Charon: Load-Aware Load-Balancing in P4

Carmine Rizzi, Zhiyuan Yao, Yoann Desmouceaux, Mark Townsley, Thomas Clausen

***Abstract*—Load-Balancers play an important role in data centers as they distribute network flows across application servers and guarantee per-connection consistency. It is hard however to make fair load balancing decisions so that all resources are efficiently occupied yet not overloaded. Tracking connection states allows to infer server load states and make informed decisions, but at the cost of additional memory space consumption. This makes it hard to implement on programmable hardware, which has constrained memory but offers line-rate performance. This paper presents Charon, a stateless load-aware load balancer that has line-rate performance implemented in P4-NetFPGA. Charon passively collects load states from application servers and employs the power-of-2-choices scheme to make data-driven load balancing decisions and improve resource utilization. Per-connection consistency is preserved statelessly by encoding server ID in a covert channel. The prototype design and implementation details are described in this paper. Simulation results show performance gains in terms of load distribution fairness, quality of service, throughput and processing latency.**

***Index Terms*—load-balancing, cloud and distributed computing**

I. Introduction

Data centers (DCs) have seen a rising amount of connections to manage [1], [2] and users expect an elevated server responsiveness [3]. Due to these conditions, applications are virtualized in replicated instances in data centers to provide scalable services [4], [5]. A given service provided in a data center is identified by virtual IP (VIP). Each application instance behind the VIP is identified by direct IP (DIP). In this architecture, load balancers (LBs) play an important role. They distribute requests from clients among application servers and maintains per-connection consistency (PCC) [1], [6].

This paper exemplifies the challenges that LBs should tackle by way of a simple heuristic load balancing mechanism, *i.e.* Equal Cost Multi Path (ECMP). As is depicted in figure 1, on receipt of a new request (step ①), ECMP LBs randomly select a server among the server pool to which the request is forwarded (step ②), based on the hash over the 5-tuple of the connection[1]. The replies are sent back directly to the client instead of traversing the LBs (step ③) in direct source return (DSR) mode. DSR mode is first proposed in [1] so that LBs avoid handling 2-way traffic and becoming a throughput bottleneck between servers and clients. Though easy to implement, ECMP is agnostic to the server load states.

C. Rizzi, Z. Yao and T. Clausen are with École Polytechnique, 91128 Palaiseau, France; emails {carmine.rizzi,zhiyuan.yao,thomas.clausen}@polytechnique.edu.

C. Rizzi, Z. Yao, Y. Desmouceaux and M. Townsley are with Cisco Systems Paris Innovation and Research Laboratory (PIRL), 92782 Issy-les-Moulineaux, France; emails {crizzi, yzhiyuan, ydesmouc,townsley}@cisco.com.

[1]The 5-tuple corresponds to IP source, IP destination, protocol number, TCP source port and TCP destination port.

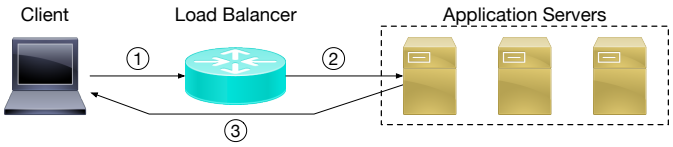


Figure 1. Network load balancer in data centers.

As ECMP randomly distribute workloads, new requests may be forwarded to overloaded servers, reducing load balancing fairness. ECMP is also not able to guarantee PCC since server pool updates change the DIP entries in the hash table, which potentially forwards subsequent packets of established connections to different servers and breaks connections.

A. Related Work

To guarantee PCC, stateful LBs keep tracking the state of the connections [1], [6]–[8]. Using advanced hashing mechanism (*e.g.* consistent hashing [6], [7]), server pool updates have little impact on the hashing table therefore the amount of disrupted connections is decreased. However, stateful LBs require additional memory space for flow tables to store connection states. When encountering DoS attacks, flow tables risk of being filled by malicious flows and no longer track benigh flows. In case of LB failures, the tracked connection states are lost and all connections via the failed LBs need to be re-established, which degrades quality of service (QoS). Stateless LBs [9]–[11] use alternative techniques to recover the right server destinations, without keeping the flows' states. They daisy-chain two possible server candidates, to retrieve a potentially changed flow-server mappings. Charon adopts stateless load balancing scheme [9], [10], [12] and encapsulates the server id inside the packet. In particular, the TCP timestamp option [13] is used to transport this information.

