HLB: Toward Load-Aware Load Balancing

Zhiyuan Yao, Yoann Desmouceaux, Juan-Antonio Cordero-Fuertes, Mark Townsley, Thomas Clausen

HLB: Towards Load-Aware Load Balancing

Zhiyuan Yao, Yoann Desmouceaux, Juan-Antonio Cordero-Fuertes, Mark Townsley, Thomas Clausen

***Abstract*—The purpose of network load balancers is to optimize quality of service to the users of a set of servers – basically, to improve response times and to reducing computing resources – by properly distributing workloads. This paper proposes a distributed, application-agnostic, Hybrid Load Balancer (HLB) that – without explicit monitoring or signaling – infers server occupancies and processing speeds, which allows making optimised workload placement decisions. This approach is evaluated both through simulations and extensive experiments, including synthetic workloads and Wikipedia replays on a real-world testbed. Results show significant performance gains, in terms of both response time and system utilisation, when compared to existing load-balancing algorithms.**

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

I. Introduction

In data centres (DCs), cloud services and network applications are associated with server clusters to provide high scalability, availability, and quality of service (QoS) [1], [2]. As a key component for efficient resource utilisation in DCs, Layer-4 *load-balancers* (LBs) distribute network traffic addressed to a given cloud service *evenly* on all associated servers, while *consistently* maintaining established connections [3]–[7].

The workflow of network LBs is depicted in Figure 1. On receipt of a new connection request ① (*e.g.*, a TCP SYN), LBs ② determine to which server the new connection is to be dispatched. Servers ③ respond to the request using direct-source-return (DSR) mode[1]; LBs thus have no access to the server-to-client side of communication. Finally, ④ the load balancing decision made upon the new connections is preserved until connection terminates.

There are two requirements for LBs:

- *Per-Connection-Consistency* (PCC): Packets from the same connection need to be forwarded to, and handled by, the same server.
- *Fairness*: LBs need to balance workloads on all servers and avoid both overloading and starvation of provisioned resources [5], [6].

If PCC is not ensured, connections will break and re-establishments may occur, which take time and degrade QoS [8], [9], thus potentially causing revenue loss for cloud providers [10]. PCC can be achieved by LB algorithms from two categories: *stateful* and *stateless*. Stateful LBs [3], [4], [11]–[14] use flow tables to store mappings between connection IDs (*e.g.*, 5-tuple hashes[2]) and servers. Stateless LBs encode connection-server mapping information in covert channels (*e.g.*, TCP timestamps) [7], or delegate the task of redirecting misrouted packets to servers [15]–[18]. Both are based on hashing to avoid connection disruption and packet misrouting, in case LBs fail or server pools update.

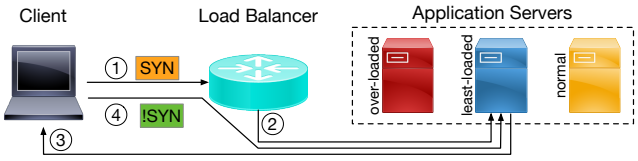


Figure 1. Workflow of Layer-4 LBs in DC.

PCC is largely explored in the literature [3], [4], [7], [11], [12], [16], [18]–[20]. Load balancing fairness and resource utilisation, in turn, are investigated in [6], [21]–[25].

A. Statement of Purpose

The overall purpose of this paper is to specifically investigate workload distribution fairness of network LBs. This paper proposes Hybrid LB (HLB), a distributed, load-aware load-balancing algorithm that infers server occupancies and processing speeds for making optimised load balancing decisions. HLB requires no explicit monitoring or signaling, which incurs additional management traffic that grows with the scale of server pools and probing frequencies[3], and reduces the effective bandwidth in the core links [11].

This paper also argues that to improve workload distribution fairness and QoS, the server load information needs to be taken into consideration, including:

- server occupancy, that indicates queuing delays,
- processing speed, determined by available resources.

Doing so allows HLB to make per-connection-level load balancing decisions and offer each server subject to a fair share of workloads. HLB estimates these factors with no additional overhead for coordination among LBs, or with servers. HLB works out of the box and requires no network or application modification, nor additional control message.

The contributions of this paper are three-fold: (i) a study of the dominating factors in load balancing performance, with a taxonomy of existing approaches, (ii) specification of a "fair"

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

Z. Yao, J.-A. Cordero-Fuertes and T. Clausen are with École Polytechnique, 91128 Palaiseau, France; emails {zhiyuan.yao,juan- antonio.cordero-fuertes,thomas.clausen}@polytechnique.edu.

Digital Object Identifier 10.1109/TNET.2022.3177163

[1]DSR is enabled for response packets from servers to clients to bypass LBs. It relieves LBs of handling 2-way traffic, improving network throughput [3].

[2]TCP 5-tuple consists of source IP address, destation IP address, source port, destination port, and protocal number.

[3]With 50-byte packets, active probing 128 servers at 10Hz generates 64kbps traffic, while the 90-th percentile of per-destination-rack flow rate is 100kbps in production [28].

TABLE I
TAXONOMY OF RELATED WORK.

LB Algorithms	Description	Related Works	Aware of Server Capacities	Aware of Server Occupancy	No Error-Prone Configurations
ECMP	Randomly assigns a server.	[11], [16], [26]	✗	✗	✓
WCMP	Assigns servers based on weights defined by provisioned resources.	[3], [4], [18], [12], [14], [19]	✓	✗	✗
AWCMP	Assigns servers based on weights defined by polled resource utilisation.	[6], [22], [24]	✗	✓	✗
LSQ/GSQ2	Assigns servers with the shortest/shorter queue occupancy based on local/global observations.	[23], [25], [27], [7]	✗	✓	✓
SED	Assigns servers with the lowest delay derived from static server weights defined by provisioned resources.	[21]	✓	✓	✗
HLB	Assigns servers with the lowest delay derived from *adaptive* server weights based on passive observations.	This work.	✓	✓	✓

LB algorithm, HLB, that requires no manual configuration, or additional interaction with servers or other LBs, (iii) evaluations, by way of simulations and testbed experiments, that compare HLB with existing LB algorithms, in various DC configurations, and under realistic network traffic.

B. Related Work

LVS (Linux Virtual Server) [21] implements a wide range of load balancing algorithms to improve fairness, however, without attaining throughput and latency characteristics meeting production requirements for DCs. Using statically configured *match action tables* or hash tables [3], [4], [12] increases throughput and reduces packet processing latencies. However, these tables do not support advanced load balancing algorithms, *e.g.*, weighted round-robin [29] or least loaded server [25], which requires dynamically managing connection-server mappings. Cheetah [7] allows dynamically registering and recovering mappings of connections and servers, by encoding mappings as cookies in covert channels in packet headers. This allows to retrieve the server handling a given connection if it is lost, *e.g.*, when an LB fails. Prism [19] *statelessly* maps connections to their hash buckets and *statefully* registers connection-server mapping information in a table of migrated connections when facing potential risks of connection disruptions, *e.g.*, during server pool updates. When servers are added or removed, it creates an independent table to track migrated connections, and updates server weights for balanced workloads distribution[4]. Integrating these algorithms [7], [19] will allow HLB to build load-aware algorithms while guaranteeing PCC for large-scale DCs.

The load-aware LB decision-making process uses the estimation of server occupancy and processing speeds, as well on use of the application of different rules (probabilistic or minimisation rules). Table I summarises the taxonomy of network LB algorithms, based on their awareness of server occupancies and processing speeds.

1) Equal-Cost Multi-Path (**ECMP**) treats all servers as equal, and is agnostic to server load state differences. It is applied in many LB mechanisms [11], [12], [16], [17], [26] that aim at minimizing performance overhead.
2) Weighted-Cost Multi-Path (**WCMP**) assigns weights to servers proportional to their provisioned resources [3], [4], [13], [14], [19], which may not correspond to their actual processing capacity. However, as available server capacities change with time in elastic DCs [2] or when workloads are co-located in a shared infrastructure [31], [32], these quantified capacities may not correspond to the actual processing capacities of servers.
3) Active WCMP (**AWCMP**) is a variant of WCMP. It periodically updates server weights, based on probed resource utilisation information (CPU/memory/IO usage) [6], [22], [33], [34]. AWCMP requires server modifications to manage communication channels and collect observations. Higher probing frequencies help achieve more accurate server load estimation, yet lead to increased volume of control messages and reduced bandwidth [6], [11].
4) Local Shortest Queue (**LSQ**) tracks for each server the number of established connections at a per-flow level [7], [25]. On arrival of new connection requests, LBs assign the corresponding connections to the server with the shortest queue based on observed traffic. Global Shortest Queue with Power-of-2-Choices (**GSQ2**) is a LSQ variant that leverages (i) the actual server queue occupancy, and (ii) the power of choices [35], [36].
5) Shortest Expected Delay (**SED**) derives the "expected delay" as server occupancy divided by statically configured server processing speed [21]. New connections are then assigned to the server with the minimal "expected delay."

Among load-aware LBs, TWF [25] obtains the actual queue lengths on each server via periodic out-of-band communications. It uses statistical models to reduce the impact of outdated observations. However, TWF assumes that all servers have the same processing speed, which is not the case with servers instantiated on heterogeneous architectures [32], malfunctioning servers, or servers running colocated workloads [31]. SED [21] statically configures server processing speeds (based on provisioned server CPU numbers) which neither reflect the actual processing speed for a given application (*e.g.*, IO-

[4]This approach of adding a table to track migrated connections is also employed in Yoda [30], a Layer-7 LB.

intensive) nor adapt to server health or operational status. HLB considers server occupancies and adaptively updates server processing speeds based on passive observations to improve load balancing performance with little additional overhead.

6LB [23] and SHELL [27] offload the fine-grained load balancing decision-making processes to servers, and allow them to hand off requests to another server using SRv6 [37], if they are already overloaded. Spotlight [6] and LBAS [24] periodically poll each server for their resource utilisation information, and either classify servers into several priority classes, or predict server load states using Ridge Regression [38], so as to dynamically update server weights. INCAB [22] tunes server weights on receipt of notifications from overloaded servers, which are defined by manually configured thresholds. Unlike these LB algorithms, HLB is an out-of-the-box LB and passively collects networking features. It requires no monitoring or signaling among networking devices, and avoids error-prone manual configurations.

C. Paper Outline

The remainder of this paper is organized as follows. Section II studies 2 dominating factors in LB performance, and discusses the challenges of implementing load-aware LBs. Section IV describes the design of HLB and its load balancing decision-making process. Section V describes the implementation details of the testbed and the simulator for conducting evaluations. Section VI presents quantitative evaluations, comparing and contrasting different load balancing algorithms under various scenarios. Section VII concludes this paper.

II. Problem Space

This section formalises the load balancing problem addressed in this paper, and defines the problem space by introducing notation and assumptions.

A load balancing system consists of one or several LBs, connected to a set of servers in a DC. The set of servers is denoted $\mathbb{S} = \{s_1, ..., s_N\}$, with $|\mathbb{S}| = N$. Given a server, $s_i \in \mathbb{S}$, its processing speed is denoted μ_i, and its total number of jobs or connections in the queue is denoted l_i.

