POLITECNICO DI TORINO
Repository ISTITUZIONALE

Load Profiling via In-Band Flow Classification and P4 With Howdah

Original
Load Profiling via In-Band Flow Classification and P4 With Howdah / Angi, A., Sacco, A., Esposito, F., Marchetto, G., Clemm, A.. - In: IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. - ISSN 1932-4537. - ELETTRONICO. - 21:1(2024), pp. 295-309. [10.1109/TNSM.2023.3299729]

Availability:
This version is available at: 11583/2980823 since: 2024-02-12T13:17:46Z

Publisher:
IEEE

Published
DOI:10.1109/TNSM.2023.3299729

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

17 June 2026

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

Load Profiling via In-Band Flow Classification and P4 with Howdah

Antonino Angi, *Student Member, IEEE,* Alessio Sacco, *Member, IEEE,* Flavio Esposito, *Member, IEEE,* Guido Marchetto, *Senior Member, IEEE,* and Alexander Clemm, *Member, IEEE*

***Abstract*—Data center traffic management challenges increase with the complexity and variety of new Internet and Web applications. Efficient network management systems are often needed to thwart delays and minimize failures. In this regard, it seems helpful to identify in advance the different classes of flows that (co)exist in the network, characterizing them into different types based on different latency/bandwidth requirements. In this paper, we propose Howdah, a traffic identification and profiling mechanism that uses Machine Learning and a load-aware forwarding strategy to offer adaptation to different classes of traffic with the support of programmable data planes. With Howdah, the sender and gateway elements inject in-band traffic information obtained by a supervised learning algorithm. When a switch or router receives a packet, it exploits this host-based traffic classification to adapt to a desirable traffic profile, for example, to balance the traffic load. We compare our solution against recent traffic engineering proposals and demonstrate the effectiveness of the cooperation between host traffic classification and P4-based switch forwarding policies, reducing packet transmission time in data center scenarios.**

***Index Terms*—load profiling, machine learning, traffic classification**

I. Introduction

In the last two decades, data centers have changed their topology in response to the increasing demands of networked applications that continue to require more data at a faster speed while requiring lower latency. Because of these requirements, new data center architectures have been proposed, focusing on ingress and egress traffic optimizations but also on better orchestration of data center internal traffic. Data center topologies have also evolved to represent multi-rooted leaf-spine or, more often, fat-trees. Such topologies have in common the presence of multiple source-destination paths to handle the high traffic volume, which can lead to the necessity of having routing strategies that deal with different traffic loads in the network, aimed at avoiding congestion, lowering delays, and still high performance.

One problem concerns ensuring that traffic is properly balanced, meaning that traffic is evenly distributed so that the maximum link utilization of any links is minimized. This way, problems such as congestion and resulting sudden packet loss or delay variations can be avoided. However, compounding the problem are aspects such as uneven link bandwidth and the differing quality of service requirements and traffic priorities across flows. In these cases, it is desirable to have routing algorithms that deal with different classes of traffic, characterized by different priorities and demands. These solutions are referred to as load-profiling routing algorithms [2], [3].

One of the most common strategies is still Equal-Cost Multi-Path (ECMP), a routing algorithm that statically hashes flows for path assignment. This algorithm is known to forward flows randomly to a path and does not consider potential congestion or link failure in the network. For this reason, ECMP might lead to uneven flow distribution and, consequently, poor performances [4]. Recent solutions attempted to overcome such ECMP limitations, and while they are all sound solutions, they either introduced additional overhead, e.g., [5], [6] or failed to apply efficient logic per-packet, e.g., [7], [8]. On the one hand, centralized schemes, such as Hedera [7], B4 [8], FastPass [9] and SWAN [10], can perform congestion-aware decisions, but demand considerable control traffic and react too slowly for volatile (data center) traffic. On the other hand, recent distributed approaches, such as CONGA [5] and HULA [6], introduce periodic network feedback that might lead to excessive overhead traffic and contribute to congestion. In line with these efforts, the research question we are addressing in this paper is: *"Can we profile the network traffic's load over uncongested paths without the need to define elaborate protocols for exchanging information amongst the switches or between the switches and a centralized controller?"*

In this paper, we answer this question with *Howdah*, a data-plane programmable architecture that enables load profiling by taking forwarding decisions via a distributed and (partially) congestion-aware logic. The idea behind Howdah is a joint optimization: minimization of collisions between flows and maintenance of high utilization inside the datacenter network.[1] In Howdah, network switches are instructed with P4 programs to run a data-driven load profiling that, rather than flows, operates over flowlets: bursts of packets in a flow, split by a sufficiently large time gap. Approaches based on flowlets have been shown [11] to be preferred as there are no packet reordering problems (since packets of the same flowlet are

This paper is an extended version of [1].

This work has been partially supported by NSF awards #1908574 and #2201536.

Antonino Angi, Alessio Sacco and Guido Marchetto are with DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: antonino.angi@polito.it, alessio_sacco@polito.it, guido.marchetto@polito.it).

Flavio Esposito is with the Department of Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: flavio.esposito@slu.edu).

Alexander Clemm is with the Futurewei Technologies Inc, Santa Clara, CA, 95050-2516 USA (e-mail: alex@clemm.org).

[1] An howdah, derived from the Arabic word *hawdaj*, which means "bed carried by a camel", is a carriage positioned on the back of an animal, typically an elephant or occasionally a camel. We called our solution Howdah since, as in the real *howdah*, it is a tiny overhead that can serve several applications and can be carried over elephant (or other) flows.

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

forwarded along the same path) and no modifications to the TCP stack are needed.

To further optimize path selection, forwarding actions are differentiated according to the type of traffic carried in the packets. Howdah's sending hosts internal to the data center and peripheral gateways run a supervised Machine Learning (ML) model to predict if each flow entails a large amount of data, *elephant flow*, or a small amount, *mouse flow*, and this traffic knowledge is transferred directly to the intermediate switches and inserted into the packet in an in-band fashion. Because elephant flows are mostly responsible for network congestion, they are routed over the fastest paths by considering the least utilized path from the switch perspective; mice, less likely to overload network nodes, are just forwarded with a weighted version of the ECMP algorithm in combination with the flowlet-based grouping.

An ML classifier can help generalize over diverse traffic patterns while reducing the classification time compared to other statistical traffic classifiers. For this reason, in Howdah we looked for a ML model that could give us the smallest overhead in terms of training and classification time, RAM and CPU percentile usage, and could also generate fewer carbon emission when compared to other tested ML models (i.e., Support Vector Machine, k-means, Random Forest, and Neural Network). After analyzing the accuracy of each model on real-world traffic, in our architecture we decided to rely on an easily explainable and decision-maker transparent model as a Decision Tree.

Then, we studied how Howdah's performance changes when different protocols are used to carry the in-band traffic information.Among different tested protocols (e.g., MPLS, New IP, IPv6), we found how IP type-of-service fields provide negligible overhead and represent the first implementation of our proposed architecture in transferring information on traffic classes. We tested such implementation in data center network scenarios and compared it to recently proposed benchmarks. Our results validate how our solution can reduce both Round-Trip-Time (RTT) and Flow Completion Time (FCT) compared to other load-profiling routing algorithms, especially for elephant flows and congested networks.

The rest of the paper is organized as follows. Section II describes the state-of-art methodologies that also focus on variations of load profiling techniques. Section III shows the considered scenario, giving a general overview of our solution design. In Section IV we focus on different methods to carry the in-band information using already defined protocols. Section V describes the traffic classification method chosen for our solution. Finally, results are shown in Section VI, and the conclusion in Section VII.

II. Related Work

Efficient balancing/profiling of traffic load among available paths is a critical issue, especially in highly stressed networks such as data centers. Many recent studies addressed this problem, proposing solutions that fully use available bandwidth resources. Although traditional and local routing strategies (e.g., the standard ECMP) are widely used in practice, their performance is suboptimal for data centers due to local, trivial, and stateless decisions that lead to split traffic without knowledge of potential congestion on the network [7], [12], [13]. Recent local approaches, such as DRILL [14], Clove [15], and PRESTO [16], attempt to solve ECMP's shortcomings while confining decisions within each switch and ignoring global information. For example, DRILL forwarding decisions are load-aware and based on the local queue occupancy, enabling operating on microsecond (packet-by-packet) timescales. PRESTO [16] instead is based on the insight that ECMP provides near-optimal load balance in a symmetric Clos where all flows are small. As such, it divides flows into "mice" that are source-routed, so they are striped across all paths without demanding load awareness. However, both solutions have to deal with the performance impact and computational bottleneck of TCP reordering, which problem is exacerbated in asymmetric topologies.

A common approach to taking more appropriate actions is to delegate forwarding logic to centralized controllers and make congestion-aware decisions, as in B4 [8], F10 [17], Mahout [18], MicroTE [19], and Hedera [7], which are based on the assumption that global congestion information is helpful to balance the load evenly. However, despite having shown near-optimal traffic engineering for inter-data center WANs, these solutions were not designed to depart from balanced loads and for highly volatile data center networks due to the coarse time scale of their control operations. A recent centralized yet performant solution is Tiara [20], a three-tier architecture composed of a programmable switch that encapsulates/decapsulates the packet, an FPGA that handles the match-action tables, and an x86 server that stores the load balancing software for the slow paths. Despite the efficiency and scalability brought by this architecture, this solution is strictly dependent on specific hardware operations, which could increase the deployment cost.

