Efficient Data-Driven Network Functions

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

***Abstract*—Cloud environments require dynamic and adaptive networking policies. It is preferred to use heuristics over advanced learning algorithms in Virtual Network Functions (VNFs) in production becuase of high-performance constraints. This paper proposes Aquarius to passively yet efficiently gather observations and enable the use of machine learning to collect, infer, and supply accurate networking state information – without incurring additional signalling and management overhead. This paper illustrates the use of Aquarius with a traffic classifier, an auto-scaling system, and a load balancer – and demonstrates the use of three different machine learning paradigms – unsupervised, supervised, and reinforcement learning, within Aquarius, for inferring network state. Testbed evaluations show that Aquarius increases network state visibility and brings notable performance gains with low overhead.**

***Index Terms*—Virtual Network Functions, high performance network, data-driven, cloud, performance evaluation**

I. Introduction

To increase network programmability, and balance the trade-off between capital expenditures and quality of service (QoS), Virtual Network Functions (VNFs) (*e.g.*, firewalls, load balancers) replace or augment dedicated hardware devices and play a significant role in large-scale data centers (DCs), running on commodity computing platforms. To dynamically monitor and configure VNFs, the routing and decision-making process (*control plane*) is dissociated from the network packets forwarding process (*data plane*).

Data-driven mechanisms based on machine learning (ML) and reinforcement learning (RL) algorithms are applied in the control plane to adaptively manage networking policies [1]–[3]. For instance, auto-scaling systems and load balancers can achieve improved QoS with reduced cost based on periodically polled resource utilisation of distributed nodes [4]. However, it is challenging to apply these algorithms in networking systems in real-time, since they require fine-grained observations of network and system states [5].

Reactive polling resource utilisation and system performance incur additional control messages [4], [6]–[8] and reduce system scalability. Fine-grained networking features are extracted offline or in simulated environments for clustering and RL algorithm development [5], [9]. Since the data plane is constrained by low-latency and high-throughput requirements [10], heuristics–which may not be adaptive to dynamic environments–prevail over advanced learning alrogithms in real-world high performance networks [7], [8], [11]–[14].

This paper proposes Aquarius, a fast and scalable data collection and exploitation mechanism that bridges different requirements for data planes (low-latency and high-throughput) and control planes (making informed decisions). Extensive performance and overhead evaluations of Aquarius in a realistic testbed show that Aquarius enable:

- **unsupervised learning + offline data analysis**: creating benchmark datasets to gain insight in different networking problems with minimal data collection overhead;
- **supervised learning + VNF management**: embedding ML techniques to achieve self-aware monitoring and self-adaptive orchestration in an elastic compute cloud;
- **reinforcement learning + online policy updates**: enabling closed-loop control to optimise routing policies and improve QoS, with no human intervention.

II. Background

This section presents the challenges of effective feature collection and data-driven VNFs in cloud DCs, and, with a comparison of related work, motivates the design of Aquarius.

A. Challenges

There is a rising trend of embedding intelligence and applying ML techniques in cloud and distributed systems to dynamically monitor and adaptively configure system parameters and characteristics (*e.g.*, server configurations, forwarding rules) [5], [8], [23], [27]. However, this raises a number of challenges and trade-offs:

Online and Reliable Feature Collection: Few reliable datasets are available and considered as benchmark for ML applications in VNFs and networking systems (*e.g.*, traffic analysis and anomaly detection) [28]–[31]. Though log-based feature collection provides abundant information for various types of applications, it incurs high overhead under heavy traffic, which leads to inaccurate and unreliable measurements and makes it hard to bring ML algorithms "online" (making inference in real time) [27].

Scalability vs. Visibility: Active probing is another way of feature collection for VNFs to monitor the system state and make informed decisions [6], [15], [16], [24]. However, this incurs additional communication overhead and requires modifications on each node to maintain management and communication channels.

B. Requirements

Based on the challenges, this paper summarizes the following requirements to enable data-driven VNFs in the cloud:

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.

This work is, in part, supported by the Cisco endowed "Internet Technologies and Engineering" chaire at École Polytechnique.

TABLE I: Comparison of data-driven VNF systems.

Property/Work	[15], [16]	[17]–[19]	[20]	[4], [21]	[1], [5]–[7]	[2], [22]	[23]	[24]	[25]	[26]	*Aquarius*
No Control Message	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓
Distributed	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
Commodity Device	✓	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓
Functionality	Framework	Management Protocol		Autoscaling	Traffic Optimisation		Traffic Classification			Framework	Framework

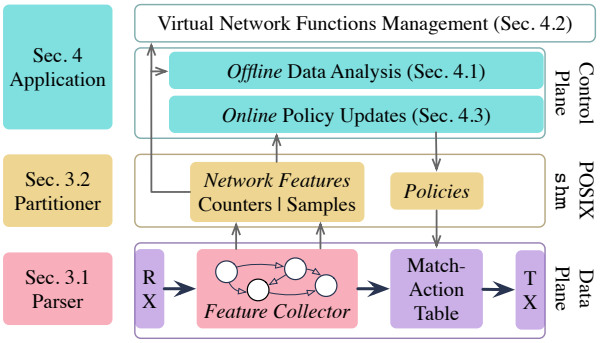


Fig. 1: Aquarius architecture overview.

