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

Antonino Angi * Alessio Sacco * Flavio Esposito ‡ Guido Marchetto * Alexander Clemm †
* Department of Control and Computer Engineering, Politecnico di Torino, Italy
‡ Computer Science Department, Saint Louis University, USA
† Futurewei Inc., USA

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

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

I. Introduction

In the last decades, datacenters have changed their topology where the demand of the global network has increased especially when new applications ask for more data at faster speeds but still requiring low latency. Because of that, a new focus was given not only to traffic in and out of the datacenter, but even within it, with the need of changing datacenter's architecture to multi-rooted leaf-spine or fat-tree topologies. These topologies have in common the presence of multiple source-destination paths to handle the high traffic volume. However, efficiently using the network resources requires a load-balancing strategy that moves network performance towards optimality. One of the most used load-balancing techniques remains ECMP (Equal-Cost Multi-Path), a routing strategy that static hashes flows for path assignment. However, because flows are randomly assigned to a path, ECMP does not take into account potential congestion in the network (as well as link failures) and leads to uneven flows distribution (as well as poor performance) [1]. Recent approaches attempted to overcome these limitations and, while they are all sound solutions, they either introduced additional overhead, e.g., [2], [3] or failed to apply efficient logic per-packet, e.g., [4], [5]. On the one hand, centralized schemes, such as Hedera [4], B4 [5], FastPass [6] and SWAN [7], can perform congestion-aware decisions, but demand considerable control traffic and react too slowly for volatile (datacenter) traffic. On the other hand, recent distributed approaches, such as CONGA [2] and HULA [3], introduce periodic network feedback that might lead to excessive overhead traffic and contribute to congestion. In line with these efforts, the research question that we are addressing in this paper is: *"Can we balance network traffic over uncongested paths without the need of defining elaborate protocols for the exchange of information between the switches or between the switches and a centralized controller?"*

In this paper, we answer this question with *Howdah*, a data-plane solution that aims at improving 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] Load profiling [8], [9] subsumes load balancing: it may be desirable to have different classes of traffic with different priorities and demands, hence it may be desirable to depart from a merely balanced load. In Howdah, network switches are instructed with P4 programs to run a data-driven load profiling that, rather than flows, operates over flowlets — burst of packets in a flow, split by a sufficiently large time gap. Approaches based on flowlets have been shown to be desirable as there are no packet reordering problems and no modifications to the TCP stack needed [10].

To further optimize path selection, forwarding actions are differentiated according to the type of traffic carried in the packets. Howdah internal sending hosts 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. Based on the fact that elephant flows are the ones 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, just flow via flowlet ECMP.

We evaluated our classifier's accuracy over real-world dat-

[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 elephant or occasionally on some other animal such as 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 elephants (or other) flows.

978-3-903176-51-5 © 2022 IFIP

acenter traffic traces and experienced how a Support Vector Machine (SVM) algorithm is simple yet effective enough to split traffic into classes accurately. Then, we studied how Howdah's performance changes when different protocols are used to carry the in-band traffic information. As discussed in Section IV, our mechanism can operate as an in-band strategy on different existing protocols, e.g., MPLS, IPv4, IPv6, or even on future Internet architectures [11]. We found how IP type-of-service fields provide negligible overhead and represent a valid implementation of our proposed architecture in transferring information on traffic classes. We tested such implementation in datacenter 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) at high network loads, especially for elephant flows.

The rest of the paper is structured as follows. Section II describes state-of-the-art techniques, while Section III outlines the scenario considered and overviews our solution design. In Section IV we study some options to carry the in-band information using already defined protocols, and Section V describes our traffic classification method. We present our results in section VI and conclude the paper 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 datacenters. Many recent studies addressed this problem, proposing solutions that fully utilize the available bandwidth resources. Although traditional and local routing strategies (e.g., the standard ECMP) are extensively used in practice, their performance is suboptimal for datacenters, given the local, trivial, and stateless decisions that lead to split traffic without knowledge of potential congestion on the network [4], [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, ignoring global information. DRILL forwarding decisions are load-aware and based on local queue occupancy, enabling operating on microsecond (packet-by-packet) timescales. PRESTO [16] is based on the insight that, in a symmetric Clos where all flows are small, ECMP provides near-optimal load balance. As such, it divides flows into "mice", that are source-routed so they are striped across all paths, without load-awareness. However, both solutions have to deal with the performance impact and computational bottleneck of TCP reordering, whose problem is exacerbated in asymmetric topologies.

