OFLoad: An OpenFlow-based Dynamic Load Balancing Strategy for Datacenter Networks

Ramona Trestian, *Member, IEEE,* Kostas Katrinis, and Gabriel-Miro Muntean, *Senior Member, IEEE*

***Abstract*—The latest tremendous growth in the Internet traffic has determined the entry into a new era of mega-data-centers, meant to deal with this explosion of data traffic. However this big data with its dynamically changing traffic patterns and flows might result in degradations of the application performance and eventually affecting the network operators' revenue. In this context there is a need for an intelligent and efficient network management system that makes the best use of the available bisection bandwidth abundance to achieve high utilization and performance. This paper proposes OFLoad, an OpenFlow-based dynamic load balancing strategy for datacenter networks that enables the efficient use of the network resources capacity. A real experimental prototype is built and the proposed solution is compared against other solutions from the literature in terms of load-balancing. The aim of OFLoad is to enable the instant configuration of the network by making the best use of the available resources at the lowest cost and complexity.**

***Keywords*—Load Balancing, OpenFlow, Software-defined Networks, Datacenter Networks.**

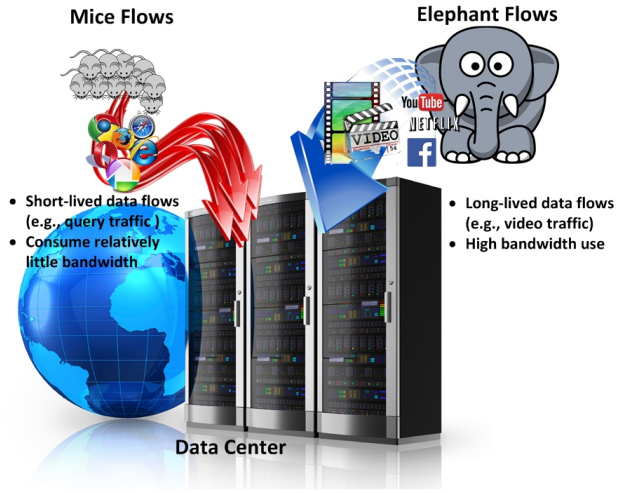

Fig. 1. Illustration of Traffic Engineering in a Datacenter

I. Introduction

THE increasing reliance on streaming mobile video from anywhere at anytime started causing new behaviour patterns in the network traffic and changes in the traffic mix. Cisco reports that the video-based traffic (TV, video on demand, Internet and P2P) will reach 80%-90% of the global consumer traffic by 2019 [1]. As mobile video becomes the dominant traffic type it also causes extraordinary changes in the network traffic patterns that in turn shape how data traffic is consumed from the Enterprise datacenters. Focusing on data communication aspects only, one of the leading cloud datacenter equipment vendors reports [2] that 75% of the total traffic present in a contemporary datacenter does not exit the datacenter. This ratio is expected to persist - if not increase - as a side-effect of trends such as the increasing number and scale of new services and end user demands. All these elevate the criticality of the in-datacenter network infrastructure, calling - among others - for agile, per-tenant isolated and Service Level Agreement (SLA) - abiding control-plane functionality. Aligned with this trend, this paper focuses on proposing new traffic management techniques targeting specific classes of datacenter network traffic, showing through real experimentation their value in maintaining a well-balanced, hot-spot free and low-latency datacenter network.

Following a bijective treatment of traffic engineering in the datacenter, literature classifies traffic flows as either short-lived (low-volume) - referred to as *mice flows* - or long-lived (high-volume) flows - referred to as *elephant flows* as seen in Fig. 1. For example, the *mice flows* could be represented by query traffic (e.g., a mobile device could ping the datacenter for a small amount of data, such as Google Search, Facebook updates, etc.) whereas the *elephant flows* could be represented by video traffic data (e.g., a mobile device requesting a Youtube stream).

The study in [3] on real datacenter traffic reported that less than 10% of the entire flow population is classified as elephant. However, the elephant flow set was found to carry more than 80% of the cumulative traffic volume exchanged within the datacenter. The latter finding motivated intensive research labour spent on deriving methods for dedicated treatment of elephant flows, while leaving mice flows to be handled by baseline, load-unaware traffic management methods. However, as the mice flows represent 90% of the flow population [3], they could have a significant impact when considered on ensemble, rather than individually. For instance, space and time correlation of trains of mice flows can be amplified, as distributed data-intensive applications scale further out in a cloud datacenter; these applications (e.g. MapReduce [4],

This work was supported by the Irish Research Council (IRC) through the Enterprise Partnership Scheme with IBM Ireland.