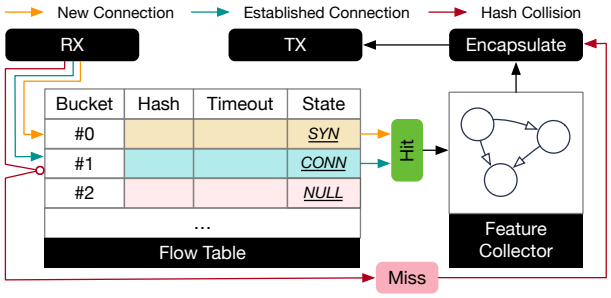


Fig. 2: Flow table data structure and workflow.

Universality: the feature collection mechanism should cover a wide range of features and be application-agnostic;

Reliability: the collected features should be representative, and have high usability and granularity;

Scalability: the feature collection and exploitation mechanism should incur minimal performance overhead and support large-scale and dynamically changing network topology;

Deployability: the mechanism should be plug-and-play and require no additional installation or configuration.

C. Related Work

Various mechanisms (summarised in Table I) dynamically configure and manage VNFs, making data-driven decisions.

ML allows inferring system states from networking features, for, *e.g.*, intrusion detection systems [23], traffic classification [32], [33], and task scheduling [1], [2]. To obtain networking features, these ML applications operate at the Application Layer. However, they are not application-agnostic and do not generalise to different use cases. Aquarius collects a wide range of features at the Transport Layer and enables generic data-driven network functions with minimal overhead.

Management and Orchestration (MANO) frameworks use centralised controllers to monitor and update VNF configurations [15], [16]. Software-Defined Network (SDN) provides programmable APIs to gather per-flow or application-level features in a centralised way, to adaptively update configurations, using network equipments that supports the OpenFlow protocol [19]. Smart Network Interface Cards (sNICs) [17] and Nitro [18] offload VNFs from host processors to dedicated hardware devices to boost performance and reduce operational cost with centralised management. Aquarius passively extracts networking features from the data plane and let VNFs make decisions in a distributed way.

Distributed VNFs also benefit from periodically polled network states (*e.g.*, CPU and memory usage), to ensure service availability, improve QoS [6], or classify networking traffic [24]. Some network functions gain more visibility via in-network telemetry (INT) [22]. However, these methods require to either deploy agents, or to modify protocol stack on network nodes, which reduce the deployability. Aquarius employs the plug-and-play design and requires no coordinated modification in the network.

Learning algorithms incurs additional inference and processing latencies. To reduce latency, dedicated hardware, *e.g.*, CGRA [26], and NetFPGA [25], helps accelerate data processing for in-network ML applications. Yet they lack flexibility when developing ML algorithms for different use cases in elastic networking systems. MVFST-RL [5] proposes–in simulators–to asynchronously update networking configurations from learning algorithms to reduce additional latency in the data plane. Aquarius can incorperate intelligence in a variety of VNFs, requiring no dedicated device, yet it is ready to be deployed in real-world systems.

III. Design

To meet the 4 requirements summarised in Section II-B, Aquarius is designed as a 3-layer architecture (Fig. 1). Aquarius embeds a feature collector at the Transport Layer in the data plane, to efficiently and passively extracts a wide range of features with high granularity and low performance overhead. It makes the features available via shared memory, for applications of ML algorithms on various use cases. This paper illustrates the design using TCP traffic, which prevails in the cloud of Content Delivery Networks (CDNs) [34].

A. Parser Layer

To balance the tradeoff between scalability and visibility, networking features which indicate system states can be passively collected from the data plane to avoid active probing and additional installations and configurations.

1) Stateful Feature Collection: Network traffic consists of flows that traverse different nodes (*e.g.*, edge routers, load balancers, servers) in the system, whose states can be traced and retrieved from the flows–along with traffic characteristics.

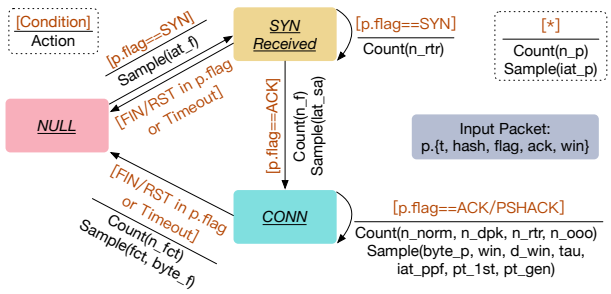


Fig. 3: A state machine of feature collector for TCP traffic.

Rationale: Stateless feature collection mechanisms (*e.g.*, sketches [9]) do not track the state of network flows, yet they can gather counters as ordinal features for ML algorithms using hashing functions, with little performance overhead. However, ordinal features contains less information than quantitative features–time-related features (*e.g.*, round-trip time, inter-arrival time, flow duration) and throughput information (*e.g.*, congestion window size, flow size), which are not captured by stateless mechanisms.

Design: Aquarius tracks flow states in bucket entries with a stateful table (Fig. 2), which can be configured to collect a wide range of features using a state machine depicted in Fig. 3. In the flow table, Aquarius stores the information of each flow into a bucket entry indexed by $hash(fid)\%M$, where fid is the flow ID (*e.g.*, the 5-tuple of TCP flows) and M is the flow table size. An entry in the flow table can be in one of three states–`SYN`, `CONN` and `NULL` (Fig. 3). The flow state transitions trigger the networking feature updates, which are described in the next subsection. In case where the bucket entry is not available when a new flow arrives, the flow is considered as a "miss" and is excluded by the feature collector.

2) Network Features: Various features can gainfully benefit decision making process for different use cases.

Rationale: As a generic feature collection mechanism, Aquarius should be able to collect as much information as possible with minimal overhead (*e.g.*, memory space consumption). Other than ordinal and quantitative features, to capture the system dynamic, it is also important to trace the timestamps of different events, for sequential ML algorithms.