A common approach to taking more appropriate actions is to delegate forwarding logic to centralized controllers and take congestion-aware decisions, as in as B4 [5], F10 [17], Mahout [18], MicroTE [19], and Hedera [4], which are based on the assumption that non-local (i.e., global) congestion information is helpful to evenly balance load. While they have shown near-optimal traffic engineering for inter-datacenter WANs, they are not designed for highly volatile datacenter networks because of the coarse timescales of their control operations.

To reach microseconds performance yet using global information, CONGA [2] operates in the data plane and makes globally optimal allocations using a distributed approach, allowing a faster reaction when handling asymmetry. 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 real-time congestion on fabric paths, CONGA combines this mechanism with the flowlet switching strategy. Although this study validates the efficacy of flowlet switching strategy to ensure efficient utilization of network resources, especially if applied to datacenters, it comes with two main limitations: first, the global congestion state at the edge switches may drastically increase and exceed the switch memory; second, its implementation is designed for custom hardware. These limitations are addressed explicitly by HULA [3], a data-plane load balancing algorithm applied on P4-based programmable switches, in which leaf switches track congestion for the best path to a destination through a neighboring switch and not for all paths, with no need to have specifically designed hardware. In particular, HULA uses probes to get information on the network status (i.e., link failure, topology change) and update switches' internal tables. Our solution uses the same load profiling principles as HULA. However, instead of 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, leading to a positive impact on performance.

A recent solution as CONTRA [20] 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 adapting forwarding decisions, we also provide some readily-available load profiling actions that can be extended and customized by the users to meet specific performance objectives. However, different from these load profiling solutions, in our local congestion-aware routing solution, the 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 forwarding decisions of the switches in the network.

III. Architecture and Protocol Design

Howdah is constituted by a data-plane load balancing system that uses a traffic classifier to differentiate forwarding actions properly. Although the traffic classifier can be deployed at the ingress of the network provider or at the sender, we simply refer to this element as *Howdah host*. As such, we can summarize the overall Howdah algorithm in a two-step process and two main architectural components: one running

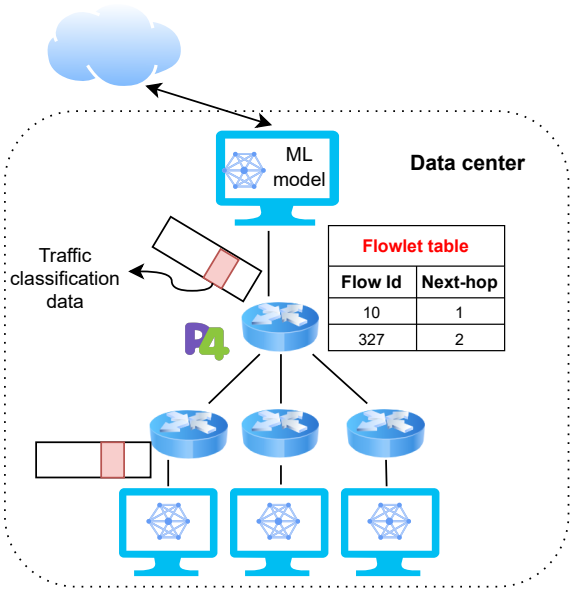


Fig. 1: Howdah overview. The system is based on the cooperation of hosts that help network nodes by inserting information about traffic classification, then processed by the P4 switches.

on local machines and gateways, and one running on P4-enabled switches, as shown in Fig. 1. At first, the host classifies the traffic using a supervised machine learning algorithm and injects the output of the classification process in an opportune field of the packet header (see Section IV). Secondly, when an intermediate switch receives the flow, it checks the type of traffic that is being sent and differentiates the actions. While mouse flows are forwarded only following the information in the packet header without the need to update each switch's statistics, this is not true for elephants, given their major impact.

