SMOOTHIE: Efficient and Flexible Load-Balancing in Data Center

Loïc Champagne, Benoit Donnet
Université de Liège, Montefiore Institute, Belgium

***Abstract*—In the context of contemporary data center networks (DCNs), optimizing resource utilization and preventing congestion are critical objectives. This paper introduces SMOOTHIE, a dynamic path load balancer specifically designed for real-time congestion management using Inband Network Telemetry (INT) for collecting network state pieces of information and Segment Routing Version 6 (SRv6) for rerouting traffic. Leveraging the programmable nature of the P4 programming language, SMOOTHIE effectively achieves its goals. Our experimental results showcase SMOOTHIE's superiority over conventional Equal-Cost Multipath (ECMP) routing and its competitive performance compared to other congestion-aware load balancing solutions. We assert that these findings can be attributed to SMOOTHIE's proactive congestion response, which minimizes the necessity for `TCP` congestion window resizing, and its ability to intelligently reroute flows onto optimal paths, thereby substantially reducing route flapping. These outcomes highlight SMOOTHIE's potential to significantly enhance network performance within DCNs. Furthermore, SMOOTHIE offers enhanced manageability, ease of maintenance, and simplified deployment through a centralized controller, further underscoring its value.**

I. INTRODUCTION

Modern Data Center Networks (DCNs) utilize multi-rooted topologies like k-ary fat-trees [1] or spine-leaf setups [2], [3]. These configurations ensure ample bisection bandwidth for numerous servers in end racks, using cost-effective commodity switches. However, achieving optimal link bandwidth utilization relies heavily on even distribution of network traffic across available paths. Poor load balancing risks link over subscription, leading to congestion and underutilized bandwidth. Effective traffic distribution prevents a few switches from bearing excessive traffic, reducing processing delays and latency for end-users, while optimizing DCN resource utilization.

Traditionally, equal-cost multi-path routing (ECMP) has been the bedrock of load balancing strategies, employing a static path selection method based on the 5-tuple flow hash digest. This encompasses source and destination IP addresses, source and destination ports, and the transport protocol. However, ECMP has notable limitations, leading to performance degradation in dynamic environments [4], [2], [3]. To address these shortcomings, various load balancing approaches have emerged. While centralized solutions like B4 [5] and SWAN [6] offer adaptability, their universal applicability is constrained (i.e., optimized for WAN architectures with path updates occurring at minute-scale intervals). Conversely, Decentralized methods exemplified by CLOVE [2] and HULA [3], swiftly re-balance traffic but strain network resources by relying on switch memory and on additional packets, potentially causing congestion. Additionally, innovative solutions like MPTCP [7], [8] and CONGA [9] modify network architecture but present their own challenges, such as kernel stack modifications for MPTCP and the replacement of network switches for CONGA.

	SMOOTHIE	CONGA	CLOVE	HULA	B4
Reliability & cong. control	✓	✓	✓	✓	✓
QoS	✓	✗	✗	✗	✗
Ease of management	✓	✗	✗	✗	✓
Convenient upgradability	✓	✗	(✓)	(✓)	✓
Use commodity switches	✓	✗	✓	✓	✓
Universally applicable	✓	(✓)	✓	✓	✗

TABLE I
COMPARISON OF ✗ THE SERVICES NOT OFFERED, (✓) PARTIALLY AVAILABLE, AND ✓ OFFERED BY SOLUTIONS.

Table I presents a comprehensive overview of the key attributes of these network solutions, namely CONGA, CLOVE, HULA, and B4, which encompass reliability and congestion control. Table I also includes SMOOTHIE, our solution discussed in this paper for dealing with modern challenges in load balancing traffic in DCNs. Notably, SMOOTHIE uniquely achieves Quality of Service (QoS) by dynamically adapting path selection to suit individual data flow requirements. It emphasizes QoS prioritization through the customization of the p-INT parameter, enabling fine-grained monitoring of critical flows. Additionally, our consideration extends to an expanded iteration of SMOOTHIE which incorporates dedicated routing, bandwidth reservation, and flow-specific enhancements. Moreover, utilizing flow-specific INT instructions to gather path-related metrics enables precise adjustments tailored to each flow's distinct demands. Both SMOOTHIE and B4 offer advantages in terms of manageability and upgradability, owing to their centralized architecture. Conversely, CLOVE and HULA exhibit limitations in upgradability due to their reliance on P4 [10] implementation only. Indeed, they are constrained by their restricted network view and the limitations of the P4 language. CONGA suffers from a lack of applicability as it is built upon custom Application-Specific Integrated Circuits (ASICs). Finally, it is worth noting that B4 is only applicable to Wide Area Network (WAN) environments.