Design: With the flow table, Aquarius allows flexible configuration of attributes, to gather the most significant features and optimise the memory usage overhead for different applications. Quantitative features are collected as samples, using reservoir sampling (Algorithm 1). Since networking environments are dynamic, it is important to capture not only the features, but also the sequential information of the system. Reservoir sampling gathers a representative group of samples in fix-sized buffer from a stream. It captures both the sampling timestamps and exponentially-distributed numbers of samples over a time window, which help analyse patterns sequentially.

B. Partitioner Layer

Cloud services have different characteristics and they are identified by virtual IPs (VIPs) (Fig. 4), corresponding to clusters of provisioned resources–*e.g.*, servers, identified by unique direct IPs (DIPs). In production, cloud DCs are subject to high traffic rates and their environments and topologies change dynamically.

Algorithm 1 Reservoir sampling with no rejection

1: $k \leftarrow$ reservoir buffer size
2: $buf \leftarrow [(0,0),\ldots,(0,0)]$ ▷ Size of k
3: **for** each observed sample v arriving at t **do**
4: $randomId \leftarrow rand()$
5: $idx \leftarrow randomId\%N$ ▷ randomly select one index
6: $buf[idx] \leftarrow (t, v)$ ▷ register sample in buffer

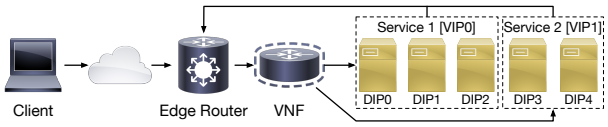


Fig. 4: Cloud service topology.

Rationale: Different cloud services should be separated to (i) avoid multimodal distributions in collected features and to (ii) allow dynamically adding or removing services. Features should be made available so that both spatial and sequential information can be easily partitioned and accessed. Even with heavy traffic and dynamically changing network topology, features should be reliable and easy to access with low latency.

Design: Aquarius organises observations of each VIP in independent POSIX shared memory (`shm`) files, to provide scalable and dynamic service management. In each `shm` file, collected features are further partitioned by egress equipments so that spatial information can be distinguished. Fig. 5 exemplifies the `shm` layout and data flow.

1) Bit-Index and Masking: The first byte in the `shm` file of a VIP defines the max number of egress equipments N (64 in Fig. 5). The N-bit *bit-index header* helps quickly identify activated egress and its corresponding memory space–the i-th bit is set to 1 if the i-th egress is active and 0 otherwise. Adding an activated egress to the system requires only to set the corresponding bit to 1 after initialising its memory space. It suffices to simply flip the bit from 1 to 0 to deactivate an egress node.

2) Independent Egress Memory Space: Each egress node has its own independent memory space, storing counters, reservoir samples, and data plane policies (actions). As depicted in Fig. 5, on receipt of the first `ACK` from the client to a specific egress node i, VNF increments the number of flows in the counters cache of node i. Quatitative features (*e.g.*, flow duration $t_3 - t_0$ gathered at t_3 in Fig. 5) can be stored in the reservoir buffer of node i using Alg. 1. Obtained features for all active egress nodes can then be aggregated and processed to make further inferences or data-driven decisions, which can be written back to the memory space of each egress node.

3) Multi-Buffering and Asynchronous I/O: Counters and actions are exchanged between cache and buffer using m-level multi-buffering with incremental sequence ID. When copying data, the sequence ID is set to 0 to avoid I/O conflicts. ML algorithms can pull the latest observations and push the latest data-driven decisions using multi-buffering with no disruption

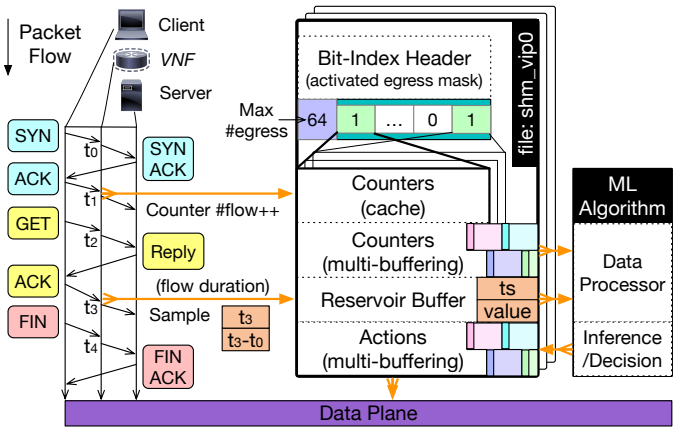


Fig. 5: Aquarius `shm` layout and data flow pipeline.

Operation / Complexity		Computation	Memory
Add / Remove VIP		$\mathcal{O}(1)$	$\mathcal{O}(kN+mN)$
Add egress node		$\mathcal{O}(1)$	$\mathcal{O}(k+m)$
Remove egress node		$\mathcal{O}(1)$	$\mathcal{O}(1)$
Register reservoir sample Update counter (cache)		$\mathcal{O}(1)$	$\mathcal{O}(1)$
Update counters / actions (multi-buffering)		$\mathcal{O}(1)$	$\mathcal{O}(N)$
Get the latest observation	1 node	$\mathcal{O}(m)$	$\mathcal{O}(k+m)$
	All nodes		$\mathcal{O}(kN+mN)$
Update action in the data plane	1 node	$\mathcal{O}(m)$	$\mathcal{O}(1)$
	All nodes		$\mathcal{O}(N)$

TABLE II: Computation and memory complexity of different operations, where k is the size of reservoir buffer, N is the number of egress nodes, and m is the level of multi-buffering.

in the data plane. This design offers an asynchronous 2-way communication interface to exchange fine-grained features and data-driven decisions with low latency.

