting RobinHood's performance advantages in FCT (Figure 4a) and throughput (Figure 4b) compared to the baselines. RobinHood significantly outperforms ECMP, Vertigo, and DIBS. Results show that RobinHood is more efficient at reducing FCT, which is highlighted by the tail of the distribution where RobinHood consistently achieves lower

FCTs, even under high load conditions.

## V. CONCLUSION

In this paper, we presented RobinHood, a novel in-network packet deflection protocol inspired by work-stealing techniques to tackle microburst congestion in modern datacenter networks. RobinHood offers an efficient mechanism for dynamically managing congestion by redistributing packet flows among neighboring switches while preserving flow-level packet ordering. Our design leverages existing programmable switch capabilities without requiring modifications of network stack at the host level or additional metadata fields.

Through comprehensive trace-driven simulations in diverse topologies, RobinHood demonstrated a substantial reduction in flow completion times and tail latency compared to state-of-the-art solutions such as ECMP, DIBS, and Vertigo. Specifically, our results show that RobinHood reduces average flow completion time by up to 22% over ECMP, and by up to 7% over DIBS and Vertigo under high-load conditions.

## ACKNOWLEDGMENT

This work has been supported by NSF awards #2201536 and #2133407.

## REFERENCES

- [1] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 78–85.
- [2] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "NAIL: A Network Management Architecture for Deploying Intent into Programmable Switches," *IEEE Communications Magazine*, vol. 62, no. 6, pp. 28–34, 2024.
- [3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.
- [4] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "Load Profiling via In-Band Flow Classification and P4 With Howdah," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 295–309, 2023.
- [5] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Micro-burst in data centers: Observations, analysis, and mitigations," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018, pp. 88–98.
- [6] G. Kumar, N. Dukkupati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," 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*, 2020, pp. 514–528.
- [7] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hppc: High precision congestion control," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 44–58.
- [8] X. Shi, L. Wang, F. Zhang, K. Zheng, and Z. Liu, "Pabo: Congestion mitigation via packet bounce," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.
- [9] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Burst-tolerant datacenter networks with vertigo," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–15. [Online]. Available: <https://doi.org/10.1145/3485983.3494873>
- [10] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "Dibs: just-in-time congestion mitigation for data centers," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: <https://doi.org/10.1145/2592798.2592806>
- [11] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 44–57.
- [12] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Practical packet deflection in datacenters," *Proceedings of the ACM on Networking*, vol. 1, no. CoNEXT3, pp. 1–25, 2023.
- [13] S. McClure, A. Ousterhout, S. Shenker, and S. Ratnasamy, "Efficient scheduling policies for Microsecond-Scale tasks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1–18. [Online]. Available: <https://www.usenix.org/conference/nsdi22/presentation/mcclure>
- [14] D. J. D. Case, R. Mundy, D. Partain, and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework," RFC 3410, Dec. 2002. [Online]. Available: <https://www.rfc-editor.org/info/rfc3410>
- [15] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Oct. 2004. [Online]. Available: <https://www.rfc-editor.org/info/rfc3954>
- [16] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.
- [17] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 3–14, 2014.
- [18] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker *et al.*, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015, pp. 1–2.
- [19] J. Bezerra, I. Brito, R. Frez, and J. Ibarra, "An adaptive and efficient approach to detect microbursts leveraging per-packet telemetry in a production network," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–6.
- [20] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "Burstradar: Practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, 2018, pp. 1–8.
- [21] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich, "Catching the microburst culprits with snappy," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, 2018, pp. 22–28.
- [22] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," Tech. Rep., 2001.
- [23] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 98–111, 2019.
- [24] D. Shan and F. Ren, "Improving ecn marking scheme with micro-burst traffic in data center networks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
- [25] J. Zhang, D. Shen, F. Dong, K. Kang, J. Jin, J. Luo, and J. Zhang, "Micro-burst aware ecn in multi-queue data centers: algorithm and implementation," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 2384–2398, 2023.
- [26] K. Kang, J. Zhang, J. Jin, D. Shen, R. Xiong, and J. Luo, "Qaecn: Dynamically tuning ecn threshold with micro-burst in multi-queue data centers," in *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2019, pp. 398–403.
- [27] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fast-pass: A centralized zero-queue datacenter network," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 307–318.
- [28] G. Kim and W. Lee, "Losspass: Absorbing microbursts by packet eviction for data center networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2717–2728, 2021.
- [29] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2001, pp. 346–353.