All LBs implement the same LB algorithm. When there are multiple LBs, the traffic (a stream of jobs) injected into the system is randomly distributed among the LBs, *e.g.*, by the edge router of the DC. Each LB thus is exposed only to a fraction of the connections in the system - those traversing that LB. The occupancy estimator of server s_i on LB j, is denoted $\tilde{l}_{ij}$ ($\tilde{l}_{ij} \leq l_i$). The observed and unobserved traffic rates on server s_i are denoted λ_i and γ_i respectively, with the total observed traffic rates as $\lambda = \sum_{s_i \in \mathbb{S}} \lambda_i$, and total unobserved traffic rates as $\gamma = \sum_{s_i \in \mathbb{S}} \gamma_i$, that subject to the system.

The following hypotheses are made for the remainder of the paper:

- **TCP** still is the most widely used protocol in content delivery networks (CDNs) [28], [39]. Further, this assumption allows experimenting using existing connection traces [40], and does not limit the generality of the results over any other connection-oriented transport protocols (*e.g.*, QUIC [41]).
- **Finite-duration connections** are assumed for a connection q with a flow-completion time (FCT) of $T(q) < \infty$. $T(q)$ is modeled as a random variable with a uni-modal distribution (*e.g.*, a long-tail distribution as in [28], [31]).
- **Non-communicating LBs**, *i.e.*, LBs do not communicate with each other. LBs implementation complexity is reduced [4], [18], especially for high-performance LBs deployed on dedicated hardware [12], [26].
- Unless otherwise specified, modeling, simulations and experiments rely on the hypothesis that traffic follows Poisson distribution.

III. Analysis of Existing LB Algorithms

Given the defined problem space, this section provides an analytic examination of performances of different LB algorithms in a simple setup, and analyse the impact of inaccuracies in their input parameters. The trade-off between performance and overhead of different design choices is discussed, and the remaining challenges, that motivate the design of HLB, are presented.

A. Stochastic Modeling and Simulation

The performance and operation of the LB algorithms described in Section I-B, as well as their sensibility to inaccuracies in their input parameters, is analysed stochastically, on a basic load balancing setup, with 2 servers with a processing speed ratio $\frac{\mu_1}{\mu_2} = 2$ (*i.e.*, server 1 is 2x faster than server 2).

Each server has a queue of size Q, such that $0 \leq l_1, l_2 \leq Q$. Traffic arrivals and departures are modeled as Poisson processes with rates λ (observed traffic), γ (unobserved traffic), and μ_1, μ_2. With sufficiently short timeslots, it can be assumed that only one arrival or departure (at most) happen at a given timeslot (*i.e.*, $\sum_{i=1}^{2}(\lambda_i + \gamma_i + \mu_i) \leq 1$); the system is then Markovian with the state (l_1, l_2), departure rates (μ_1, μ_2), and arrival rates $(\lambda_1, \lambda_2, \gamma_1, \gamma_2)$. For simplicity, the system works at *nominal* capacity (*i.e.*, $\lambda + \gamma = \mu$). In these conditions, Table II describes the traffic arrival rate λ_i assigned to server i using different LB algorithms. Note that this section studies LB algorithms (ECMP, WCMP, LSQ, SED) that correspond to fundamentally different design choices, while AWCMP and GSQ2 are variants of WCMP and LSQ respectively.

With $s_i(n)_{l_i}$ denoting the probability (or probability density function), of server s_i to have a queue length of l_i at time-step n, the transition of server occupancies between two time-steps can be described as:

$$\begin{aligned} s_i(n)_{l_i} - s_i(n-1)_{l_i} \quad = \quad & (\lambda_i + \gamma_i) \cdot s_i(n-1)_{l_i - 1} + \\ & + \mu_i \cdot s_i(n-1)_{l_i + 1} - \\ & - (\lambda_i + \gamma_i + \mu_i) \cdot s_i(n-1)_{l_i} \end{aligned}$$

for $0 < l_i < Q$ (corner cases are treated accordingly).

Figure 2 depicts the LB performance of each LB algorithm, measured as the weighted service duration of a connection ($\sum_{i \in \{1,2\}} \frac{l_i}{l_1 + l_2} \frac{l_i}{\mu_i}$), for different configurations.

When the LB observes 100% traffic (*i.e.*, $\gamma = 0$) and assigns server weights based on actual processing speeds $\frac{w_1}{w_2} = \frac{\mu_1}{\mu_2} = 2$, the WCMP and SED have the best performance. By considering the state of the queues, LSQ is able to largely

TABLE II
TRAFFIC DISTRIBUTION IN 2-SERVERS LB SYSTEM.

Algorithm	λ_i
ECMP	$\frac{1}{2}$
WCMP (AWCMP)	$\lambda \cdot \frac{\mu_1}{\mu_1+\mu_2}$
LSQ (GSQ2)	$\lambda \cdot Pr\{i = \arg\min_{j=1,2} \tilde{l}_j\}$
SED	$\lambda \cdot Pr\{i = \arg\min_{j=1,2} \frac{\tilde{l}_j+1}{\tilde{w}_j}\}$

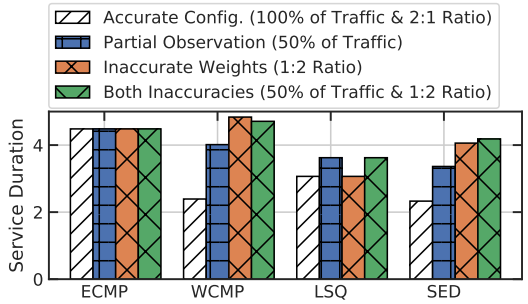


Figure 2. Load balancing performance for a cluster of 2 servers with different processing speeds ($\frac{\mu_1}{\mu_2} = 2$) under various scenarios for algorithms that consider different factors under system steady state ($\lambda + \gamma = \mu_1 + \mu_2$).

outperform ECMP. When the LB observes only 50% of traffic (*i.e.*, $\gamma = \lambda$) and the other 50% of traffic is uniformly split between the two servers ($\gamma_1 = \gamma_2$), LSQ and SED outperform WCMP, which is agnostic to instant server occupancy. However, partial traffic observation substantially degrades the performance of LSQ and SED. As a LSQ variant, GSQ2 gets global observations thus it is not subject to any impact from this source of inaccurate observation.

When LBs have inaccurate server weights (*e.g.*, in case of misconfiguration, $\frac{w_1}{w_2} = \frac{1}{2}$, while $\frac{\mu_1}{\mu_2} = 2$), WCMP and SED exhibit degraded performance even when the LB sees all the traffic ($\gamma = 0$). As a WCMP variant, AWCMP derives server weights from servers and may avoid the negative impact of misconfigurations, though with additional communication overheads. Taking both server occupancies and processing speeds into account, SED makes more informed load balancing decisions. However, while LSQ is only sensitive to partial observation, the performance of SED can be degraded by both inaccuracy sources: (i) partial observations on server occupancies, and (ii) inaccurate server weights.

B. Challenges

Section III-A shows that, performance degrades with 2 sources of inaccuracies (partially observed traffic, and misconfigured server weights), which are found to be present in production DCs [4], [23], which are challenging to resolve.

A single LB allows to observe all traffic, but also constitutes a single point of failure [42]. Multiple LBs are thus deployed for reliability, leading to partial observations.

Existing load-aware LBs gather observations of server occupancies either by actively probing or passively observing networking traffic. As such, these mechanisms are exposed to the trade-offs between performance and overhead:

- *Estimating occupancies* based on passive traffic observation at the LBs requires tracking connection states – whereas incurs substantial underestimation of server occupancies, if multiple LBs exists in the system [25].

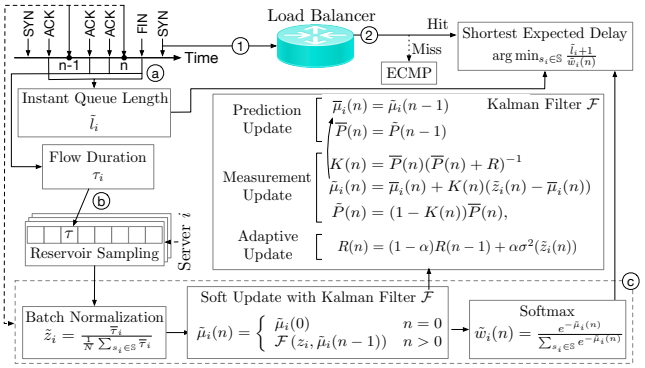


Figure 3. HLB workflow overview. Step ① and ② represent the decision making process on arrival of new flows. In step ⓐ−ⓒ, HLB collects networking observations and periodically learns server load states.

- *Actively probing server load information* allows an LB to obtain accurate but delayed server occupancies. Higher probing frequencies may increase load balancing fairness – but maintaining additional communications incurs management traffic and complexity [6].

While it is possible to avoid inaccuracies due to partial or delayed observations (as in [23], [27]), this requires server modifications to accommodate further feedback mechanisms.

Apart from partial observations and delayed updates, it is hard to determine the optimal weights for different applications that rely on different resources [43]. Besides, explicit weights configuration cannot capture or adapt to the dynamic networking environment. "Correctly" assigning weights to servers is therefore challenging in cloud DCs because:

- servers may have different provisioned resources;
- colocated workloads not captured by the LB may reduce available resources [31] on shared infrastructures;
- applications may have different profiles (*e.g.*, CPU-intensive, IO-intensive) and consume provisioned resources differently, whose impact is hard to quantify [44].

IV. HLB DESIGN

HLB dynamically distributes workloads on servers using estimations of both server occupancies and processing speeds. HLB estimates server weights and queue occupation from passive observations on network connections, by sampling flow durations and counting ongoing connections, respectively. HLB minimizes: (i) instant server load state estimation errors due to inaccurate observations, (ii) mismatches between assigned server weights and actual server processing speeds, and (iii) performance and management overhead, to improve load balancing performance.

HLB consists of two components: (i) a server state observation mechanism, and (ii) an algorithm that uses observed server states to place the incoming connection onto a server.

The first component tracks connection states using reserved memory locations (*buckets*) in flow tables, and extracts server state observations, without additional control or signaling. As depicted in Figure 3, on receipt of new packets, HLB:

ⓐ inspects headers, and passively gathers observations (numbers of ongoing flows $\tilde{l}_i$ and flow durations τ),

ⓑ gathers statistical flow duration distributions on each server with reservoir sampling,

ⓒ at each time-step n, periodically learns from gathered flow durations and updates estimated processing speeds of each server with Kalman filters.

For the second component, when receiving ① a new connection request, HLB ② computes the hash digest of the connection ID and maps the connection to a corresponding bucket in its flow table. HLB then integrates estimated server occupancies and processing speeds using the SED rule, to generate server state estimations, for all servers. The server with the lowest estimated load then receives the connection. HLB uses an adaptive approach based on passively collected observations of network connections, with no additional monitoring or management overhead – in contrast to SED, which relies on manual server weight configurations.