From these observations, it becomes evident that an efficient load balancing strategy is needed to address the increasing demands on modern network infrastructures. This paper introduces SMOOTHIE, a dynamic path load balancer designed to tackle these challenges. Leveraging Inband Net-

work Telemetry (INT) [11] and Segment Routing Version 6 (SRv6) [12], SMOOTHIE detects and responds to congestion in real-time, enhancing traffic distribution. Developed in the P4 programming language, SMOOTHIE capitalizes on its programmability and flexibility, aligning with the rise of programmable packet-processing pipelines [13], [14]. These architectures offer configurable stateful packet processing at line rates, allowing operators to adapt load balancing schemes to their deployment needs and compile them for the hardware. Although SMOOTHIE integrates P4, its use is optional as many switch manufacturers already support INT and SRv6 [15], [16]. Its inclusion primarily aims to offer customization options to prospective users.

This paper makes the following contributions:

- Introducing SMOOTHIE, a novel network load balancing approach that utilizes standard network switches without altering tenant virtual machines' network stack. It offers centralized management without introducing additional network packets.
- We leverage the concept of "the power of two choices" [17] to strike a balance between ideal and rapid decisions when it comes to rerouting. We also demonstrate that this approach significantly reduces the occurrence of route flapping.
- Our experiments demonstrate that we are able to rival state of the art load balancing while using an easier to manage centralized load balancing strategy.
- We showcase an easy way to manage INT monitoring granularity as well as showing the impact of heavy real time monitoring on the network performance.
- Our source code is freely available to the research and industrial community.

The remainder of this paper is organized as follows: Sec. II provides the required background for the paper; Sec. III discusses SMOOTHIE, our efficient load balancing technique; Sec. IV explains the methodology we followed for evaluating SMOOTHIE; Sec. V discusses the results we obtain; Sec. VI positions SMOOTHIE with respect to the state of the art; finally, Sec. VII concludes this paper by summarizing its main achievements.

II. BACKGROUND

This section discusses the required technical background for the remainder of the paper. In particular, Sec. II-A introduces *in-band telemetry* (INT) and Sec. II-B focuses on Segment Routing.

A. In-Band Telemetry

Inband Network Telemetry (INT) is a data plane framework designed to autonomously gather and report network state information, removing the need for control plane intervention. Within this model, packets carry header fields as 'telemetry instructions' for network devices. These instructions, programmable within the data plane, can be tailored to specific flows, dictating the network state information to collect.

The collected network state information can be sent directly to a dedicated telemetry monitoring system or encapsulated within packets traversing the network. Dedicated traffic sinks handling INT retrieve and potentially report these results, facilitating monitoring of the observed data plane state.

Traffic sink functionalities of INT include tasks such as OAM (Operations, Administration, and Maintenance) [18], real-time control loops for traffic engineering adjustments, and immediate actions for urgent conditions like congestion or data-plane violations.

This architectural model supports applications including network troubleshooting, performance monitoring, congestion control, advanced routing, and network data plane verification. For further insights and evaluations, readers can explore Jeyakumar et al. [19].

B. Segment Routing

Segment Routing [12] (SR) is a source routing concept that uses an ordered list of *segments*, defining specific packet handling instructions. These segments can enforce topological or service-related requirements. Over time, SR has found applications in various areas like network monitoring, traffic engineering, and failure recovery [20]. There are two primary implementations: SR over MPLS (SR-MPLS)[21] and SR over IPv6 (SRv6)[22].

SR features multiple segment types, with the most common being *node segments* and *adjacency segments*. Node segments represent the least cost path between routers and a prefix, while adjacency segments signify a link between two routers.