R. Trestian is with the School of Science and Technology, Middlesex University, United Kingdom (e-mail: r.trestian@mdx.ac.uk)

K. Katrinis is with IBM Research–Ireland, IBM Technology Campus, Damastown Industrial Estate, Dublin 15, Ireland (e-mail: katrinisk@ie.ibm.com)

G.-M. Muntean is with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University, Dublin, Ireland (e-mail: gabriel.muntean@dcu.ie).

web search, NoSQL databases) exhibit group communication patterns (one-to-many, many-to-one) even for short-lived communication (e.g. data block lookup or heartbeat messages), which in turn can lead to concentration of mice flows to a specific destination rack. In fact, the non-uniform distribution of active flows to switch ports is reported in [5]. In summary, higher volume "flows"[1] can be created due to temporal and spatial (same destination rack) correlation of ensembles of mice flows; and while these flows qualify for being treated as elephant flows, they will still get uncaught by previously proposed elephant flow management methods due to their inherent inability to treat flow aggregations. Equally, using load-unaware traffic management solutions (e.g. local random hashing) to spread the mice flows equally among multiple paths is far from optimal, as some of the links might be already loaded. This could lead to waste of bandwidth, congestion on some links, and degradation of application performance due to increased latency [6]. Moreover, as most of the datacenter bandwidth is consumed by elephant flows, they could cause latency and service disruption for mice flows without an intelligent traffic engineering solution in place.

Motivated by the above, this paper proposes **OFLoad, an OpenFlow-based dynamic load balancing scheme for datacenter networks** that achieves optimal path configuration and workload-optimization of the datacenter traffic. To better handle the impact of the many-to-one traffic patterns within a datacenter network, we propose a two-stage design framework for load-balancing. First stage copes with the elephant flow placement by defining the problem of minimizing the network congestion under the constraint to route the elephant flow traffic in a single path, minimum hop manner. The second stage makes use of an adaptive aggregator that aggregates the traffic of the mice flows and routes them in a weighted multi-path manner, keeping the same path for a certain flow at any given time. The problem is defined as a multi-commodity flow problem aiming at minimizing the network congestion under the constraint of minimum hops and minimum number of paths allocated. The load-balancing is achieved by defining a new multiplicative weighted multi-path routing algorithm which heavily impacts the already loaded paths by allocating them a lower weight value, thus less traffic will be routed through the loaded paths. An experimental prototype is built to validate the use of the OpenFlow-based technology and the benefits of the proposed OFLoad solution.

The rest of the paper is organized as follows: Section II discusses the related works and Section III presents the problem statement and the design of the proposed OFLoad solution. Section IV covers the experimental evaluation of OFLoad highlighting its benefits and finally the conclusions and future works are covered in Section V.

II. Scope and Related Works

To provide increased flexibility, resilience, and bisection bandwidth, the datacenter network topologies are densely interconnected offering support for multiple paths between pairs of end hosts. To this end, load-balancing techniques are needed to make efficient use of the bisection bandwidth abundance. In autonomous systems, the typical routing algorithms are based on the shortest path computation. For example, a well-known routing algorithm is the Open Shortest Path First (OSPF) [7] where the shortest path between a source and a destination pair is determined in advance and all the traffic is directed through this path. However this solution does not provide load-balancing over multiple paths. Because of the dense interconnected nature of the datacenter network topologies, using single-path routing without any load-balancing mechanism will not make efficient use of the network capacity. Moreover, this could lead to congestion despite still having enough unused bandwidth in the network. This problem can be avoided by using the ECMP routing, which is widely supported by the existing commodity switches. ECMP makes use of multiple shortest paths to randomly distribute the traffic and achieve load-balancing. However, ECMP cannot guarantee good load-balancing as the path selection is done statically at each router without any information of the current network-wide traffic. Thus, by using ECMP overload may still appear on certain links within the network.

Another technique which splits flows evenly and uses random routing is Valiant Load Balancing (VLB) [8]. Randomly spreading the traffic over multiple paths without considering the uneven flow sizes could lead to short-term congestion on some links. In the case of Spanning Tree Protocol (STP), all the traffic traverses a single tree which leaves many links unused. This technique avoids routing loops; however in high traffic load conditions congestion may occur resulting in packet loss.

The rapid adoption of OpenFlow [9] and Software Defined Networking (SDN) has introduced significant changes in today's enterprise datacenter network architectures and revenue models. SDN enables the network operators to control and efficiently move the data flows through the network to achieve load-balancing. This enables usage of new ways of managing the network components and the datacenter traffic. Within the research community, increasing number of OpenFlow deployments are expanding across the university campuses fuelled by several large-scale projects such as: GENI [10], FEDERICA [11], AKARI [12], OFELIA [13], SPARC [14], etc. All these networks are used for measurements of the user-generated traffic and performance analysis of different aspects of SDN (e.g., application differentiation, predictability, scalability, load-balancing, robustness, etc.). To this end, researchers have proposed various traffic management solutions that rely on a centralized controller for route configuration. The centralized controller mostly leverages on the OpenFlow technology for switch state maintenance and traffic statistics gathering and needs to scale up to meet datacenter traffic demands. As the intelligent decisions, topology virtualization and policies lie on the control plane network, its reliability becomes crucial [15].