Both computation and memory space complexity is presented in Table II. The whole dataflow is asynchronous and avoid stalling in the data exchange process in both the data plane and the control plane.

C. Implementation

This paper implements Aquarius as a plugin to the Vector Packet Processor (VPP) [10]. The flow table size is configured as $M = 65536$. Each sampled network feature is a 2-tuple of a 32-bit float timestamp and a 32-bit value. The reservoir buffer size is $k = 128$ for each feature per egress equipment. The `shm` file of each VIP consists of 6KB 3-level multi-buffering counters and 832KB reservoir sampling buffers. In this paper, Aquarius is implemented to be able to collect 8 ordinal features (counters) and 65 quantitative features in total.

IV. Applications

This section shows 3 applications of Aquarius in cloud DCs in the context of 3 key VNFs–traffic classification, resource prediction and auto-scaling, and Layer-4 load balancing, along with 3 different ML paradigms.

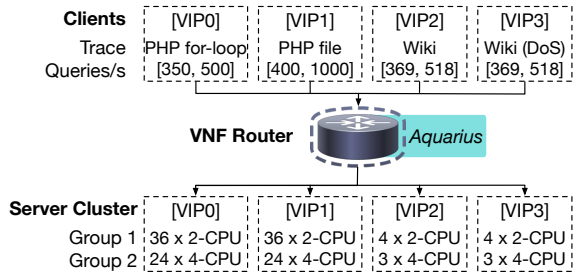
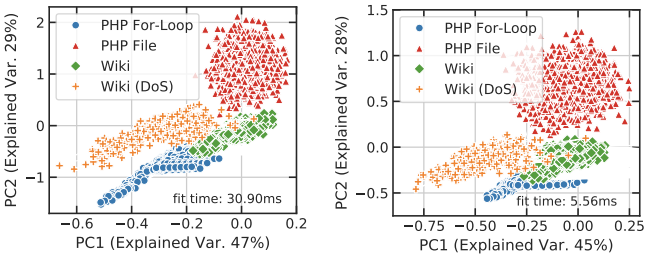


Fig. 6: Network topology for traffic classification.



(a) PCA clusters (all features). (b) PCA clusters (25 features).

Fig. 7: PCA analysis and 2D visualisation.

A. Traffic Classification

As one of the key VNFs in the cloud, traffic classification allows distinguishing different types of traffic [23]–[25], to allocate appropriate resources and achieve service level agreements. It also helps detect anomalies and security threats to prevent potential damages or losses [23].

1) Task Description and Testbed Configuration: This section shows the capability of Aquarius to collect reliable features and conduct traffic classification with unsupervised ML algorithms. A testbed is implemented using Kernel-based Virtual Machine (KVM), where a virtual router embedded with Aquarius forwards different types of traffic to 4 VIPs (Fig. 6). In VIP0, a simple PHP `for`-loop script on each server takes requests for given number of iterations and replies with proportional sizes. The flow duration and size follow an exponential distribution as in [34]. In VIP1, static files of different sizes are served on each server[1] as in [6], to represent IO-bound applications. In VIP2 and VIP3, each application server is an independent replica of an Apache HTTP server that serves Wikipedia databases. Two samples of 600s duration are extracted and replayed from a real-world 24-hour replay [35]. In VIP3, an additional 5000 queries per second SYN flooding traffic is applied to simulate a DoS attack.

2) Principal Component Analysis (PCA) and Clustering: PCA is conducted to visualise the 4 clusters for the 4 different types of network traces in a 2D projection (Fig. 7a). Among the 4 traces, PHP `for`-loop is pure CPU-bound and PHP file is pure IO-bound. The `Wiki` trace consists of both queries for SQL database (CPU-bound) and static files (IO-bound), thus its cluster is located between the former 2 traces. The `Wiki`

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

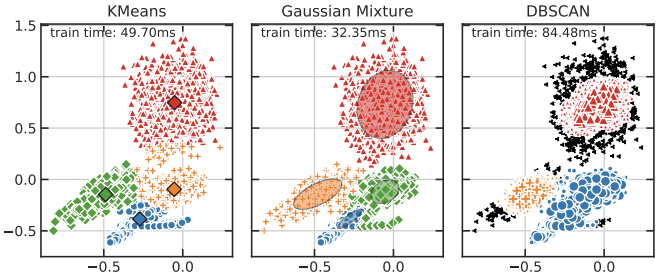


Fig. 8: Unsupervised clustering using 25 features.

Configuration		0 Feature	11 Features	73 Features	PCAP
First Packet	CPU Cycles	938.232	1635.838	2609.019	1295.284
	Delay (μs)	0.361	0.629	1.003	0.498
	Difference	1.000×	1.744×	2.781×	1.381×
Data Packet	CPU Cycles	576.357	1583.798	2602.684	885.041
	Delay (μs)	0.222	0.609	1.001	0.340
	Difference	1.000×	2.748×	4.516×	1.536×
CPU Usage (%)		26.687	40.858	49.716	31.480
CPU Difference		1.000×	1.376×	1.675×	1.060×
RAM Usage (GiB)		2.652	2.719	2.744	3.305
RAM Difference		1.000×	1.025×	1.034×	1.246×