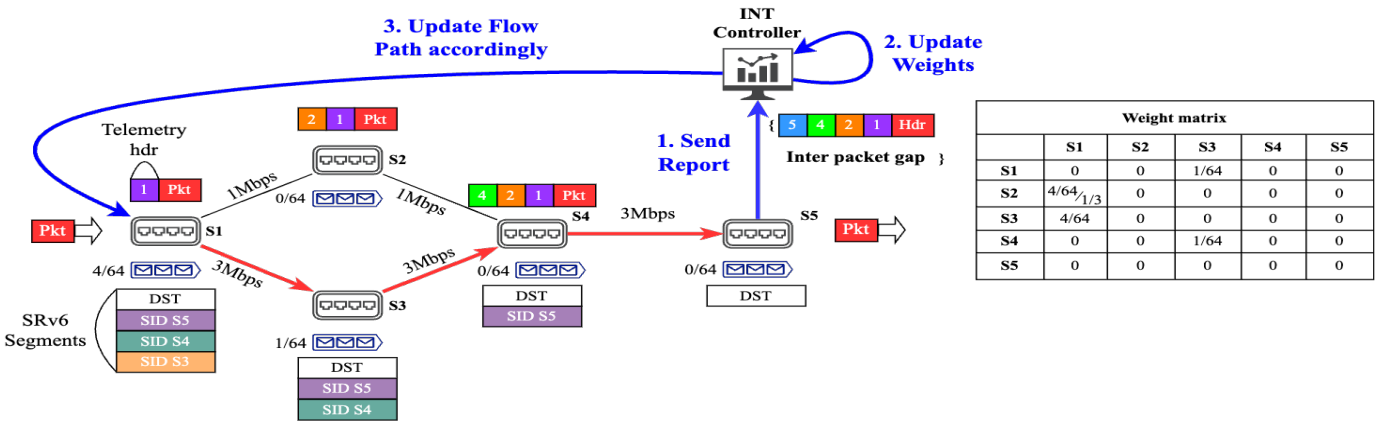
In SR-MPLS, segments are identified by a unique 20-bit label called *Segment IDentifier* (SID), implemented as MPLS labels. Each label in the MPLS stack represents a segment, guiding packet forwarding.

SRv6, however, uses 128-bit IPv6 addresses as SIDs, enabling deployment over non-MPLS networks or areas without MPLS, such as data centers. This implementation simplifies deployment as it only requires advertising IPv6 prefixes. SIDs are encoded within the `Routing` Extension Header known as SR Header [23] in SRv6.

III. SMOOTHIE

The overall functioning of SMOOTHIE is illustrated in Fig. 1. SMOOTHIE has been designed to work on a private domain, such as a datacenter, and is responsible to efficiently and flexibly load balance traffic between the domain ingress point and the domain egress. SMOOTHIE applies ECMP to distribute flows across paths and adapts path assignments based on network congestion.

SMOOTHIE is based on three main steps: first, SMOOTHIE seamlessly collects telemetry data through INT (Sec. III-A) to detect potential congestion. This telemetry data is exported, by the egress switch, to the controller (Sec. III-B). The controller is responsible to detect congestion and, to do so, maintain state information about the network. Finally, once congestion has been detected, the controller computes a new path for a flow and the domain ingress is responsible to force traffic following that new path through SRv6 (Sec. III-C).



Weight matrix					
	S1	S2	S3	S4	S5
S1	0	0	1/64	0	0
S2	$4/64_{1/3}$	0	0	0	0
S3	4/64	0	0	0	0
S4	0	0	1/64	0	0
S5	0	0	0	0	0

Fig. 1. Routing evolution in SMOOTHIE. The initial path is [S1, S2, S4, S5] which is subsequently altered by the controller after processing telemetry data. The new route, encoded through SRv6, becomes [S1, S3, S4, S5] and is illustrated as the red path. Queue occupancy, denoted as x/y (where x represents the current occupancy and y is the queue size), is displayed below the switches.

A. *InBand Network Telemetry with* SMOOTHIE

First, SMOOTHIE operates by collecting telemetry data through an inband approach (i.e., Inband Network Telemetry – INT), seamlessly embedding real-time network performance and traffic data into packets.

