Adaptive Network Load Balancing at the End Host for Traffic Bursts in Data Centers

Qingyu Shi*†, Huang Huang†, Xiaocui Li*†, Chuang Li*†, Wenzhi Cao*†, Limei Liu*†
*School of Advanced Interdisciplinary Studies, Hunan University of Technology and Business, Changsha, China
†Xiangjiang Laboratory, Changsha, China

***Abstract*—The network load balancing mechanism plays a pivotal role in enhancing transmission performance in modern cloud data centers. Conventional flowlet-based approaches at host side offer a balance between performance and deployment simplicity. However, their passive load balancing strategy restricts rerouting opportunities, and lacks precision in congestion detection as it necessitates at least one round-trip time (RTT) to acquire end-to-end congestion feedback. To overcome the performance loss caused by the above limitations, we propose BurstLoader, an enhanced flowlet-based mechanism that adapts to varying traffic burst intensities and improves congestion detection accuracy. BurstLoader proactively reroutes congested flows when no new flowlets are detected, while simultaneously avoiding the rerouting of flowlets that are in good transmission states. Furthermore, BurstLoader incorporates delay and its gradient for a more nuanced and precise congestion detection. The extensive experiments demonstrate that BurstLoader achieves a significant reduction in flow completion time (FCT) by up to 48% compared to other flowlet-based solutions deployed at the end host, while maintaining competitive performance even against schemes that require custom switches under realistic workloads.**

***Index Terms*—data center networks, load balancing, burst traffic.**

I. Introduction

The escalating performance requirements for high bandwidth and low latency in cloud data center applications, particularly those involving online data-intensive services [1]–[4], pose a formidable obstacle for data center networks (DCNs). In data centers, topologies with multiple roots, like the fat-tree and leaf-spine, are commonly implemented. These designs ensure a variety of potential routing paths for communication between any pair of end hosts. Efficiently balancing traffic across these multiple paths holds the key to enhancing performance for data center applications [5]–[8]. Currently, Equal Cost Multiple Path (ECMP) forwarding [9] serves as the standard load balancing strategy in DCNs. This method employs a hash function to direct traffic across various available paths, leveraging unique packet header attributes to make these assignments. However, ECMP falls short due to hash collisions and its inability to adapt to dynamic traffic and asymmetric topologies. To address these limitations, numerous promising alternatives to ECMP have been designed. These alternatives aim to provide greater adaptability by sensing congestion, load intensity, or network asymmetry, thereby enhancing performance. By leveraging these advanced load balancing techniques, DCNs can better meet the escalating performance demands of applications.

As we know, burst traffic is typical in DCNs and has been studied extensively in load balancing [5], [10]–[13]. Flowlet-based solutions emerge as particularly effective in dividing burst traffic, thereby enhancing load balancing performance. Flowlets essentially refer to bursts of packets originating from a single flow, distinctively separated by substantial gaps. When rerouted across different paths at appropriate intervals, flowlets typically result in minimal packet reordering. Nevertheless, the generation of flowlets is largely influenced by traffic characteristics. Schemes that rely solely on flowlet switching tend to be reactive in nature, and they may not always respond promptly to congestion during periods of smooth workload. Some other solutions (e.g., RPS [14], Presto [15], DRILL [16] and AG [17]), implemented either at end hosts or switches, disseminate fixed or dynamic switching units across multiple paths in order to optimize the utilization of link resources. They can experience packet reordering under asymmetric network or require customized hardware for complex packet scheduling algorithms. Furthermore, several solutions, such as PLB [18] and Hermes [19], which employ proactive load balancing strategies. These strategies are based on a comprehensive assessment of flow status, encompassing factors like the flow size, the sending rate, and the network resource utilization ratio. They can detect and reroute flows with unnormal status (such as severe congestion and network failures). However, these schemes fail to capitalize on the opportunity of flowlet scheduling during traffic bursts, thus sacrificing potential load balancing prospects for performance improvement.

According to our observation, how to ensure efficient load balancing without customized hardware under both highly bursty and low bursty traffic is an unsolved problem. Therefore, we present BurstLoader to achieve adaptability to varying traffic burst intensities. BurstLoader monitors flow status at end hosts, facilitating the decision-making process for flowlet switching. First, BurstLoader employs various events such as retransmissions and timeouts, along with the flow's sending rate, the estimated remaining flow size based on the amount already sent, and the end-to-end latency of flows, to assess the status of the flow. The end-to-end latency in BurstLoader can be measured with sufficient precision with the help of DPDK or NIC hardware. Besides, BurstLoader reroutes new flowlets in appropriate flow status to explore more rerouting opportunities, while avoids rerouting flowlets in high transmission efficiency. In summary, we have made

three principal contributions:

- Our empirical analysis examines the limitations inherent in current load balancing schemes, revealing an important need for enhanced adaptability to varying traffic burst intensities.
- We present BurstLoader, a simple yet efficient load balancing mechanism running at end hosts to adapt to varying traffic burst intensities. BurstLoader promptly responds to congestion and failures by leveraging real-time flow status to initiate proactive rerouting when the flowlet timeout has not occurred. Meanwhile, BurstLoader performs flowlet switching only for flows in appropriate transmission state.
- We evaluate BurstLoader through extensive experiments, showing that BurstLoader achieves up to 48% better FCT than other flowlet-based solutions deployed at host side, while maintaining competitive performance even against schemes that require custom switches under realistic workloads.