A. Observation Extraction from The Data Plane

As the LB sees only traffic from clients and to servers within the DC, and not the return-traffic, HLB statefully maintains connection states using flow tables, and estimates (i) server occupancies ($\tilde{l}_i$) by counting the number of ongoing connections per server, and (ii) server processing speeds by collecting flow durations on each server.

1) Stateful Observation Extraction: TCP connections are identified by their 5-tuples and are statefully tracked in flow tables. As depicted in Figure 4a, for the purpose of the LB, a connection exists in one of the three states. On receipt of the first TCP `SYN` packet from the client, the LB selects a server s_i to which the new connection is assigned. The LB also registers the connection state `SYN`, along with the connection ID and other corresponding information, in a bucket in its flow table. Once the connection is established, its state is updated to `CONN` (connected) on reception of the first data packet. On connection termination (reception of `FIN` or `RST` packets), or in the case of connection timeout, the connection state is reset to `NULL` to evict the registered connection from the flow table, and the bucket is available for a new connection.

The state machine in Figure 4a allows dynamically (i) tracking the number of ongoing connections $\tilde{l}_i$ when connections transit from `SYN` to `CONN` or from `CONN` to `NULL`, and (ii) collecting samples of flow durations from connections in state `CONN`, without interrupting the data plane.

2) Flow Table Workflow: HLB stores connection states in a flow table (Figure 4b). Each bucket comprises: (i) the hash digest of the TCP 5-tuple (`hash`) as connection ID, (ii) the target server ID assigned by the LB (`DIP`), (iii) connection "liveness", renewed on receipt of new packets (`timeout`), (iv) the first data packet arrival time (`T0`), for computing flow durations, and (v) the state of the connection (`state`).

A new connection is hashed and registered in corresponding bucket in the flow table, along with its assigned `DIP`s. Subsequent packets of the established connection are encapsulated with the target `DIP` as destination and forwarded to the corresponding server. When a bucket is not available on receipt of a new connection request, the connection gets a "miss" and

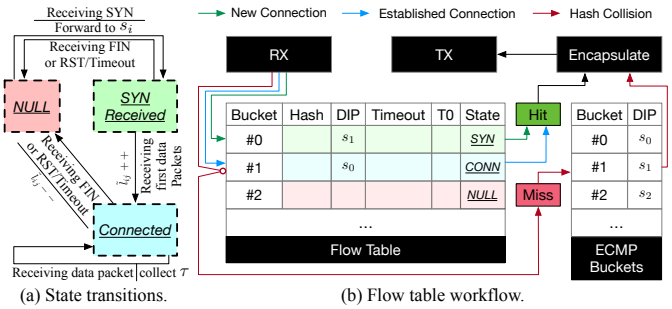


Figure 4. HLB's observation collection mechanism.

is excluded by both the observation extraction and the load-aware load balancing process. In that case (hash collision[5]), HLB falls back to ECMP for the "miss"-ed connection yet guarantees PCC. A "miss" can happen in 2 cases:

- if there is no available entry for new connections;
- if no matched entry is found for established connections;

where available buckets have `NULL` state. For connections registered in buckets that have `SYN`/`CONN` states, the counter of ongoing connections $\tilde{l}_i$ is not incremented until the first data packet is received, so that the counter is not corrupted under `SYN` flooding attacks. $\tilde{l}_i$ is decremented only if the connection ends and its state transits from `CONN` to `NULL`[6].

B. Load-Aware Load Balancing Algorithm

HLB estimates server occupancies, and server processing speeds, with l_i and τ_i, respectively, extracted from the data plane as described in Section IV-A. To minimize the additional processing and memory overhead, HLB uses reservoir sampling to collect flow durations for each server. HLB then processes collected flow duration samples using Kalman filters, to smoothly adapt server weights $\tilde{w}_i(n)$ at each time-step n, and uses both parameters, $\tilde{l}_i$ and $\tilde{w}_i$, for making load-aware load balancing decisions.

1) Sampling Flow Durations to Estimate Processing Speeds: HLB collects flow duration information on each server using *reservoir sampling*, which is a statistical mechanism that helps gather a representative group of samples, in fix-sized buffers from a stream, with minimized computational overhead and memory usage for high-performance data planes [45], [46]. Reservoir sampling collects an exponentially-distributed number of samples over time and gives more importance to "fresher" observations.

The procedure of reservoir sampling that implements the state machine from Figure 4a, which collects flow durations in connection state `CONN`, is shown in Algorithm 1. The arrival times t of data packets received from an established connection with server s_i are compared with the arrival time of the first packet $t0$ stored in $t0Table$ to compute flow durations τ. These flow durations τ, along with their corresponding packet arrival times t, are stored in a fix-sized buffer $tauBuf$ of the

[5]To reduce hash collision probability, each bucket can have multiple entries. The implementation detail is omitted since it is out of the scope of this paper.

[6]Similar DDoS mitigation mechanism using flow tables is proposed in [19], but it is out of the scope of this paper.

Algorithm 1 Collect flow durations with reservoir sampling

```
N ← number of servers
k ← reservoir buffer size
L ← flow table size
buf ← [(0,0),...,(0,0)]                    ▷ Size of k
tauBuf ← [buf,...,buf]                     ▷ Size of N
t0Table ← [0,...,0]                        ▷ Size of L
stateTable ← [0,...,0]                     ▷ Size of L
for each packet (towards s_i from query q arriving at t) do
    qid ← Hash5Tuple(q)
    i ← s_i.id
    if t0Table[qid] == 0 and SYN packet then
        t0Table[qid] ← t                   ▷ store t_0
        stateTable[qid] ← SYN
    else if t0Table[qid] ! = 0 and not SYN packet then
        lastState ← stateTable[qid]
        if lastState == CONN then
            randomId ← rand()
            idx ← randomId%N               ▷ randomly select one index
            τ ← t − t0Table[qid]           ▷ calculate duration
            tauBuf[i][idx] ← (t, τ)        ▷ register τ in buffer
        else
            stateTable[qid] ← CONN
        end if
        if FIN packet then
            t0Table[qid] ← 0               ▷ evict finished query
            if lastState == CONN then
                stateTable[qid] ← NULL
            end if
        end if
    end if
end for
```

corresponding server s_i. These buffers thus serve as a snapshot that captures a statistical distribution of flow durations on each server, and are made available for data processing.

HLB uses flow durations to estimate and infer server processing speeds for the following reasons. First, since connections addressed to the same network application are expected to terminate with FCTs of a certain distribution $T(q)$ given sufficient privisioned resources, observed flow durations are correlated to server processing speeds depending on available resources in each server (*e.g.*, overloaded CPUs, drained memory space, congested IO). Second, flow duration is collected on receipt of each new packet of the connection in state `CONN`, thus provides measurements with high granularity (update frequency) and reduced delay. Third, as an estimator of server processing speeds, flow duration can be generalized for connection-less transport protocols (*e.g.*, UDP).

2) Periodic Processing Speed Inference with Kalman Filter: With flow durations gathered in reservoir buffers, HLB computes the average flow duration $\overline{\tau}_i$ on server i as observed by the LB, and then derives the normalized server processing duration measurement $\tilde{z}_i(n) = \frac{\overline{\tau}_i}{\frac{1}{N}\sum_{s_i \in \mathbb{S}} \overline{\tau}_i}$ at each time-step n. Between different time-steps, the samples of $\tilde{z}_i$ may have high variance, and their values can change significantly. As a function of flow duration τ_i, $\tilde{z}_i$ is correlated to server processing speeds. In addition, $\tilde{z}_i$ also depends on the distribution of $T(q)$, which may vary in time. To decouple the possibly abrupt variations of $T(q)$, and adapt to actual server states, HLB uses Kalman filters $\mathcal{F}$ to smoothly update server processing duration estimations $\tilde{\mu}_i(n)$ at step n:

$$\tilde{\mu}_i(n) = \begin{cases} \tilde{\mu}_i(0) & n = 0 \\ \mathcal{F}\left(\tilde{z}_i, \tilde{\mu}_i(n-1)\right) & n > 0 \end{cases},$$

where $\tilde{\mu}_i(0)$ is initialised as 0.5 on all servers.

The Kalman filter takes streams of measurements observed over time and tracks estimated system state as well as the level of uncertainty. It works in the following 2-step process:

- Prediction update:

$$\overline{\mu}_i(n) = \tilde{\mu}_i(n-1)$$
$$\overline{P}(n) = \tilde{P}(n-1),$$

- Measurement update:

$$K(n) = \overline{P}(n)(\overline{P}(n) + R)^{-1}$$
$$\tilde{\mu}_i(n) = \overline{\mu}_i(n) + K(n)(z_i(n) - \overline{\mu}_i(n))$$
$$\tilde{P}(n) = (1 - K(n))\overline{P}(n),$$

where $\overline{\mu}_i$ is the predicted processing duration, $\overline{P}$ is the expected prediction error, R is the measurement variance, $\tilde{P}$ is the expected estimation error, and K is the Kalman gain.

The only parameter to be tuned is the measurement variance R, which can be configured based on the expected noise in measurements z_i. The value of R can be increased if the flow durations of input traffic vary a lot. To avoid manual configuration, R can be adaptively estimated using the variance of measurements $\sigma^2(z_i)$, and can be smoothly updated as $R(n) = (1-\alpha)R(n-1) + \alpha\sigma^2(z_i(n))$, where $\alpha = 0.01$ helps regularize the variation in z_i.

3) Merging Occupancy and Processing Speed: The server processing duration estimation from the latest step $\tilde{\mu}_i(n)$ is used to calculate the weight $\tilde{w}_i$ assigned to each server in the following form:

$$\tilde{w}_i(n) = \frac{e^{-\tilde{\mu}_i(n)}}{\sum_{s_i \in \mathbb{S}} e^{-\tilde{\mu}_i(n)}} \in (0,1),$$

which normalizes the negation of server processing duration estimations, and creates a probability distribution centered around the servers with higher estimated processing speeds.

After obtaining both measurements, *i.e.*, server occupancy $\tilde{l}_i$ and inferred processing speed $\tilde{w}_i(n)$, a score is computed from these two factors using SED. During the time interval of a step n, the target server s_i is selected by:

$$\arg\min_{s_i \in \mathbb{S}} \frac{\tilde{l}_i + 1}{\tilde{w}_i(n)},$$

where the added 1 on the numerator takes the new incoming connection into account. This form gives priority to servers with high estimated processing speeds and low occupancies.

V. Experimental Setups

To evaluate LB performances in different realistic setups, subject to both partial traffic observations and potential server weights misconfigurations as described in Section III, a physical testbed is configured and deployed, and an event-based simulator is implemented. This allows testing with realistic network traces – when available – and large-scale simulated scenarios.

A. Testbed

Experiments are conducted on a testbed running network traces on physical servers. The experimental platform consists of VMs representing clients, an edge router, load-balancers, and Apache HTTP server agents as depicted in Figure 5.