DevoFlow [16], [17] is a flow management solution that differentiates between the mice and elephant flows and reports the elephant flows to the DevoFlow controller which maintains the visibility over the elephant flows only. The detection of the elephant flows is done at the edge switch by introducing

[1]The classical definition of a five-tuple flow is here overloaded to embrace a flow of bytes between a rack pair.

a threshold for the transferred bytes. The authors use static multi-path routing and the microflow path is randomly selected according to a pre-computed probability distribution. Similarly, Hedera [18], another OpenFlow-based flow management solution aims to maximize the bisection bandwidth with minimal scheduler overhead or impact on active flows. Initially, all flows are assumed to be mice flows and the default switch behavior is to forward the flows on equal-cost paths, similar to ECMP. When elephant flows are detected at the edge switches, Hedera uses placement algorithms to compute and allocate good paths for them. Mahout [19] detects the elephant flows and the central controller routes the elephant flows only on the best available paths (the least congested). The mice flows are routed using static load-balancing scheme (e.g., ECMP) without sending them to the controller.

Hercules [20] represents an integrated control framework which combines four controllers into one framework: (1) SPAIN [21] and (2) HBR [22] - multi-pathing controllers which pre-install a set of VLANs to be used for multi-pathing and distribute active flows over them, the distribution decision is dependent on the current traffic conditions; (3) QoS controller [23] - automatically slices the network resources to meet QoS requirements of multiple tenants, computes the resource provisioning based on the flow's QoS requirements and current link traffic load, makes use of priority tags and rate limits; (4) Mahout controller - for detection and distribution of elephant flows to avoid flow collisions, complete knowledge of incoming traffic is assumed.

Other traffic engineering solutions have been proposed in the literature that focus on load balancing. Long et al. [24] propose LABERIO, a load-balanced routing mechanism for OpenFlow-based datacenter networks. However, LABERIO relies on the information of traffic demand and does not differentiate between data flows. The authors assume that each single flow has the same size of 500M. FlowBender [25] is an end-host-driven load balancing scheme that changes a flow's path selectively and only in response to congestion in the network. However, as FlowBender does not have a global view of the network, when congestion is detected, it randomly reroutes the flow on a new path which might as well be congested. Another approach that moves the network load balancing functionality out of the datacenter network hardware into the software-based edge host was proposed by He et al. in [26] and is referred to as Presto. Presto performs sub-flow load balancing by breaking the flows into equal size chunks of 64KB units of data named flowcells routed in a round-robin manner among the available paths. However, splitting the flows at the switch for load balancing could result in instability and low throughput as well as out of order packet delivery. Niagara [27] is another traffic-splitting solution for commodity switches which offers similar performance to MicroTe [28]. A promising distributed congestion-aware load balancing solution is proposed in [29], [30] and referred to as Expeditus. Expeditus is targeted at general 3-tier Clos topologies and conveys load balancing to network edge using local congestion monitoring.

In our previous work, we proposed MiceTrap [31], an OpenFlow-based traffic engineering approach that employs mice flow aggregation together with a weighted routing algorithm to spread the traffic across multiple paths and achieve load-balancing. However, even though MiceTrap achieves good load-balancing, the weighted routing algorithm is using an additive function when computing the path weights and does not consider the impact of the number of hops on path.

A. Motivational Example

Researchers have put important efforts in understanding network traffic flow characteristics for various applications. Several classification schemes have been proposed in the literature that differentiate the flows based on: size (*mice* and *elephant* flows), duration (*dragonfly* and *tortoise*), or rate (*cheetahs* and *snails*). To differentiate between each type of flows within these classes, thresholds are defined. However the values of these thresholds differ in the literature. For example in the case of differentiation between mice and elephant flows, several values for the threshold have been proposed, including: 100MB [8], 10MB [5], or 100KB [19] [20] [32]. Farrington et al. in [33] define a mouse flow as a flow with rate $<$ 15Mb/s. Nevil et al. in [34] look at the duration of the flows and classify the flows as: very short dragonflies, lasting up to 2s; short flows lasting up to 15min.; and long running tortoises, lasting more than 15min. The authors state that only 2% of the flows last longer than 15min. and carry more than 50% of the total bytes, whereas 45% of the flows last less than 2s. Lan et al. [32] classify the flows based on their rate, such as cheetahs flows with rate $>$ 100KB/s and snails flows with rate $<$ 100KB/s. The authors state that more than 70% of the cheetah flows have a size that is less than 10KB.