In the rest of this section, we detail our design decisions, focusing on how functionalities are split between end-hosts and P4-enabled switches.

A. Host-based Traffic Classification within Howdah Hosts

While switches implement load profiling and traffic engineering decisions, senders and gateways traditionally house the traffic classification logic used during the forwarding process of the switches. Since our network scenario is constituted by a datacenter topology, we assumed that the sending hosts for East-West (internal) traffic can be easily instructed to classify the traffic via an ML model and insert this information in the packet itself (see the details of our design in Section V). Conversely, for the North-South traffic (from/to outside), we assume that external hosts may not implement any ML classification logic. Our gateway also applies the same traffic classification algorithm before letting packets in the datacenter network, along with other packet filtering operations that are common in a datacenter. For traffic directed outside, the classification data is stripped away before leaving the datacenter.

Why traffic classification. Datacenters typically encounter a variety of traffic classes hosting diverse services. Among them, we can enumerate on-demand video delivery, storage and file sharing, web search, social networks, cloud computing, financial services, recommendation systems, and interactive online tools [21], [22]. These applications present different traffic characteristics and distribution of flow arrivals, flow sizes, and flow duration [23]. As an example, flows generated by web search queries are usually much smaller and shorter than flows of batch computing jobs. Instead, high-performance computing (HPC) jobs like Hadoop, transfer petabytes of data during the shuffle MapReduce phase [24].

Such a variety of applications leads to the creation of long-lived connections, as well as short microbursts on the same network. As common in the network management literature [18], [25], we refer to long-lived flows as "elephants", and short microbursts as "mice". The goal of our load profiling solution is *to provide high bisection bandwidth for throughput-sensitive and latency-sensitive flows without introducing excessive delay on remaining flows, by properly balancing traffic loads among the available links.* In line with recent studies [26], [27] that have pointed out the importance of classifying traffic in "elephants" and "mice", we also argue that long-lived flows must be identified to take appropriate actions and better orchestrate traffic.

Moreover, while in this paper we only consider two different types, the Howdah architecture, along with the P4 language, provides flexibility to generalize on multiple differentiation traffic types, e.g., bandwidth $vs.$ delay-sensitive applications, or web $vs.$ database $vs.$ HPC traffic.

Why host-based classification. A possible place where classifying packets would be the switch itself. However, the hardware characteristics of network nodes badly fit the learning procedure of an ML model, resulting in poor performance. Besides, given the strict packet scheduling of datacenter switches, the application of ML models would either negatively affect the packet forwarding process or necessitate a specific software and hardware design. To guarantee fast packet forwarding, the literature has presented valuable examples of switches collecting flow metrics but delegating the ML learning phase to a centralized controller [4], [5], [28]. However, both a per-flow statistic and a sampling mechanism do not scale: The bandwidth between switches and the controller is limited, so transferring statistics becomes the bottleneck in this traffic management scheme. Moreover, collecting statistics per flow would consume significant switch resources, while sampling detection, i.e., sampling only a small fraction of incoming packets, would lead to accurate detection of elephant flows only after 10K packets [29]. In light of this, we argue that the host and the gateway are the optimal places for detecting elephant flows in datacenters. First, they have better visibility over the frequency and amount of application data generated, while network nodes can be biased by network congestion. Second, the application layer of datacenter programs can be augmented with our Howdah layer, and this option is favored by the single administrative domain and software uniformity of

common datacenters. Third, there are likely GPUs or general-purpose CPUs on the hosts that better fit the ML classification process.

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 – without the need for reordering when those arrive at the destination [10]. It has been shown how this method allows higher granularity while providing better performance [2].