To improve load balancing fairness, different mechanisms are proposed to evaluate server load states before making load balancing decisions. Segment Routing (SR) [14] and power-of-2-choice [15] are used in [7], [9] to daisy chain 2 servers and let them decide, based on their actual load states, to which server a new flow is assigned. Another approach is to periodically poll servers' instant "available capacities" [8]. Ridge Regression is used in [16] to predict server load states and compute the relative "weights". In [17], the servers are clustered based on their load states, where clusters with less workload are prioritized. The servers notify the LBs about load state changes if their resource consumption surpasses pre-defined thresholds. LVS [18] presents a heuristic that combines the queue lengths of active flows and provisioned server capacity to determine server load states. Unlike prior arts, Charon passively polls and retrieves the server load when

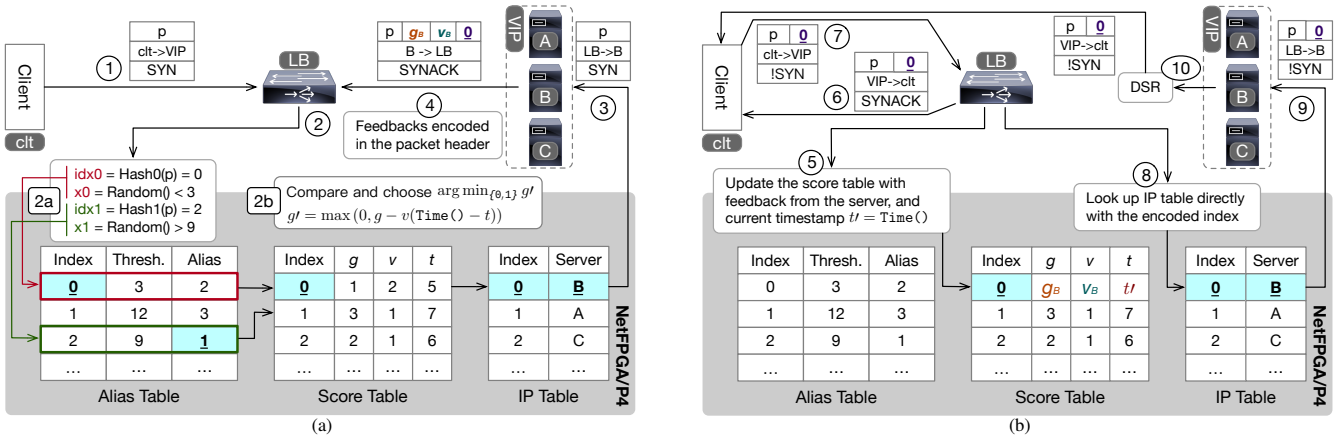


Figure 2. Charon overview.

a new flow is assigned to it. The feedback is used to predict the future server load states and make informed and fair load balancing decisions, which improves resource utilization and QoS.

To optimize performance in terms of throughput and latency, hardware solutions are proposed. SilkRoad implements LB functions on dedicated hardware device [19], while other designs implement a hybrid solution combining software and hardware LBs [3], [20]. As a hardware solution, Charon is realized on a NetFPGA board using P4-NetFPGA toolchain [21] and achieves low jitter and delay.

B. Statement of Purpose

This paper proposes Charon, a stateless, load-aware, hardware load balancer. This paper targets the fore-mentioned 3 aspects of LB performance:

Availability: encapsulates the chosen server id in the covert channel of packet headers. Different covert channels are available (*e.g.* `connection-id` of QUIC connections and the least significant bits of IPv6 addresses) [12]. This paper uses the higher-bits of TCP timestamp options.

Fairness: Charon makes load balancing decisions on predicted server load states based on passive feedback from the application servers with actual load states encoded in SYNACK packets. Two factors are integrated at the same time, *i.e.* queue lengths and processing speed.

Performance: Charon implement all functionalities on programmable hardware to boost performance and achieve low latency and high throughput.

Virtual simulations show promising results and performance gain using Charon. Physical testing also demonstrates the high throughput of the board.

C. Paper Outline

The rest of this paper is organized as follows. In section II the overview of Charon is described. Section III presents the design choices of Charon. Section IV presents the implementation of Charon and section V shows the results obtained. Section VI concludes the paper.

II. Overview

Charon relies on 3 tables and 1 server agent to achieve stateless load-aware load balancing on NetFPGA. 2 tables are constructed and managed by the control plane. The *Alias Table* allows to select servers based on various weights with low computational complexity and low memory space consumption. The *IP Table* is used to map server id to actual IP address. 1 table, namely the *Score Table*, is updated in the data plane on per-flow basis.