Moreover, Wu et al. in [5] state that 80% of the flows have inter-arrival time between 400us and 40ms and their duration is less than 10s, whereas 20% of flows last longer than 10s. They also noticed that within one second interval at one edge switch the number of active flows is between 1000 and 2000 and the flows are not uniformly distributed on each link, causing high utilization on some links. Thus, large flows will coexist with many small flows as the authors state that 90% of the mice flows have duration overlap with one or more large flows. All these observations, motivate us to state that although the elephant flows are responsible for carrying the major proportion of bytes within the network, a large number of mice flows could be sufficient to create significant loss in the elephant traffic throughput, when considered as an aggregate.

For example, considering the case of a many-to-one traffic pattern, where multiple synchronized servers send data to the same receiver in parallel, it is feasible to assume that the mice flows are not independent and that for the same destination Top-of-Rack (ToR), there are a certain number of mice flows arriving near one to each other within a certain time interval. Moreover, let us assume a mouse flow size of 100KB, as most of the previous works do, and a mean mouse flow duration of 2s (based on [34] where the authors state that 2% of the flows last longer than 15min, 45% of the flows last less than 2s, and the rest of 53% is between 2s and 15min). Thus, the mouse flow rate would be 0.40Mbps. Furthermore, considering that there are 2000 active mice flows at a ToR switch, that means that the average rate of the aggregate mice flows would

be around 800Mbps. Even if we consider 1000 active mice flows at a ToR switch their aggregated rate would be around 400Mbps. If we take a smaller mean mouse flow duration, the numbers increase even more. In either case it is obvious that when considered as an aggregate the mice flows could have an important impact. Consequently, using ECMP to spread the amount of mice flows equally among the paths is far from optimal, as some of the links might be already loaded. This could lead to waste of bandwidth, unbalanced link utilization leading to hot-spot links, critical degradation of the application performance and latency [6]. In this context, applying traffic engineering solutions on 10% of the flows (the elephant flows) in a datacenter only, may not be very effective [3].

B. Contributions

In the context of previous work, the main contributions of this paper are as follows:

- OFLoad is proposed, an OpenFlow-based two-stage design framework for data center networks that handles the aggregated mice flow traffic and achieves load-balancing;
- a new multiplicative weighted multi-path routing algorithm is proposed based on the link utilization and the number of hops within a path to route the aggregated mice flow traffic;
- a real experimental prototype is built to validate the benefits of the OpenFlow mice flow aggregation and the performance of the proposed solution is analyzed and compared with ECMP and MiceTrap in terms of load-balancing.

III. Load Balancing Problem Formulation and OFLoad Proposed Architecture

This section defines the general communication datacenter network model, formulates the load-balancing problem addressed and describes the proposed OFLoad architecture.

Figure 2 illustrates the concept behind the proposed OFLoad solution, which addresses the load-balancing problem in a datacenter network. This load-balancing problem is formulated as a two-stage problem. In the first stage the elephant flows are routed on the path with the minimum number of hops and the minimum link utilization in single-path manner. The mice flows are routed as per default switch behavior, defined as wildcarded rules and ECMP routing. The aim is to reduce the network congestion. In the second stage, an adaptive aggregator is used to group the mice flows into bundles which are then spread across multiple weighted paths, maintaining the same path for a certain flow at any given time. The goal is to make use of multi-path weighted routing to reduce the network congestion considering the minimum number of paths and hops on paths, the weighted cost of each path and the exiting path load/number of existing flows on the path.

A. OFLoad Architecture

The architecture of OFLoad, the proposed OpenFlow-based load balancing solution is illustrated in Figure 3. The architecture includes the end-host, which integrates an elephant flow