In INT version 2.1 [24], metadata plays a crucial role in network monitoring and optimization. Elements like "Ingress Interface Identifier" and "Node Identifier" help identifying packet transit paths, aiding in diagnosing routing anomalies. "Ingress" and "Egress timestamps" offer insights into packet processing times, while metrics like "Interface Utilization" streamline resource allocation. "Average Queue Length" assists in buffer management, queue performance, and congestion detection. These real-time insights enable prompt automatic corrective actions to address network issues.

Nonetheless, the inclusion of such detailed pieces of information in all packets does introduce a notable overhead, which can adversely impact both flow completion times and application-level performance. In response to this concern, we have introduced the Proportional INT (p-INT) monitoring mechanism, enabling fine-grained control over monitoring granularity. With p-INT, this granularity can be precisely adjusted by selecting the proportion of packets to which telemetry headers are appended.

Furthermore, INT provides an indispensable capability for selectively gathering metrics tailored to specific flows within a network infrastructure. Much akin to the concept of Quality of Service (QoS), INT can be meticulously configured to focus its monitoring efforts on critical flows, thereby providing enhanced visibility and granular insights into their performance. This selective approach empowers network administrators to prioritize the monitoring of mission-critical or high-priority data streams, ensuring they receive the utmost attention and resources. Concurrently, INT can curtail in-band overhead for non-critical flows, alleviating unnecessary strain on network resources. This fine-grained control over flow-specific metric collection improves the allocation of monitoring resources, elevating network efficiency and reliability to align closely with the specific needs and priorities of the organization.

B. SMOOTHIE *Controller*

The second part of SMOOTHIE involves the *controller*, responsible for finding alternative routes during network congestion. The pseudo-code of the controller inner work can be seen on Algo. 1. It detects congestion by monitoring idle time between packets in various flows. Congestion is then inferred when these intervals exceed a predefined threshold, τ, prompting the controller to choose an alternative path. Unlike flowlet-based methods, we assess idle time at the egress, enabling proactive congestion management, which helps preempt `TCP` congestion window resizing and ensures `UDP` protocol compatibility. In terms of value, τ needs to balance between being high enough to avoid false positives from bursts and low enough for early congestion detection. Which is due to the fact that τ can be exceeded in three scenarios: `TCP` congestion window resizing (indicating packet loss or congestion), bursts from other flows on the same path, or actual congestion.

Within the new path selection process, we harness the concept of "the power of two choices" [17] in the realm of randomized load balancing. This technique stems from the intriguing observation that when distributing n balls randomly into n bins, the maximum number of balls in any single bin typically hovers around $\mathcal{O}(\log n)$ with a high degree of certainty. However, by employing a strategy of making two random choices and directing the ball into the bin with fewer balls, the maximum number drops significantly to $\mathcal{O}(\log \log n)$. This "power of two choices" strategy effectively ensures that the balls are more uniformly distributed among the bins, significantly improving load balancing. By intelligently selecting the bin with the fewest balls, we mitigate the risk of any single bin becoming excessively crowded.

This paradigm, known as the "balls into bins" model, finds natural applications [25], [26] in tasks such as scheduling, load balancing, hashing, and other common problems. Remarkably, despite its apparent simplicity, the use of two random choices

Algorithm 1 Receive and Process INT Report at the controller.

//Receive INT *report from Egress*
//Parse INT *report into an object called* `report`

$G \leftarrow$ *the graph object representing the topology*
$src\_host \leftarrow G.get\_host(report.src\_ip)$
$dst\_host \leftarrow G.get\_host(report.dst\_ip)$
for $i,\ hop$ **in** $report.hop\_metadatas$ **do**
    if $i == 0$ **then**
        $dst \leftarrow hop.switch\_id$
        continue
    else
        $src \leftarrow hop.switch\_id$

    //Update weights
    $G.edges[src, dst][w1] \leftarrow \frac{hop.queue\_occupency}{hop.queue\_size} \div \frac{G.node\_to\_node\_bw(src,dst)}{G.max\_bw}$

    $dst \leftarrow src$

//Indicates that the parameter τ has been exceeded
if $report.update\_path$ **then**
    //Generate two paths of minimal length
    $paths \leftarrow G.generate\_paths(src\_host, dst\_host, 2)$