The workflow is exemplified in figure 2. When a SYN packet reaches the LB (step ①), Charon employs power-of-2-choices and applies 2 hash functions to the 5-tuple of the packet. The 2 hashes are then used as indexes in the "Alias Method" [22] (step 2a) to generate 2 random server candidates based on their relative weights, which is explained in section III. Referring to the Score Table, Charon calculates and compares the load states of the 2 candidate servers (step 2b). The server with lower score is assigned to the new flow. In the example of Fig. 2a, the IP of the selected server is retrieved from the IP Table as server B (step ③). At step ④, along with the reply to the connection request, the agent on server B encapsulates its load state information and its server id in the packet header. In this paper the server load state is encoded inside the key option field of the GRE header [23], which encapsulates the original IP packet[2]. This "passive feedback" design differs from other LBs and reduces communication overhead with respect to periodic polling mechanisms yet keeps LBs informed before application servers reach a critical load level. On reception of the SYNACK packet from the server (DSR is disabled for step ④), Charon updates the load state information in the Score Table. The packet is decapsulated and the response is forwarded back to the client (step ⑥). The server id (0 in the example) is preserved in the higher bits of the TCP timestamp option. In this way, the subsequent packets from the same flow (step ⑦) contains the server id, which helps Charon retrieve the server's IP address (step ⑧) from the IP Table and redirect immediately to the right server (step ⑨). The

[2]IPv6's flow id field can also be exploited to store server load information. Charon chooses the key option field of GRE header to achieve better compatibility between IPv4 and IPv6.

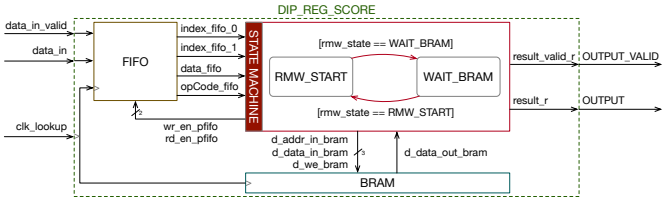


Figure 3. Schematic of `dip_reg_score` module.

server can directly answer to the client using DSR mode (step ⑩) till the end of the flow.

III. Design

The first building block of the design of Charon is the Alias Method. It is a probabilistic algorithm which, given initial weights, generates a table of probabilities and "aliases". The role of the Alias Method is to distribute with higher chance the flows to servers with higher weights. The weights are derived from servers' instant load states and are updated periodically. The update time interval is 1s and the choice is explained in section V.

As shown in figure 2a, each entry of the Alias Table has a *threshold* and an *alias*. The former determines which value is chosen, while the latter is the alternative index with respect to the entry initially selected. Generating a server candidate requires 2 input values, *i.e.* , an entry index and a random number. If the random number is bigger than the threshold, the output of the Alias Method is the alias, otherwise the entry index. In the given example, four values are taken into considerations when generating 2 server candidates. The `idx0=0` and `idx1=2` are the two initial entry indexes of the table. Given that the random value is smaller than the threshold `x0` < 3, the first output is the entry index 0. Similarly, since `x1` ≥ 9, the second output is the alias 1.

The 2 values obtained from the Alias Method are then used as the ids of the 2 server candidates. Their associated scores are computed with the function $g^0 = max(0, g - v * (Time() - t))$, where g^0 is the new score, g is the previous score of the server, v is the "velocity" or the server processing speed, $Time()$ is a function that returns current timestamp and t is the previous timestamp. The 3 variables, g, v and t, are saved in the Score Table. The score g is the amount of work remaining or the number of active flows on the server to execute. The processing speed v is derived from the average flow completion time (FCT) on the server side. The timestamp t corresponds to the last time the score was updated. The time difference `Time()` $- t$ measures the elapsed time since last update. The intuition of this function is to predict the remaining amount of tasks or queue length that a server needs to accomplish. A higher score translates into a busier server. The $max()$ function guarantees that the score stays non-negative. Once the scores of the servers are computed, the server with lower score is assigned to the flow. In the example in figure 2a, supposing that $Time() = 8$ then the scores of index 0 and 1 are $g_0^0 = max(0, 1 - 2 * (8 - 5)) = 0$ and $g_1^0 = max(0, 3 - 1 * (8 - 7)) = 2$ respectively. The selected