To make our switches programmable and make them control plane independent, we instruct them with P4, a programming language for protocol-independent packet processes [30]. Such a language enables to program packet processing pipelines in packet forwarding ASICs and allows defining 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 provides many advantages compared to 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. Moreover, P4 offers a switch abstraction that is independent of the actual hardware: P4 programs are compiled to a target-independent representation (front-end), and then compiled again to different specific platforms, e.g., NetFPGA.

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 of switches can be used to specify behavior 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 on the basis of this information and the port utilization. Concretely, we make use of a table that contains the hash of the arriving flowlet, helpful to record the last time a flowlet belonging to a certain flow was seen, and compute the difference between the stored value and the arriving time of a new flow. If such a difference is below T, whose value is chosen according to other state-of-the-art techniques [3], then the switch forwards the flowlet to the stored best-hop; otherwise, the switch recognizes a new flowlet, computes the hash of the 5-tuple composed of the protocol, IP source & destination address, TCP source & destination port, and finally stores the current best next-hop. We recall the concept of *load profile* as the desired load on an outgoing link of the switch, enabling the user to specify how to split traffic over these links [8]. One common scenario is an even load on the links of the switch, but other circumstances may demand unequal balance if links have different features or traffic have different priority. Our P4-enabled switches can be effortlessly adapted to implement the desired policy.

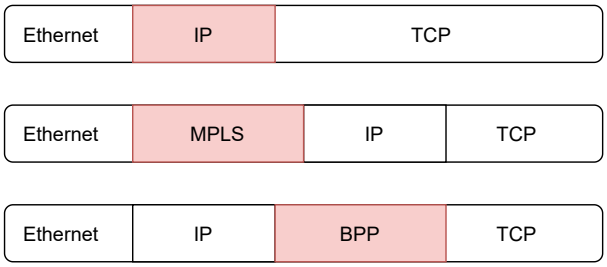


Fig. 2: Possible policies for Howdah's packet header. Traffic classification information is inserted directly into packet, leading to a small impact on the switch forwarding process.

In the special case of load balancing, forwarding rules are applied on top a flowlet-based version of ECMP packet forwarding: like traditional ECMP is based on selecting next-hop hashing the 5-tuple, but rather than per-flow, decisions are per-flowlet. In the case of mice, the switch simply forwards the packet to the next best hop according to the flowlet-based version of ECMP. Otherwise, in the case of elephant traffic, the selection of the next hop also considers the least recently used (LRU) port. As they are more prone to cause congestion in the network, we consider the frequency of port utilization, and we also need to update the statistics about the network in each switch. As such, aside from the hash function computation, the switch needs to update this utilization metric for each arriving packet. Despite being simple, this LRU criterion effectively avoids congestion– and reduces delay – as the flowlet is sent throughout different ports where the probability of sharing the bandwidth with other ongoing flows is reduced.

IV. In-Band Traffic Knowledge Policy

Recent studies have pointed out that additional network information can reach a considerable amount of bytes and become some of the heaviest packets in the network [31]. A significant countermeasure is provided by the In-band network measurement, which is becoming highly used in a variety of network management applications to add network information directly into each packet's data.

For this reason, we use in-band network management in our solution and configure the switch to forward the packet to the next hop, 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. 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 the viability of this architecture, and to demonstrate that the network programming framework can indeed be used to support applications with real-time networking demands without needing to deploy custom hardware in networking devices or even controllers. In particular, 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 has been designed as the way to specify a datagram's priority and request a route for low-latency, high-throughput, or highly-reliable service. These 8-bits have been split to serve the function of Differentiated Services Code Point (DSCP), with 6 bits, and Explicit Congestion Notification (ECN), with 2 bits. Although the behavior of the router in response to these values is not specifically defined, IP ToS definitions are widely found in Unix implementations. For this reason, they appear as 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 scalability and generability of the solution, which is unable to accommodate a broader range of application requirements and switch's actions specification.