To achieve microsecond performance while still using global information, CONGA [5] operates in the data plane and makes globally optimal allocations using a distributed approach, allowing a faster reaction in the presence of asymmetries. Using a leaf-to-leaf mechanism, in which switches at the edge (leaves in Clos networks) gather and analyze congestion feedback from remote switches to estimate congestion on fabric paths in real-time, CONGA combines this mechanism with the flowlet switching strategy. This study confirms the effectiveness and efficient utilization of network resources of flowlet-based forwarding, especially when applied in data centers. It has, however, two main limitations: first, the global congestion state at the edge switches can increase dramatically and exceed the switch memory; second, its implementation is designed for custom hardware. These limitations are explicitly addressed by HULA [6], a data-plane load balancing algorithm applied to P4-based programmable switches, in which leaf switches track congestion for the best path to a destination through a neighboring switch rather than for all paths, without requiring specifically designed hardware. Specifically, HULA uses probes to obtain network status information (i.e., link failure, topology change) and update the switches' internal tables. In a similar way, our solution uses the same principles for a data-plane load profiling strategy. However, instead of

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

using Top of Rack (ToR) switches to send probes, we send additional information in each packet so that each switch can better handle traffic congestion, which positively impacts performance.

A recent solution as CONTRA [21] provides a performance-aware routing that can adapt to traffic changes at hardware speed, allowing the users to specify network policies to rank network paths given their current performance. After a verification process, the CONTRA compiler decomposes these non-arbitrary policies into P4 switch local programs opportunely adapted to the topology. Nevertheless, HULA's policy is the default and best-performing setting.

Inspired by the idea of customizing forwarding decisions, we also provide some out-of-the-box load profiling actions that can be extended and adapted by the users to achieve specific performance goals. Unlike these load profiling solutions, however, in our local congestion-aware routing solution, switches are not the only ones doing all the work, but they are assisted by the host machines for the traffic type identification. This host-based traffic classification is inserted in-band and then used for the switches' forwarding decisions in the network.

Other recent proposals as Application Aware Networking (APN) [22] allow senders to convey information about specific flows for fine-granularity traffic steering and network resource adjustment. However, this framework requires multiple additional entities involved in the process, e.g., controller, edge node, head-end, and mid-point, which hinder deployment. In addition, they allow any client to convey metadata about traffic, assuming that senders act in good faith and are truthful. In contrast, our approach makes no such assumptions. For external traffic, the metadata insertion is performed by nodes of the network provider, while internal machines are under control.

III. Architecture and Protocol Design

Howdah is an architecture for data-plane programmability customized for load profiling. In particular, the switch attempts to maximize network resources using a load profiling approach combined with a classifier to label the flow and properly differentiate forwarding actions according to it. Such a traffic classifier can be deployed at the ingress of the network provider or at the local sender. Either way, in the context of our solution, we refer to this element as *Howdah host*.

Our Howdah algorithm is composed of two steps and two main architectural components: one running on local machines or gateways, and one running on P4-enabled switches (Fig. 1). As shown in the figure, the host classifies the traffic before sending it out throughout the network using a decision tree (D-Tree) model and injects the classification label into an appropriate field of the packet header (details in Section IV). Such a classification can be applied to both unencrypted and encrypted packets, i.e., flows whose payload is encrypted. When an intermediate switch receives a packet, it examines the output of the classification and differentiates its forwarding actions accordingly. While mouse flows are forwarded based only on the information in the packet header, without the need to update each switch's statistics, this is not true for

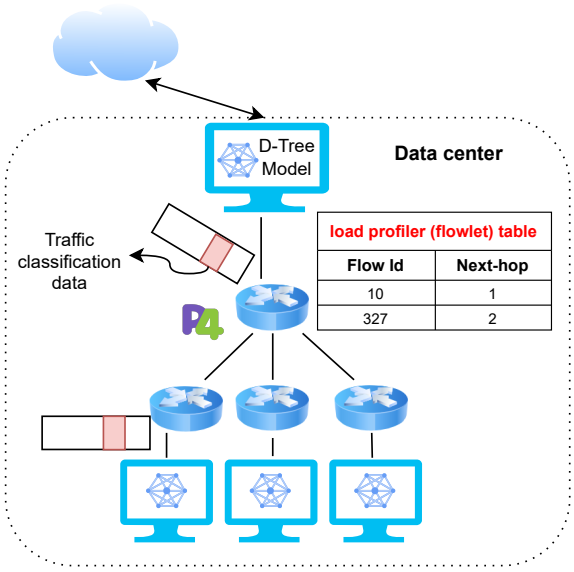


Fig. 1: Howdah overview. The system is based on the cooperation of hosts that help network nodes by inserting traffic classification information, which is then processed by the P4 switches. The switches run the load profiling algorithm that is differentiated by traffic type.

elephants, since they have a more considerable impact on network congestion. For this latter type, the switches forward the packet to the next hop port according to the least recently used (LRU) strategy. In the remainder of this section, we motivate this design choice and describe the algorithms of both Howdah host and P4-enabled switches.

A. Host-based Traffic Classification within Howdah Hosts

While switches implement load profiling and traffic engineering decisions, senders and gateways are designed to contain the traffic classification logic used, in turn, during the forwarding process of the switches. Since our network scenario consists of a data center topology, we assume that for East-West (internal) traffic, the sending hosts can be easily instructed to classify traffic via an ML model and insert this information into the packet header itself (see Section V for more details). Conversely, we assume external hosts may not implement any ML classification logic for the North-South traffic (to/from outside). Most importantly, external traffic classification could not be trusted. For this reason, when a packet is originated outside and arrives at our network, our gateway applies the same traffic classification algorithm before letting packets into the data center network, along with other packet filtering operations that are common in a data center. For traffic directed outside, instead, the classification data is stripped away before it leaves the data center.

Howdah for traffic classification. Data centers typically face a variety of traffic classes since they host multiple services. Among them, we can cite on-demand video delivery, storage and file sharing, web search, social networks, cloud computing, financial services, recommendation systems, and interactive online tools [23], [24]. These applications present different traffic characteristics and distribution of flow arrivals,

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

flow sizes, and flow duration [25]. For one, the data streams generated by web search queries are usually much smaller and shorter than the flows generated by batch computing jobs. Instead, high-performance computing (HPC) jobs, e.g., Hadoop, may transfer petabytes of data during the shuffle MapReduce phase [26]. Moreover, recently there has been a rapid increment of latency-sensitive and interactive applications, such as video and voice applications.

Such various applications lead to the emergence of both long-lived connections and short microbursts in the same network [27], [28]. As typical in the network management literature [18], [29], we refer to long-lived flows as "elephants" and short microbursts as "mice."

The common goal of a load profiling solution is *to provide high bisection bandwidth for throughput-sensitive and latency-sensitive flows without unduly delaying remaining flows by distributing "available" bandwidth across a set of candidate routes to match the characteristics of incoming QoS requests.* In line with recent studies [30], [31] that have pointed out the importance of classifying traffic into "elephants" and "mice", we also argue that long-lived flows must be identified to take appropriate actions and better orchestrate traffic. Not only the application treatment can be differentiated, but the network congestion can be alleviated if the load is properly balanced among the available (and redundant) links. Recent studies have pointed out how East-West traffic of a data center is responsible for traffic volume that is one order of magnitude larger than North-South traffic [32]. Avoiding bottlenecks is thus extremely important for all the traffic flowing in the topology.

It can be noted that, although we consider only two different types in this paper, the Howdah architecture, together with the P4 language, provides the flexibility to generalize on multiple types of differentiated traffic, e.g., bandwidth $vs.$ delay-sensitive applications, or web $vs.$ database $vs.$ HPC traffic.

Why host-based classification. A possible place for classifying packets would be the switch itself. However, the hardware characteristics of the network nodes are a poor fit for the learning procedure of an ML model, resulting in poor performance. Furthermore, given the strict packet scheduling of switches in data centers, the application of ML models would either negatively impact the packet forwarding process or necessitate a specific software and hardware design. To ensure fast forwarding of packets, the literature has presented valuable examples of switches that collect flow metrics but delegate the ML learning phase to a centralized controller [7], [8], [33]–[35]. However, both a per-flow statistic and a sampling mechanism do not scale: the bandwidth between the switches and the controller is limited, making the statistics transmission a bottleneck in this traffic management scheme. Moreover, collecting per-flow statistics would consume significant switch resources, while sampling detection, i.e., sampling only a small fraction of incoming packets, would result in accurate detection of elephant flows only after 10K packets [36]. In light of this, we argue that the host and the gateway are the optimal places for elephant flow detection in data centers. The application layer of data center programs can be augmented with our Howdah layer, and this option is favored by the single administrative domain and software uniformity of common data centers. In addition, there are likely GPUs or general-purpose CPUs on the hosts that are better suited for the ML classification process than the processing resources of typical network nodes. Lastly, hosts and gateways have good visibility into the patterns of application traffic being generated.