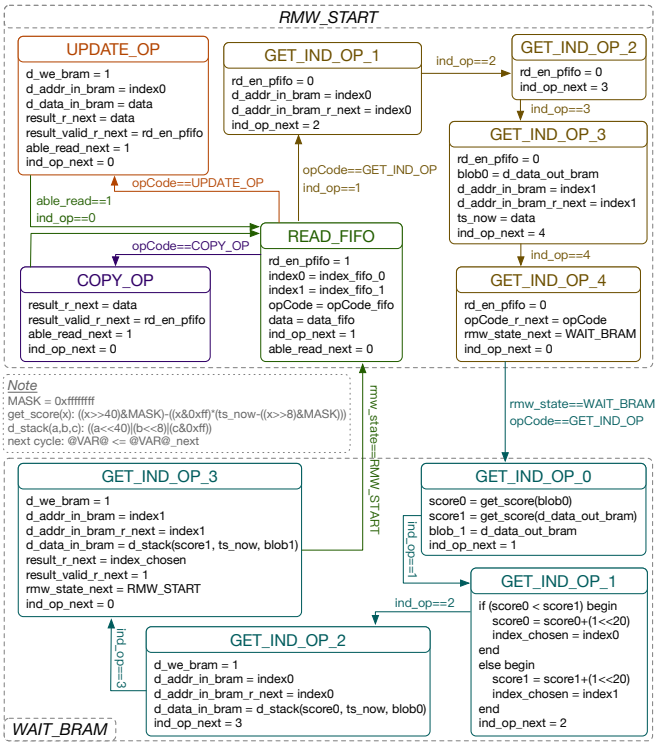


Figure 4. State machine in the `dip_reg_score` module.

server is the server with index 0, which is then mapped to server B in the IP Table. The power-of-2-choices is applied as it has lower computational complexity than calculating the minimum yet if offers recognizable performance gains [7]. For this reason, Charon better handles large-scale DCs.

The main function of Charon is implemented in a single Verilog module called `dip_reg_score`. This design choice depends mainly from the read and write actions that should be executed on multiple indexes. Figure 3 shows the architecture of this module. It takes as input `data_in_valid`, `data_in` and `clk_lookup` and as output `OUTPUT_VALID` and `OUTPUT`. The core logic of `dip_reg_score` locates in the `STATE_MACHINE` block. It interacts with `FIFO` and `BRAM` (Block RAM). The former receives and stores the inputs of the block, while the latter is used to save the server load states information. Figure 4 depicts in detail the workflow of `STATE_MACHINE`. It consists of two states `RMW_START` and `WAIT_BRAM`. Each state can be further decomposed into several states. The initial state in `RMW_START` is `READ_FIFO`. In this state the input saved in the FIFO are extracted. Depending on the `opCode` value, different operations can be executed. Three operations are available: `UPDATE_OP`, `COPY_OP` and `GET_INDEX_OP`. The code `UPDATE_OP` is used to update the server load states in the Score Table given the feedback extracted from the `SYNACK` packets sent by the application servers. The code `COPY_OP` is a buffering operation. It copies the data received from the input into the output. The code `GET_INDEX_OP` is executed when a `SYN` packet reaches the LB. It extracts the server load states with the 2 given server indexes. The reading operations require 2 clocks for each

value, which yields 4 clocks in total for reading 2 blobs from BRAM. In the WAIT_BRAM state, with the fetched blobs, the two scores are then computed. and stored in the Score Table. The server with lower score is selected and its corresponding score is increased by 1 unit task so that the new flow can be taken into account immediately. The two scores are then stored back in the Score Table before Charon forward the flow to the chosen server.

IV. Implementation

This paper implements Charon using P4-NetFPGA. P4 is a programming language in the family of Software-Defined Networking (SDN) technologies [24]. It is used to program network devices' data plane and has been applied in other LBs [11], [16], [19]. P4 has high flexibility. It can be translated into Verilog and the created module can be compiled into bitstream files using Vivado toolkit [25].

Figure 5 shows the workflow of Charon in P4's PISA model. It can be split into 3 parts: *Parser*, *Match-Action Pipeline* (MAP) and *Deparser*. The Parser separates different packet headers depending on the values of different fields. Packets with unexpected packet headers are dropped while the others will be forwarded to the MAP. In MAP, the main logic of P4 takes place using tables, which are a collection of keys and values. A possible input could be an IP address. Using longest-prefix match, one key is matched, associated to which an action can be executed as, for instance, setting the egress port value. Multiple tables can be applied during one packet processing. Charon uses 2 tables for the two server indexes obtained from the Alias Method. The last step is the Deparser, which recollects all the header information and sends a new packet out of the egress port.

Despite the flexibility of P4, it presents several limitations. For instance, an external function is necessary to create memory cells and store additional information. Depending on the hardware targets, different languages can be used to describe these external modules. Verilog is adopted for NetFPGA, which is the reason why the external module dip_reg_score is implemented in Verilog. Its complexity cannot be expressed by P4 language. This block is mainly accessed for SYN and SYNACK packets. For other types of packets, it is used as a buffer. The other external modules of Charon implemented in Verilog are the IP table, current timestamp calculation and server id extraction from the TCP timestamp option. The IP table is a fundamental component used to redirect packets coming from the client. Timestamp calculation takes place when a score computation or update happens. Server id extraction is used for any packet traversing the LB with the presence of TCP timestamp option, except for SYN packets because in this case the LB has not yet assigned any server to the flow.