In the remainder, supported by an empirical study, we show the background and our motivation in section II. Then we present our BurstLoader mechanism in section III. We evaluate BurstLoader and show the superiority of BurstLoader compared to other schemes in section IV. Finally, we briefly introduce the essential related work in section V and summarize our work in section VI.

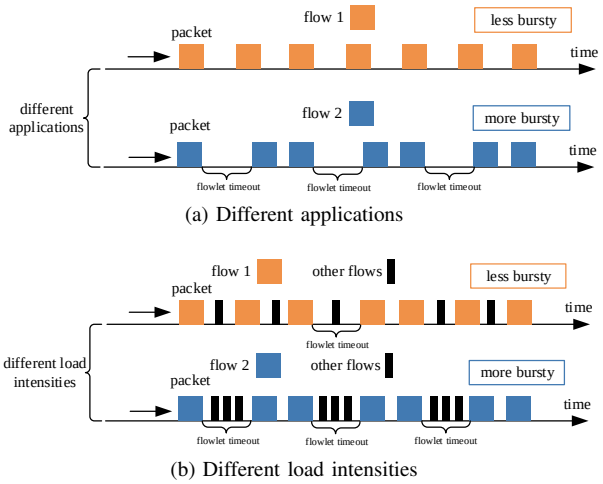
II. Background and Motivation

Data center networks usually have network asymmetries and traffic bursts [5], [11], [17], [19], [20]. In addition to the above two characteristics, we observe that data center traffic has the varying traffic burst intensities. For example, as shown in Fig. 1, the changes of traffic bursts mainly come from changes in application types and load intensities. Fig. 1a shows that the flow 2 is more bursty than the flow 1 since they come from different applications. Fig. 1b shows that the flow 2 is more bursty than the flow 1, where the inter-arrival time of adjacent packets of flow 2 increases due to network congestion caused by increased load.

Many flowlet-based load balancing mechanisms have been designed to adapt to network asymmetries and traffic bursts, which typically utilize a fixed flowlet timeout threshold to generate new flowlets [5], [11], [21]. However, the existing schemes cannot adapt to varying traffic burst intensities to provide efficient load balancing, which can be analyzed from the following two aspects.

A. Drawbacks of Existing Scheduling Policies

Flowlet switching has proven effective in minimizing packet reordering for load balancing in scenarios with asymmetric network and burst traffic [11]. Multiple solutions (e.g., CONGA [21], CLOVE [22] and LetFlow [11]) utilize flowlet switching to achieve precise load balancing. Nevertheless, the utilization of flowlet switching with a fixed flowlet timeout value presents two significant constraints:



(a) Different applications

(b) Different load intensities

Fig. 1: Varying traffic burst intensities.

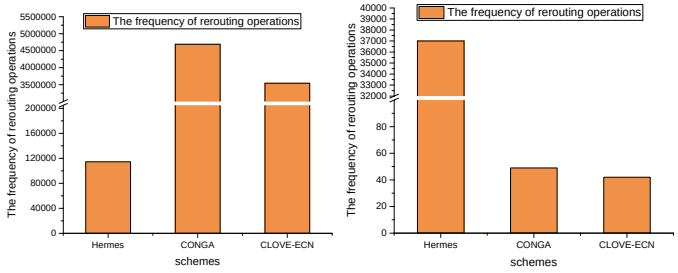
- The immediate rerouting of congested flows to alternative paths is hindered when there are limited flowlet timeout events.
- The frequent occurrence of flowlet timeout events may lead to the frequent rerouting of flows in good transmission status, exacerbating congestion mismatch and packet reordering issues [19].

Therefore, prior schemes utilizing flowlet switching falter in adapting to varying traffic burst intensities. The performance degradation caused by the incapacity to adapt to varying traffic burst intensities is experimentally demonstrated below.

We quantify the inherent shortcomings of various load balancing mechanisms in their response to the varying traffic burst intensities. We have implemented CLOVE-ECN [22], CONGA [21], and Hermes [19] within the NS3 simulation environment [23], utilizing two realistic workloads (web-search and data-mining). CLOVE-ECN distributes flowlets on multiple paths at end hosts. CONGA employs per-flowlet switching at the switch. Hermes implements load balancing in packet grain, leveraging the flow status at the end host to facilitate proactive rerouting.

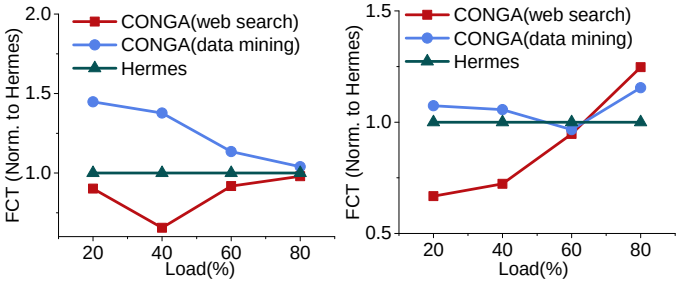
We count the frequency of rerouting operations in two realistic workloads to demonstrate the adaptability of different solutions to varying traffic burst intensities. The results are shown in Fig. 2. Given the higher burstiness of the web-search workload compared to the data-mining workload, we can observe that flowlet-based schemes trigger rerouting more frequently under the web-search workload. Conversely, Hermes exhibits a higher rate of rerouting under the data-mining workload.