MPLS. Multiprotocol Label Switching (MPLS) is a routing technique based on the key idea that packet-forwarding decisions are made solely on the contents of labels assigned to data packets, without the need to examine the packet itself. MPLS works by prefixing packets with an MPLS header, containing one or more labels and forming a label stack. Each entry in the label stack contains four fields; among them comes the 3-bit for Traffic Class field, typically used for QoS. An MPLS-compliant version of Howdah would use this field to carry the information about the traffic flow. Possibly, paths per flow are reserved in advance by means of the Label Distribution Protocol (LDP). This approach provides remarkable flexibility, with more thorough traffic engineering decisions, at the cost of an additional packet header, and an additional protocol, such as LDP, for label distribution, or administrator effort to set up the paths on each network device.

BPP. Big Packet Protocol (BPP) is a novel approach to customize packet-based networking behavior, based on the introduction of the concept of a BPP Collateral: a block of data (metadata and forwarding instructions) carried with the header and the user payload, used to inject meta-information and guide intermediate switches on how to process those packets [11]. Such BPP Collateral can be viewed as three smaller blocks: BPP header, BPP command block, and BPP metadata block. While the command and the metadata blocks are optional, the header is essential because it contains information on the BPP version, block length, some error handling, and the "next header" or "next BPP block" field, which links BPP blocks to each other. The command block is used for condition type purposes or to evaluate some true/false statements; the metadata block, as the name already suggests, might be used to gather more information on the flow type or to perform counting on hops or other values of interest. In light of this, we envision our solution to work optimally when the traffic knowledge is included in the metadata field. We can, however, easily observe how this option implies a larger amount of additional bytes, nearly 30 bits, but can be easily extended to support precise service level guarantees and to facilitate other traffic engineering operations.

Howdah essentials. Given the building blocks of Howdah exposed previously, we envisioned our solution to work with multiple protocols carrying the traffic information. In this paper we limit our attention to some options that can be used to inform the switches about the traffic type, as shown in Fig. 2. However, we argue that other possible protocols, e.g., IPv6 or VXLAN, can be employed given the programmability of our P4 switches. Whatever the protocol carrying the classification output, our solution involves a cooperation hosts-switches towards optimized forwarding decisions: when the packet is ready to be sent, the host adds a flow type bit useful to the switches to distinguish between an elephant or a mouse flow and react accordingly. The Howdah's header is used by the switch to distinguish between an elephant flow and a mouse one by using a bit in this block: "0" if mouse, "1" if elephant. In conclusion, this header field is used to inject meta-information directly into the packet allowing guidance through the network, where our Howdah switches, based on this value, apply load balancing at the granularity of flowlets.

V. Howdah Traffic Classification

One key aspect to consider in our system is traffic classification, as it impacts how packets are forwarded. In this section, we describe the process running on the host machines responsible for accomplishing the classification task.

A. SVM with SGD

In Howdah we model each flow using a Support Vector Machine (SVM) algorithm, a supervised machine learning algorithm typically used for classification and regression purposes. Its goal is to classify and label the data by finding a hyper-plane that better divides items in a dataset and maximizes the distance between the support vectors. This hyper-plane can be of two types: a linear and an N-dimension one. A linear hyper-plane splits the space (and consequently the dataset) into two categories (or classes), while the N-dimension one splits the space in N-dimensions (and classes). SVM is highly used for big datasets, especially for document classification or sentiment analysis. In the prototype presented in this paper, we make use of a linear hyper-plane because we need to classify our data using only two labels: elephants and mice flows.

The considerable dimension of our dataset, however, can cause issues during the learning process. The literature has presented alternative gradient descent (GD) techniques to solve this issue, where three of the most common optimization strategies are:

- Batch gradient descent (BGD), defined as a GD that at each iteration takes the whole training set and computes an average of all gradients;
- Stochastic gradient descent (SGD), defined as a GD that deals with randomness in the training dataset using only a simple sample at each iteration;
- Mini-batch gradient descent (MBGD), a mixture between BGD and SGD where the training set is divided into many groups. However, it needs to know the size of each group ("mini-batch size") as an additional variable for the algorithm.