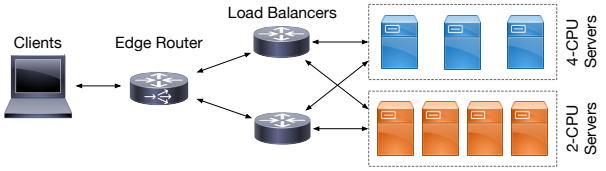


Figure 5. An example of network topology with two groups of 7 servers.

1) Load-Balancers: HLB, along with other LB algorithms, is implemented as a plugin to VPP (the Vector Packet Processor) [46], a performant packet-processing stack that runs on commodity CPUs. The number of buckets (160 bits/entry) in the flow table for stateful LB algorithms is set to 65536. Each load balancer is connected to all application servers.

2) Apache HTTP Servers: Running on each server VM, Apache HTTP servers gather two metrics every 200ms as "ground truth"-s for the occupancies: CPU utilisation, and the number of busy Apache worker threads[7]. The Apache servers use the `mpm_prefork` module to boost performance. Each server has 32 worker threads, and the TCP backlog is set to 128. In the Linux kernel, the `tcp_abort_on_overflow` parameter is enabled, so that a TCP `RST` will be triggered when the queue capacity of TCP connection backlog is exceeded, instead of silently dropping the packet and waiting for a `SYN` retransmit. With this configuration, similar to [5], [23], the FCT measures application response delays rather than potential TCP `SYN` retransmit delays. Servers are organized into 2 groups, where server capacities within a group are identical, yet may be different between the 2 groups.

3) System Platform: Depending on the scale of experiments, the testbed resides on 2 to 4 physical machines, each with a 48-CPU Intel Xeon E5-2690 CPU. An 8-CPU traffic generator, representing the clients, and a 4-CPU edge router node, run on one machine. The other machine hosts 4-CPU VMs running LB instances on VPP. The number of CPUs of Apache HTTP servers may vary from 2 to 8 under different configurations. All VMs are on the same Layer-2 link, with statically configured routing tables. The mean round-trip-time (RTT) between 2 network nodes is 0.322ms and the standard deviation of RTT is 0.037ms.

4) Wikipedia Replay: In order to evaluate LB performance under realistic workloads, LB algorithms are evaluated in a DC setup that provides typical Web services. To emulate Wikipedia server clusters, on each server instance, an instance of MediaWiki[8] of version 1.30 is installed along with the `memcached` daemon and a MySQL database server. The database server is populated by a copy of the English version of Wikipedia database dump [40]. The sizes of Wiki pages follow a long-tail distribution, whose average and standard deviation are both 12KiB. With the configured *WikiLoader* tool [47], each server is an independent replication of the Wikipedia server. The traffic generator is used to generate a MediaWiki access trace and to record page response times.

[7]CPU utilization is calculated from the file `/proc/stat` and the amount of Apache busy threads is assessed via Apache's *scoreboard* shared memory.

[8]https://www.mediawiki.org/wiki/Download

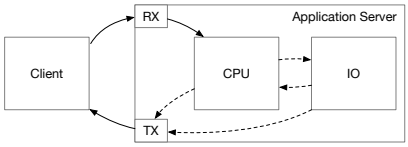


Figure 6. Illustration of the processing states of connection requests. Solid and dashed arrows represent deterministic and non-deterministic procedures respectively.

B. Simulator

To compare and contrast the performance of load balancing algorithms in various scenarios, in particular those difficult or where no network trace exists to evaluate in testbeds (*e.g.*, large-scale DC networks), an event-driven simulator is implemented, based on hypotheses described in Section II. The simulator implements the network topology as in Figure 5, where each load balancer is connected to all servers.

Real-world network applications can be CPU-bound or IO-bound [48], [49]. The simulator allows configuring applications that require multi-stage processes switching between CPU/IO queues (Figure 6). For instance, a connection request for a 2-stage application is first processed in the CPU queue, then in the IO queue, before being sent back to the client.

Two different processing models are used for CPU and IO queues, respectively. A FIFO model is defined for CPU queues, and connections that arrive when no CPU is available will be blocked in a backlog queue until there is an available CPU. IO is simulated as a simple processor sharing model, in which the instantaneous processing speed is the inverse of the number of connections in the IO queue. The backlog queue length of each server is configured as 64. Connections that arrive when the backlog queues are full will be rejected, with 40s timeout. Communication latency between 2 nodes is uniformly distributed between 0.1ms and 1ms.

C. Benchmark LB Algorithms

All the 6 LB algorithms described in Section I-B are implemented to be evaluated in the simulator. Similarly to SED, AWCMP is implemented to be aware of the server speed difference ratio, and with the server occupancies (*i.e.*, queue lengths) on each server polled periodically. The default update frequencies of server weights are the same for AWCMP and HLB (every 0.5s).

In the simulator, an *Oracle* LB algorithm is implemented, which distributes connections to the server which is expected to finish all its job with the lowest delay (including the new connection). The Oracle LB is aware of the remaining time of each connection, which is otherwise not observable for layer-4 LBs. By adding the Oracle LB, the load balancing performance of HLB and other LB algorithms can be compared to the potential upper bound of performance, corresponding to "perfect" network and server state observations.

For algorithms that consider instant server occupancies, power-of-2-choices can be applied additionally. Besides GSQ2, the simulator thus also implements LSQ2, SED2, HLB2, and Oracle2, to study the impact of partial observations and suboptimal load balancing decisions.

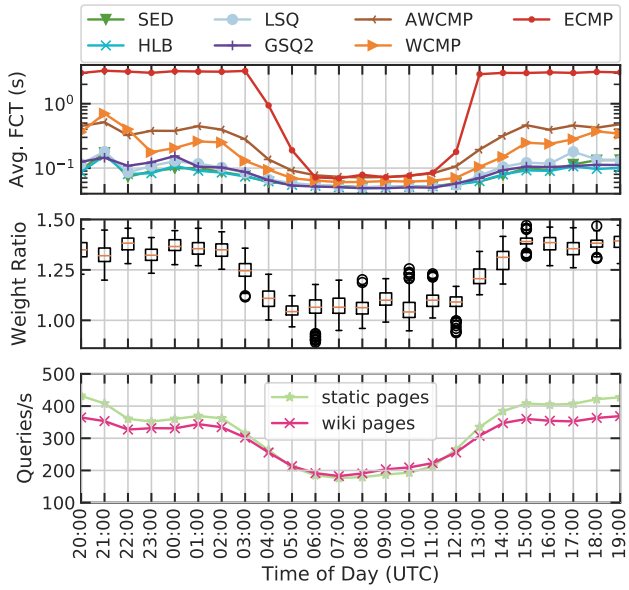


Figure 7. [Testbed] 24-hour Wikipedia trace replayed using different LB algorithms. Average FCTs (top), ratio between weights assigned to the 2 groups of servers by HLB (middle), and traffic rate (bottom) are depicted.

VI. Evaluation

Simulations and experiments are conducted (≥ 10 runs for each setup) to answer the following questions:

- How does the performance of HLB, when subjected to different traffic rates, compare with existing LB algorithms (Section VI-A);
- What is the impact of heterogeneity in server capacities on load balancing performances (Section VI-B);
- Are partially observed server occupancies representative of server load states (Section VI-C);
- How does HLB perform using different configurations of system parameters (Section VI-D);
- Can HLB adaptively react to dynamic networking environments (Section VI-E);
- What is the performance overhead (Section VI-F).

A. Performance with Different Traffic Rates

This section presents an overall performance evaluation of HLB, compared to other LB algorithms, when subjected to different traffic rates with both a real-world network trace replay and a large-scale simulation.

1) 24-Hour Trace Evaluation: Samples of 600s duration are extracted from the 24-hour Wikipedia trace and replayed on the testbed. The results are depicted in Figure 7.

During the off-peak period from `5:00` to `11:00` UTC when servers are under-utilised, all LB algorithms show similar performances. As traffic rates grow, HLB sees less increase in FCT compared with other LB algorithms, which is indicative of improved resource utilisation and performance gains achieved by the load balancing decisions using HLB.

LSQ and GSQ2 assume all servers have the same processing capacities, and aim at maintaining equal queue lengths on all servers. Under heavier traffic, servers with less processing capacities receive more workloads than they can process, and

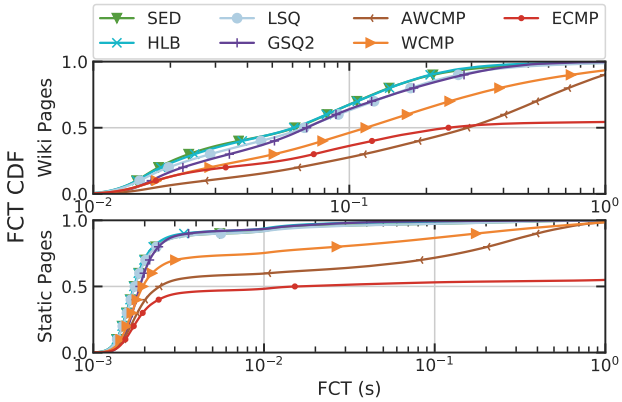


Figure 8. [Testbed] FCT CDF comparisons for two types of requests in the 24-hour Wikipedia replay.

their queues grow full. LSQ and GSQ2 thus become less performant than SED and HLB during peak period.

As depicted in the middle plot in Figure 7, HLB has no *a-priori* knowledge of server capacity differences (the ratio of CPU numbers between the 2 server groups is 2), yet it is able to passively learn these differences from observations, and achieve similar performance as SED. During off-peak hours, as servers have enough processing capacities, thus no additional queuing delay occurs, HLB does not differentiate server processing speeds. When subjected to heavier traffic rates, less powerful servers become "overloaded" and see higher queuing delays, which increase their corresponding flow durations. The increased flow durations thus inform HLB of the server processing speed differences.

Figure 8 depicts the FCT CDF of each LB algorithm for two types of requests: (i) static pages, and (ii) Wiki pages[9]. For both types of requests, HLB, SED, LSQ and GSQ2 show notable performance gain when compared with other LB algorithms. For Wiki pages, which are more computationally expensive (CPU-bound) to load than static pages, HLB achieves 23.66% and 26.43% less 90p FCT than LSQ and GSQ2 respectively. Of particular note is to mention, that HLB achieves the same performance as SED with no requirement of manual configurations of server weights. For static pages, which are IO-bound, HLB achieves 38.22% less 90p FCT than SED.

2) Workloads Distribution: To understand the workload distribution, this section studies 6 resource utilisation metrics: mean CPU usage, fairness, overprovision factor, mean number of busy threads, fairness of number of busy threads, and finally the average FCT. Given a random variable X, the fairness of X is defined as $F = \frac{\mathbb{E}(X)^2}{\mathbb{E}(X^2)} \in [0, 1]$ [50]. The overprovision factor of X is computed as the maximum load over the average load at each time step $\frac{\max(X)}{\overline{X}} \in [1, \infty)$ [4].