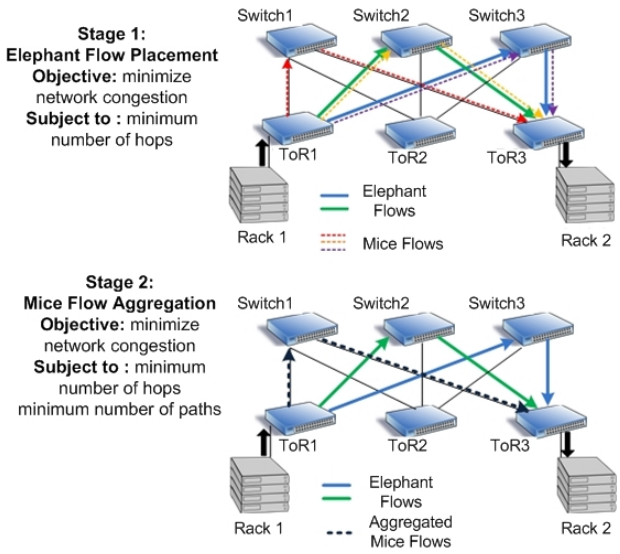

Fig. 2. General Concept of OFLoad

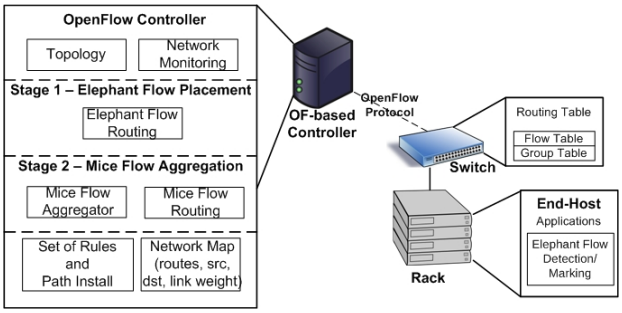

Fig. 3. OFLoad Architecture

detection/marking mechanism, the custom OpenFlow-based switch with support for OpenFlow v1.3, and the custom OpenFlow controller that manages the two-stages of the OFLoad load-balancing, as already described. These three components are described in more details next.

1) The End-Host - Elephant Marking Mechanism: The problem of differentiating between mice flows and elephant flows has already been well researched and various solutions are available, as elaborated in the previous section. In general the elephant detection and marking mechanism can be integrated at the end-host by using a kernel-level shim layer, similar to the approach introduced in [18]. The mechanism makes use of a shim layer integrated in the end-host that monitors TCP socket buffers. Given a predefined rate threshold, the shim layer identifies and marks the flow as an elephant flow (e.g. by using the Differentiated Services (DS) field in the IPv4 header), when the number of bytes in the buffer exceeds the threshold over a given time window. As soon as a flow is tagged as an elephant flow, it is sent to the controller and is handled by the Elephant Flow Routing

module (see Figure 3) which will select the best routing path for that particular flow.

2) OpenFlow-based Switch:
The OpenFlow-based switch is a custom switch with support for OpenFlow v1.3. OpenFlow v1.3 adds support for multipath routing, by using the group tables along with the flow tables. For example, when a switch receives a packet from a flow and if there is no matching in the forwarding table, the switch will forward the packet to the controller. The controller decides the way the packet will be forwarded and sends the rule to the switch. The switch will then use this rule for the following packets of the same flow. Multipath routing is enabled by adding the ability to a flow to point to a group. Each group is composed of a set of group action buckets, and each group bucket contains a set of actions to be applied to matching flows. Each bucket contains a weight field which defines the buckets share of the traffic processed by the group.

In the first stage the mice flows are wildcarded, while the elephant flows are routed in a single-path manner. Thus, the elephant flows will have a switch forwarding table entry that will map the flow to the minimum hop path. In the second stage, to efficiently use the bisection bandwidth, all the incoming mice flows that match the flow-entry destination ToR are pointed to a group. The group table contains the action buckets with each bucket corresponding to a possible path the aggregated mice flow may take to reach its destination. However, the packets belonging to the same flow will follow the same path at any given moment, avoiding the possibility of the packet re-ordering problem.

3) OpenFlow-based Controller:
The OpenFlow-based Controller consists of the following blocks: (1) *Topology block* - stores information about all the links currently up in the network. The topology discovery is done by generating link events using the Link Layer Discovery Protocol (LLDP) packets; (2) *Network Monitoring block* - keeps track of the links load by periodically collecting load information from the switch ports counter; (3) *Elephant Flow Routing block* - computes the path for the elephant flows, (4) *Mice Flow Aggregator block* - aggregates the mice flows based on the same destination ToR, (5) *Mice Flow Routing block* computes the multiple paths the mice flows could take to reach the destination, (6) *Set of Rules and Path Installation block* - sends the computed paths and rules to the switch; (7) *Network Map block* - stores a map of the network.

As we have seen in the previous section, traffic engineering and monitoring in data center networks has been well researched. Benson et al. [28] show that a significant fraction of the data center traffic is predictable on short time-scales of 1 or 2 seconds. Moreover, the authors argue that for 70% of the ToR pairs the traffic remains stable for an average of 1.5 to 2.5 seconds. Thus, the *Network Monitoring block* could collect updates every one second. Based on the averaged information received, the routing components will compute the set of paths and rules to be installed into the Openflow switches. The minimum re-computation interval could be set to 1 second [28]. Moreover, the aggregator component will group the most popular flows that match the same flow-entry destination ToR by using the short-term history, such as the last two polling messages. Additionally, the mice flow grouping reduces the control traffic overhead, as instead of having a rule for each flow the controller will send a group message installing a single rule matching the ToR destination. To further reduce the control overhead a mechanism similar to [28] could be also integrated. In this case, only one server per rack is responsible for collecting and aggregating the network statistics for the entire rack. Thus, every server in the data center collects traffic monitoring information every 0.1 seconds and sends it to the designated server every second. The designated server will update the controller only if some thresholds are reached.