Why ML-based classification. While traditionally elephant and mouse classification was performed by means of statistics, we argue that an ML-based classification is faster and more accurate. As mentioned earlier, traffic classification can occur either at the host or on the network side. In the first case, as in Mahout [18], the metrics considered during the decision process are the buffer occupancy. This means that any packet has to wait before being sent in order to check the buffer. However, as demonstrated in Section VI-H, our classification lasts μs, as opposed to the ms of Mahout. Other host-based detection methods may still need to wait until the communication has started before deciding the flow size. On the other hand, for a network decision, for example inside an SDN controller as in ZOOM [37], the decision can consider the number and the size of flows currently in the network. Although in this case the classification is based on current data and thus accurate, this process requires a statistics polling interval and transients in the order of seconds. The usage of new data-driven algorithms, as in Howdah, allows reducing the overall process (classification + label stack) while achieving notable accuracy as demonstrated in Section VI-B.

B. P4-compatible Switches

The main task of the switch is to profile flowlets – bursts of packets belonging to the same flow separated by a significant time interval – to avoid possible side effects at the destination. It has been shown how forwarding flowlets over the same path avoids the possibility of later packets arriving at their destination sooner than others, which might result in a need for additional buffering and a negative impact on Quality of Experience [11]. Moreover, flowlet-based decisions (rather than flow-based) allow higher granularity while providing better performance [5].

To make our switches programmable and easily extended to any possible protocol used (Section IV), we instruct them with P4, a programming language for protocol-independent packet processes [38]. Such a language enables the programming of packet processing pipelines in packet forwarding ASICs and allows the definition of custom parsing rules and new protocol logic. P4's control model follows the SDN architecture and involves a separate control plane to deploy commands directly on networking devices. This approach offers many advantages over a hardware implementation: the user can modify the size of all variables and registers according to the topology of interest and the workload demands. For example, since Howdah can work with different packet header formats (see Section IV), the packet parsing can be smoothly adapted to meet the desired header policy. In addition, P4 provides a switch abstraction that is independent of the actual hardware: P4 programs are compiled into a target-independent repre-

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

sentation (front-end) and then recompiled to different specific platforms, e.g., NetFPGA [39].

Howdah switch forwarding. In our solution, we redefine P4 tables to apply match-action entries to implement our load-profiling actions. In general, P4 tables can be used by switches to specify behaviors such as preliminary next-hops, multicast groups, and ISO-OSI level-2 forwarding using MAC addresses. With Howdah, once the hosts have inserted the information about the traffic type, our P4 switches forward the packets based on port utilization and the included traffic type information. In particular, we make use of a table stored inside the P4 switch registers that contains the hash of the incoming flowlet and that records the output port and the last time a flowlet belonging to a certain flow was seen. This value is helpful in computing the difference between the stored timestamp and the new flow's arrival time. If such a difference is below *T*, whose value is chosen according to other state-of-the-art techniques [6], then the switch forwards the flowlet to the stored best-hop. Otherwise, the switch detects a new flowlet, computes the hash of the 5-tuple composed of the protocol, IP source & destination address, TCP source & destination port, and finally selects the best next-hop for the current 5-tuple. It is worth noting that the 5-tuple hashing is performed directly inside the switch, and there are no controllers involved. This approach allows a reduction of possible delays that might occur when interacting with a controller [40].

We recall the concept of *load profile* as the desired load on an outgoing link of the switch, which allows the user to specify how to split traffic over these links [2]. A common scenario is to load the switch's links (load balancing) evenly, but other circumstances may demand unequal distribution if links have different characteristics (e.g., link capacity) or the traffic has different priorities. Our P4-enabled switches can be effortlessly customized to implement the desired profiling policy.

More formally, consider a system with N different paths between a particular source and destination, and let W denote the overall load of the system. A load-balanced system would tend to distribute its load equally amongst all paths, making the actual bandwidth on each path as close as possible to W/N. A load-profiled system would tend to distribute its load so that the probability of satisfying the QoS requirements of incoming flow requests is maximized. This goal can be achieved by having differently loaded paths that maximize the likelihood of satisfying the bandwidth requirements. One simple algorithm for load profiling is based on assigning a weight q_i to any switch's port i, representing the probability of choosing the path. The switch, then, performs a weighted choice when selecting the output links so as to match the traffic profile chosen by the user [41]. Ports with a higher weight are chosen with a higher probability and hence more frequently than ports with a lower weight.

In our solution, forwarding rules are applied on top of a flowlet-based version of ECMP packet forwarding with weights. Like traditional ECMP, the next-hop selection is based on hashing the 5-tuple, but instead of per flow, decisions are made per flowlet. In the case of mice, the switch simply forwards the packet to the best next-hop according to the weighted flowlet-based version of ECMP, i.e., it chooses a path with a weighted probability to avoid congesting a path quickly. Otherwise, in the case of elephants, the next-hop selection also considers the least recently used (LRU) port. Since ports with higher utilization are more prone to cause congestion in the network, we also consider the frequency with which paths of a given port are selected. To do so, we need to update also the statistics about the network in each switch. Aside from calculating the hash function, the switch updates this utilization metric for each incoming packet. Despite being simple, this LRU criterion effectively avoids congestion – and reduces delay – because the flowlet is sent over different ports where it is less likely to share the bandwidth with other ongoing (large) flows.

IV. In-Band Traffic Knowledge Policy

Recent studies have pointed out that additional network information can reach a significant amount of bytes and some of the heaviest packets on the network [42]–[44]. Although their main purpose is to check if the network is congested or if requirements are being met, telemetry metrics can not excessively harm application data. One important countermeasure is provided by the In-band network measurement, which is increasingly used in various network management applications to insert network information directly as part of packet data, either as payload or header.

For this reason, in our solution we use in-band network management and configure the switch to forward the packet to the next hop by taking into account the additional data contained in the packet itself. By combining in-band flow information with P4, we reduce the control traffic of traditional SDN architectures, e.g., OpenFlow, where the switches communicate with the controller to decide flow rules (see Section VI-F for a numerical comparison). As explained in Fig. 1, the control traffic is now carried in the header of the packets. In what follows, we describe three possible algorithms to prove this architecture's viability and show that the network programming framework can indeed be used to support applications with real-time networking requirements without the need for custom hardware in networking devices or even controllers. Specifically, we examine the following alternatives and identify the advantages and disadvantages of each of them.

IP Type of Service. The Type of Service (ToS) field of IPv4 has been designed to indicate the priority of a datagram and request a route for a low-latency, high-throughput, or highly-reliable service. These 8-bits have been split to perform the Differentiated Services Code Point (DSCP) function with 6 bits, and Explicit Congestion Notification (ECN) with 2 bits. Although the router's behavior in response to these values is not specifically defined, IP ToS definitions are widely found in Unix implementations. For this reason, they appear to be the most viable approach to introduce our traffic classification data in combination with our programmable switches. Results in Section VI confirm the low overhead introduced by this solution. However, the limited bits available also limit the

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

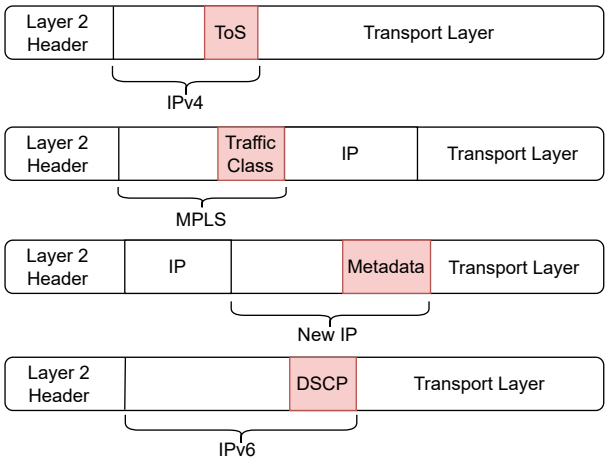


Fig. 2: Possible protocols for Howdah's packet header. Traffic classification information is inserted directly into the packet (in red), with little impact on the switch-forwarding process.

scalability and generability of the solution, which is unable to accommodate a broader range of application requirements and the switch's actions specification.

MPLS. Multiprotocol Label Switching (MPLS) works by prefixing packets with an MPLS header that contains one or more labels, forming a label stack. Each entry in the label stack contains four fields, including the 3-bit for Traffic Class field, typically used for QoS. An MPLS-compatible version of Howdah would use this field to carry the traffic flow information. Possibly, paths per flow are reserved in advance by means of the Label Distribution Protocol (LDP), and also profile information can be easily carried. This approach provides remarkable flexibility with more thorough traffic engineering decisions, but at the cost of an additional packet header and an additional protocol, such as LDP, for label distribution, or the administrator effort to set up the paths on each network device. Moreover, there are also new proposed variants of MPLS that would allow for the encoding and processing of metadata as part of a label stack [45].