In this paper some design choices are configured for these external blocks as follows. The target number of servers is defined as 16, which yields $\mathcal{O}(16)$ memory space complexity with the 3 tables[3]. The FIFO memory inside the dip_reg_score module has a queue length of 64. As a small-scale prototype implementation, the Vivado simulations have been applied only to 1 of the 4 possible Ethernet interfaces. Another assumption of this paper regards the server id encapsulation in the TCP timestamp option. To be able to encode up to 16 servers, the the server id takes 4 higher-bit length among the total 32 bits timestamp value[4]. To simplify the P4 code, the only option considered in the TCP header is the timestamp option.

The server agent is implemented as a VPP plugin [26] on each AS, which uses an Apache HTTP. This VPP node corresponds to a modified version of GRE, which encodes the instant server load state in the key option field and encapsulates the original IP packet. The number of Apache busy threads, which can be retrieved from Apache scoreboard is used in this paper as server load state and the score of the server. Other metrics for the score could be applied for different applications.

V. Evaluation

This section evaluates Charon from 3 perspectives, (i) acceptance rate of covert channel modification in packet headers, (ii) performance gain in terms of load balancing fairness and quality of service, and (iii) throughput and additional processing latency using P4-NetFPGA implementation.

A. Covert Channel Acceptance

To understand how the Internet would react to the presence of timestamp option in the TCP header, requests are sent from Paris to random sets of over 60k distinct IP addresses. The results obtained are the following:

NO CONNECTION = 45019
SUCCESS = 12876
FAILURE = 5787
TOTAL = 63682

The code NO CONNECTION is the number of connections which have not received any response regardless of the presence of the timestamp option. The code SUCCESS is the number of connections that have answered to a packet with the timestamp option. The code FAILURE is the number of connections that have not answered to a packet with the timestamp option but answered to packets without timestamp option. Pruning the first case where the IP addresses can not associated to any device or service is not available and analyzing only SUCCESS and FAILURE cases gives an acceptance rate of 68.99%. This experiment does not study the different geographic locations of the clients and servers or other factors yet it validates that the stateless design of Charon works for most end hosts. It is also in accordance with the high acceptance rate (over 86%) obtained by experimenting on a larger scale of testbed in [27].

[3] 16 is small yet can be updated at ease.

[4] The assumed maximum number of bits used for encoding server id is 8, *i.e.* 256 servers in total, which is sufficient for modern DCs [2]. Any change in the timestamp value that modifies more than 24 bits is ignored.

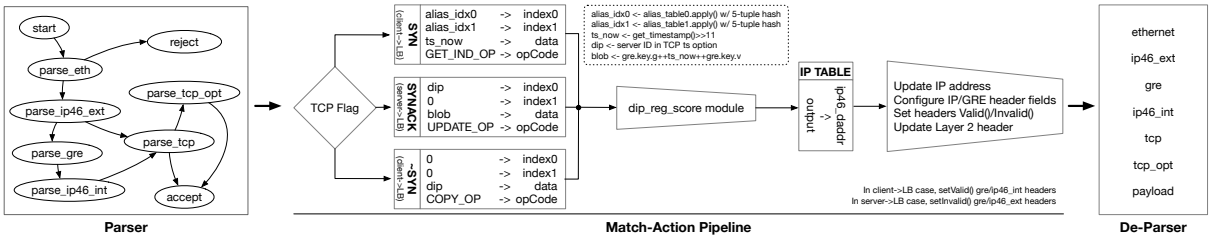
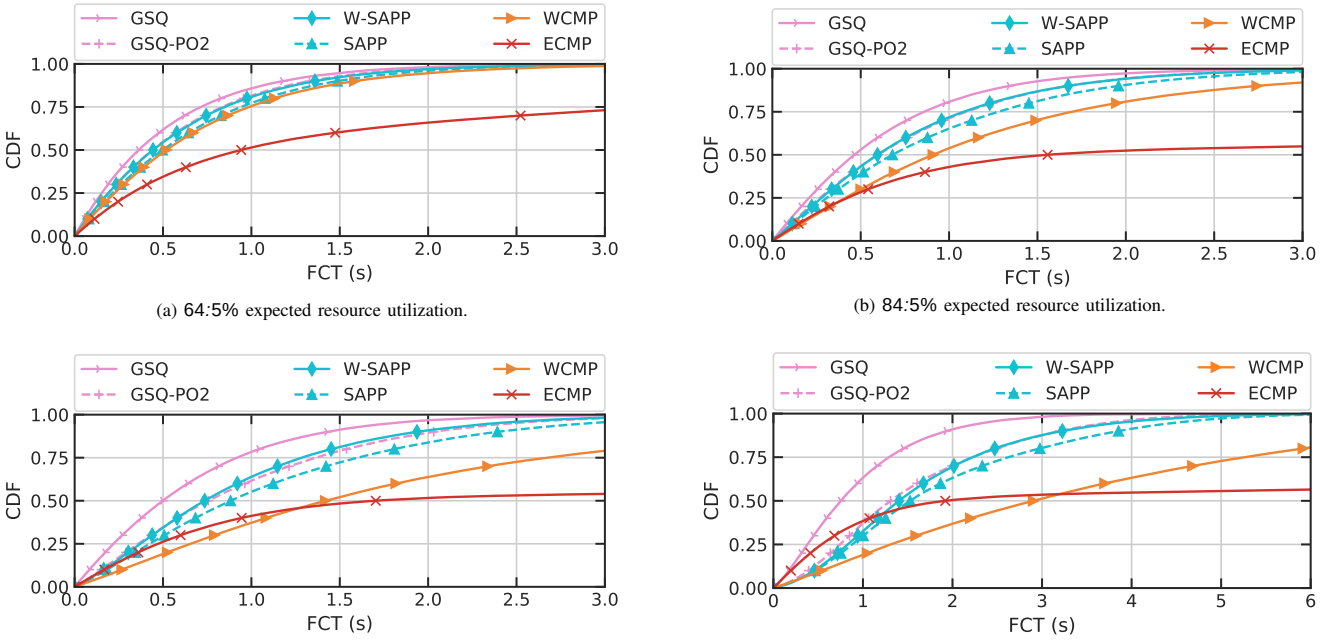