The performances of the 4 best performing LB algorithms in the 24-hour trace evaluation, are further analysed – still on a test platform with servers of different capacities. As depicted in Figure 9, SED achieves balanced average CPU usage between the 2 server groups, thus SED balances the average FCT, since it is aware of both server occupancies and

[9]Wiki pages are identifiable by the string `/wiki/index.php/` in URLs.

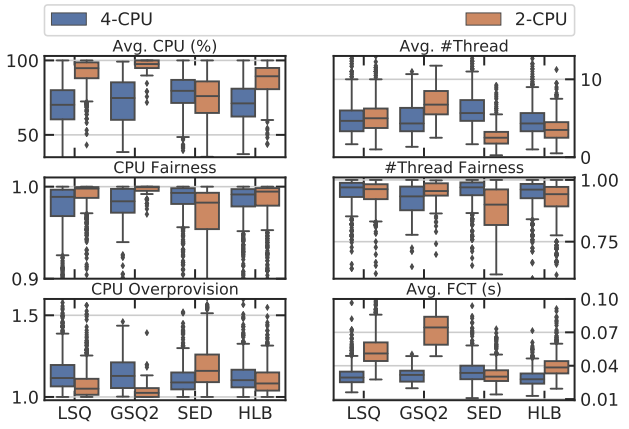


Figure 9. [Testbed] Comparison on server resource utilisations using network traces from hour 20:00 (800 queries/s) in the 24-hour Wikipedia replay.

processing speeds, thus assigns 2.3x connections to 4-CPU servers than to 2-CPU servers. Unlike SED, LSQ and GSQ2 balance queue lengths between the two server groups. They ignore the capacity differences, and overloads 2-CPU servers which experience FCTs increased by 71% and 131%, using LSQ and GSQ2 respectively, over FCTs on 4-CPU servers. HLB learns to give less aggressive weights than SED without any *a-priori* knowledge and assigns 35% more requests to 4-CPU servers than to 2-CPU ones. The queue lengths between the 2 groups of servers are less imbalanced than SED yet more proportional to their processing speeds than LSQ and GSQ2.

3) Large-Scale Simulation: To study LB performance in large-scale DC networks, simulations are conducted in a setup with 4 LBs and 128 servers, half of which has 1 CPU each, while the other half has 2.

The input traffic is a Poisson stream of single-stage CPU-bound application queries. The exponential distribution of FCTs, $T(q) \sim Exp(0.5)$, has an average of 500ms. Traffic rates are normalized with respect to the total provisioned resrouces. Results are obtained from multiple runs, each consisting of 80k network connection requests[10].

As depicted in Figure 10, the server occupancy is the dominant factor of LB performance when traffic rates are heavier, and LB algorithms that are occupancy-aware achieve better performance. Consistent with the testbed experiments in Section VI-A1, HLB yields a lower FCT than other LB algorithms – even though HLB has no *a-priori* knowledge about the server capacity difference. HLB achieves similar performance to the Oracle from moderate traffic rates up to 90% expected resource utilisation – when the average FCT becomes more than 5x higher than the expected FCT (200ms) – which covers most cases in DC networks [31]. Under 88.5% expected resource utilisation, HLB achieves 24.64% and 25.59% less 90p FCT than LSQ and SED respectively.

The **take-away** for these experiments and the subsequent simulations is that, even without manual *a-priori* configurations, HLB achieves better load balancing performance by learning server capacity differences, which allows fair distribution of workloads to servers. Tracking server occupancies further allows HLB to improve load balancing performance, when the servers are subjected to heavy traffic rates.

[10]There are 5 runs in total. From each run, only results from the interquartile range of the simulation time are used for analysis, to guarantee that all the metrics are collected under the Poisson stream of input traffic.

TABLE III
CONFIGURATIONS WITH DIFFERENT SERVER CAPACITY RATIOS.

Capacity Ratio n	1x	2x	4x
Testbed Group 1	5 × 2-CPU	4 × 2-CPU	2 × 2-CPU
Testbed Group 2	5 × 2-CPU	3 × 4-CPU	2 × 8-CPU
Simulator Group 1	64 × 1-CPU	64 × 1-CPU	64 × 1-CPU
Simulator Group 2	64 × 1-CPU	64 × 2-CPU	64 × 4-CPU

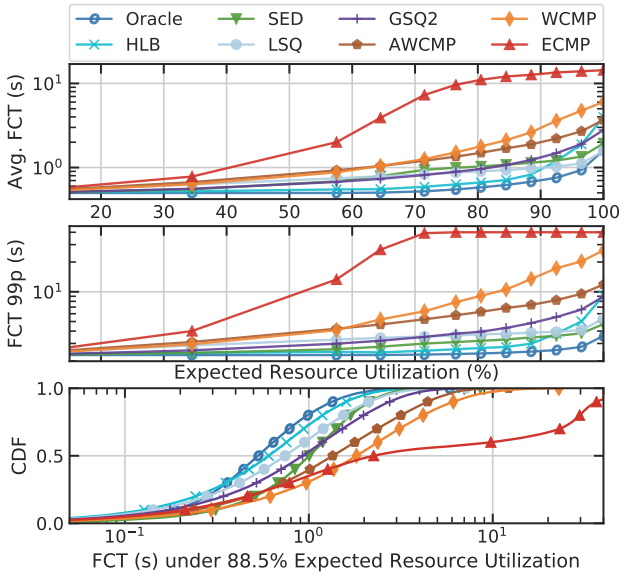


Figure 10. [Simulator] FCT comparison using 2x server capacity ratio under various traffic rates.

B. The Impact of Heterogeneity in Server Capacities

In view of the results from Section VI-A, this section will further explore the impact of heterogeneity in server capacity.

1) Testbed Experiments: 3 different configurations of server with a total of 20 CPUs, as per Table III, are tested with all the different LB algorithms.

As depicted in Figure 11, for larger differences between server processing capacities, SED and HLB outperform LSQ and GSQ2. The ratio of server weights computed by HLB between the two groups of servers[11] are lower than the ratio of provisioned resources. This is because the estimation of server processing capacities is based on flow durations, which capture not only the server processing time, but also the queuing delays, which are not proportional to server processing capacities. This causes HLB to "under-estimate" the server processing speed differences between the two groups of servers, yet HLB still achieves lower FCT than SED, since the traffic rate does not push the resource utilisation to the limit.

2) Large-Scale Simulation: Two Poisson streams of input traffic with $T(q) \sim Exp(0.5)$ are applied and consume respectively 70% and 90% provisioned resources on average.

[11]If not specified, the ratios used in this paper are calculated as the average value of the second group of servers over the average value of the first group of servers.

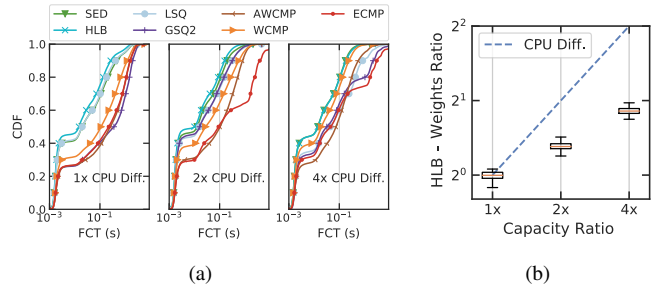


Figure 11. [Testbed] Comparison using different server capacity ratios using trace from hour `23:00` (680 queries/s). Figure (a) compares FCT CDF under 3 ratio configurations of CPU capacity differences using different LB algorithms. Figure (b) compares the server weights ratio between the two server groups generated by HLB with the actual provisioned server capacity ratios.

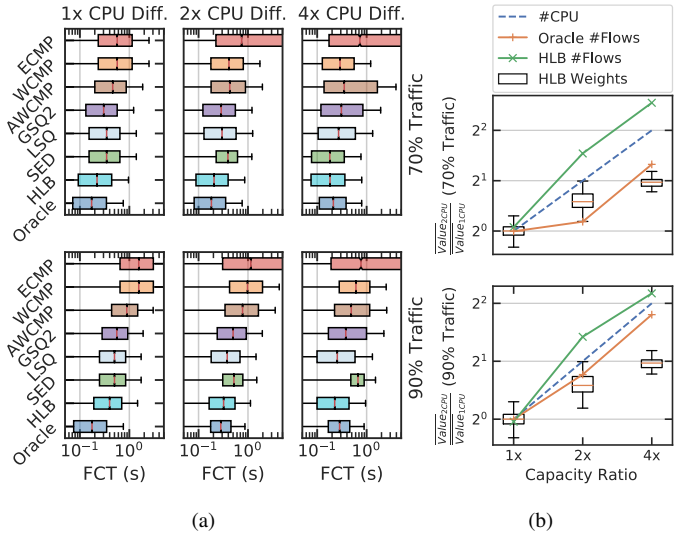


Figure 12. [Simulator] Comparison using different server capacity ratios under 70% (top) and 90% (bottom) expected resource utilisation. Figure (a) compares the FCT distribution and figure (b) compares the ratio of weights and load distribution between two groups of servers.

Three setups are configured as per Table III, with the results of the simulation depicted in Figure 12.

The results, when subjected to moderate traffic rates, show similar trends as the experiments in Section VI-A. As depicted in Figure 12a[12], when all servers have the same processing capacity, SED exhibits performance equivalent to LSQ.

With moderate traffic rate (70% expected resource utilisation), HLB and SED are the optimal LB algorithms, especially when the server capacity differences grow. With heavy traffic rate (90% expected resource utilisation), however, LSQ becomes better than SED. This is because the FCT is composed of the network delay, the queuing delay, and the server processing delay. At high resource utilisation, the queuing delay becomes dominant, whereas at low resource utilisation, the server processing time becomes significant.

Another observation that can be obtained from Figure 12a is the performance degradation of SED at high resource utilisation. This is because the partial observations on network traffic in presence of 4 LBs, make the server load state evaluation

[12]The boxplots used in this paper are standard boxplots, showing the interquartile ranges and the medians.

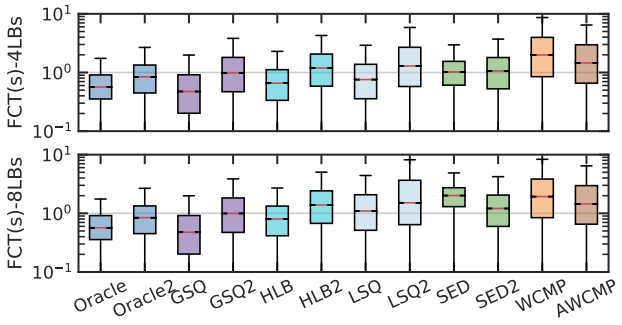


Figure 13. [Simulator] The impact of the application of power-of-2-choices on load balancing performance under 90% expected resource utilisation.

function of SED de-correlated from the actual server load state. For instance, when the provisioned resource difference ratio is $1:4$, SED assigns 12.14 times more workloads to powerful servers, causing them to be overloaded. This will be studied further in Section VI-C.