Moreover, the primary performance metric employed in Fig. 3 is FCT, and for a clearer visualization of the results, we normalize the FCT relative to Hermes. As depicted in Fig. 3a, CONGA demonstrates an 8% improvement in performance over Hermes. Conversely, under the data-mining workload, Hermes outperforms CONGA by over 10%. This disparity can



(a) Web-search (b) Data-mining

Fig. 2: The number of rerouting under different workloads.



(a) Overall avg FCT (b) Small flow(<100KB) avg

Fig. 3: FCT statistics in different traffic burst intensities (normalized to Hermes).

be attributed to CONGA's ability to capitalize on traffic bursts, thereby affording it more rerouting opportunities, while Hermes does not capture new flowlets but sense congestion by monitoring flow status to balance load. The result shows that CONGA performs better than Hermes in highly bursty traffic. However, Hermes achieves better FCTs than the former in low bursty traffic. Furthermore, as Fig. 3b shows, the average FCTs for small flows exhibit a pronounced increase with CONGA as the load intensity increases. As evident from Fig. 2, under the web-search workload, the frequency of rerouting operations in CONGA is several orders of magnitude higher than that observed in Hermes due to excessive flows being broken into new flowlets, which leads to congestion mismatch and increased out-of-order packets. Therefore, current flowlet switching cannot adapt to varying traffic burst intensities. And schemes rerouting flow according to flow status cannot achieve good performance because they lack the use of flowlet rerouting opportunities.

B. Drawbacks of Existing Path Congestion Detection

Existing congestion-aware load balancing mechanisms at host side usually select the least congested or relatively uncongested path for rerouting flows [21], [22], [24]–[26], so the accurate detection of path congestion is crucial for ensuring optimal transmission performance. The congestion detection in current load balancing uses one-way delay, queuing delay, ECN or other congestion feedbacks to determine the degree of path congestion. In the following, we use

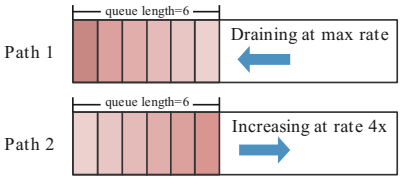


Fig. 4: The one-way delay cannot differentiate congestion degree.

absolute congestion information to refer to the class of above congestion feedbacks. However, the existing schemes lack the perception of congestion trend, so it is difficult to quickly and accurately evaluate the degree of path congestion, which leads to wrong path selection, especially under varying traffic burst intensities.

We illustrate the above problem in Fig. 4, and assume that the load balancing mechanism uses queuing delay to estimate the degree of path congestion. Two transmission paths exist with the same queuing delay in Fig. 4, but one is experiencing increased congestion, and the other is experiencing congestion relief. Current schemes do not consider the congestion trends in different paths and may choose the path where congestion is being aggravated, resulting in performance degradation. Therefore, only using queue length or delay cannot comprehensively evaluate the congestion degree of the path. The above analysis inspires us to consider combining absolute congestion information and congestion trend to improve the accuracy of congestion detection.

All in all, in the face of the varying traffic burst intensities, there exist drawbacks of scheduling policy and path congestion detection in current load balancing schemes. We will provide a detailed exposition of our design in the subsequent section, addressing the aforementioned issues.

III. Design

In this section, we elaborate on the design details of BurstLoader. This novel scheme adapts to varying traffic burst intensities for load balancing at end hosts with no switch modification, which can be implemented in the hypervisor. Fig. 5 overviews BurstLoader, which mainly consists of two modules: traffic analysis module and traffic routing module.

- **Traffic Analysis Module:** BurstLoader closely monitors flow status via various flow events (including retransmissions and timeouts), sending rates, remaining flow size, and end-to-end latency. By doing so, it is able to accurately assign a specific state to each flow. Besides, BurstLoader checks whether the flow is experiencing flowlet timeout.
- **Traffic Routing Module:** BurstLoader adopts XPath [27] to achieve explicit routing path control at host side. Besides, BurstLoader selectively and proactively reroutes flows with the bad transmission state, while rerouting flowlets that are in the appropriate transmission state.

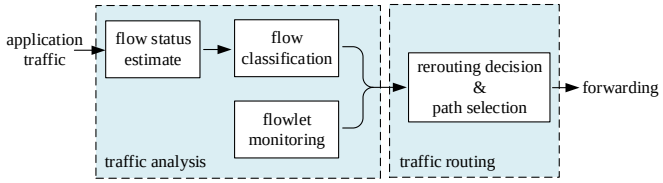


Fig. 5: BurstLoader overview.

A. Traffic Analysis Module

The traffic analysis module contains flow classification and flowlet monitoring. It gathers flow statistics by monitoring flow events, such as packet retransmissions and timeouts, as well as assessing the sending rate, the estimated remaining flow size based on the bytes already transmitted, and the end-to-end latency. BurstLoader detects flow status at packet granularity, then assigns a specific state to each flow.