B. Problem Description and Definitions

A datacenter *network topology* is represented by a connected graph $G = (V, E)$ where V is the set of nodes and E represents the directed set of links/edges. Let us assume that $N = |V|$ and $M = |E|$. A *simple path p* is defined as a finite sequence of distinct nodes $p = (v_0, v_1, ..., v_h)$, such that $\forall n \in [0, h-1]$,(v_n,v_{n+1}) $\in E$. A *commodity* is represented by the pair of nodes (i, j) $\in V \text{x} V$ where node *i* represents the source node and node j represents the destination node of the commodity. Any given commodity has assigned a non-negative demand of $\gamma_{(i,j)}$, and let us assume that β represents the set of all commodities with positive demand such that $\beta = \{(i,j) | (i,j) \in V \text{x} V, \forall \gamma_{(i,j)} > 0\}$. If $|\gamma| > 1$ it means that the network has a multi-commodity flow demand.

Let $P_{(i,j)}$ be the set of simple directed paths from the source i to the destination j in the network, and $\forall p \in P_{(i,j)}$ and link $e \in E$ there is a $\Delta_e(p)$ that represents the probability that the link e appears multiple times along the path p.

Furthermore each link $e \in E$ has assigned a link weight $w_e \in Z^+$ and a link capacity $c_e \in Z^+$. The link weights w_e define the level of the link load with $w_e \in [0, 1]$. The higher the weight, the less loaded the link is. It is considered that the OpenFlow-based Controller has an image of the entire network stored in the Network Map block so that the link state of each node in the network is collected from the OpenFlow-based switch ports counters. For example, the link load could be reported by polling the respective switch port using standard OpenFlow mechanisms.

Definition 1. Given a network topology $G(V, E)$, a *path flow* is defined as a real valued function $f : P \to R^+ \cup \{0\}$ that satisfies the following properties:

- Capacity Constraints: $\forall e \in E, \sum_{p \in P} \Delta_e(p) f(p) \leq c_e$
- Flow Demand: $\forall (i,j) \in V \text{x} V, \sum_{p \in P_{(i,j)}} f(p) = \gamma_{(i,j)}$

Definition 2. Given a network topology $G(V, E)$ and a path flow $f : P \to R^+ \cup \{0\}$, a *link flow* of a comodity $(i, j) \in V \text{x} V$ is defined as a real valued function $f : P \to R^+ \cup \{0\}$, that satisfies the following: $\forall e \in E, f_e^{(i,j)} \stackrel{def}{=} \sum_{p \in P} \Delta_e(p) f(p)$ where $f_e \stackrel{def}{=} \sum_{(i,j) \in V x V} f_e^{(i,j)}$.

Definition 3. Given a network topology $G(V, E)$ and a link flow f_e, the *link congestion factor* is defined as f_e / c_e.

Definition 4. Given a network topology $G(V, E)$ and a link flow f_e, *the network congestion factor* is defined as the maximum link congestion factor value within the network such as: $max_{e \in E} \{f_e / c_e\}$. The network congestion factor is used to

provide information about the network congestion within the network and the aim is to minimize it.

Definition 5. Given a network topology $G(V, E)$, a path $p \in P_{(i,j)}$, a *path weight* $W(p)$ of path p is defined as a multiplicative function of its containing link weights w_e and the number of hops within the path M_p, such that $W(p) = (\prod_{p \in P} w_e/M_p)/\sum_{p \in P}(\prod_{p \in P} w_e/M_p)$, and $\sum_{p \in P} W(p) = 1$.

Definition 6. Given a network topology $G(V, E)$ and a path $p \in P_{(i,j)}$, the *available capacity* $C(p)$ of path p is defined as the capacity of the bottleneck link, such that $C(p) = Min_{e \in E}\{c_e\}$.

Definition 7. Given a network topology $G(V, E)$, a path $p \in P_{(i,j)}$ and link $e \in E$, a *link weight* w_e within the path p is defined as a function of the link congestion factor such that: $w_e = 1 - f_e/c_e$.