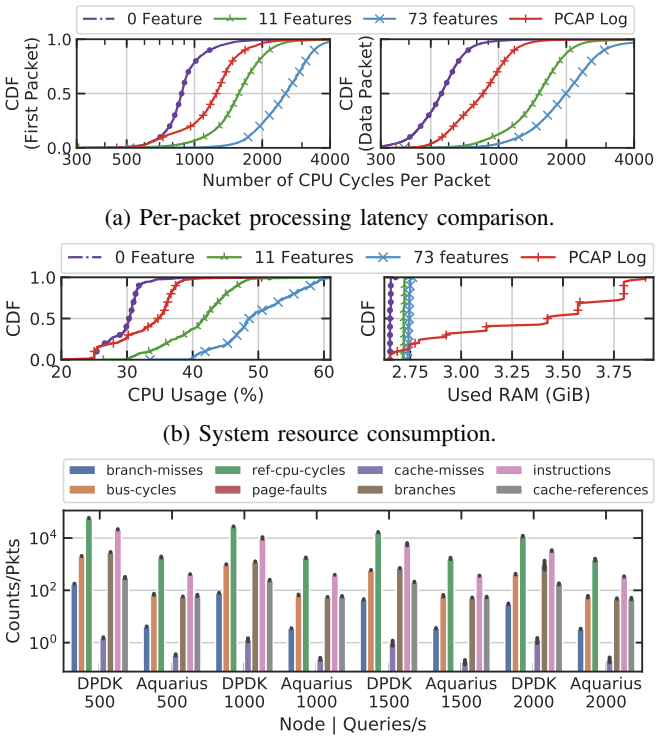
TABLE III: Per-packet processing overhead (on 2.6GHz CPU) and system resource consumptions (avg.) comparison.

trace under DoS attack, however, can be clearly noticed as an independent cluster. More features give multi-dimensional observations, yet at the cost of higher computation and memory overhead. PCA helps reduce feature dimensionality from 73 to 25, while preserving data representation. As depicted in Fig. 7b, using 25 features still gives clear clustering results, yet it reduces data processing time from 30.90ms to 5.56ms.

3) Unsupervised Learning: As depicted in Fig. 8, when applying unsupervised learning algorithms, K-Means and Gaussian Misture are able to generate clusters similar to the ground truth, while they require the number of expected clusters (4) as input. Gaussian Mixture model has the shortest fit time and can be an interesting candidate for online traffic classification. In case where the number of clusters is not known *a priori*, DBSCAN [36] can distinguish the potential security threat, based only on a predefined distance of 0.1.

4) Overhead Analysis: To study the feature collection overhead, Aquarius is compared with a vanilla router which collects 0 feature and a router logging packet information in the memory using `pcap`. Under 500 queries/s PHP `for`-loop traffic towards a 176-CPU server cluster, when collecting 11 features or collecting all 73 features, Aquarius incurs different overhead (Table III and Fig. 9a). On a 2.6GHz CPU, the additional per-packet processing delays are trivial comparing with the typical round trip time (higher than 200μs) between network equipments. The mean CPU usage of Aquarius is 1.376× and 1.675× higher than the vanilla router when collecting 11 and 73 features respectively (Fig. 9b). As expected, log-based feature collection mechanism does not scale in terms of memory consumption[2]. As depicted in Fig. 9c, the system performance of Aquarius shows advantages over RX interface

[2]The results can be machine-dependent. This paper aims at showing the order of magnitudes, rather than providing a precise quantification.



(a) Per-packet processing latency comparison.

(b) System resource consumption.

(c) System performance metric comparison

Fig. 9: Aquarius feature collection overhead.

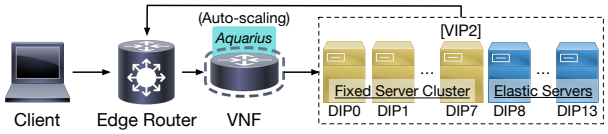


Fig. 10: Network topology for autoscaling system.

driven by DPDK on various metrics when using different traffic rates.

Take-Away: Aquarius gathers fine-grained and reliable datasets, which allow feature engineering and conducting in-depth data analysis. Its fast and configurable design help achieve the right balance between visibility and performance.

B. Resource Prediction and Auto-Scaling

To minimize operational cost while guaranteeing QoS, cloud operators [11]) need to elastically provision server capacities.

1) Task Description and Testbed Configuration: This section shows the capability of Aquarius as a platform to adapt supervised ML algorithms to infer resource utilisation with no actively signaling. 600s samples extracted from the real-world 24-hour Wikipedia trace are replayed on the network topology depicted in Fig. 10. Workloads are randomly distributed among running servers (2-CPU each) by way of ECMP. A learning task can be framed as predicting server load states (CPU usage) on each server with the same set of features as in Sec. IV-A. The predicted utilisation can be then used to plan and re-scale server cluster to guarantee QoS with reduced operational cost. This task consists of 2 steps–offline model training and online prediction.