    //Compare paths weights
    if $w\_path(paths[0]) < w\_path(paths[1])$ **then**
        $p \leftarrow paths[0]$
    else
        $p \leftarrow paths[1]$
    if $w\_path(current\_path) < w\_path(p) + \delta$ **then**
        return
    if "path already exported" **then**
        return

    //Export p to the first switch on the path (i.e., ingress)
    $export\_path(p, p[1])$

proves to be highly effective. It is worth noting that, should one opt for more than two choices ($d > 2$) when assigning each ball, the potential for improvement diminishes, as the maximum load can be approximated as $\frac{\mathcal{O}(\log \log n)}{\log d} + \mathcal{O}(1)$. Encouraged by these findings, we have adopted this strategy for route selection in the context of SMOOTHIE.

Therefore, the process of selecting a new path involves randomly picking two paths of minimal length and then selecting the one with the lowest weight. The path weight is determined by the sum of the weights of its constituent links, calculated as:

$$Link\ Weight = \frac{QueueOccupancy}{QueueSize} \div \frac{LinkBandwidth}{MaxBandwidth}.$$

Here, Queue Occupancy is obtained through INT telemetry headers, and Max Bandwidth represents the network's maximum capacity. After selecting the optimal routing path based on these metrics, the chosen path is then communicated to the ingress switch for further action.

Furthermore, the controller's potential impact can be assessed empirically by calculating the achieved average throughput and dividing it successively by the average packet length and the p-INT proportion. This calculation provides an estimation of the average packet processing load anticipated for the controller per second.

C. Flow Redirection

To facilitate efficient packet forwarding along the chosen route, our network switches employ Segment Routing v6 (SRv6). SRv6 has emerged as the preferred choice for traffic steering in our network due to its innate support for network functions and its seamless integration capabilities with cloud services [27].

Once the controller has selected the most suitable path, it is encoded as a node segment (i.e., the list of all hops to traverse) using SRv6.

It is worth noting that our routing information (i.e. SRv6 path) is exported to the ingress switch via industry-standard communication protocols such as Thrift or gRPC, ensuring seamless integration within our network ecosystem and compatibility with various network management tools.

IV. METHODOLOGY

We conduct experiments using the P4utils [28] framework and Mininet [29]. The setup uses a host with a 16-core Intel Xeon Processor E5-2683 v4 and 64 GB of RAM. As for the chosen topology, it is a spine-leaf with 12 switches, connecting 8 clients and 8 servers.

CLOVE and ECMP have both been implemented using the implementation provided on the P4learning GitHub repository [30]. Furthermore, in order to ensure consistency with SMOOTHIE, both CLOVE and ECMP implementations have been adapted to support IPv6. Specifically, for CLOVE, the congestion-aware load balancing implementation from P4learning has been carefully modified to align as closely as possible with the CLOVE-Int variant [2], which serves as a relevant baseline for our comparative analysis. This adaptation process involved, among others, the integration of IPv6 support while preserving the core principles and functionality of the original CLOVE implementation.

To replicate authentic network traffic patterns, we utilize a web search workload derived from operational DCNs at Microsoft [31]. Our experiments follow a simple client-server communication model, where each client randomly selects a server and initiates a persistent `TCP` connection. The size of data flows sent by clients is determined by sampling from the empirical Cumulative Distribution Function (CDF) of the web search workload. The inter-arrival rate of flows on a connection follows an exponential distribution, with the mean adjusted to control the desired network load.

Cognizant of perturbations introduced by concurrent processes and external factors, each experiment underwent ten iterations with a consistent random seed. Our reported outcomes

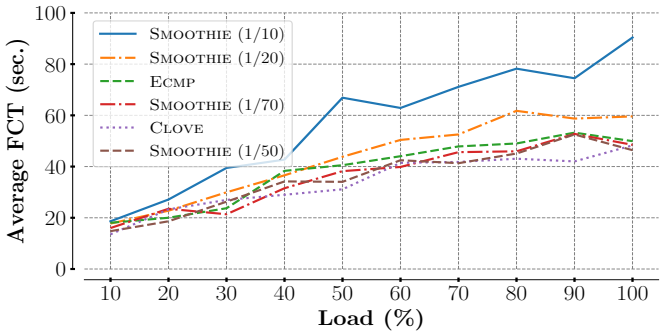


Fig. 2. Comparison of SMOOTHIE with CLOVE and ECMP with respect to average flow completion time over different load. The threshold τ has been fixed to four RTTs. SMOOTHIE is tested with different p-INT values.