The flow state contains the following four categories, which works together with the flowlet monitoring function as input data for the rerouting decision.

- **Failure state:** The TCP retransmission ratio of flow is high, or the flow has experienced many consecutive timeouts.
- **Bad state:** Flows experiencing high end-to-end delay is considered in this state.
- **Good state:** Small flows characterized by a high sending rate are in this state.
- **Intermediate state:** Flows that do not belong to the above states are in this state.

The application traffic will undergo the process of traffic analysis module and subsequently be allocated to a particular state. The load balancing algorithm of BurstLoader is shown in Algorithm 1, where the flow classification is located in lines 1-22. Firstly, in the event of network failures, such as switch and link malfunctions, or severe congestion, numerous packets are unable to reach their destination hosts promptly or are dropped within the network. Additionally, these packet losses can prompt TCP retransmissions, further compromising network performance. To address this, BurstLoader calculates the flow retransmission ratio, adopting a threshold P_t of 1% as referenced in prior research [19], for every few milliseconds. It also keeps track of the number of timeouts, setting a threshold N_t of 3, to facilitate the selective and proactive rerouting of flows (lines 3-6). By doing so, BurstLoader aims to minimize the impact of network failures and congestion on overall network performance. The rerouting procedure will select the least congested path from the other paths that remove the current path, which helps avoid transmission over high packet loss or failed network links.

Moreover, BurstLoader efficiently and proactively reroutes flows encountering network congestion to optimize traffic distribution. The improvement of FCTs through rerouting events is contingent upon various factors, including the remaining flow size and the flow sending rate. Initially, rerouting can trigger packet reordering and congestion mismatches, potentially causing a decrease in the sending rate. And, over time, the sending rate on the alternative path is observed to increment gradually in our experiments. Consequently, if the current transmission rate is already substantial, achieving significant performance improvements through rerouting becomes challenging. Furthermore, rerouting flows with a minor remaining size often yields limited benefits.

Algorithm 1 Load balancing algorithm.

Require: each packet in the sending flow f
Ensure: the selected path p
1: $X \leftarrow$ *set of all paths*
2: **if** f is not a new flow **then**
3: $p \leftarrow$ *old path*
4: **if** $f.timeout > N_t$ or $f.retrans > P_t$ **then**
5: $X \leftarrow X - p$
6: $p \leftarrow Rerouting\ procedure(X)$
7: **else if** $f.delay \geqslant T_{delayHigh}$ **then**
8: **if** $f.size \geqslant S$ and $f.rate \leqslant R$ **then**
9: $p \leftarrow Rerouting\ procedure(X)$
10: **end if**
11: **else if** $f.size < S$ **then**
12: update $f.updateTime$
13: **return** p
14: **else if** $f.flowlet\_timeout$==$true$ **then**
15: $p \leftarrow Rerouting\ procedure(X)$
16: **else**
17: update $f.updateTime$
18: **return** p
19: **end if**
20: **else**
21: $p \leftarrow Rerouting\ procedure(X)$
22: **end if**
23: update $f.updateTime$
24: **return** p

Based on the aforementioned analysis, we implement proactive rerouting that takes into account various flow status parameters, including the flow sending rate, remaining flow size, and one-way delay, to optimize network utilization. To accomplish this, we utilize the size of the flow already transmitted to estimate the remaining flow size [28]. Additionally, we employ the flow table to document the transmission rate and analyze the one-way delay to identify congestion within the flow. As detailed in lines 7-9 of Algorithm 1, BurstLoader actively reroutes flows experiencing congestion, indicated by a one-way delay exceeding $T_{delayHigh}$. When rerouting, we prioritize flows with an relatively large remaining flow size (S) and a relatively low sending rate (R). By referring to the previous work [19] and our experiments, we set $T_{delayHigh}$ to 180 microseconds, which is calculated as the base RTT plus 1.5× of the one-hop delay. S is set to 600KB, and R is established as 30% of the link capacity.

At the end of the traffic analysis, to mitigate the potential negative impact of frequent rerouting on small flows in flowlet

switching, we disallow the transmission path alteration for small, uncongested flows (as specified in lines 14-17). This approach ensures that only larger or congested flows undergo rerouting, minimizing any potential disruption to small flows. We utilize one-way delays (denoted as $T_{delayHigh}$) to define the level of congestion a flow is experiencing, and we have established a fixed threshold for the remaining flow size (S) to identify small flows. Assuming a packet has not been allocated a new state during the execution of lines 2-13 in Algorithm 1, BurstLoader will enter the rerouting procedure to balance the load in response to the flowlet timeouts. Considering the system overhead, we do not use a complex benefits model to calculate performance gains before rerouting flows and a more fine-grained and low-overhead benefits model is left as our future work.

B. Traffic Routing Module

This module performs the comprehensive rerouting decision and completes the path selection for each flow. Unlike prior flowlet-based schemes, BurstLoader employs a selective flowlet rerouting mechanism specifically designed to address traffic bursts. Besides, it proactively reroutes flows experiencing severe congestion or network failures even without new flowlets captured.