Figure 5. P4 workflow.



(a) 64:5% expected resource utilization.

(b) 84:5% expected resource utilization.

(c) 92:5% expected resource utilization.

(d) 100% expected resource utilization.

Figure 6. CDF of FCT of different LB designs at various traffic rate.

B. Load Balancing Fairness

A virtual simulator is built with 2 LBs and 64 application servers with different processing capacities[5] to study the load balancing performance in terms of workload distribution fairness. 3 episodes of 50k clients' requests for flows that last 500ms on average are simulated with Poisson traffic at different variances. The traffic rates are normalized by the total server cluster processing capacities. Figure 6 depicts the cumulative distribution function (CDF) of FCT, which is the time necessary to complete a flow. Different LB designs are compared. Global shortest queue (GSQ), as the name suggests, chooses the application server with the shortest queue. It is an oracle solution that can be achieved assuming that the LBs are aware of the actual queue lengths on each server and no computational overhead is incurred when computing the minimum queue length. It represents the best performance a LB can achieve. GSQ-PO2 applies power-of-2-choices over GSQ. Similar to GSQ, the LBs are assumed to be aware of the exact instant queue lengths on each server. Unlike GSQ, GSQ-PO2 selects 2 random server candidates and then picks the one with a shorter queue. It represents the theoretical best performance Charon can achieve. ECMP randomly selects the application servers and is the most widely employed load balancing mechanism. Weighted Cost Multi-Path (WCMP) selects the application servers according to its statically configured weights which are proportional to the server processing capacities. W-SAPP denotes the implementation of Charon. SAPP corresponds to a simplified version of Charon, where the 2 server candidates are chosen using a uniform distribution instead of using weighted sampling with the Alias method. The main difference between SAPP and W-SAPP is the probabilistic method that W-SAPP applies to obtain the 2 choices. The weights, which are used later as probabilities, are defined using the relative processing capacities of application servers. As depicted in figure 6, the performances of both SAPP and W-SAPP are similar to GSQ, which is considered as the method that takes the perfect choice. Improvements can be observed for W-SAPP over SAPP, which is the reason why the Alias Method is used in Charon. Despite its limited additional complexity, W-SAPP would be able to catch the different capacities of servers.

Another interesting metric to evaluate load balancing fairness is the Jain's fairness index [28], which computes the

[5]Half of the application servers have 2 times higher processing capacities than the other half.

Utilization	GSQ	GSQ2	W-SAPP	SAPP	WCMP	ECMP
64.5%	**0.59**	0.54	0.52	0.51	0.43	0.34
76.5%	**0.64**	0.59	0.56	0.57	0.47	0.47
84.5%	**0.68**	0.63	0.61	0.60	0.49	0.49
92.5%	**0.69**	0.67	0.66	0.66	0.54	0.51
100%	0.75	0.74	**0.76**	0.74	0.63	0.52

TABLE I
JAIN'S FAIRNESS INDEXES OF DIFFERENT LBS AT DIFFERENT TRAFFIC RATES.

fairness of workload distribution. Considering n servers each one with a particular amount of flows x_i, the fairness index is computed as $\left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2 \cdot \left(\frac{\sum_{i=1}^{n} x_i^2}{n^2}\right)^{1}$. The maximum and minimum values that the index can reach are respectively 1 and $\frac{1}{n}$ with n is the number of servers. If the index reaches value 1, it means that the load has been fairly distributed. The worst case is when the index is equal to $\frac{1}{n}$ which proves that only one server has taken all the flows.