In our Howdah classifier, we make use of stochastic gradient descent (*SGD*) since it can efficiently deal with a large dataset like ours while also reducing the computation cost efficiently.

B. Howdah Classifier Methodology

Using an SVM algorithm in combination with SGD (referred to as SVM in the following), our Howdah's hosts classify the type of traffic before transmission and tag the packet accordingly. 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, the features list of our ML model consists of 5-tuple: source IP address, destination IP address, source port number, destination port number, and transport protocol (such as TCP or UDP). The output of this classification process is a binary label indicating whether it is an "elephant" or "mouse".

Any host in our datacenter, as well as the gateway, should run a modified instance of either kernel-level network services or application-level socket instances. While the literature has shown profitable usage of a shim layer on the end hosts [18], [23], in our prototype we considered the latter option and our results validate the efficacy VI. To further simplify the operations over the host machines, we apply the classification process only if necessary. In detail, protocols known to bring little contribution 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), the Howdah classifier runs before the sending, and the output label is set in the packet header. Our SVM classifier enables an accurate traffic classification while not incurring an excessive burden for the host machine, as detailed in Section VI. 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.

VI. Evaluation

In this section, we illustrate our evaluation results that guided our solution design and validated Howdah's benefits.

A. Evaluation Settings

To validate our solution over a datacenter-like network, we deploy Howdah over Mininet, a network emulator that allows reproducing arbitrary virtual networks for fast simulations. Being specifically designed for software-defined networking (SDN), it can also support P4-compatible switches via behavioral model version 2 (bmv2), which allows compiling a P4 program into packet-processing actions of C++11 software switches. We limit our attention to a load balancing problem. We replicate in Mininet a leaf-spine topology with 10 server racks connected to their related switches and each of these connected to other 4 switches, as shown in Fig. 3; where every link has 100 Mbps bandwidth. We use the $iperf3$ tool to reproduce different traffic workloads and to induce congestion in the network.

We then tested the traffic classifiers when the input is three realistic workloads, taken from publicly available

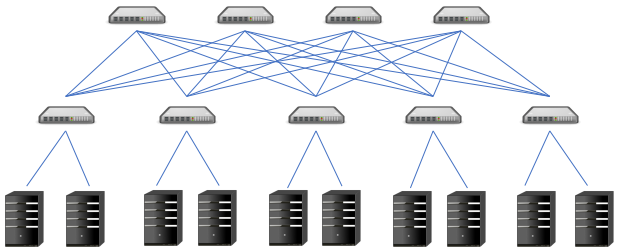


Fig. 3: Network topology used throughout the experimental evaluation.

datasets [32]. We extracted three different datasets and stored them in a .pcap file, corresponding to three captures obtained during the same day in the same datacenter but at diverse time instants. We refer to them as: "US-UNV-1" composed of $887,647$ items, "US-UNV-2" composed of $913,026$ items, and "US-UNV-3" composed of $887,647$ 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 [3], 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.

Traffic classifier benchmarks. We compare our Howdah classifier against three recent solutions and a well-known (and widely used) model. First, a Random Forest (RF) model-based technique as in [26], where the solution of this study classifies flows with the goal of optimizing the incast completion time on different buffered switches using elephant-based traffic. Second, [27], investigates 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". Thirdly, given its popularity, we consider a Neural Network (NN) classifier, whose layout is made with three fully connected layers: The first two hidden layers consist of 12 and 8 nodes and use the rectified linear unit activation function; the third layer, the output layer, has one node and uses the sigmoid activation function. The number of layers and nodes for our classifier was tried in different experiments to find the one that maximizes the metric measures of our classification problem.