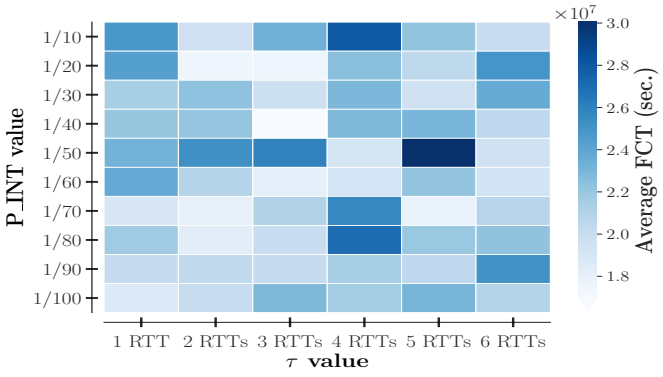


Fig. 3. SMOOTHIE results with respect to parameters τ and p-INT variation. Lighter colors represent superior results, highlighting ideal parameter combinations. This analysis informs refined configurations, ensuring efficient load balancing in dynamic network environments.

represent the averaged results from these runs. Employing a fixed seed across multiple trials mitigates random variance, ensuring consistent initial conditions and stabilizing our findings. This method bolsters the reliability and robustness of our study, yielding a more steadfast and indicative estimation of the system's behavior, immune to stochastic influences.

In line with prior research [2], [3], we assess the overall performance using the average Flow Completion Time (FCT). This metric ensured that all flows, including the majority of smaller flows, received equal consideration.

V. RESULTS

In Fig. 2, we delve into the sensitivity of SMOOTHIE to variations in monitoring granularity, highlighting the crucial necessity of meticulous parameters (p-INT and τ – see Sec. III) tuning for its effective operation. As illustrated in Fig. 2 and 3, the choice of parameters can exert a significant influence on the performance and outcomes of the load balancer.

To tackle this challenge and efficiently optimize our parameters, we adopted a Grid-search approach, as depicted in Fig. 3. Grid-search is a systematic method for parameter tuning that involves exhaustively testing combinations of parameter values within predefined ranges. Additionally, we leveraged insights gleaned from the experiences of our peers, as documented in CLOVE [2] and HULA [3]. These insights served as a valuable starting point in our quest for the optimal configuration of SMOOTHIE's parameters.

Fig. 3 displays SMOOTHIE's sensitivity to p-INT and τ parameters. The X-axis represents τ, the idle time threshold between packets for congestion detection in round-trip times (RTT). The Y-axis indicates p-INT values, the proportion of packets with appended telemetry headers. The color bar in Fig. 3 signifies average FCT (measured in seconds) across multiple iterations, with lighter shades denoting better performance.

The main parameter to configure is τ congestion threshold, which should be chosen to be small enough in order to alleviate congestion at its early onset. The smaller it is, the sooner we could be avoiding severe congestion episodes, but the more rerouting false alarms we might be getting and it might increase packet reordering at the receiver. τ, as recommended in previous works [9], [32], [2], should be set between one and five RTTs.

Furthermore, Fig. 3 reveals that lower p-INT values drive SMOOTHIE closer to behavior resembling ECMP routing. In this context, SMOOTHIE maintains initial route assignments akin to ECMP, displaying insensitivity to network conditions and refraining from flow rerouting (see Sec. III for details). Notably, performance deterioration is evident when p-INT exceeds 1/70.

Selecting optimal parameters in systems like SMOOTHIE is intricate, influenced by traffic patterns, network architecture, switch configurations, and broader network objectives. Precision in these settings is vital for efficient load balancing and traffic management, adapting to the network's nuances and evolving requirements through real-time data and performance objectives.

In conclusion, Fig. 3 shows that the ideal range for τ falls within the range of 1 to 5 Round Trip Time(s) (RTTs), while the ideal range for p-INT appears to be between 1/40 and 1/70. In the subsequent experiments, we used a value of τ which is four RTTs.