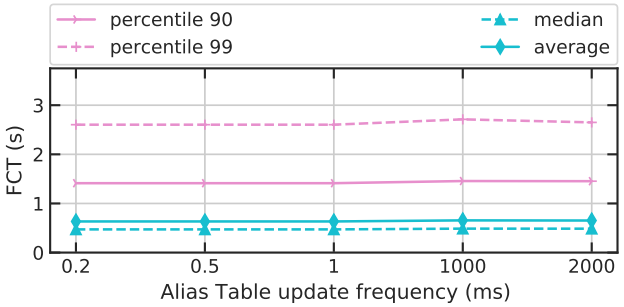
Using the same configuration as in previous simulations, the fairness indexes of different LB designs are computed. These values have been obtained periodically computing the fairness over the remaining workload of each one of the ASes during the simulation execution. These results take also in consideration the different capacities of the servers. As shown in table I, ECMP and WCMP have the worst performance. Random choices do not guarantee a fair distribution of flows. On the other hand, GSQ and GSQ-PO2 get the best fairness. They have perfect knowledge of the flows and they always choose the server with the shortest queue length. W-SAPP and SAPP achieve similar performance to GSQ and GSQ-PO2. Although the fairness indexes of SAPP and W-SAPP are not so different, W-SAPP takes into account the processing capacities of the servers which can be useful when their capacities are different inside the same server cluster. The Alias Method in Charon however, uses dynamic weights to select a subset of candidates.

Another important parameter to analyze is the update time intervals of the Alias Table. If the update time interval is too high, the LB choice would not reflect the real-time load states of the application servers. For this reason, different update time intervals are simulated.

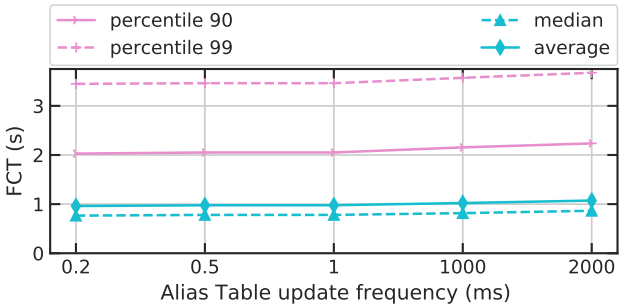
The results of the simulations are depicted in figure 7. 4 values have been taken into consideration: percentile 90, percentile 99, median and average of FCT at 2 different rates. Five different time intervals of Alias Table updates have been used: 0.2 ms, 0.5 ms, 1 ms, 1 s and 2 s. The plots show a slight improvement of FCT when the update time interval is lower (higher update frequency). This difference is too small to justify shorter time interval update. The LB are not significantly influenced by a real-time update of the weights. However, it has to be taken in consideration that these simulations are virtual. Physical experiments are necessary to further justify this assumption.

C. P4-NetFPGA Implementation Performance

The performance of NetFPGA is compared with respect to software solutions. On the NetFPGA, the `reference_NIC`



(a) 64:5% expected resource utilization.



(b) 92:5% expected resource utilization.

Figure 7. FCT of different Alias Table update interval LB designs at various traffic rate.

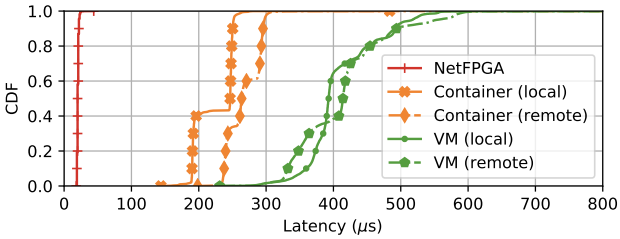


Figure 8. Latency CDF of different implementation.

bitstream file is loaded. This program returns on the PCI the packets that the NetFPGA receives from the Ethernet interface. At the same way a packet sent to the PCI is then sent from the Ethernet interface. The NetFPGA behaves as a NIC. In this way, it is possible to find at which time the packet sent through the PCI traverses one of the four Ethernet interfaces and it reaches the host machine. Figure 8 shows the Cumulative Distribution Function (CDF) of the latency. The Round-Trip Time (RTT) of ping packets of other software solutions are also depicted. In particular, four cases are considered: when there are two containers (dockers) or two VMs on the same machine or on different machines. The performance of the NetFPGA largely outperforms the software solutions, which makes NetFPGA a preferred solution in terms of performance.

To demonstrate the performance of the Verilog module `dip_reg_score`, Vivado behavioural simulations have been executed. A burst of 600 packets is sent to the NetFPGA board. The delay between packet arrival and departure is shown in figure 9.