Load balancer benchmarks. We compare our approach against two of the most recent load balancer systems: CONGA [2] and HULA [3]. Note that even a more recent solution, CONTRA [20], 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. Moreover, we are not hardware-specific thanks to the P4 language, nor do we need any centralized controller. Finally, ECMP is used as a baseline.

TABLE I: Performance comparison of datacenter traffic classification for different ML models. Tests are performed over three datasets, and SVM shows acceptable accuracy and generability.

	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

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). Considering two classes – elephants and mice– the definition of positive and negative classes for a binary classifier is simplified. Then, we focus on the most relevant ML measures: accuracy, precision, recall, and f1-score, according to the definitions: (i) Accuracy: the fraction of the test 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 is a way of combining 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 measured the classification performance over the remaining samples in this dataset and over the other two files. We report the obtained results in Table I. 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.

We can observe how, despite providing high precision, this unsupervised approach poorly performs in comparison to other supervised alternatives. Moreover, both RF and NN perform well over the three datasets but provide lower accuracy and F1-score than our SVM. Focusing on the RF classifier, we can notice a good accuracy when tested on the same dataset in which it is trained. However, this model does not give comparable performance when applied to the other datasets. On the other hand, our enhanced SVM model provides overall more intriguing performance metrics, along with more generability: its performance is satisfactory even when applied to other datacenter workloads.

C. Packet Header Impact

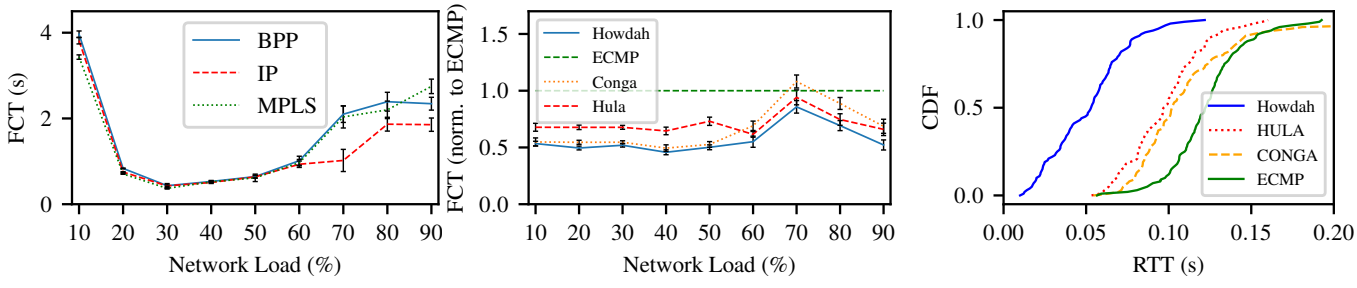
As explained in Section III, Howdah can work when combined with multiple protocols responsible for adding extra information directly in the packet header. Among them, in this paper, we specifically focus on IP, BPP, and MPLS, although more options are available. In Fig. 4a, we study the impact of the diverse header format in terms of 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. The error bars in the graph refer to the 95% confidence intervals.

Since using the IP fields would require no additional bytes and zero overhead, it can be seen that when the traffic load increases, the benefits of this option are evident, leading to the lowest FCT. However, when the congestion is minimal and traffic load is less than 50%, the advantages of IP are minimal too. This motivates us to use the IP header as the default option in the following tests, even though the alternatives are valid and provide more flexibility.