New IP. Other possible protocol candidates that support ancillary data, which can be used to carry classification information (and more), include New IP [46], encoding ancillary data in a so-called flexible packet contract, and its precursor Big Packet Protocol (BPP) [47]. They provide an extendable approach to adapting packet-based networking behavior based on the introduction of the concept of a "contract": a block of data (metadata and forwarding instructions) carried with the header and user payload that can be used to inject ancillary information that provides guidance to intermediate switches on how to process these packets. In the context of our solution, traffic classification data and other data that may be useful to determine the proper treatment of a flow (such as information about service level guarantees) are included in the metadata field. In addition, the ancillary data can further be augmented to provide simultaneous support for additional functions, such as telemetry collection that can be used to refine traffic profiling further.

IPv6. IPv6, the most recent version of IP, includes an 8-bit field in its header called Traffic Class. In turn, this field is divided into two sub-fields used respectively for traffic classification and congestion management: Differentiated Services Code Point (DSCP) with 6 bits, and Explicit Congestion Notification (ECN) with 2 bits. In the IPv6-compliant version of Howdah, we consider adding the traffic information in the DSCP field, given its designed scope similar to the ToS of IPv4. The total IPv6 header occupies 320 bits, twice the size of the IPv4 header, since IPv6 addresses are 128 bits each. While the IPv6 adoption is increasing year by year and almost 50% of connections to Google happen over IPv6 [48], only 29.2% of all networks in the global BGP routing table support the IPv6 protocol [49]. Our idea of inserting traffic information in the IPv6 header can be used in conjunction with Segment Routing over IPv6 (SRv6), which allows routers to use parameters encoded using IPv6 extension headers [50] to perform special operations. Since Segment Routing Headers (SRH) [51], a well-known example of such an extension header, allows in principle also other parameters to be conveyed and processed, it is conceivable that it is compatible with our architecture.

Howdah essentials. Given the building blocks of Howdah presented earlier, we envisioned that our solution can work with multiple protocols carrying the traffic information. In this paper, we limit our attention to a few options that can be used to inform the switches about the type of traffic, as shown in Fig. 2. In detail, the figure shows, colored in red, where Howdah can insert the classification label on different protocols (IPv4, MPLS, New IP, IPv6), while all the other protocols' fields are intentionally not mentioned. However, we argue that given the programmability of our P4 switches, other possible protocols, such as VXLAN, can also be used. The essential idea of our solution does not change regardless of which protocol carries the classification result, and our solution involves hosts-switches cooperation towards optimized forwarding decisions: When the packet is ready to be sent, the host adds a flow type bit that helps the switches distinguish between an elephant flow and a mouse flow and react accordingly. The switch uses the header of Howdah in this binary differentiation of flows: "0" if mouse, "1" if elephant. In summary, this header field is used to inject meta-information directly into the packet to provide guidance through the network, where our Howdah's switches perform load profiling at the granularity of flowlets based on this value.

V. Howdah Traffic Classification

One important aspect at our system's core is traffic classification, as it impacts how packets are forwarded. This section describes the process that runs on the host machines responsible for accomplishing such classification task.

A. Decision Tree Model

In Howdah, we classify each flow using a decision tree, a predictive ML model that uses a tree-like structure to make decisions based on various input variables [52]. The main goal of a decision tree classifier is to predict the class of an input by building a tree-like structure where each internal node represents a feature and each leaf node represents a class, using

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

logical tests to identify relevant features and make accurate predictions based on trained data. Decision trees can be used for classification and regression tasks, especially when there are many possible values for each variable.

One key parameter in decision trees, often shortened as D-Tree, is the depth of the tree. A reasonable value of depth can help prevent overfitting, which occurs when the decision tree has too many branches and leaves [53]. Overfitting can result in poor performance on new data, as the decision tree may not generalize well to unseen data. By finding the optimal value for tree growth, we can ensure that the decision tree is neither too complex nor too simple and that it has the ability to predict the outcome of new situations accurately. For this reason, in Howdah we used a k-fold Cross-Validation procedure, an evaluation technique useful to find the optimal maximum depth for our trained dataset. In detail, this technique splits the dataset into k equal-sized subsets and iterates over each fold as the validation set while training the model on the remaining k-1 folds. This process allows us to evaluate the performance of the decision tree model on different subsets of the data, which contributes on reducing the bias and variance of the trained model.

B. Howdah Classifier Methodology

As mentioned, we used a decision tree to classify the traffic type for three main reasons. First, the structure of a D-tree resembles the decisions made by many networking systems, such as flow scheduling [54] and ABR algorithms [55], which make decisions based on rules. Second, they are lightweight for networking systems, bringing further benefits to resource consumption and decision latency. As detailed in Section VI-H, our decision tree classifier enables an accurate traffic classification while not incurring an excessive burden for the host machine. Third, decision trees have properties of expressiveness and high faithfulness because they are non-parametric and can represent very complex policies. Model interpretation is an important part of an ML process, as it can help understand the inner workings of the chosen model and ensure that it makes accurate and fair predictions. Therefore, using such an interpretable model can facilitate the monitoring and debugging of network operations (see Section VI-C for further details).

Howdah's hosts classify the type of traffic before transmission and inject the classification label directly inside the packet using a decision tree algorithm. Our supervised classifier acts over an input space of $1 \times N$, where 1 refers to the fact that it just considers a single packet, and N is the cardinality of features considered. In particular, our decision tree model is trained on a features list composed of five elements: source and destination IP address, source and destination port number, and transport protocol (i.e., TCP or UDP). Our packet interceptor can easily obtain this list and can work even with encrypted data, which frequently happens in data center applications. The output of this classification process is a binary label, 0 or 1, indicating whether it is a "mouse" or an "elephant", respectively.

Any host in our data center, as well as the gateway, should run a modified instance of either kernel-level network services or application-level socket instances. Since the literature has shown profitable usage of a shim layer on the end hosts [18], [25], in our prototype we considered the same option, and our results validate the efficacy, as shown in Section VI. To further simplify the operations over the host machines, we apply the classification process only if necessary. In detail, protocols known to contribute little to network congestion, such as ICMP, are automatically labeled as mouse flows. On the other hand, for unknown protocols and transport protocols that may be heavy (i.e., TCP and UDP), Howdah's classifier runs before the sending, and the output label is set in the packet header. It must be noted that even though forwarding is flowlet-based, the classification is per flow, thus reducing the number of times classification is executed. The traffic class is thus included in each packet of the flow, but the classification process is done only once.

Clarifying example. When a connection is established, before sending the flow, the host classifies it using the pre-trained D-Tree model. Assuming we are relying on the IPv4 version of Howdah, we encapsulate this information inside the ToS field of the header. It is important to notice that while the classification is flow-based, the forwarding is flowlet-based, reducing the amount of time the classifier has to be called. The receiving switch then computes the hash of the flowlet, taking into account the 5-tuple (protocol, IP source & destination address, TCP source & destination port), stores into its registers a flowlet table with the hash and the current timestamp, and chooses the output port according to the flow type. If mice, just using the weighted version of the ECMP algorithm for the flowlets; if elephants, via the least recently used (LRU) strategy. When another flowlet arrives, which belongs to a previously observed flow, the switch calculates the time difference between the current timestamp and the one stored in the table. If the difference is less than or equal to a specified value, denoted as T, the switch forwards the flowlet to the next hop indicated in the table. However, if the difference exceeds T, the switch recalculates the hash of the 5-tuple, updates the timestamp to the current value, and selects a new output port based on the LRU strategy. This concept is particularly important for latency-sensitive applications, where being routed over the less congested path is crucial. Short-lived applications, sometimes encapsulated in protocols diverse from TCP/UDP, are thus used to balance the overall network congestion.

VI. Evaluation

In this section, we illustrate the evaluation results that helped us develop our solution confirming Howdah's benefits. First, we summarize the experimental settings of our campaign. Then, we discuss the performance of our classifier and explain its behavior. We then evaluate the performance of our solution and compare it to a centralized version. Finally, we measure the resources consumed when running our proposed ML model.

A. Evaluation Settings

To validate our solution over a data center-like network, we deploy Howdah over Mininet, a network emulator that allows

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

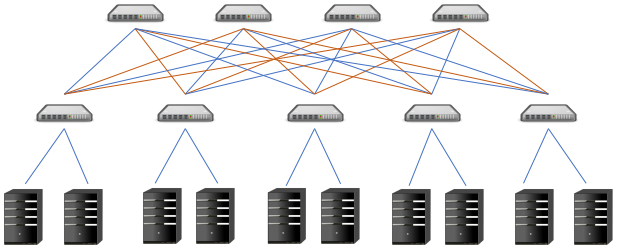


Fig. 3: Network topology used throughout the experimental evaluation. Blu links have 100 Mbps, and orange ones have 150 Mbps of bandwidth.

reproducing arbitrary virtual networks for fast simulations. Designed for SDN networks, Mininet can be used in combination with the behavioral model version 2 (bmv2) to configure P4-programmable switches in a simulated environment. To do so, Mininet compiles a P4 program into packet-processing actions of C++11 software switches, allowing network programmers to test and debug their code before deploying it on actual hardware. In Howdah, we focus on a load profiling problem. To set up our simulation environment, we build a leaf-spine topology composed of 10 server racks connected to their related switches, and each of these connected to other four switches, as shown in Fig. 3. In this load profiling scenario, to the orange links (150 Mbps), we assign a weight of 2 and to the blu links (100 Mbps) a weight of 1, to favor faster links. We use the $iperf3$ tool to reproduce different traffic workloads and to induce congestion in the network so that we can verify how the network behaves at different network loads.