As depicted in Figure 12b, HLB achieves better performance in all the tested scenarios, by dynamically adjusting weights based on the inferred server states: the ratio of server weights calculated by HLB between the two server groups is correlated to, yet lower than, the actual ratio of provisioned resources. When comparing the number of distributed network flows, while HLB uses a different strategy than the Oracle, and prioritizes the servers with higher processing capacities, HLB is adaptive and achieves good performance under different scenarios.

The **take-away** from this set of experiments and simulations is, that when an LB algorithm considers server processing capacities when making decisions, it is important that this information is accurate – at least in as much as the provisioned resource difference ratio is accurate. This can be done either through *a-priori* configurations (as in SED), or through observing and learning (as in HLB). Likewise, as the simulations showed that the impact of the provisioned resources (number of CPUs) on a server on the FCT depends on the overall resource utilisation, it is important that the weight for a given server can be adaptive also to the traffic rate; especially when LBs have only partial observations on server load states.

C. The Representativeness of Partial Observations

As learned in Section VI-B2, that when there are multiple LBs, the performance of SED degrades. This section therefore studies the representativeness of partial observations of the server occupancies and suboptimality of power-of-2-choices.

1) Partial Observations on a Large-Scale: Following the Section VI-B2, this section uses the `2x` simulator configuration in Table III to study the impact of partial observations in a large-scale DC network. Two configurations with 4 and 8 LBs are applied to compare different degrees of partial observations. The FCT of the input traffic has the same distribution as in Section VI-B2, *i.e.*, $T(q) \sim Exp(0.5)$. In addition to the studied LB algorithms, this section also studies the power-of-2-choices variants of LB algorithms, that take the server occupancies into considerations: GSQ, HLB, LSQ,

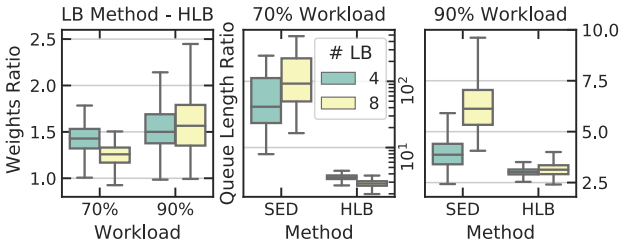
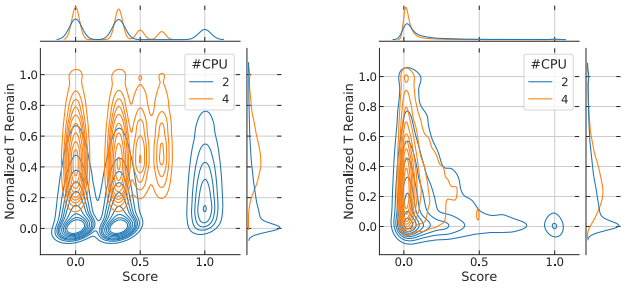


Figure 14. [Simulator] Different numbers of LBs give different levels of partial observations, which impact the weights ratio between the two groups of servers computed by HLB (left), and the ratio of queue lengths between the two groups of servers under different traffic rates (middle and right).



(a) SED score $\frac{\tilde{l}_i+1}{w_i}$. (b) HLB score $\frac{\tilde{l}_i+1}{\tilde{w}_i}$.

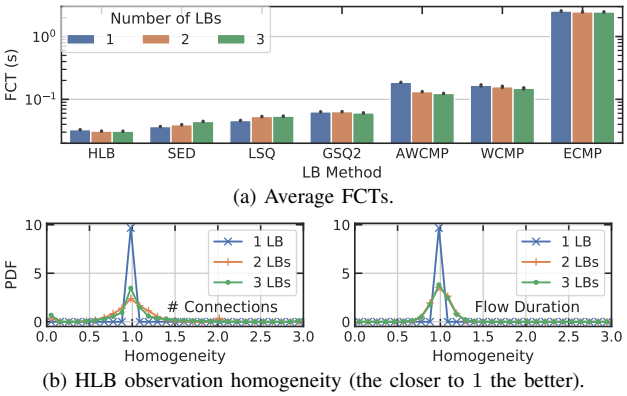
Figure 15. [Simulator] With 8 LBs, under a traffic rate that consumes 90% resource utilisation, the correlation between normalized residual processing time and computed server scores using SED and HLB.

SED, and the Oracle, since they may be potentially impacted by partial observations.

Figure 13 depicts the FCT distributions using different LB algorithms. As expected, the application of power-of-2-choices degrades the performance of GSQ, LSQ, and HLB for saving compute cycles. SED, however, shows the opposite results and achieves lower FCT with SED2 when there are 8 LBs.

2) Understanding Workloads Distribution for SED and HLB: Figure 14 depicts the queue length ratios between the two groups of servers when using SED and HLB, along with the server weights computed by HLB. SED prioritizes, and steers most workloads to servers with higher weights. With more LB nodes, the server occupancy observations become more partial and less representative of the actual server occupancies, and the estimations computed by SED are not correlated to the actual workloads on the servers (Figure 15a). Based on this incorrect estimation, SED assigns 74.2% network traffic to more powerful servers in presence of 8 LBs, leading to worse results than the randomness induced by the power-of-2-choices of SED2. The estimations of HLB, on the other hand, are more accurate and make the two groups of servers undertake similar workloads (Figure 15b).

3) Partial Observations on Experimental Testbed: To verify the observations obtained from Section VI-C1 and Section VI-C2, the 2x testbed configuration in Table III is used with various numbes of LBs. This section applies 600s Wikipedia trace with an average traffic rate of 680 queries per second. As depicted in Figure 16a, the performances of SED and LSQ degrade when the presence of more LBs make their observations on server occupancies more partial and less representative of the actual server occupancies. AWCMP



(a) Average FCTs.

(b) HLB observation homogeneity (the closer to 1 the better).

Figure 16. [Testbed] Comparison using different number of LB nodes.

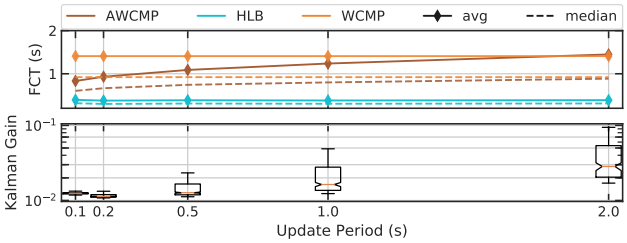


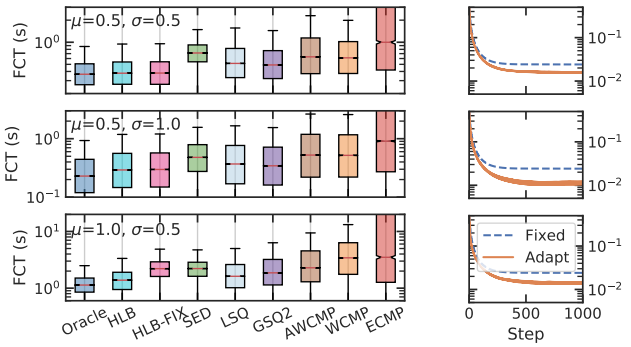
Figure 17. [Simulator] Comparison using different weights updating frequency under 90% expected resource utilisation.

obtains better performance since the presence of more LBs increases the polling frequency of the group of LBs thus improving observation granularity. Normalising statistically significant measurements across servers, HLB is less impacted by the factor of partial observations among all LB algorithms. In a setup with M LBs passively observing N servers, denote $z_{ij} = \frac{x_{ij}}{\sum_j x_{ij}}$ where x_{ij} is the measurement of server j made by LB i at a given time step. The distribution of $M\frac{z_{ij}}{\sum_i z_{ij}}$ is compared in Figure 16b to measure the HLB observation homogeneity across LBs *w.r.t.* global measurement distribution. The homogeneity of number of ongoing connections is less centered and has increased outliers with more LBs, yet the one of flow durations is less sensitive to the growth of LB node numbers, which helps HLB gainfully use observed information for server load ranking.

The **take-away** from these experiments and simulations is that, more partial observations can be less representative of the measured system. When having only partial observations on server occupancies available, such as is the case for a multi-LB set-up, the superiority of combined metrics when using HLB emerges. Using Kalman filters, HLB accumulates reliable observations on server processing speeds over time in the history, and predicts the server processing speeds at the next time-step. Integrating both server occupancy and processing speed, HLB is less sensitive to partial observations.

D. Sensitivity Analysis

This section studies the potential impacts on LB performances of different conditions and system parameters, namely, (i) weights update frequency, (ii) flow table size, and (iii) RTT between clients and servers.



(a) Measured FCT Distribution (s). (b) Kalman Gain.

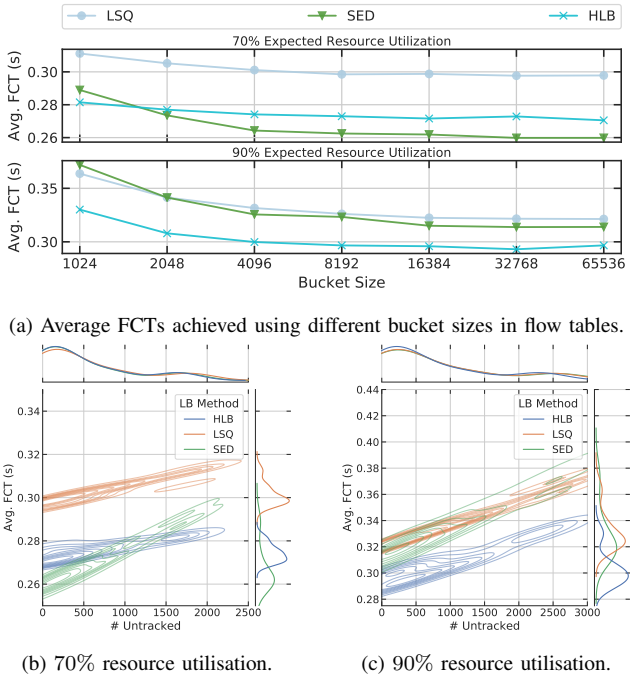
Figure 18. [Simulator] Different input traffic FCT distributions under 90% expected resource utilisation.

1) Weights Update Frequency: AWCMP and HLB periodically update server weights. The intervals between two consecutive updates is a system parameter. In this section, 5 periods are applied to study their impacts. Figure 17 shows that higher weight updating periods degrade AWCMP performance. Since AWCMP uses the same set of weights during a complete time interval, the correlation between server weights and server load states decreases, until the next update. HLB on the other hand, is less affected by the update interval because of its adaptive Kalman gain. When the update interval is short, HLB generates higher Kalman gains, and assigns more weights to the newly computed load state estimations to catch up with the dynamics of the environment.