Algorithm 1 outlines the rerouting logic that is promptly invoked for each packet to assess the flow status and consult the flowlet table. Specifically, as delineated in lines 3-9 of Algorithm 1, in the event of flows experiencing failure or bad state, BurstLoader initiates the rerouting procedure. This decision is based on the assessment that the current path exhibits a risk of network failures or severe congestion, necessitating the rerouting. Furthermore, when a flow, already in an intermediate state, encounters a subsequent flowlet, the rerouting mechanism is triggered, as indicated in lines 14-15 of Algorithm 1. BurstLoader dynamically reroutes flows encountering failures, timeouts, or congestion, and efficiently utilizes the rerouting potential presented by flowlet switching.

The rerouting procedure is described in Algorithm 2 for the path selection. Intuitively, the rerouting path should be at the lowest congestion degree to balance traffic. However, if all flows were to choose a single path during periods of burst traffic, that path could rapidly become congested, leading to significant increases in transmission latency. To mitigate the herd effect, Algorithm 2 employs a probabilistic selection process for the forwarding path. The selection of the available path set is conducted between lines 3-23 of Algorithm 2 as illustrated, in which the path with a congestion degree lower than the old path is iteratively included from all paths. The path selection ratio is the difference between the congestion degree Δ and the Ψ_{max} in the set of available paths. Consequently, a path with a lower degree of congestion is more likely to be selected as the forwarding path in lines 24-30 of Algorithm 2, reflecting a probabilistic approach that favors less congested routes.

In the Algorithm 2, BurstLoader needs to determine the congestion degree of each path. Initially, BurstLoader records

Algorithm 2 Rerouting procedure.

Require: the input flow f, the congestion degree of each path Ψ, the set of paths X
Ensure: the rerouted path p

```
    Map[] ← 0
 2: X_lower ← ∅
    if f is not a new flow then
 4:     p ← old path
    else
 6:     p ← random path
    end if
 8: for each path i in X do
        if Ψ_i < Ψ_p then
10:         X_lower ← X_lower ∪ i
        end if
12: end for
    if X_lower == ∅ then
14:     return random path in X
    end if
16: for each path j←1 to n in X do
        if j ∈ X_lower then
18:         Δj ← Ψ_max − Ψ_j
            Map[j] ← Map[j − 1] + Δj
20:     else
            Map[j] ← Map[j − 1]
22:     end if
    end for
24: Ψ_rand = rand() × Ψ_max
    for each path k←1 to n in Map do
26:     if Ψ_rand > Map[k − 1] and Ψ_rand ≤ Map[k] then
            p ← path_k
28:         break
        end if
30: end for
    return p
```

the timestamp of each packet in the optional fields of the TCP header to calculate the RTT and one-way delays [26], [29]. Due to the varying traffic burst intensities, the one-way delay estimated by packet timestamp is insufficient to accurately assess congestion degrees of different paths, as we analyze in section II-B. In fact, several congestion control protocols have integrated the queue length and its gradient to sense congestion quickly [12], [30]. Inspired by them, BurstLoader defines the congestion degree of the path as the product of the sum of packets being transmitted in the network and their varying gradients, formally expressed in Eq. (1),

$$\Theta(t) = (q(t) + C \cdot B) \cdot (\frac{\Delta q(t)}{\Delta t} + C) \tag{1}$$

where $q(t)$ is the sum of all queue lengths, C is the link bandwidth, B is the base round trip time and $\frac{\Delta q(t)}{\Delta t}$ refers to the queue length gradient. Since we cannot obtain the queue

length, we calculate $\frac{\Theta(t)}{C^2}$ in Eq. (2),

$$\frac{\Theta(t)}{C^2} = (\frac{q(t)}{C} + B) \cdot (\frac{\Delta q(t)}{C \cdot \Delta t} + 1) \tag{2}$$

and then, using the fact that $\frac{q(t)}{C} + B = \theta$ (one-way delay) and $\frac{\Delta q(t)}{C \cdot \Delta t} = \overline{\theta}$ (gradient of one-way delay), we rearrange the Eq. (2) as follows,

$$\Psi(t) = \frac{\Theta(t)}{C^2} = \theta \cdot (\overline{\theta} + 1) \tag{3}$$

Therefore, we monitor the one-way delay and its gradient to update $\Psi(t)$ in Eq. (3), which defines the congestion degree of the path in Algorithm 2.

To summarize, BurstLoader enhances load balancing performance by concurrently considering the status of individual flows and the dynamics of flowlets at host side while having no modification to hardware and network protocols.

IV. Evaluation

We conduct an evaluation of BurstLoader, comparing it with the state-of-the-art load balancing mechanisms to investigate the enhancement in performance with NS3 large-scale simulations [23]. Our evaluation aims to demonstrate whether BurstLoader can achieve better performance compared to other schemes under varying traffic burst intensities.

Topology: We construct an 8×8 leaf-spine network topology in NS3, featuring 20 Gbps links and a server count of 128. This design ensures eight distinct equal-cost paths between any host pair, interconnected through diverse switches. Consequently, we implement a 2:1 oversubscription ratio at the leaf level to meet standard configurations in common data center networks [21]. To compare BurstLoader with other schemes under asymmetric network scenario, we selectively reduce the capacity of 20% of leaf-to-spine links from 20 Gbps to 4 Gbps.