D. Load Balancing Effectiveness

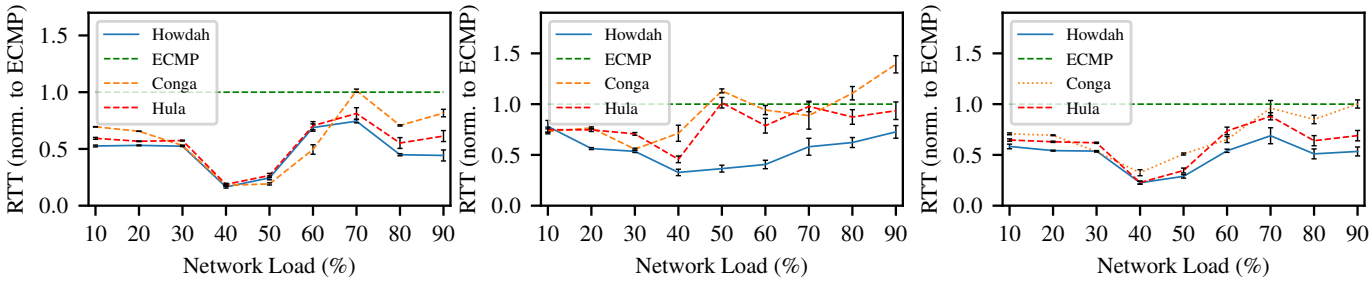
After evaluating our predictive model and the impact of different packet header formats, we studied the load-balancing effectiveness in a datacenter scenario by comparing Howdah against the other benchmark solutions. The 10 servers in Fig. 3 are used to send packets so that traffic replicates the datacenter 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-balancing, normalizing all values obtained to a baseline algorithm as ECMP. As shown in Fig. 4b, 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 datacenter, assuring a less congested network configuration. We then move our attention to another key metric, the RTT, and we consider a specific network load, 70%, evaluating the cumulative distribution function (CDF) of the RTT 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. 4c, 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. 5. Our comparison differentiates the "elephant" from



(a) (b) (c)

Fig. 4: (a) Flow Completion Time (FCT) performance for different packet headers, measuring their impact. (b) FCT comparison for benchmark load balancing solutions. (c) CDF for RTT of benchmark solution when the network load is at 70%.



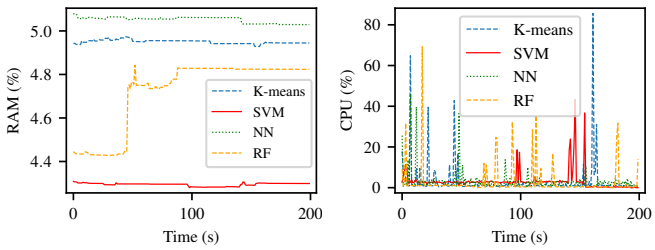
(a) Elephants (b) Mice (c) Average

Fig. 5: 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.

"mouse" flows to better analyze the behavior. Starting from elephant traffic, Fig. 5a 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 is unable to react to higher loads. If then we compare the RTT when sending mice traffic (Fig. 5b), it is even more visible this Conga's behavior, 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. 5c, we observe that when the load is low (10% to 40%) the network is not considerable 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.

E. 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 on board host machines, which are typically running resource-consuming processes. A lightweight yet accurate classifier is thus essential. To this end, we study the consumption of memory and CPU of different ML models during the training learning phase and report



(a) (b)

Fig. 6: (a) RAM and (b) CPU consumption of the considered classifiers during the training process. Our SVM model allows the lowest memory occupancy and a limited processing requirement.

results in Fig. 6. We can observe how our SVM classifier consumes the lowest amount of RAM (Fig. 6a), while also limiting the consumption of CPU (Fig. 6b). The reduced memory footprint required by SVM, even when no specific hardware is utilized, validates our design and motivates our assumption to run the learning process on host machines.

VII. Conclusion

In this paper we presented Howdah, an in-band load balancing technique for programmable switches, whose pillar is the cooperation host-switch: the host classifies sending traffic using a specifically trained ML model, i.e., SVM, and inserts it directly into the packet; the switch 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 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.

ACKNOWLEDGMENT

This work has been partially supported by NSF awards #1836906 and #1908574.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[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] 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.

[21] 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.

[22] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy, "A formal model of network policy analysis," in *Proc. of the 1st IEEE International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE, 2015, pp. 516–522.

[23] 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.

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

[25] 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," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 782–792, 2016.

[26] 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.

[27] A. Chhabra and M. Kiran, "Classifying elephant and mice flows in high-speed scientific networks," *Proc. INDIS*, pp. 1–8, 2017.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.