The adaptive Kalman filter also allows adapting server weights accordingly when facing input traffic with different FCT distributions. Three lognormal distributions of FCTs are applied on the 2× simulator configuration as in Table III with 4 LBs. As depicted in Figure 18, when facing input traffic with different FCT distributions, HLB is able to achieve performance close to the Oracle, without manual configuration or additional control messages. The measurement noise R of HLB-FIX is configured as 0.5 while HLB has no hard-coded R. As depicted in Figure 18b, HLB has different convergence of Kalman gain corresponding to different FCT distributions. It helps HLB achieve better performance than HLB-FIX when the average FCT of input traffic is higher.

2) Pitfalls of Statefulness: LSQ, SED, and HLB track connection states in flow tables, and the number of buckets in flow tables is another system parameter. Flow tables with more buckets can store more connection states, and can therefore enable higher observation accuracy. Untracked connections will be statelessly forwarded to a random server by looking up ECMP bucket tables, without considering server load states. However, managing large flow tables consumes more memory space, which is costly on dedicated hardware [12], [27].

To investigate performance degradation when memory space is limited, simulations are conducted on a DC network with 4 LBs and 256 servers. Half of the servers have 2 CPUs while the other half have 4 CPUs. Reducing bucket sizes from 65536 to 1024 leads to more untracked flows, and thus degraded load balancing performance. Figure 19 shows that HLB is more robust to traffic rate changes and less sensitive to the flow



(a) Average FCTs achieved using different bucket sizes in flow tables.

(b) 70% resource utilisation. (c) 90% resource utilisation.

Figure 19. [Simulator] Comparison using different flow table bucket size.

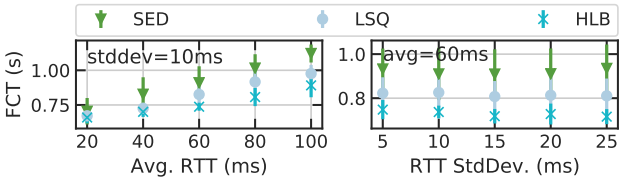


Figure 20. [Testbed] FCT (avg. ± stddev) comparison under different RTT distributions between clients and servers.

table size than is LSQ and SED. LSQ and SED use only the observations of server occupancies, while HLB can infer server processing speeds based on measured flow durations, and thus it is less impacted by untracked flows. HLB achieves the best performance with the minimal bucket size, which makes it an interesting candidate for hardware implementations.

3) RTT Between Clients and Servers: Evaluations above have demonstrated that HLB can improve load balancing performance for intra-DC services, where clients locate within the same DC network. To understand whether HLB can benefit the use cases where clients connect to servers through the Internet, this section studies the impact of different distributions of RTT between clients and servers. Intuitively, given a request from a client, the load balancing decision is not biased by the RTT between this client and the server cluster. HLB makes load balancing decisions and assigns servers based on its estimations of server load states, which depends on the distribution of sampled flow durations. The flow duration measurements consist of server processing time and RTT between clients and servers. Since requests with different RTTs are indiscriminately distributed across servers yet server processing time varies depending on instant server load states, HLB normalises flow durations across servers and reserves the variance of server processing speeds. Therefore HLB is not sensitive to different RTT distributions.

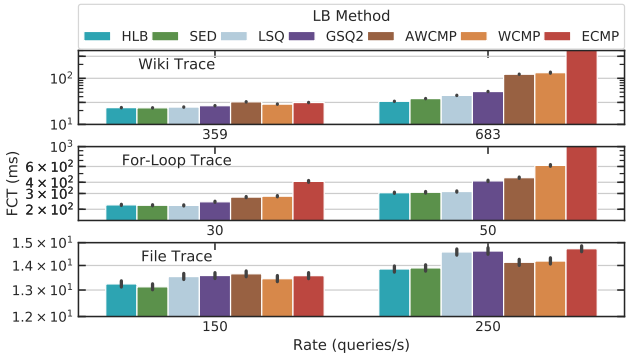
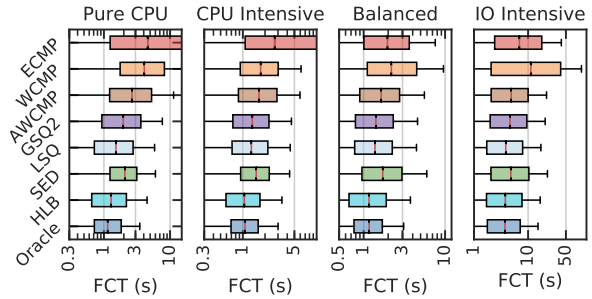
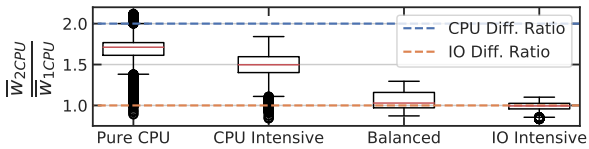


Figure 21. [Testbed] Comparison with different types of network applications.



(a) FCT distribution for different types of applications



(b) Server weight ratio between two server groups assigned by HLB.

Figure 22. [Simulator] Simulation results with 3-stage application queries under 90% expected resource utilisation.

To provide an empirical study, using the same setup as in Secion VI-A1, the RTT between clients and the edge router is shaped using `netem` to follow Pareto normal distributions with different means and standard deviations [51]. Added delays have 25% dependency on their previous values. As depicted in Figure 20, the FCT grows linearly with the increase of the mean RTT. With the combination of reservoir sampling and Kalman filters, HLB removes the unimodal RTT distribution so that the processed flow durations still reflect server processing speed differences. In all scenarios, HLB remains superior to LSQ and SED. This shows that HLB is not sensitive to the change of RTT between clients and servers.

The **take-away** in this section is, that HLB is less sensitive to server weights updating frequency than are active LB algorithms. It also requires less memory space than other stateful LB algorithms, which makes it more hardware-friendly. HLB is not sensitive to RTT between clients and servers thus it can potentially benefit more than just intra-DC applications.

E. Response to Heterogeneous and Dynamic Environments

This section studies the adaptability of HLB when facing heterogeneous traffic and dynamic DC setups.

1) Adaption to Different Types of Input Traffic: This section studies the adaptivity of HLB for different types of network applications. Besides Wikipedia trace, another two types of poisson traffic are applied. A PHP `for`-loop script that runs for a given number of iterations, simulates CPU-bound applications with $T(q) \sim Exp(0.2)$. To simulate IO-bound applications, a farm of static files with different sizes[13] are created and queried. Moderate and high traffic rates are applied for the 3 types of applications in testbed as in Figure 5. As depicted in Figure 21, LSQ achieves similar performance as SED and HLB for `for`-loop trace but does not perform better than ECMP for the file trace. AWCMP achieves lower FCT under CPU-bound traffic, especially when traffic rate is high. SED and HLB have the best performance for all traces.

TABLE IV
FOUR CONFIGURATIONS WITH DIFFERENT APPLICATION TYPES.

Application Type	Pure CPU	CPU Intensive	Balanced	IO Intensive
Avg. CPU Time (s)	1.	0.75	0.5	0.25
Avg. IO Time (s)	0.	0.25	0.5	0.75

Though flow duration is affected by many factors including expected workloads, instant server occupancy and different types of provisioned resources (*e.g.* CPU, IO, networking conditions), by collecting multiple samples (128 per server) of flow durations for the same VIP (and thus for the same application), we obtain a statistical representation of the flow duration distribution on each server. This allows to derive and compare the overall server processing speed for the given application. Using flow duration as an indicator of server load states saves us from profiling different applications (*e.g.* resource dependencies) and allows to generalize to different types of applications.

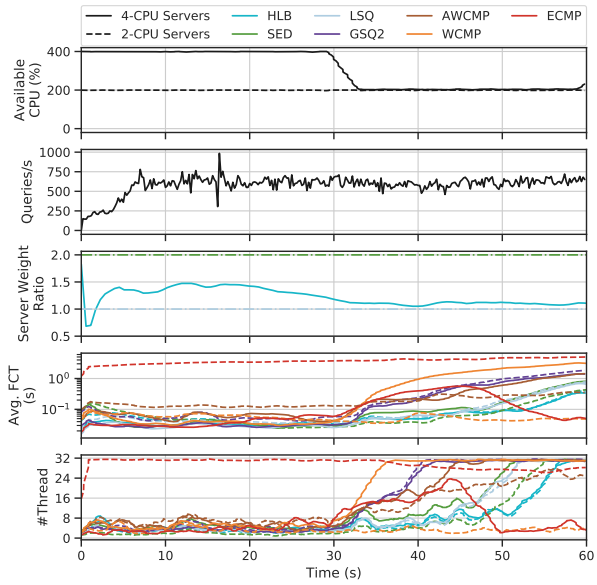
On a larger scale, simulations are conducted with 4 LBs and 128 servers using the `2x` configuration as in Table III. In this section, a 3-stage application whose queries follow CPU-IO-CPU processing stages is compared with a pure CPU application. Both CPU and IO processing time follow exponential distributions and the aggregated average FCT is 1s. The four different types of network applications are configured as in Table IV. As depicted in Figure 22, with different provisioned resource ratios for CPU (`2x`) and IO (`1x`) queues, HLB has better performance for all types of applications, with weights adaptive to the requirements of different types of applications.

2) Adaption to Processing Speed Changes: This section shows the ability of HLB to detect changes in server processing speeds, *e.g.*, when VMs are migrated to a new server. Using the `2x` testbed configuration with 2 LBs, additional CPU-bound workloads are applied on the 4-CPU server group starting from 30s. As depicted in Figure 23, under heavy Wikipedia traffic, HLB adapts server weights over time and achieves better performance than other LB algorithms. It is able to infer that the processing speeds of the two groups of servers become similar to each other after 30s.

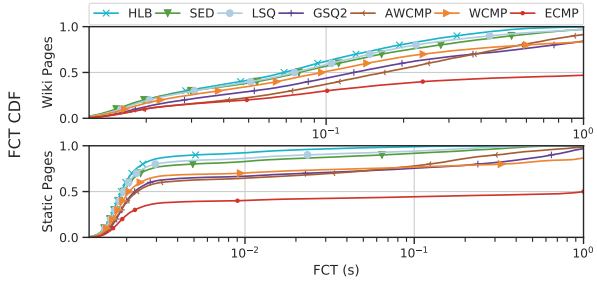
F. Overhead Analysis

To compare the additional processing latency of HLB, one 4-CPU LB and a 176-CPU server cluster is deployed on 4 physical machines. The number of CPU cycles per packet and

[13]The sizes of files are 100KB, 200KB, 500KB, 750KB, 1MB, 2MB, and 5MB. 50 files are randomly generated for each size.

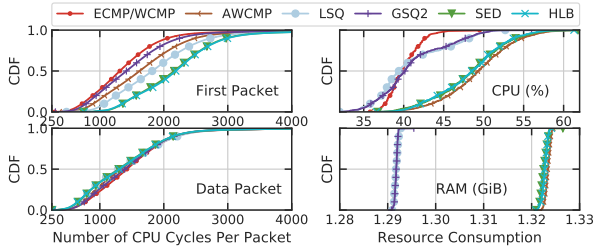


(a) Additional workloads are applied on servers with 4 CPUs after 30s.