We then tested the traffic classifiers when the input is composed of three realistic workloads, taken from publicly available datasets [56]. We extracted three different datasets and stored them in a .pcap file, corresponding to three captures obtained during the same day in the same data center but at diverse time instants. We mentioned them: "US-UNV-1" with $887,647$ items, "US-UNV-2" with $913,026$ items, and "US-UNV-3" with $887,647$ items. By scanning these files, we extracted the necessary features for each flow, and the flow label is assigned based on the total bytes exchanged by the flow: if this number is greater than D or the connection lasts more than L seconds, it is an elephant; otherwise, it is a mouse. As in [6], the threshold D is set to 1700 bytes while L is 10 seconds since we experienced these values are realistic, and the label assignment is not strongly imbalanced. Additionally, as in [5], [6], we set the flowlet gap to $100\mu s$ since we realized it is a reasonable value to identify flowlets belonging to the same flow when evaluating the workload datasets. In this set of experiments, hosts H1 and H4 sent background traffic using the *iperf3* tool and regularly adjusted the bandwidth to increase the network load and, thus, generate congestion. At the same time, H1 sends these trace-driven packets to all the other hosts (H2 - H10) and modifies the packet size according to the flows of interest (elephants or mice) while collecting and showing all the relevant metrics, such as FCT and RTT.

Traffic classifier benchmarks. We compare our Howdah classifier against four well-known and widely used ML models. First, a Support Vector Machine (SVM) model technique as in our previous work [1], where we had it combined with a stochastic gradient descent (SGD) technique to deal with large datasets while also reducing the computation time. Second, we considered a Neural Network (NN) classifier, composed of three fully connected layers where the first two hidden ones were made of 12 and 8 nodes and used the rectified linear unit activation function; the third layer, the output one, was composed of one node and used the sigmoid activation function. We tested different numbers of layers and nodes for our NN classifier to finally find the optimal combination that maximizes the performance metrics of our classification problem. Thirdly, a relevant study [31] investigates both supervised and unsupervised ML methods to identify flow types based on traffic characteristics. Its prediction proposes an unsupervised ML solution that uses a clustering technique as $k-means$, to predict classes, labeling each flow "elephant" or "mouse". Finally, a Random Forest (RF) model-based technique as in [30], where the solution of this study classifies flows intending to optimize the incast completion time on different buffered switches using elephant-based traffic.

Load profiler benchmarks. We compare our approach against two of the most recent solutions: CONGA [5] and HULA [6]. Note that even a more recent solution, CONTRA [21], employs HULA as its default approach. Differently from them, we do not use out-of-band probes because it is overhead traffic, but we inject network information directly inside the packet. Finally, ECMP is used as a baseline.

B. Traffic Classification Accuracy

To estimate the performance of our model, we use the standard notation TP for true positive, TN for true negative, FP for false positive, and FN for false negative. In particular, we pair TP and TN as the numbers of elephants correctly predicted (TP) or mice correctly predicted (TN); and FP and FN as the numbers of elephants erroneously predicted (FP) or mice erroneously predicted (FN). As mentioned before, in Howdah we have a binary classification – elephants and mice – that simplifies the definition of positive and negative classes. To compare different ML methods, we computed the most relevant performance metrics for these algorithms: accuracy, precision, recall, and f1-score, according to the definitions: (i) Accuracy: the ratio on which the model provides a correct prediction: *accuracy* = $\frac{TP+TN}{TP+TN+FP+FN}$. (ii) Precision: the fraction of true positives that are effectively and correctly classified as positives on the total of positives: *precision* = $\frac{TP}{TP+FP}$. (iii) Recall: the fraction of positives on the total of the real positives: *recall* = $\frac{TP}{TP+FN}$. Finally, (iv) F1-score combines both precisions and recall measures and it is defined as their harmonic average: *F1-score* = $\frac{2*TP}{2*TP+FP+FN}$.

After having trained all the considered classifiers over the 80% of samples in $US-UNV-1$, we computed the performance metrics over the remaining 20% of it and over the other two datasets. Table I shows the results of the performance comparisons. To obtain these metrics even in the case of unsupervised learning, i.e., $k-means$, we combine it with SVM, to fall in the classification task and convert the unsupervised results into classification performance metrics.

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

TABLE I: Performance comparison of data center traffic classification for different ML models. Tests are performed over three datasets

	US-UNV-1				US-UNV-2				US-UNV-3			
	Accuracy	*Precision*	*Recall*	*F1-score*	*Accuracy*	*Precision*	*Recall*	*F1-score*	*Accuracy*	*Precision*	*Recall*	*F1-score*
SVM	0.94	0.96	0.94	0.97	0.94	0.96	0.94	0.96	0.99	0.99	0.99	0.99
NN	0.93	0.92	0.99	0.96	0.93	0.93	0.99	0.95	0.99	0.99	0.99	0.99
k-means	0.83	0.99	0.83	0.91	0.84	0.99	0.84	0.91	0.97	0.99	0.98	0.99
RF	0.99	0.99	0.99	0.99	0.93	0.93	0.93	0.93	0.97	0.97	0.97	0.97
D-Tree	0.99	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.98

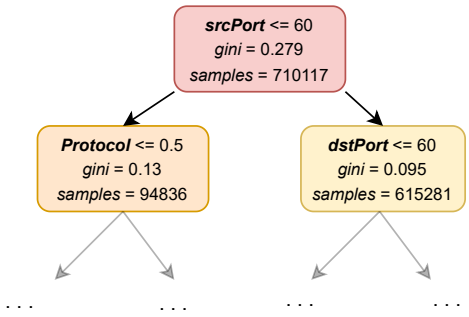


Fig. 4: First levels in the depth of our trained decision tree. Source and destination ports, along with the protocol used, are the principal features used during classification. More in detail, "gini" is a parameter assessing the quality of the decision tree branch, and "samples" is the number of items in the branch.

Despite providing high precision, we can observe how this unsupervised approach performs poorly compared to other supervised alternatives. Moreover, both RF and NN perform well over the three datasets but provide lower accuracy and F1-score than our D-Tree. Focusing on the RF classifier, we can notice good accuracy when tested on the same dataset in which it is trained. However, this model does not perform comparably to the other datasets. On the other hand, our enhanced D-Tree model provides more intriguing performance metrics and more generability: its performance is satisfactory even when applied to other data center workloads.

C. Interpreting classification with a decision tree

One reason behind our choice of D-Tree for traffic classification is the rich expressiveness of such models that allows their interpretation [57]. Interpretation for classification techniques refers to understanding and explaining how the classification model makes predictions. Also, it can make the user understand the factors that are most leading in making a prediction, help identify biases in the model, and explain the model's predictions. There are different ways to interpret a classification model, depending on the type of model and the specific techniques used. Focusing on D-Tree models, they can be interpreted by following the path that an input takes through the tree to arrive at a prediction. We report our D-tree's first levels of depth after having trained it over the training set (80%) of $US-UNV-1$ in Fig. 4. The figure shows that the root node, i.e., the node that starts the tree, splits the tree into two descending branches according to the transport protocol source port. This first branch, along with the second level, is an indicator of the principal features used in the decision. This splitting criterion is done according to the Gini index, a value used to evaluate the quality of a split along a particular attribute. In detail, the Gini index is computed by subtracting 1 at the sum of the squares of the probability of each class belonging to each node, and it is used to determine which attribute to split on at each step in the learning process [58]. A low value of the Gini index means a relatively pure subset of the data, whereas a high value indicates a mix of different classes. The goal of a decision tree is to split the data according to the lowest obtained Gini index, which will lead to a more accurate and interpretable model. The figure also shows the samples that are contained in each node. The root node, for example, contains $710,117$ samples since we considered the 80% of $US-UNV-1$ dataset, which comprises $887,647$ items in total. More in detail, all name fields were encoded into numerical values, as seen in the first level of depth, where the transport protocol (i.e., TCP, UDP) was converted into numerical values (i.e., 0, 1) for being manipulated by the machine learning algorithms. We can observe how the source and destination ports, as well as the protocol, are the major factors that dictate the decision process. This outcome is in line with the rule-based decision as in [18]; however, data-driven learning allows us to adapt and generalize over diverse traffic patterns.