In our study, as illustrated in Fig. 2, we present compelling empirical findings that highlight the performance advantages of the SMOOTHIE algorithm over both ECMP and CLOVE. Notably, SMOOTHIE demonstrates superior load balancing performance when compared to ECMP. Moreover, even when pitted against the sophisticated CLOVE algorithm, SMOOTHIE maintains a high level of competitiveness, outperforming CLOVE, particularly in scenarios characterized by a 100% workload.

In Fig. 4, we can observe that SMOOTHIE employs intelligent congestion flow rerouting strategies, as opposed to the random rerouting methods employed by CLOVE. Notably, SMOOTHIE reroutes a flow at most twice before achieving convergence, while CLOVE, in 27% of cases, needs to reroute flows more than five times to achieve the same outcome. Additionally, it is worth noting that in the worst-case scenario, CLOVE had to reroute flows a staggering 37 times

Fig. 4. After rerouting a flow, we assess the route flapping prior to achieving convergence. As depicted in this Figure, our analysis reveals that, in the case of SMOOTHIE, a specific flow experiences at most two reroutes. This finding underscores the superior path selection optimization of SMOOTHIE compared to CLOVE, in the context of route flapping quantification.

before achieving convergence. This observation underscores the importance of mitigating route flapping, which can disrupt network stability and performance [33]. Route flapping, characterized by frequent and unpredictable changes in network paths, can lead to increased latency, packet loss, and overall network instability, which are detrimental to network performance. It is the reduction of route flapping that plays a pivotal role in explaining why SMOOTHIE outperforms its counterparts, especially in high workload scenarios.

Our thorough analysis strongly supports SMOOTHIE as the preferred load balancing solution for contemporary networks. Its advantages, including centralized management simplifying configuration, and reduced monitoring impact using innovative INT signaling instead of header-heavy feedback packets, are key. SMOOTHIE's capacity to make informed decisions based on comprehensive network state suits dynamic architectures. Its impressive ability to mitigate route flapping reduces network instability and disruptions, solidifying its position as an ideal load balancing solution. Compared to decentralized alternatives, SMOOTHIE efficiently offloads processing burden from switches, prioritizing their core function of packet forwarding.

VI. RELATED WORK

Traditionally, the equal-cost multi-path routing (ECMP) technique has been the cornerstone of load balancing strategies. ECMP operates on a static path selection method, where the decision to route traffic is primarily based on the 5-tuple flow hash digest, encompassing source and destination IP addresses, source and destination ports, and the transport protocol. However, ECMP exhibits limitations, including performance degradation [4], [2], [3] under a dynamic environment.

To address these limitations, alternative load balancing strategies have emerged. Systems like HEDERA [4] and B4 [5] adopt a centralized control paradigm. These solutions continuously monitor the network's state and periodically update load-balancing decisions in response to changing conditions. While these approaches excel in specific network topologies and operational contexts, they may not be universally applicable (i.e., optimized for WAN architectures with path updates occurring at minute-scale intervals).

In contrast, decentralized load-balancing mechanisms like CLOVE [2] and HULA [3] have gained popularity for their rapid traffic re-balancing. However, these approaches use switch memory to store network-wide state information, potentially straining resources in large deployments. Additionally, these solutions rely on active probes and feedback mechanisms which introduce extra packet overhead, potentially contributing to congestion. Moreover, they often lack the path decision flexibility of centralized counterparts, relying on limited information for routing decisions.

Ultimately, a final classification of load balancers emerges, denoting those that intricately alter the network architecture, either at the host level, exemplified by MPTCP [7], [8], or at the switch level, typified by CONGA [9].

VII. CONCLUSION

This paper introduced SMOOTHIE, a novel load balancing solution tailored for private domains such as datacenters. Leveraging Inband Network Telemetry (INT), SMOOTHIE facilitates real-time network performance monitoring, enabling proactive congestion management. The p-INT parameter provides fine-grained monitoring control, prioritizing critical flows and reducing overhead for non-critical ones. Using a "power of two choices" strategy for route selection, SMOOTHIE enhances load balancing and congestion control while reducing computation overhead.

In comparison to traditional ECMP and the advanced CLOVE algorithm, SMOOTHIE consistently outperforms them, especially in high-load scenarios. This advantage primarily arises from SMOOTHIE's ability to minimize route flapping, optimizing resource usage and minimizing packet reordering at receivers.