The path weight is then used to distribute the aggregated mice flow traffic. The higher the path weight value the more traffic is routed through that path. It has been shown, that in case of decision making problems, the main drawback of using a simple additive weighted function is that a poor value for one parameter (e.g., heavily loaded link) could be heavily out-weighted by a very good value of another parameter [35] values. For example, in our case, considering a path with multiple hops on which several link segments present a light load but only one link is heavily loaded. If we use a simple additive function, the path weight will still get a good overall value and will determine the heavily loaded link segment on the path to become the bottleneck. However, by using a multiplicative function we overcome this problem by penalizing the alternatives with poor parameters values more heavily. Thus, the path with the heavily loaded link segment will receive a poor weight value meaning that less traffic will be routed through that path, avoiding in this way a bottleneck situation.

Considering all these aspects the two main problems of each stage are formulated in the following sections. Both stages are trying to minimize the network congestion factor subject to different conditions.

C. First Stage - Elephant Flow Placement

The first stage handles the elephant flow placement. The objective is to assign the elephant flows to the paths with minimum number of hops and minimum load such that network congestion is minimized.

1) Problem Formulation:

Given a network topology $G(V, E)$, for each link $e \in E$ with the link capacity c_e, we compute a link weight $w_e \in [0, 1]$, and for each commodity $(i, j) \in V\text{x}V$ marked as an elephant flow from the source node i to the destination node j with a demand of $\gamma_{(i,j)}$, find the shortest path $p \in P_{(i,j)}$ for the path flow $f(p)$ that minimizes the network congestion factor and reduces the number of hops within the path, M_p. Thus the elephant placement problem can be formulated as follows:

$$\text{minimize:} \quad max_{e \in E}\{f_e/c_e\}$$

$$\text{subject to:} \quad \sum_{p \in P} f(p) \leq c_e, \qquad \forall e \in E$$

$$\sum_{p \in P_{(i,j)}} f(p) = \gamma_{(i,j)}, \qquad \forall (i, j) \in V\text{x}V$$

2) Elephant Flow Path Selection:

After a flow is marked as an elephant flow by the end-host, the first packet of the flow arrives at the OpenFlow-based switch, which routes the packet to the Controller. At the Controller side, the Elephant Flow Routing block will compute the set of shortest paths, based on the minimum number of hops, between the source and destination of the elephant flow. The path with the lowest path utilization (e.g., minimum link congestion factor) will be selected as the target path. After the best path is computed, the Controller installs the rules into the OpenFlow switches among the path. The elephant flow is routed in a single-path manner on the best selected path, thus all the other packets appertaining to the elephant flow will follow the same route.

D. Second Stage - Mice Flow Aggregation

The second stage handles mice flow aggregation and path selection. The objective is to aggregate the mice flows and route them in a multi-path manner, by identifying a set of shortest paths such that the load of the most utilized links in the network is minimized.

1) Problem Formulation:

Given a network topology $G(V, E)$, for each link $e \in E$ with the link capacity c_e we compute a link weight $w_e \in [0, 1]$, and for each commodity represented by the aggregated mice flows, $(i, j) \in V\text{x}V$ with demand $\gamma_{(i,j)}$, and a path restriction $K_{(i,j)}$ find the shortest path set $\in P_{(i,j)}$ for the path flow $f(p)$ that minimizes the network congestion factor. Thus the aggregated mice flows placement problem can be formulated as follows:

$$\text{minimize:} \quad max_{e \in E}\{f_e/c_e\}$$

$$\text{subject to:} \quad \sum_{p \in P^*_{(i,j)}} W(p)f(p) \leq C(p), \quad \forall p \in P^*_{(i,j)}$$

$$\sum_{p \in P^*_{(i,j)}} f(p) = \gamma_{(i,j)}, \qquad \forall (i, j) \in V\text{x}V$$

where $P^*_{(i,j)} \subseteq P_{(i,j)}$, the set of shortest paths such that $|P^*_{(i,j)}| \leq K_{(i,j)}$.

2) Adaptive Mice Aggregation and Path Selection:

Initially, the default OpenFlow switch behavior is to use wildcards and forward the flows using ECMP. When a flow is marked as an elephant flow, the Controller will install the specific rules into the switches so that the elephant flow is routed in a single-path manner. The controller collects flow-related information and statistics from the OpenFlow switches periodically by using custom OpenFlow polling messages. Based on a short-term history (e.g. last two polling messages) the Mice Flow Aggregator groups the most popular flows that match the same flow-entry destination ToR. The Mice Flow Routing block will compute the set of shortest paths between the source and destination ToR and the weights for each specific path. To take full advantage of the datacenter network bisection bandwidth available, as well as to protect the application performance, OFLoad routes the mice flow aggregates using a weighted multi-path routing algorithm.