Algorithm 2 Auto-scaling Rule

$n\_servers\_min, n\_servers\_max \leftarrow 8, 14$ ▷ Server number range
$\mathbb{S} \leftarrow$ Initial set of running servers
$cpu\_lo, cpu\_hi \leftarrow 0.7, 0.8$ ▷ Desired CPU usage range
for each time step **do** ▷ Δt = 250ms
  $\delta \leftarrow 0$ ▷ Initialize server state counter
  $y(\mathbb{S}) \leftarrow$ CPU usage prediction of 16 steps ahead
  $threshold \leftarrow \lceil \frac{|\mathbb{S}|}{3} \rceil$ ▷ Threshold that triggers scaling actions
  for $s \in \mathbb{S}$ **do**
    if $y(s) < cpu\_lo$ **then**
      $\delta++$ ▷ Increment δ if s is under-loaded
    else if $y(s) > cpu\_hi$ **then**
      $\delta--$ ▷ Decrement δ if s is over-loaded
  if $\delta > threshold$ **and** $|\mathbb{S}| > n\_servers\_min$ **then**
    $\mathbb{S} \leftarrow downscale(\mathbb{S})$
    skip 8 time steps ▷ Cool-down period
  else if $\delta < -threshold$ **and** $|\mathbb{S}| < n\_servers\_max$ **then**
    $\mathbb{S} \leftarrow upscale(\mathbb{S})$
    skip 8 time steps ▷ Cool-down period

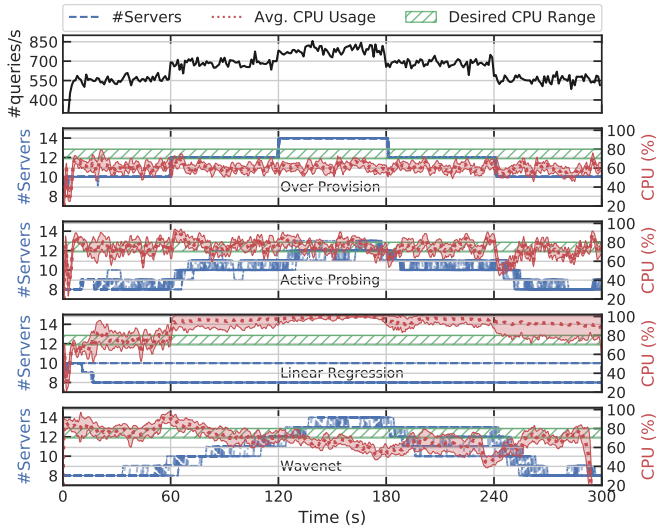
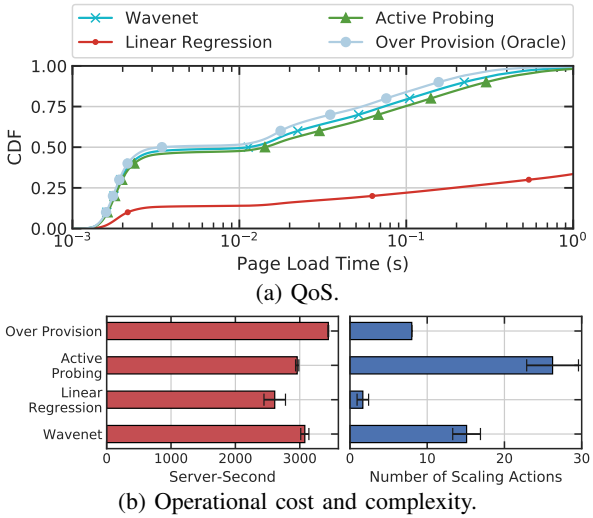


Fig. 11: Comparison of online auto-scaling performance using different algorithms. The (discrete) numbers of running servers are plotted for each run in dashed lines, while CPU usage is summarised as avg. ± stddev across 30 runs.

2) Offline Model Training: To predict the resource utilisation of server clusters using networking features, 12 widely used ML algorithms are selected to cover different families of ML algorithms, *e.g.*, sequential and non-sequential, parametric and non-parametric, linear and non-linear [8]. The first 23-hour samples are applied on 10 servers to gather datasets for offline model training. The distribution of the ground truth CPU usages in the training set covers the other two datasets so that the prediction task is feasible, yet the ML models have not seen the datasets for evaluations. Based on the offline training performance evaluation, 1 non-sequential model (linear regression) and 1 sequential model (WaveNet [37]) are selected to be applied for online auto-scaling[3].

3) Online Auto-Scaling: To test the online performance of offline-trained ML models, a 300s Wikipedia replay trace sample of the last hour (unseen by the ML models) is synthesized

[3]The whole comparison among all 12 models and additional experimental results will be included supplementary details.



(a) QoS.

(b) Operational cost and complexity.

Fig. 12: Trade-off between QoS and cost using different autoscaling mechanisms.

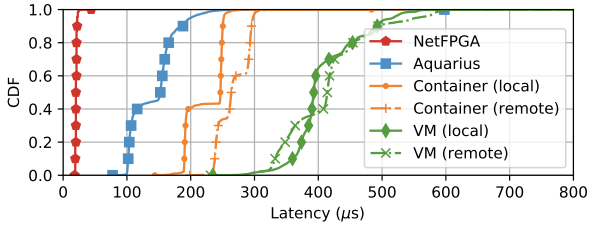
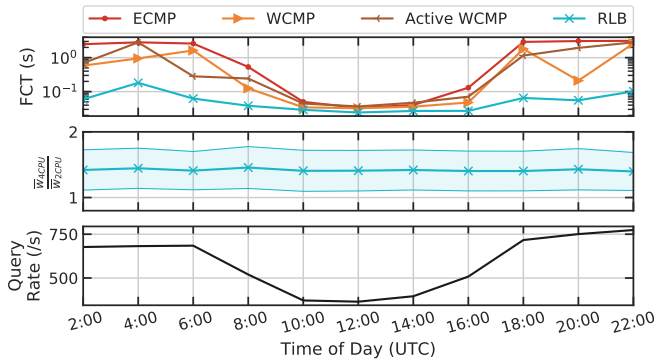


Fig. 13: Feature collection latency comparison between Aquarius and active probing techniques.