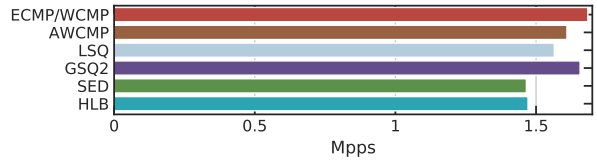


(b) FCT CDF comparisons for two types of requests.

Figure 23. [Testbed] HLB is able to adapt to changed environments without manual configurations or additional control messages.



(a) Processing latency and resource consumption comparison.



(b) Throughput comparison.

Figure 24. [Testbed] Overhead analysis.

resource consumption are compared in Figure 24a under 10 runs of 2000 queries/s of Poisson traffic (more than 1150.76 average concurrent connections). The first packets are those that register new-coming connections in the flow table while the data packets are the subsequent packets that are matched in the flow table. As HLB calculates and compares the score of each server when assigning servers to connections, it consumes on average 871 cycles (0.34μs on 2.6GHz CPU) more than does ECMP for each connection. Compared with ECMP, HLB incurs on average 8% additional CPU usage and 31MiB additional RAM usage. Assuming LBs see one `SYN` packet, one data packet and one `FIN` packet in each connection, the average packet throughput for each LB algorithm on a 2.6GHz CPU are compared as depicted in Figure 24b. HLB achieves 87.38% throughput of ECMP.

VII. Conclusion

This paper has proposed, and studied the performance of HLB – a load-aware Layer-4 LB. Based on passively gathered networking observations extracted from the data plane, HLB is able to estimate both server occupancies and processing speeds, which are identified in this paper as two key factors in load balancing performance, with no *a-priori* knowledge or manual configurations. HLB can be deployed without modifications on the target network, since it requires no additional management traffic or active signaling. Evaluated in both simulations and testbed experiments, HLB offers better load balancing performance than existing LB algorithms. It is also able to adapt to dynamic DC environments and variant workloads.

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[2] 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.

[3] 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.

[4] 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.

[5] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "Srlb: The power of choices in load balancing with segment routing," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 2011–2016.

[6] A. Aghdai, C.-Y. Chu, Y. Xu, D. Dai, J. Xu, and J. Chao, "Spotlight: Scalable transport layer load balancing for data center networks," *IEEE Transactions on Cloud Computing*, 2020.

[7] T. Barbette, C. Tang, H. Yao, D. Kostić, G. Q. Maguire 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)*, 2020, pp. 667–683.

[8] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," Cloud Workshops at OOPSLA09, [Online] Available: https://corescholar.libraries.wright.edu/knoesis/78, 2009.

[9] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM Sigcomm computer communication review*, vol. 39, no. 1, pp. 50–55, 2008.

[10] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 51–66.

[11] 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.

[12] 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*, 2017, pp. 15–28.

[13] Facebook, "Katran," [Online] Available: https://github.com/facebookincubator/katran, 2018.

[14] S. Shi, Y. Yu, M. Xie, X. Li, X. Li, Y. Zhang, and C. Qian, "Concury: A fast and light-weight software cloud load balancer," p. 14, 2020.

[15] V. Olteanu and C. Raiciu, "Datacenter scale load balancing for multipath transport," in *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, ser. HotMIddlebox '16. ACM, 2016, p. 20–25, event-place: Florianopolis, Brazil.

[16] J. T. Araújo, L. Saino, L. Buytenhek, and R. Landa, "Balancing on the edge: Transport affinity without network state," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 111–124.

[17] GitHub, "GLB Director," [Online] Available: https://github.com/github/glb-director, 2020.

[18] 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.

[19] R. Cohen, M. Kadosh, A. Lo, and Q. Sayah, "Lb scalability: Achieving the right balance between being stateful and stateless," *IEEE/ACM Transactions on Networking*, 2021.

[20] ——, "Hardware syn attack protection for high performance load balancers," in *2021 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2021, pp. 9–16.

[21] W. Zhang *et al.*, "Linux virtual server for scalable network services," in *Ottawa Linux Symposium*, vol. 2000, 2000.

[22] A. Aghdai, M. I.-C. Wang, Y. Xu, C. H.-P. Wenz, and H. J. Chao, "In-network congestion-aware load balancing at transport layer," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2019, pp. 1–6.

[23] 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.

[24] 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.

[25] G. Goren, S. Vargaftik, and Y. Moses, "Distributed dispatching in the parallel server model," *arXiv:2008.00793 [cs]*, Aug 2020, arXiv: 2008.00793.

[26] 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.

[27] 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)*. IEEE, 2018, pp. 418–423.

[28] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, 2015, p. 123–137, event-place: London, United Kingdom.

[29] W. Wang and G. Casale, "Evaluating weighted round robin load balancing for cloud web services," in *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, 2014, pp. 393–400.

[30] R. Gandhi, Y. C. Hu, and M. Zhang, "Yoda: a highly available layer-7 load balancer," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 21.

[31] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, and L. Mao, "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," p. 10, 2019.

[32] A. Kumar, I. Narayanan, T. Zhu, and A. Sivasubramaniam, "The fast and the frugal: Tail latency aware provisioning for coping with load variations," in *Proceedings of The Web Conference 2020*, 2020, pp. 314–326.

[33] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 5.

[34] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 6, pp. 3670–3682, 2017.

[35] 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.

[36] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM journal on computing*, vol. 29, no. 1, pp. 180–200, 1999.

[37] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[38] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[39] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *ACM CoNEXT '10*, Philadelphia, PA, December 2010.

[40] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Elsevier Computer Networks*, vol. 53, no. 11, pp. 1830–1845, July 2009.

[41] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.

[42] Facebook Engineering, "Reinventing Facebook's data center network," Mar 2019.

[43] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: modeling techniques and their applications," *Journal of Internet Services and Applications*, vol. 5, no. 1, pp. 1–17, 2014.

[44] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 191–205.

[45] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[46] The Fast Data Project (fd.io), "Vector Packet Processing (VPP)," [Online] Available: https://wiki.fd.io/view/VPP, 2017.

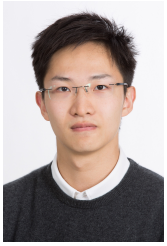
[47] E.-J. van Baaren, "Wikibench: A distributed, wikipedia based web application benchmark," *Master's thesis, Dept. Comput. Sci., VU University Amsterdam, Amsterdam, The Netherlands*, 2009.

[48] A. Hadoop, "Apache hadoop," *URL http://hadoop. apache. org*, 2011.

[49] A. Spark, "Apache spark," *Retrieved January*, vol. 17, p. 2018, 2018.

[50] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, 1984.

[51] X. Chen, H. Kim, J. M. Aman, W. Chang, M. Lee, and J. Rexford, "Measuring tcp round-trip time in the data plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, 2020, pp. 35–41.



Zhiyuan Yao received the M.Sc.T from École Polytechnique, Palaiseau, France, in 2019. He is currently pursuing an industrial Ph.D. jointly between École Polytechnique's networking team and Cisco Systems PIRL, under supervision of Mark Townsley (Cisco Systems) and Thomas Clausen (École Polytechnique). His research interests include high-performance networking, load-balancing, data-center optimization algorithms, and machine learning for systems.

Yoann Desmouceaux received the Diplôme d'Ingénieur from École Polytechnique, Palaiseau, France, in 2014, the M.Sc. degree in Advanced Computing from Imperial College, London, U.K., in 2015, and the Ph.D. degree in computer networking from Université Paris-Saclay, France, in 2019. He is currently working as a Software Engineer with Cisco Systems. His research interests include high-performance networking, IPv6-centric protocols, load-balancing, reliable multicast, and data-center optimization algorithms.

Juan-Antonio Cordero-Fuertes is an associate professor at École polytechnique. He graduated in Mathematics ("Licenciatura", M.Sc) and Telecommunication Engineering (B.Sc+M.Sc, "Ingeniería Superior") at the Technical University of Catalonia (UPC, Spain) in 2006 and 2007, respectively. He got his Ph.D. at École polytechnique in 2011, with a dissertation on the optimization of link-state routing protocols for operation in MANETs and compound (wired/wireless) Autonomous Systems. He was a postdoctoral researcher at the Université catholique de Louvain (UCL, Belgium) and the Hong Kong Polytechnic University (Hong Kong SAR, PRC), before joining faculty at École polytechnique, in 2016. His research and scientific interests include routing protocols and information dissemination algorithms, and the modeling, analysis and optimization of distributed, adaptive systems in dynamic, heterogeneous networking scenarios.

Mark Townsley is a Cisco Fellow, Professor Chargé de Cours at École Polytechnique, and co-founder of the Paris Innovation and Research Laboratory (PIRL). Before Joining Cisco in 1997, he held positions at IBM, the Institute for Systems Research (ISR) and the Center for Satellite and Hybrid Communications Networks (CSHCN) at the University of Maryland. Mark served as IETF Internet Area Director from 2005-2009, IETF L2TP Working Group Chair from 1999-2005, IESG Liaison to the Internet Architecture Board (IAB), and IETF Pseudowire WG Technical Advisor. Mark was the lead developer of the original implementation of L2TP in Cisco IOS as well as lead author of IETF L2TP protocol specification (RFC 2661). One of the original architects of the World IPv6 Day and Launch, Mark contributed significantly to the deployment of IPv6 on the internet, including lead author of RFC 5969, IPv6 Rapid Deployment (6RD). In 2011, Mark co-founded the IETF Homenet Working Group, and served as chair until 2017. In addition to his Faculty appointment at École Polytechnique, Mark lectures on Future Internet Architectures at Telecom Paris Tech (TPT), and serves on the steering committee for the joint TPT-Polytechnique Advanced Computer Networking master's degree. Mark holds a Bachelor of Science (summa cum laude) degree in Electrical Engineering from Auburn University and a Master's degree in Computer Science (magna cum laude) from the Johns Hopkins University Applied Physics Laboratory.

Thomas Clausen is a graduate of Aalborg University, Denmark (M.Sc., PhD – civilingeniør, cand.polyt), and has, since 2004 been on faculty at Ecole Polytechnique, France's premiere technical and scientific university, where as a professor, he holds the Cisco endowed "Internet of Everything" academic chaire. At Ecole Polytechnique, Thomas leads the computer networking research group. He has developed, and coordinates, the computer networking curriculum, and coordinates the M.Sc.T programme "IoT: Innovation and Management". He has published more than 100 peer-reviewed academic publications, and has authored and edited 24 IETF Standards. Thomas has also consulted for the development of IEEE 802.11s, and has contributed the routing portions of the ITU-T G.9903 standard for G3-PLC networks – upon which, e.g., the current SmartGrid & ConnectedEnergy initiatives are built. Thomas is a senior member of the IEEE, and was named an "IEEE Computer Society Distinguished Contributor", as part of the 2021 inaugural class.