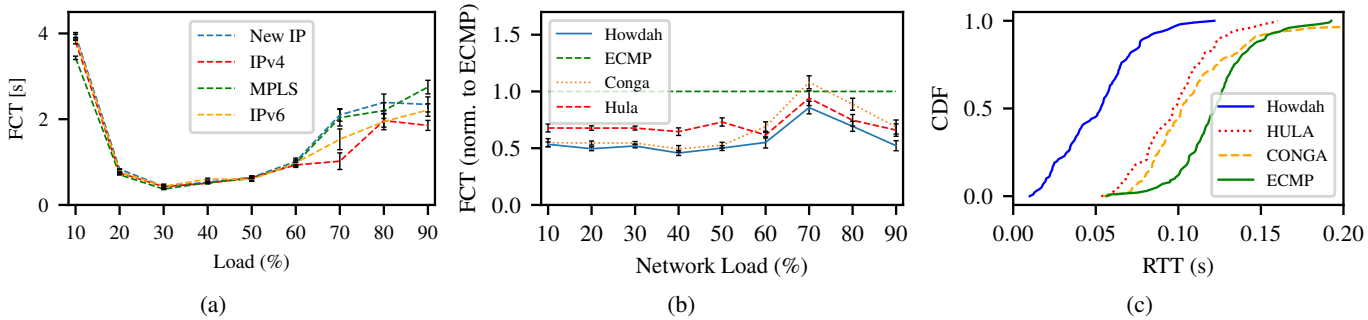
D. Packet Header Impact

As explained in Section III, Howdah can work when combined with multiple protocols responsible for adding extra information directly in the packet header. Among them, in this paper, we specifically focus on IP, New IP, MPLS, and IPv6, although more options are available. In Fig. 5a, we study the diverse header format's impact on flow completion time (FCT) for different network loads. FCT is defined as the time when the first packet is sent until the last one is received and represents a key performance metric when speaking about network congestion [59]. The error bars in the graph refer to the 95% confidence intervals.

The first noticeable advantage of a load profiling technique can be observed when the network load is at 20%. While at 10%, the link congestion is not detected and considered not relevant, for an increment in the traffic, our strategy can effectively split the multiple flows over the available paths. Later, when the congestion is reduced and the traffic load is less than 50%, we can see that the advantages of having no overhead as in IPv4 are minimal and all options show similar FCTs. This suggests that the burden introduced by additional bytes in the packet header is minimal. However, it can be seen

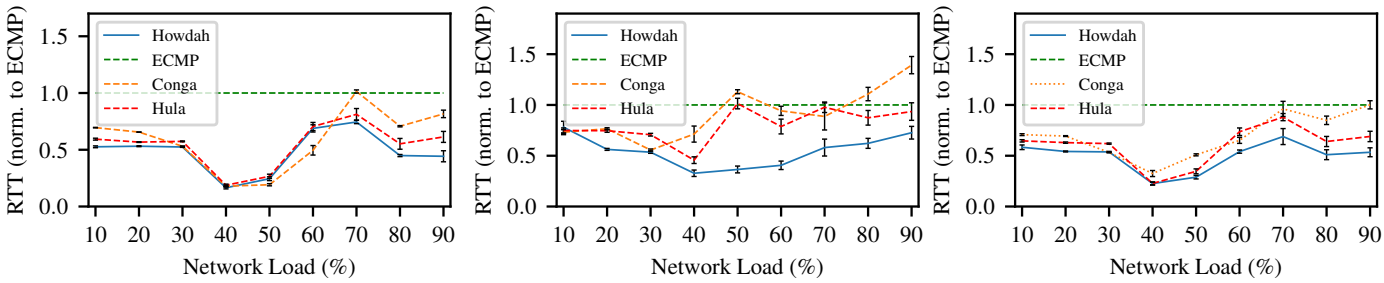
Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729



(a) (b) (c)

Fig. 5: (a) Flow Completion Time (FCT) performance for different packet headers, measuring their impact. (b) FCT comparison for benchmark load profiling solutions. (c) CDF for RTT of benchmark solution when the network load is at 70%.



(a) Elephants (b) Mice (c) Average

Fig. 6: RTT evolution at varying network load, for (a) elephant flows, (b) mouse flows, and (c) on average. Differentiating action per traffic type leads Howdah to attain the lowest RTT overall.

that when the traffic load is high, more than 70%, the benefits of no additional bytes as in IPv4 are evident and result in the lowest FCT. Although all of these alternatives are valid and provide even more flexibility, we use the IP header as the default option in the following tests, given the minimal intervention required on the host side.

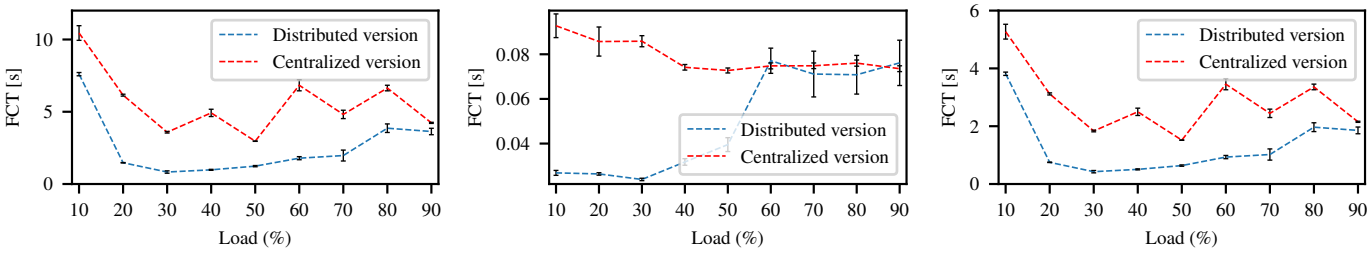
E. Load Profiling Effectiveness

After evaluating our predictive model and the impact of different packet header formats, we studied the load-profiling effectiveness in a data center scenario by comparing Howdah against the other benchmark solutions. The 10 servers in Fig. 3 are used to send packets so that traffic replicates the data center workload described in [12]. This allows us to consider an increasing network load by varying the number of receiving servers (from 1 to 9). First, we compare the FCT obtained by Howdah and the other benchmark solutions for load-profiling, normalizing all values obtained to a baseline algorithm as ECMP. As shown in Fig. 5b, Howdah can stably minimize the FCT for all network loads considered. While HULA performs well at high network load, CONGA provides the best results at low load. Our solution, instead, attains the lowest FCT for any type of traffic in the data center, assuring a less congested network configuration. We then focus on another key metric, the RTT, and consider a specific network load, 70%, to evaluate the RTT's cumulative distribution function (CDF) for sending traffic. By plotting the CDF, we can study the distribution of RTT values with a particular focus on tails. As visible in Fig. 5c, our solution not only diminishes the RTT on average compared to state-of-the-art but also lowers the RTT of the transmission of the most long-lived packets. In particular, all responses are received by 0.12 seconds after the request is sent, representing the minimum among all the alternatives considered.

Moreover, to generalize our findings and study the behavior at different network loads, we also report the RTT evolution in Fig. 6. Our comparison differentiates the "elephant" from the "mouse" flows to better analyze the behavior. Starting from elephant traffic, Fig. 6a shows the RTT normalized to ECMP and demonstrates that the more network loads, the more notable improvements are brought by our solution. Although for a load ranging from 50% to 60%, we can notice CONGA slightly outperforming Howdah, we can also observe how CONGA cannot react to higher loads. If we compare the RTT when sending mice traffic (Fig. 6b), this CONGA's behavior is even more visible and occasionally performs worse than ECMP. On the other hand, Howdah achieves better performance, and the advantages for mouse flows are the most prominent. Averaging the results for the two types of traffic in Fig. 6c, we observe that when the load is low (10% to 40%), the network is not considerably congested, and Howdah, CONGA, and HULA achieve almost the same RTTs. However, when the load increases, Howdah increases its advantage. This enforces what was already shown in FCT behavior and demonstrates how our traffic classification, combined with differentiated actions from switches, enables achieving better results overall.

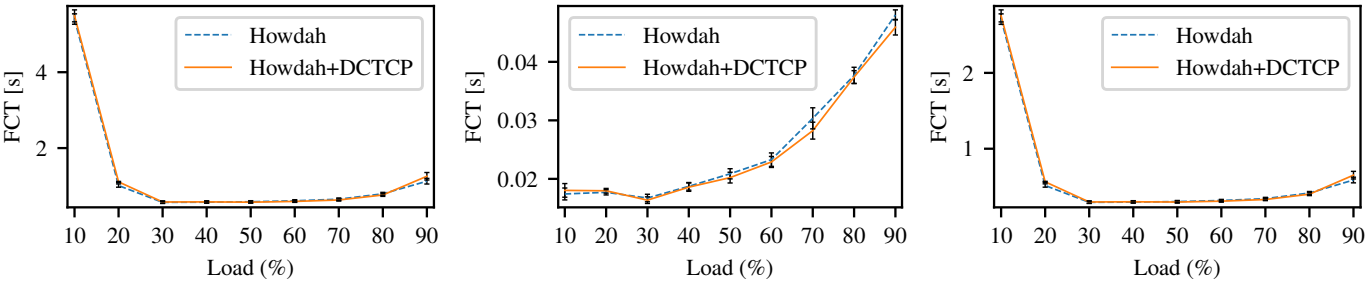
Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729



(a) Elephants (b) Mice (c) Average

Fig. 7: FCT evolution at varying network load when compared to a centralized version in OpenFlow, for (a) elephant flows, (b) mouse flows, and (c) on average.



(a) Elephants (b) Mice (c) Average

Fig. 8: FCT evolution at different network loads when the DCTCP control congestion algorithm coexists compared to a Howdah only implementation, for (a) elephant flows, (b) mouse flows, and (c) on average.