Overall, SMOOTHIE stands out as a promising load balancing solution for modern networks. Offering centralized management, reduced monitoring overhead, and dynamic congestion control, it adapts to diverse network architectures, leveraging holistic network state for informed decision making. A valuable addition to private domain networks like datacenters.

SOFTWARE ARTEFACTS

All the code discussed in this paper (i.e., SMOOTHIE, mininet topologies, experiments) is freely available on https://github.com/Advanced-Observability.

ACKNOWLEDGMENTS

This work is supported by the CyberExcellence project funded by the Walloon Region, under number 2110186.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, August 2008.
[2] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proc. ACM CoNEXT*, November 2017.

[3] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symposium on SDN Research (SOSR)*, March 2016.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *Proc. USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2010.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, October 2013.

[6] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, August 2013.

[7] C. Raiciu, S. Barré, C. Plunkte, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, August 2011.

[8] A. Ford, C. Raiciu, M. J. Handley, and O. Bonaventure, "TCP extension for multipath operation with multiple addresses," Internet Engineering Task Force, RFC 6824, January 2013.

[9] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, and N. Yadav, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, August 2014.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[11] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. Wobker, "In-band network telemetry via programmable dataplanes," August 2015.

[12] C. Filsfils, S. Previdi, L. Ginsberg, b. Decraene, S. Litkowski, and R. Shakir, "Segment routing architecture," Internet Engineering Task Force, RFC 8402, July 2018.

[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," August 2013.

[14] INTEL, "Intel® tofino™," see https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html.

[15] "Int support on nx-os," see https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/92x/programmability/guide/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x_chapter_0100001.html.

[16] "Int support on junos os," see https://www.juniper.net/documentation/us/en/software/paragon-insights/data-ingest-guide/topics/concept/pi-inband-telemetry-overview.html#:~:text=Inband%20Network%20Telemetry%20(INT)%20is,new%20ports%20in%20flow%20paths.

[17] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, October 2001.

[18] T. Mizrahi, N. Sprecher, E. Bellagamba, and Y. Weingarten, "An overview of operations, administration, and maintenance (OAM) tools," Internet Engineering Task Force, RFC 7276, June 2014.

[19] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," in *Proc. ACM SIGCOMM*, August 2014.

[20] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, c. Filsfils, P. Camarillo, and F. Clad, "Segment routing: A comprehensive survey of research activities, standardization efforts, and implementation results," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 182–221, november 2020.

[21] A. Farrel and R. Bonica, "Segment routing: Cutting through the hype and finding the IETF's innovative nugget of gold," *IETF Journal*, vol. 13, no. 1, July 2017.

[22] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and z. Li, "Segment routing over IPv6 (SRv6) network programming," Internet Engineering Task Force, RFC 8996, February 2021.

[23] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 segment routing header (SRH)," Internet Engineering Task Force, RFC 8754, March 2020.

[24] The P4.org Applications Working Group, "Int v2.1 specification." November 2020, see https://p4.org/p4-spec/docs/INT_v2_1.pdf.

[25] S. Lumetta and M. Mitzenmacher, "Using the power of two choices to improve Bloom filters," *Internet Mathematics*, vol. 4, no. 1, pp. 17–33, 2007.

[26] C. Cooper, R. Elsässer, and T. Radzik, "The power of two choices in distributed voting," in *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, July 2014.

[27] S. Koutstaal, "Bright future of SRv6," May 2023, last Accessed: Sept. 14th, 2023. [Online]. Available: https://tinyurl.com/3xtmb998

[28] ETH Networked Systems Group, "P4utils," https://github.com/nsg-ethz/p4-utils.

[29] K. Kaur, J. Singh, and N. Ghumman, "Mininet as software defined networking testing platform," in *Proc. International Conference on Communication,Computing & Systems (CCCS)*, August 2014.

[30] "p4learning," see https://github.com/nsg-ethz/p4-learning/tree/master.

[31] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, August 2010.

[32] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[33] H. woo Park, I. Y. Yeo, H. Jang, and N. Kim, "Study on the impact of big data traffic caused by the unstable routing protocol," *Indian Journal of Science and Technology*, vol. 8, p. 59, 2015.