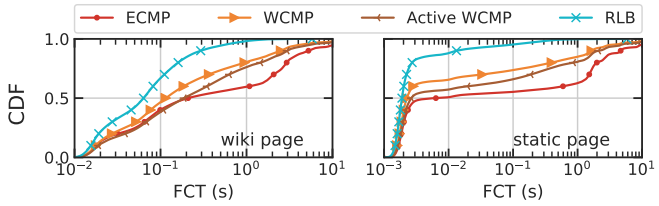
to have scheduled changing traffic rates every 60s. Based on the CPU usage predictions of running servers $y(\mathbb{S})$, a simple heuristic (Alg. 2), which approximate the threshold-based autoscaling policy as in [11], is proposed to keep the CPU usage of $\frac{2}{3}$ servers within the desired range ($70 \sim 80\%$). Using the same counter Δ for over- and under-loaded servers reduces the variance induced by imbalanced workload distributions. As a comparison to [11], a threshold-based active probing mechanism is implemented, whose predicted CPU usage for running servers $y(\mathbb{S})$ come from periodic polling. An "oracle" benchmark is implemented to over-provision the number of servers proportional to the scheduled traffic rates.

4) Results: As depicted in Fig. 11, active probing keeps the average CPU usage within the desired range, however, it requires frequent scaling events and leads to oscillating CPU usage with high variance. Linear regression is simple yet not robust when applied for an online auto-scaling system. Its under-estimated server load states lead to over-loaded servers. WaveNet takes sequential features as input and is more robust when applied online. It keeps the average CPU usage close to the desired range with less oscillations.

As depicted in Fig. 12, WaveNet is able to provide better QoS than active probing–78.37ms less page load time (26.04%) at 90th percentile and 35.70ms less (30.45%) on average–with 3.99% additional server-second cost, and 42.44% less scaling events. When over-provisioning the server cluster, the page load time is lower than using WaveNet by



(a) Mean FCTs (top), ratio between weights assigned to the 2 groups of servers by RLB (middle), and traffic rates (bottom).



(b) Peak-hour (query rate higher than 500/s) FCT distribution.

Fig. 14: Wikipedia trace replayed using different LBs.

67.13ms at 90th percentile and 28.55ms average, though it requires 11.86% more server-second operational cost than WaveNet.

Aquarius parses features stored in the local shared memory with no control messages, achieving more than 94.18μs less median latency than typical VM- and container-based probing mechanisms (Fig. 13).

Take-Away: Aquarius enables agile development and online deployment of learning algorithms to improve network performance. It makes features quickly accessible while saving management bandwidth for data transmission.

C. Traffic Optimisation and Load Balancing

As a key component in cloud DCs, Layer-4 load balancers (LBs) distribute workloads across servers to provide scalable services. Yet, in-production LBs requires human intervention which can lead to server weights mis-configurations.

1) Task Description and Testbed Configuration: This section shows that Aquarius can apply RL algorithms to self-configure server weights and optimise load balancing performance with no-prior knowledge of the system. The configuration of VIP2 (Fig. 6) is applied–replaying the `Wiki` trace and load balancing on 2 groups of servers of different processing capacities. The task is to extract and infer server processing capacity information from networking features and make informed load balancing decisions. 3 benchmark LB algorithms are implemented–(i) ECMP [13], (ii) statically configured WCMP [38], and (iii) active WCMP [6] based on polled server job queue lengths. Both ECMP and WCMP are based on the kernel-bypassing implementation of Maglev [12] in VPP, which is the state-of-the-art LB.

2) RL Algorithm: This paper implements and evaluate RLB [3]–implemented and evaluated in simulators–in a realistic testbed using Aquarius. With Aquarius, RLB (i) counts

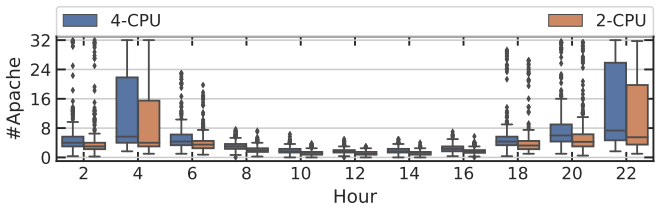
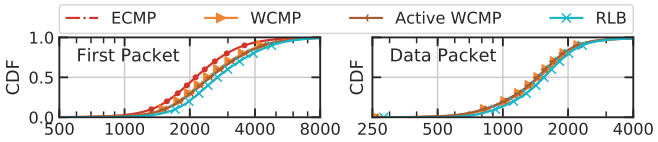
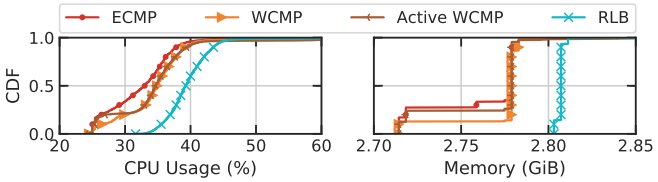


Fig. 15: Query distribution (number of busy Apache threads) on 2 groups of application servers.



(a) Per-packet processing latency comparison.



(b) System resource consumption.

Fig. 16: Overhead comparisons.

ongoing flows $\tilde{l}_i$ on servers and (ii) asynchronously updates (every 250ms) server weights $\tilde{w}_i$ derived from flow durations τ_i sampled in reservoir buffers. On receipt of new requests, RLB assigns servers based on $\arg\min_i \frac{\tilde{l}_i+1}{\tilde{w}_i}$, which prioritizes servers with higher processing speed and shorter queue lengths. Different from [3], which uses actively probed ground truth information, this paper derives the reward from locally observed features (FCT) collected by Aquarius, which requires no active signalling between LBs and servers. RLB is trained using the first hour of `Wiki` trace sample for 20 episodes. The trained RLB model is then tested on unseen traffic and compared with other LB algorithms.