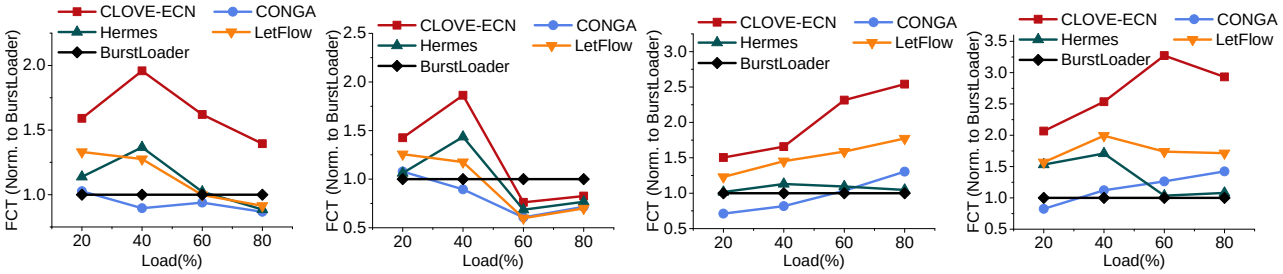
Workloads: We utilize two realistic workloads (web-search [1] and data-mining [8]) derived from operational data centers to simulate traffic dynamics for our evaluations. These workloads, as demonstrated in prior studies [11], [19], [21], [24], exhibit heavy-tailed characteristics, where the majority of flows are small, yet a minor fraction of large flows accounts for a significant portion of the total bytes. Notably, the web-search workload tends to be more bursty, while the data-mining workload exhibits a smoother yet more skewed distribution, with approximately 95% of all data bytes belonging to merely 3.6% of flows exceeding 35MB [21]. Consequently, load balancing under the data-mining workload poses a greater challenge for flowlet-based schemes. To replicate these workloads, we generate flows between randomly selected senders and receivers within various leaf switches, following Poisson processes with varying traffic intensities.

Methodology: To demonstrate the performance improvements offered by BurstLoader, we conduct a comparative analysis with several state-of-the-art schemes that includes CLOVE-ECN, Hermes, LetFlow and CONGA, while employing DCTCP [1] as the underlying transport protocol. We normalize the FCT to BurstLoader in order to better visualize the results. To ensure fairness across various schemes, we maintain a uniform flowlet timeout value.

Under the web-search workload: As shown in Fig. 6, BurstLoader attains comparable performance to CONGA, maintaining a performance gap ranging from -15% to 2%, without relying on customized switches. All other schemes demonstrate similar performance except CLOVE-ECN, particularly under heavy workload conditions. Given the bursty nature of web-search workload and the generation of numerous new flowlets, schemes employing flowlet switching in network switches can swiftly converge to a balanced load distribution once sufficient flowlets are enough. CLOVE-ECN utilizes ECNs to detect path congestion and is implemented at the end hosts. This approach provides inferior visibility compared to schemes implemented within the switch. Moreover, BurstLoader possesses the capability to proactively reroute flows and employs more precise congestion information for rerouting, surpassing CLOVE-ECN by up to 48%. Hermes enhances performance by employing a congestion-aware strategy for per-packet switching. LetFlow, a congestion-oblivious approach employing in-network random-hashing flowlet switching, resembles CONGA and cannot consistently respond proactively to congestion. BurstLoader, by seizing favorable flowlet rerouting opportunities and proactively reroute flows experiencing congestion or failures, outperforms LetFlow and Hermes by 21-24% and 12-26% at 20-40% load, respectively. The above experimental results illustrate that BurstLoader is capable of proactively rerouting flows in response to real-time flow status assessments, while simultaneously capitalizing on flowlet switching to optimize load balancing.

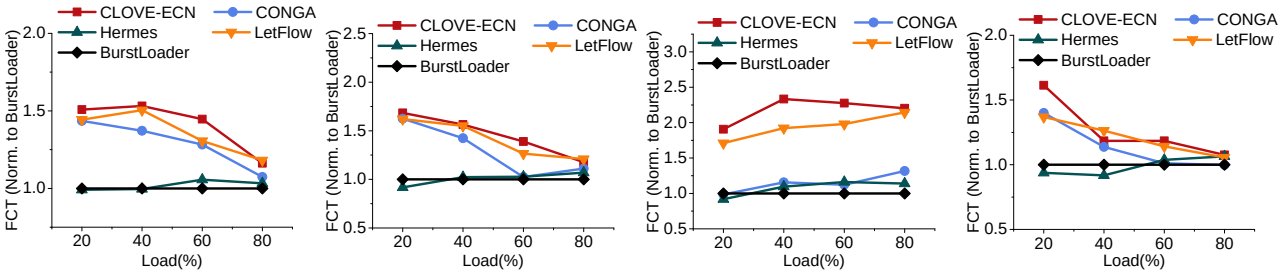
Under heavy loads approaching 80%, BurstLoader improves the average and 99th percentile FCTs for small flows by significant margins of 23-60% and 29-65%, respectively. Pure flowlet-based approaches encounter challenges such as packet reordering and congestion mismatch at high loads. However, BurstLoader mitigates these issues by continuously monitoring flow status to make informed load-balancing decisions. This approach prevents the unnecessary rerouting of small flows with high sending rates while selectively rerouting other congested flows. While Hermes also addresses congestion mismatch through comprehensive sensing and cautious per-packet rerouting, as seen in Fig. 6c and Fig. 6d, BurstLoader outperforms Hermes in terms of average FCTs for small flows by 2-11%. This is beacuse in addition to sensing the flow status for rerouting, BurstLoader reroutes new flowlets for flows in a suitable transmission state, thereby maximizing the utilization of the bursty characteristics of traffic.