F. Centralized Approach

To validate our distributed schema, we compare it against a centralized approach in which forwarding decisions are taken by an SDN controller. Although a centralized solution comes with a slow control loop due to the controller interaction, it can perform more sophisticated decisions. We developed a centralized version of Howdah where switches are instructed with OpenFlow [60], one of the most known and deployed protocols in the SDN area. In this setting, all the packets that do not have a predefined route will automatically be forwarded to the controller, which handles them according to the rules added in the controller. These rules might depend on many factors, such as source/destination addresses, transport protocol source, flags inside the packet, or even network conditions. Since we compared against different parameters to consider in packet forwarding in previous sections, herein, we study how a centralized version of Howdah would perform. We deployed such a version, simply called Howdah-centralized, in which the SDN controller is implemented in Ryu framework [61] and, just as the distributed one described throughout the paper, runs in the Mininet simulation environment. Since OpenFlow switches cannot support hash-based routing [62], [63], the hash is computed and stored on a controller table. In particular, the controller, running the ML model, classifies the incoming flow. When elephant flows are recognized, it checks on the flowlet hash table if the hash is present. If so, it simply installs the rule on the switch using as matching the IP destination and port destination. If not, it computes the hash, adds a new entry in the table, and then installs the rule on the switch. When the prefixed timeout expires, it removes installed flow routes.

Fig. 7 shows the comparison results between the two Howdah versions: the distributed version, written with P4, and the centralized one, written with OpenFlow. We focus in particular on the type of flows sent (elephants, mice), but also on the average between these two classifications in the same network conditions as before. In particular, we computed the FCT for all flows in terms of seconds at a varying network load and level of network congestion induced with the *iperf3* tool. We can observe how our distributed version performs better than the distributed one in all flow types. If we look at the elephant flows in Fig. 7a, where even when the network is not much congested (from 10% to 50% of network load), the controller interaction in OpenFlow leads to a bigger FCT. When the network starts to get congested, and the load is greater than 50%, we can see that both versions start to perform similarly, and with a congested network (90%), even a distributed solution achieves almost the same FCT as the centralized one. This behavior is even more visible in the mice flows (Fig. 7b) where just as for the elephant flows, when the network is not congested, the interaction with the controller leads to a slower forwarding time and, consequently, a greater FCT. Meanwhile, both versions achieve almost the same FCT when the network becomes more congested. However, if we consider the average FCT of all flows in Fig. 7c, we can conclude that the distributed setting can drastically reduce the FCT and the overhead. This outcome finally validates our approach and the importance of having host-based classification and a data-plane forwarding process.

G. Howdah in Conjunction with CC algorithms

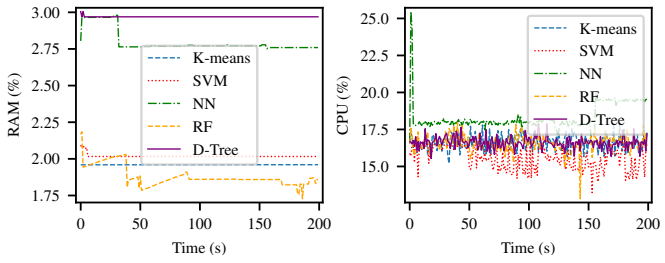
In evaluating Howdah, we also considered how it behaves when an in-network congestion control (CC) algorithm is present, taking for this experiment the well-known data-center TCP (DCTCP) [12]. With DCTCP, switches mark packets'

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

TABLE II: Overhead of the considered ML models both during training (T) and classification (C) in terms of time spent, RAM and CPU consumption, and CO_2 emission. The D-Tree model appears as a lightweight model.

	Tr. time [s]	**Class. time [μs]**	**CO_2 (T) [mg]**	**CO_2 (C) [mg]**	**RAM (T) [%]**	**RAM (C) [%]**	**CPU (T) [%]**	**CPU (C) [%]**
SVM	0.927	0.0264	1.933	0.0417	2.013	2.022	15.554	14.523
NN	170.869	4.081	416.6	1.8391	2.766	2.758	18.328	18.207
k-means	0.647	0.07974	2.013	0.0728	1.959	1.954	16.621	15.583
RF	83.439	10.562	192.4	3.1312	1.872	1.851	16.58	16.983
D-Tree	1.0202	0.107	1.733	0.0481	2.969	2.93	16.619	16.506



(a) (b)

Fig. 9: (a) RAM and (b) CPU consumption of the considered classifiers during the execution at the host side.

Explicit Congestion Notification (ECN) field to notify the sender that there is congestion in the network, i.e., the buffer occupancy of the switch exceeds a fixed small threshold. The sending host reacts by reducing the sending rate, using the fraction of marked packets as a factor: the larger the fraction, the bigger the decrease factor. Fig. 8 compares the FCT evolution of two implementations: one using only Howdah and the other using both Howdah and the DCTCP algorithm under different network loads. In Fig. 8a, we observe that despite the large size of the elephant flows, both implementations achieve the same performance at all network loads. This suggests that the load profiling mechanism that Howdah adopts is effective enough and able to collaborate with other mechanisms. Similarly, in Fig. 8b, both implementations achieve the same FCT for the mouse flows. Although there is a slight difference between 50% and 80% of network load where the solution Howdah + DCTCP performs better than Howdah alone, the difference is only 2.106ms. Overall, as shown in Fig. 8c, both implementations perform similarly on average, demonstrating that Howdah performs as a good profiler both alone and in combination with other mechanisms.

H. Resource Consumption

Finally, we consider the impact of the traffic classifier on the host machines. One of the challenges faced by the design and implementation of Howdah is the efficiency in terms of processing time, especially onboard host machines, which are typically running resource-consuming processes. A lightweight yet accurate classifier is thus essential. To this end, we study the memory and CPU consumption of different ML models during the training learning phase and execution phase, reporting results in Fig. 9. The considered machine consists of a 2.6 GHz 6-core CPU and 16GB RAM. We can observe how, although our D-tree classifier consumes the highest amount of RAM (Fig. 9a), this quantity is negligible and can be found in any device, even on the resource-constrained ones. The CPU consumption (Fig. 9b), however, is similar to other algorithms and less than the Neural Network model. Even when no specific hardware is utilized, the reduced computing resources required by D-Tree validate our design and motivate our assumption to run the learning process on host machines.

In addition to memory footprint and computation resources, we also evaluated the overhead of training and running the ML models (Table II). We report the time, CO_2, RAM, and CPU for both training and classification processes, where the training occurs over 80% of the $US-UNV-1$ dataset and the classification over the remaining 20% of it. The classification time is the time to classify an unknown flow when it must be sent. As shown in Table II, the Decision Tree, similarly to k-means and SVM, requires a limited training time, while neural networks and RF demands more time to converge. The classification time of the D-Tree is also minimal and negligible. This result is extremely important as it validates our hypothesis to run the classification before sending any packet. On the contrary, despite being very accurate, RF is slower in its operations. Concerning the energy efficiency of the models, different studies [64], [65] have proven that training ML models are highly polluting, especially when there are many parameters and massive datasets. For this reason, we estimated the environmental footprint that our code left, in terms of *mg* of CO_2 emissions, measured with the CodeCarbon python library [66]. This library uses two main factors to compute the CO_2 emission: the carbon intensity per kWh of electricity needed for the computation (in gCO2/kWh), and the power consumed by the infrastructure where the code is running (in kWh). The multiplication of these two factors gives the overall carbon emission of the running code. It is important to notice that the computation is made by the library itself, considering values such as different carbon intensities of electricity per country. We can observe from the table how the training time also impacts the carbon emission with supervised models such as NN and RF that, by needing more time to train, also have a more considerable energy impact, emitting more CO_2. On the contrary, our D-tree model is shown to be one of the greenest options for the classification, with similar emissions to SVM. Quite surprisingly, the CO_2 emissions of RF during classification are the highest, even more than deep learning models. This result is possibly due to the complexity raised by the presence of multiple trees and will be analyzed in depth in the future. Lastly, averaging the RAM and CPU consumptions of the models, we have confirmation of results in Fig. 9, where the RAM usage of D-tree is the highest among alternatives but still limited, and the CPU is comparable to benchmarks.

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

VII. Conclusion

This paper presented Howdah, an in-band load profiling solution, whose pillar is the cooperation host-switch: the host classifies sending traffic using a specifically trained ML model, i.e., Decision Tree, and inserts it directly into the packet; the switch, programmed using P4 language, takes packet forwarding decisions based on the information of the flow type and on the status of the network itself. By letting each switch locally decide the best next-hop per packet, our solution assures link failure resistance and the ability to adapt to topology changes. Throughout the paper, we also explored possible protocols that can be used to include in-band information about the ongoing traffic type. Results demonstrate that overall, and especially at high network loads, our solution reduces RTT and FCT more than the state-of-the-art techniques. Moreover, the model chosen introduces little overhead to the system, validating our design of delegating the classification task to the host process. In the future, in order to improve load profiling, traffic classification, and forwarding decisions further, we plan to explore new strategies that use a more fine-grained classification for a multi-class classification that identifies more applications (e.g., video streaming, interactive call) or insert additional data beyond the classification itself.