3) Results: As depicted in Fig. 14a, during off-peak hours, servers are under-utilised and all algorithms show similar performances in terms of QoS. As traffic rates grow, RLB achieves lower FCT for both static pages and Wikipedia pages when compared with other LB algorithms (Fig. 14b), since RLB is able to dynamically adjust server weights. As depicted in Fig. 14b, for Wikipedia pages the 90-th percentile FCT of RLB (0.292s) is 18.15x shorter than Maglev ECMP and 8.56x shorter than Maglev WCMP, and the 95-th percentile FCT of RLB (0.552s) is 17.82x shorter than Maglev ECMP and 7.82x shorter than Maglev WCMP. For static pages, the 90-th percentile FCT of RLB (0.013s) is 351.15x shorter than ECMP and 124.43x shorter than WCMP, and the 95-th percentile FCT of RLB (0.088s) is 109.09x shorter than ECMP and 37.50x shorter than WCMP. RLB is trained to learn server processing speed differences and assigns higher weights, thus more queries, to more powerful servers (Fig. 15). When using RLB, 4-CPU servers handle respectively 1.258× and 1.523× more tasks than 2-CPU servers under 676.92 and 372.01 queries/s traffic.

4) Overhead Analysis: As depicted in Fig. 16a, throughout all test runs, RLB consume on average 692.89 more CPU cycles (0.26μs on 2.6GHz CPU) than ECMP, as it computes and compares the server scores when making load balancing decisions. Fig. 16b depicts CPU and memory consumptions of all LBs. On average, RLB incurs $0.22\times$ additional CPU usage, and 45.99MiB memory usage, and achieves 87.38% throughput of ECMP.

Take-Away: Aquarius enables closed-loop control (RL) to dynamically adapt to networking systems and optimise performance. It empowers real-world deployment and evaluation of learning algorithms developed in simulated environments.

V. Conclusion

Networking features and system state information help VNFs make informed decisions, and intelligently manage and update networking policies in cloud DCs. Actively collecting features and system state information entails substantial control signalling and management overhead, in particular in large-scale DC networks. This paper has proposed Aquarius, a framework that collects, infers and supplies accurate networking state information with little additional processing latency, in a scalable buffer layout. The paper has illustrated significant performance gains of using of Aquarius for various ML-based VNFs and evaluated experimentally the impact of Aquarius in the system performance.

References

[1] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," *arXiv preprint arXiv:1810.01963*, 2018.

[2] 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.

[3] Z. Yao, Z. Ding, and T. H. Clausen, "Reinforced workload distribution fairness," in *5th Workshop on Machine Learning for Systems at 35th Conference on Neural Information Processing Systems*, 2021.

[4] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacgümüş, "Smartsla: Cost-sensitive management of virtualized resources for cpu-bound database services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1441–1451, 2015.

[5] V. Sivakumar, T. Rocktäschel, A. H. Miller, H. Küttler, N. Nardelli, M. Rabbat, J. Pineau, and S. Riedel, "Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions," *arXiv preprint arXiv:1910.04054*, 2019.

[6] 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.

[7] Z. Yao, Y. Desmouceaux, J.-A. Cordero-Fuertes, M. Townsley, and T. Clausen, "Hlb: Toward load-aware load balancing," *IEEE/ACM Transactions on Networking*, 2022.

[8] S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy, "On the use of ml for blackbox system performance prediction." in *NSDI*, 2021, pp. 763–784.

[9] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 561–575.

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

[11] "Amazon elastic compute cloud," https://aws.amazon.com/ec2/.

[12] 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.

[13] 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.

[14] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: high precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 44–58.

[15] OPNFV, "Open Platform for NFV (OPNFV) Project Portal," https://www.opnfv.org/, 2019.

[16] "OpenStack Project Portal," https://www.openstack.org/, 2019.

[17] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. Lakshman, "Uno: Uniflying host and smart nic offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 506–519.

[18] "AWS Nitro System," https://aws.amazon.com/ec2/nitro/.

[19] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "Policycop: An autonomic qos policy enforcement framework for software defined networks," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–7.

[20] P. Gonçalves, J. L. Oliveira, and R. L. Aguiar, "An evaluation of network management protocols," in *2009 IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 2009, pp. 537–544.

[21] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–40, 2018.

[22] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 323–335.

[23] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *arXiv preprint arXiv:1710.00811*, 2017.

[24] K. Lalitha and V. Josna, "Traffic verification for network anomaly detection in sensor networks," *Procedia Technology*, 2016.

[25] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM workshop on hot topics in networks*, 2019, pp. 25–33.

[26] T. Swamy, A. Rucker, M. Shahbaz, and K. Olukotun, "Taurus: An intelligent data plane," *arXiv preprint arXiv:2002.08987*, 2020.

[27] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[28] K. Shafi and H. A. Abbass, "Evaluation of an adaptive genetic-based signature extraction system for network intrusion detection," *Pattern Analysis and Applications*, vol. 16, no. 4, pp. 549–566, 2013.

[29] "Predict," https://www.predict.org/.

[30] "NSL-KDD," http://nsl.cs.unb.ca/NSL-KDD/.

[31] "CAIDA," https://www.caida.org/.

[32] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[33] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[34] 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, event-place: London, United Kingdom.

[35] 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.

[36] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, 2017.

[37] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[38] 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.