Under the data-mining workload: As shown in Fig. 7, BurstLoader outperforms all alternative schemes in most scenarios. Contrary to the web-search workload, the data-mining workload comprises a higher proportion of large flows and exhibits significantly less bursts. Under such circumstances, BurstLoader achieves a performance gain of 6-



(a) Overall avg FCT (b) Large flow (>10MB) avg (c) Small flow (<100KB) avg (d) Small flow 99^{th} percentile

Fig. 6: FCT for the web-search workload (normalized to BurstLoader).



(a) Overall avg FCT (b) Large flow (>10MB) avg (c) Small flow (<100KB) avg (d) Overall 99^{th} percentile

Fig. 7: FCT for the data-mining workload (normalized to BurstLoader).

33% compared to CONGA and LetFlow. Previous flowlet-based schemes, such as CLOVE-ECN, LetFlow, and CONGA, encounter challenges in promptly reacting to congestion due to the scarcity of new flowlets under low-burst workloads. Conversely, BurstLoader estimates flow status to proactively reroute flows upon detecting congestion or failures. Besides, BurstLoader outperforms Hermes by 3-5% at 60-80% load by rerouting new flowlets properly to enhance performance. For both average FCTs of small flows and overall 99th percentile FCTs, BurstLoader maintains consistent and robust performance, as shown in Fig. 7c and Fig. 7d. This is primarily attributed to BurstLoader's ability to proactively reroute large flows experiencing congestion or failures while avoiding performance loss caused by over-rerouting small flows.

According to the above experiments and analysis, BurstLoader is more adaptive to varying traffic burst intensities under different workloads compared with the schemes with customized switches. Furthermore, compared with the schemes residing in the end hosts, BurstLoader can seize more opportunities to schedule traffic and thus improve load balancing performance.

V. Related Work

We briefly discuss related work that has informed and inspired our design.

The centralized mechanisms (e.g., Hedera [7], MicroTE [31] and FastPass [32]) employ centralized schedulers to monitor global network state and schedule large flows in multiple paths. But they have long scheduling intervals and cannot adapt to varying traffic burst intensities in data center networks.

Some in-network solutions employ custom switches to balance traffic. RPS [14] and DRILL [16] are prone to experience packet reordering and congestion mismatch under asymmetric topology [17], [19]. Besides, the performance optimization effect of flowlet-based schemes (e.g., LetFlow [11], CONGA [21] and HULA [24]) may be affected under smooth traffic. Several schemes (e.g., AG [17], APS [33], TLB [34], BurstBalancer [13], DRW [35] and Halflife [5]) adjust the switching granularity to adapt to the dynamic traffic and network asymmetries. However, it is difficult or expensive for them to monitor the transport layer information of each flow in the switch, which may cause performance issues under failed switches, such as silent random packet drops and packet blackholes [19], [36]. Some solutions are optimized for lossless data centers and do not fall within the scope of our comparison [37], [38].

Many schemes perform load balancing at end hosts. Presto [15] routes flowcells to balance load at the network edge. CLOVE-ECN [22] leverages per-flowlet weighted round robin at end hosts to route flowlets, and the path weights are calculated according to ECN signals residing in ACKs. MPTCP [39], as a transport protocol, routes several sub-flows simultaneously over multiple paths. Hermes [19] exploits ECN signals and coarse-grained RTT measurements to sense congestion on multiple paths for load balancing. PLB [18]) reroutes flows that experience congestion, and it reroutes a connection by changing the IPv6 Flow Label on its packets, which switches include as part of ECMP/WCMP. Though

schemes rerouting flows at packet granularity based on flow status can make proactive load balancing decisions, they lose the rerouting opportunity based on traffic bursts and hardly deploy accurate path congestion detection as described in BurstLoader.

VI. Conclusion

BurstLoader adapts to varying traffic burst intensities by taking into account both flow status and flowlets at end hosts. It is worth noting that we use several fixed thresholds to divide the flow status. We will study adaptive dynamic thresholds to optimize deployment and performance in future work. Besides, BurstLoader deploys a more accurate congestion detection compared to other host-based schemes. BurstLoader has no modification to existing hardware and network protocols and provides competitive performance even against schemes that require custom switches under realistic workloads.

VII. Acknowledgments

This work was supported by the National Key Research and Development Program of China (2021YFC3300603, 2023YFC3306204); the National Natural Science Foundation of China (62376092); the MOE (Ministry of Education in China) Project of Humanities and Social Sciences (23YJCZH183); the Natural Science Foundation of Hunan Province of China (2022JJ40129, 2023JJ40236); the Open Project of Xiangjiang Laboratory (23XJ01012, 22XJ03014).

References

[1] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proceeding of the ACM SIGCOMM*, 2010, pp. 63–74.

[2] W. Li *et al.*, "Flow scheduling with imprecise knowledge," in *Proceeding of the USENIX NSDI*, 2024, pp. 95–111.