References

[1] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "Howdah: Load profiling via in-band flow classification and p4," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 46–54.

[2] I. Matta and A. Bestavros, "A load profiling approach to routing guaranteed bandwidth flows," in *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications*, vol. 3, 1998, pp. 1014–1021.

[3] A. V. Ventrella, F. Esposito, and L. A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 80–87.

[4] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, 2013, pp. 151–162.

[5] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan *et al.*, "CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '14)*. New York, NY, USA: ACM, 2014, p. 503–514.

[6] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research (SOSR '16)*. Association for Computing Machinery, 2016, pp. 1–12.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI '10)*, vol. 10, no. 8. USENIX Association, 2010, pp. 89–92.

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

[9] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized" zero-queue" datacenter network," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '14)*. ACM New York, NY, USA, 2014, pp. 307–318.

[10] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '13)*. ACM New York, NY, USA, 2013, pp. 15–26.

[11] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic Load Balancing without Packet Reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, p. 51–62, Mar. 2007.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '10)*. ACM New York, NY, USA, 2010, pp. 63–74.

[13] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies (CoNEXT '13)*, 2013, pp. 49–60.

[14] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro Load Balancing for Low-Latency Data Center Networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. New York, NY, USA: Association for Computing Machinery, 2017, p. 225–238.

[15] 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 (CoNEXT '17)*, 2017, pp. 323–335.

[16] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[17] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, 2013, pp. 399–412.

[18] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM '11)*. IEEE, 2011, pp. 1629–1637.

[19] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the seventh conference on emerging networking experiments and technologies (CoNEXT '11)*, 2011, pp. 1–12.

[20] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding *et al.*, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1345–1358.

[21] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*. USENIX Association, 2020, pp. 701–721.

[22] Z. Li, S. Peng, D. Voyer, C. Li, P. Liu, C. Cao, and G. Mishra, "Application-aware Networking (APN) Framework," Internet Engineering Task Force, Internet-Draft draft-li-apn-framework-06, Sep. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-li-apn-framework/06/

[23] A. Sacco, F. Esposito, G. Marchetto, G. Kolar, and K. Schwetye, "On Edge Computing for Remote Pathology Consultations and Computations," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 9, pp. 2523–2534, 2020.

[24] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 495–511.

[25] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2017.

[26] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[27] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim *et al.*, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM Conference on Data communication (SIGCOMM '09)*. Association for Computing Machinery, 2009, pp. 51–62.

[28] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement (IMC '09)*, 2009, pp. 202–208.

[29] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu, "Predicting inter-data-center network traffic using elephant flow and sublink information,"

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

IEEE Transactions on Network and Service Management, vol. 13, no. 4, pp. 782–792, 2016.

[30] K. B. Nougnanke, Y. Labit, and M. Bruyere, "ML-based Incast Performance Optimization in Software-Defined Data Centers," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–6.

[31] A. Chhabra and M. Kiran, "Classifying elephant and mice flows in high-speed scientific networks," *Proc. of the International Workshop on Innovating the Network for Data Intensive Science (INDIS '17)*, pp. 1–8, 2017.

[32] C. S. Inc., "Cisco global cloud index: Forecast and methodology, 2016–2021," 2021. [Online]. Available: https://virtualization.network/Resources/Whitepapers/0b75cf2e-0c53-4891-918e-b542a5d364c5_white-paper-c11-738085.pdf

[33] A. Sacco, F. Esposito, and G. Marchetto, "RoPE: An Architecture for Adaptive Data-Driven Routing Prediction at the Edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 986–999, 2020.

[34] N. Farrington, G. Porter, S. Radhakrishnan *et al.*, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM Conference on Data communication (SIGCOMM '10)*. Association for Computing Machinery, 2010, pp. 339–350.

[35] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Supporting Sustainable Virtual Network Mutations with Mystique," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2714–2727, 2021.

[36] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04)*. Association for Computing Machinery, 2004, pp. 115–120.

[37] S. Gebert, S. Geissler, T. Zinner, A. Nguyen-Ngoc, S. Lange, and P. Tran-Gia, "Zoom: Lightweight sdn-based elephant detection," in *28th International Teletraffic Congress (ITC 28)*, vol. 2. IEEE, 2016, pp. 1–6.

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

[39] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4− >NetFPGA Workflow for Line-Rate Packet Processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 1–9.

[40] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, 2015, pp. 1–6.

[41] I. Matta, A. Bestavros, and M. Krunz, "Load profiling based routing for guaranteed bandwidth flows," *European Transactions on Telecommunications*, vol. 10, no. 2, pp. 165–181, 1999.

[42] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic In-Band Network Telemetry," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. ACM New York, NY, USA, 2020, pp. 662–680.

[43] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Partially Oblivious Congestion Control for the Internet via Reinforcement Learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1644–1659, 2022.

[44] M. Baldi, G. Marchetto, and Y. Ofek, "A Scalable Solution for Engineering Streaming Traffic in the Future Internet," *Computer Networks*, vol. 51, no. 14, pp. 4092 –4111, 2007.

[45] S. Bryant and A. Clemm, "Token cell routing: A new sub-ip layer protocol," in *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 153–159.

[46] R. Li, K. Makhijani, and L. Dong, "New IP: A Data Packet Framework to Evolve the Internet," in *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2020, pp. 1–8.

[47] R. Li, A. Clemm, U. Chunduri, L. Dong, and K. Makhijani, "A New Framework and Protocol for Future Networking Applications," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, ser. NEAT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 21–26.

[48] Ipv6 statistics. Accessed: 2023-1-7. [Online]. Available: https://www.google.com/intl/it/ipv6/statistics.html

[49] Ipv6 usage. Accessed: 2023-1-7. [Online]. Available: http://v6asns.ripe.net/v/6?s=+ALL

[50] S. Deering and R. Hinden, "RFC8200: Internet protocol, version 6 (IPv6) specification," 2017.

[51] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 segment routing header (SRH): RFC8754," 2020.

[52] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[53] M. Bramer, "Avoiding overfitting of decision trees," *Principles of data mining*, pp. 119–134, 2007.

[54] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-Agnostic flow scheduling for commodity data centers," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*. USENIX Association, 2015, pp. 455–468.

[55] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[56] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*, 2010, pp. 267–280.

[57] H. Blockeel and L. De Raedt, "Top-down induction of first-order logical decision trees," *Artificial intelligence*, vol. 101, no. 1-2, pp. 285–297, 1998.

[58] S. Tangirala, "Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 2, pp. 612–619, 2020.

[59] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[60] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[61] Ryu controller. Accessed: 2022-12-7. [Online]. Available: https://ryu-sdn.org/

[62] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based server load balancing gone wild," in *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 11)*, 2011.

[63] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble routing for datacenter networks," in *2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2010, pp. 1–12.

[64] J. Cowls, A. Tsamados, M. Taddeo, and L. Floridi, "The ai gambit: leveraging artificial intelligence to combat climate change—opportunities, challenges, and recommendations," *Ai & Society*, pp. 1–25, 2021.

[65] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, 2020.

[66] V. Schmidt, K. Goyal, A. Joshi, B. Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, and S. Luccioni, "Codecarbon: estimate and track carbon emissions from machine learning computing," 2021.

Antonino Angi received his M.Sc. degree in Computer Engineering (major in Data Science) from Politecnico di Torino, Italy in 2020, and he is currently enrolled in a Ph.D. program at the same university. His research interests include protocols for network architecture and management; applying Natural Language Processing (NLP) and Machine Learning algorithms to Software Defined Networks (SDN) and Intent-based Networks (IBN), used in conjunction with data-plane programming languages.

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3299729

Alessio Sacco is an Assistant Professor at Politecnico di Torino. He received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2018 and 2022, respectively. His research interests include architecture and protocols for network management; implementation and design of cloud computing applications; algorithms and protocols for service-based architecture, such as Software Defined Networks (SDN), used in conjunction with Machine Learning algorithms.

Flavio Esposito is an Associate Professor with the Department of Computer Science at Saint Louis University (SLU). He received a M.Sc. degree in Telecommunication Engineering from the University of Florence, Italy, and a Ph.D. in computer science from Boston University in 2013. Flavio's main research interests include network management, network virtualization, and distributed systems. Flavio is the recipient of several awards, including several National Science Foundation awards and the Comcast Innovation Award in 2021.

Guido Marchetto received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently an Associate Professor with the Department of Control and Computer Engineering. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.

Alexander Clemm is a Distinguished Engineer in Futurewei's Future Networks and Innovation Group in Santa Clara, California. He has been involved in networking software and management technology throughout his career, most recently in the areas of high-precision networks and future networking services. He has served on the Organizing Committees of many management and network softwarization conferences. He has around 50 publications, 50 issued patents, and several books and RFCs. He holds an M.S. in computer science from Stanford University and a Ph.D. from the University of Munich, Germany.

Authorized licensed use limited to: Politecnico di Torino. Downloaded on August 01,2023 at 07:42:11 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.