The difference between the first 16 packets and the remaining packets is due to the nature of the packets sent. The first

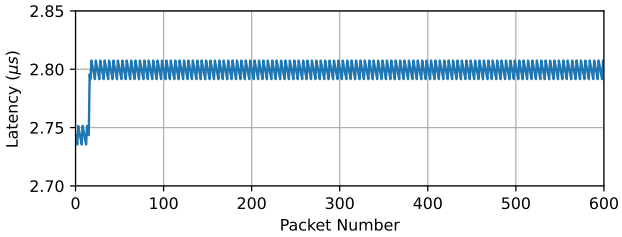


Figure 9. Delay in packet departure with respect to the number of packets sent.

batch of 16 packets traverse a shorter path as they upload the IP addresses in the IP table. The remaining packets are TCP SYN packets. They traverse the longest path in which the destination application server of the flow is chosen. The delay is almost linear to the number of cycles required for packet processing and stays constant. The sinusoidal shape is because of the jitter, which is low. This plot shows the high performance of the designed module and its efficiency in executing the proposed LB algorithm.

VI. Conclusion

This paper proposes Charon a stateless, load-aware, hardware load balancer in DCs, which (i) fairly distributes connections' requests, (ii) avoids connections breaks, and (iii) avoids additional latency due to its presence. The design choices of Charon makes it suitable for implementation on programmable hardware. Simulation results show Charon improves load balancing fairness and helps achieve better quality of service than other LB mechanisms. Evaluations of throughput and processing latency demonstrate the advantage of hardware implementations.

References

[1] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu *et al.*, "Ananta: Cloud scale load balancing," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 207–218, 2013.

[2] Facebook Engineering, "Reinventing Facebook's data center network," Mar 2019. [Online]. Available: https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/

[3] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 27–38, 2015.

[4] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[5] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[6] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer." in *NSDI*, 2016, pp. 523–535.

[7] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "6lb: Scalable and application-aware load balancing with segment routing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 819–834, 2018.

[8] A. Aghdai, C.-Y. Chu, Y. Xu, D. H. Dai, J. Xu, and H. J. Chao, "Spotlight: Scalable transport layer load balancing for data center networks," *arXiv preprint arXiv:1806.08455*, 2018.

[9] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, and T. Clausen, "Stateless load-aware load balancing in p4," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Sep 2018, p. 418–423.

[10] J. T. Araújo, L. Saino, L. Buytenhek, and R. Landa, "Balancing on the edge: Transport affinity without network state," 2018, p. 111–124.

[11] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 125–139.

[12] T. Barbette, C. Tang, H. Yao, D. Kostić, G. Q. M. Jr., P. Papadimitratos, and M. Chiesa, "A high-speed load-balancer design with guaranteed per-connection-consistency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 667–683. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/barbette

[13] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323, Sep. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7323.txt

[14] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8402.txt

[15] 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, 2001.

[16] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, and F. R. Yu, "Fast switch-based load balancer considering application server states," *IEEE/ACM Transactions on Networking*, p. 1–14, 2020.

[17] A. Aghdai, M. I.-C. Wang, Y. Xu, C. H.-P. Wen, and H. J. Chao, "In-network congestion-aware load balancing at transport layer," *arXiv preprint arXiv:1811.09731*, 2018.

[18] W. Zhang, "Linux virtual server for scalable network services," 2000.

[19] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. ACM, 2017, p. 15–28, event-place: Los Angeles, CA, USA.

[20] R. Gandhi, Y. C. Hu, C.-K. Koh, H. H. Liu, and M. Zhang, "Rubik: Unlocking the power of locality and end-point flexibility in cloud scale load balancing." in *USENIX Annual Technical Conference*, 2015, pp. 473–485.

[21] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4-netfpga workflow for line-rate packet processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–9. [Online]. Available: https://doi.org/10.1145/3289602.3293924

[22] J. Smith and S. Jacobson, "An analysis of the alias method for discrete random-variate generation," *INFORMS J. Comput.*, vol. 17, pp. 321–327, 2005.

[23] T. Li, D. Farinacci, S. P. Hanks, D. Meyer, and P. S. Traina, "Generic Routing Encapsulation (GRE)," RFC 2784, Mar. 2000. [Online]. Available: https://rfc-editor.org/rfc/rfc2784.txt

[24] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *Journal of Network and Computer Applications*, vol. 67, pp. 1–25, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516300297

[25] "Vivado." [Online]. Available: https://www.xilinx.com/support/university/vivado.html

[26] "Vpp." [Online]. Available: https://wiki.fd.io/view/VPP

[27] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend tcp?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 181–194.

[28] R. Jain, D. M. Chiu, and H. WR, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *CoRR*, vol. cs.NI/9809099, 01 1998.