[3] J. Shao *et al.*, "Racecc: A rapidly converging explicit congestion control for datacenter networks," *Journal of Network and Computer Applications*, vol. 217, p. 103673, 2023.

[4] Z. Li, M. Li, J. Liu, and S. Tang, "Understanding the flooding in low-duty-cycle wireless sensor networks," in *Proceeding of the ACM ICPP*, 2011, pp. 673–682.

[5] S. Liu *et al.*, "Halflife: An adaptive flowlet-based load balancer with fading timeout in data center networks," in *Proceeding of the ACM EuroSys*, 2024, p. 66–81.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceeding of the ACM SIGCOMM*, 2008, pp. 63–74.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceeding of the USENIX NSDI*, 2010, pp. 89–92.

[8] A. Greenberg *et al.*, "VL2: a scalable and flexible data center network," in *Proceeding of the ACM SIGCOMM*, 2009, pp. 51–62.

[9] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992*, 2000.

[10] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's burstiness with flowlet switching," in *Proceeding of the ACM HotNets*, 2004.

[11] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proceeding of the USENIX NSDI*, 2017, pp. 407–420.

[12] 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 (ToN)*, vol. 28, no. 1, pp. 98–111, 2020.

[13] Z. Liu *et al.*, "BurstBalancer: Do less, better balance for large-scale data center traffic," in *Proceeding of the IEEE ICNP*, 2022, pp. 1–13.

[14] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proceeding of the IEEE INFOCOM*, 2013, pp. 2130–2138.

[15] K. He *et al.*, "Presto: Edge-based load balancing for fast datacenter networks," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 465–478.

[16] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proceeding of the ACM SIGCOMM*, 2017, pp. 225–238.

[17] J. Liu, J. Huang, W. Li, and J. Wang, "AG: Adaptive switching granularity for load balancing with asymmetric topology in data center network," in *Proceeding of the IEEE ICNP*, 2019, pp. 1–11.

[18] M. A. Qureshi *et al.*, "Plb: congestion signals are simple and effective for network load balancing," in *Proceeding of the ACM SIGCOMM*, 2022, p. 207–218.

[19] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proceeding of the ACM SIGCOMM*, 2017, pp. 253–266.

[20] M. Guo *et al.*, "Towards distributed flow scheduling in ieee 802.1qbv time-sensitive networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 20, no. 5, 2024.

[21] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proceeding of the ACM SIGCOMM*, 2014, pp. 503–514.

[22] N. Katta *et al.*, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceeding of the ACM CoNEXT*, 2017, pp. 323–335.

[23] "NS3," Accessed April 29, 2024. [Online]. Available: https://www.nsnam.org/.

[24] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proceeding of the ACM SOSR*, 2016, p. 10.

[25] X. Diao, H. Gu, X. Yu, L. Qin, and C. Luo, "Flex: A flowlet-level load balancing based on load-adaptive timeout in dcn," *Future Generation Computer Systems (FGCS)*, vol. 130, pp. 219–230, 2022.

[26] Q. Shi, F. Wang, D. Feng, and W. Xie, "ALB: Adaptive load balancing based on accurate congestion feedback for asymmetric topologies," in *Proceeding of the IEEE IWQoS*, 2018, pp. 1–6.

[27] S. Hu *et al.*, "Explicit path control in commodity data centers: design and applications," in *Proceeding of the USENIX NSDI*, 2015, p. 15–28.

[28] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proceeding of the USENIX NSDI*, 2015, pp. 455–468.

[29] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 537–550.

[30] V. Addanki, O. Michel, and S. Schmid, "PowerTCP: Pushing the performance limits of datacenter networks," in *Proceeding of the USENIX NSDI*, 2022, pp. 51–70.

[31] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proceeding of the ACM CoNEXT*, 2011, p. 8.

[32] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proceeding of the ACM SIGCOMM*, 2014, pp. 307–318.

[33] J. Liu, J. Huang, W. Lv, and J. Wang, "APS: Adaptive packet spraying to isolate mix-flows in data center network," *IEEE Transactions on Cloud Computing (TCC)*, vol. 10, no. 2, pp. 1038–1051, 2022.

[34] J. Hu *et al.*, "Adjusting switching granularity of load balancing for heterogeneous datacenter traffic," *IEEE/ACM Transactions on Networking (ToN)*, vol. 29, no. 5, pp. 2367–2384, 2021.

[35] F. Fan, H. Meng, B. Hu, K. L. Yeung, and Z. Zhao, "Roulette wheel balancing algorithm with dynamic flowlet switching for multipath datacenter networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 29, no. 2, pp. 834–847, 2021.

[36] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 139–152.

[37] J. Hu, Y. He, W. Luo, J. Huang, and J. Wang, "Enhancing load balancing with in-network recirculation to prevent packet reordering in lossless data centers," *IEEE/ACM Transactions on Networking (ToN)*, vol. 32, no. 5, pp. 4114–4127, 2024.

[38] J. Hu *et al.*, "Load balancing with multi-level signals for lossless datacenter networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 32, no. 3, pp. 2736–2748, 2024.

[39] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proceeding of the ACM SIGCOMM*, 2011, pp. 266–277.