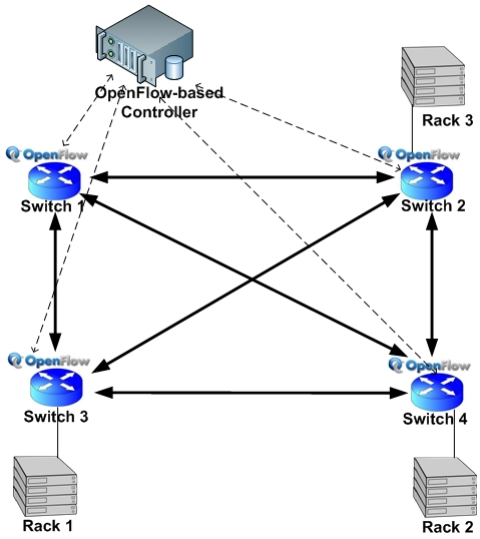

Fig. 4. Experimental Prototype Setup

The weights are dynamically computed using a multiplicative function as given in Definition 5. When using a multiplicative function, the links that are already loaded will be impacted more heavily by receiving a low value weight, such that less traffic is routed on those links. Once computed, the path weight is updated in the action bucket weight field of each path within a specific group. To avoid the re-ordering problem, each demand from a node i is required to route its entire flow on the same path within the group. To this end, a hash function on a packet is defined in the switch to distribute the flows across the multiple weighted paths by sending the packets appertaining to the same flow on the same path.

IV. Experimental Prototype, Results and Discussions

This section describes the real life experimental prototype setup and the experimental scenarios, presents the results and performs result analysis. The aim of this section is to validate the benefits brought by the proposed OFLoad mechanism in terms of grouping and routing the mice flows within a datacenter to enable load balancing. As previously seen, most of the solutions in the literature use ECMP for mice flow routing whereas the elephant flows are routed using the shortest least congested path. Thus, the performance of OFLoad is compared against MiceTrap and ECMP, where the mice flows are routed using ECMP and the elephant flows are routed on shortest least congested path similarly to the approaches in the literature.

A. Experimental Prototype Setup

The real life experimental prototype setup is illustrated in Figure 4. It consists of four OpenFlow-based switches connected in a clique topology, three racks for hosts, and one OpenFlow-based controller.

As most of the hardware OpenFlow-based switches work with OpenFlow version 1.0 only, which does not offer support for the group option, an alternative is the use of OpenFlow-based software switches. Consequently, the OpenFlow-based switches used in this experiment are based on the OpenFlow 1.3 Software Switch (OFSoftSwitch13[1]) that offers support for the OpenFlow group option, integrated in OpenFlow version 1.3.

One OpenFlow-based controller that offers support for OpenFlow version 1.3 is the nox13oflib controller[2], which is a compatible version of the NOX controller. Nox13oflib is based on the Niciras NOX Zaku[3] controller with the OpenFlow processing library being replaced with Oflib from OpenFlow 1.3 Software Switch.

The traffic generator used to generate traffic between the hosts in the racks is iperf[4], a widely used network traffic generator that can create TCP and UDP data streams between end hosts.

Tcpdump[5] is used to capture all the traffic in the experimental test-bed. The traffic is captured at each port of the switches to analyse how the packets are routed within the network.

Tcptrace[6] and TRace Plot Real-time[7] (TRPR) are used to analyze the output files from the tcpdump packet sniffing program.

All the machines used in the experimental test-bed are running Red Hat Enterprise Linux Server release 6.2 x86 (64-bit). Each machine has two Intel Xeon Processors X5670 (2.93 Ghz, 6-core, 4 flops per cycle) for improved performance.

B. Experimental Prototype Limitations

There are some limitations imposed by the use of the OpenFlow-based software switch which might depend on the underlying hardware on which OFSoftSwitch13 is running. In this situation, everything is handled by the system CPU, and therefore the switch performance is also influenced by the CPU power. To determine the limitations imposed by the software switches a set of experiments were conducted with the following observations:

1) OFSoftSwitch13 Delay:

The delay introduced by the software switch was measured considering two topology scenarios:

- One OFSoftSwitch13 Topology - one softswitch is used between two hosts (e.g. Host A and Host B), and the controller. Host A pings Host B, and the path taken by the first packet is Host A $\rightarrow$ Softswitch $\rightarrow$ Controller $\rightarrow$ Softswitch $\rightarrow$ Host B with a latency of 1.71ms including the messages exchanged between the softswitch and the controller for path setup. After the path setup the average RTT is 0.456ms.

[1]OpenFlow 1.3 Software Switch (OFSoftSwitch13) - https://github.com/CPqD/ofsoftswitch13

[2]NOX13oflib - https://github.com/CPqD/nox13oflib

[3]NOX Zaku Controller - http://www.noxrepo.org/nox

[4]Iperf - http://sourceforge.net/projects/iperf/

[5]Tcpdump - http://www.tcpdump.org/

[6]Tcptrace - http://www.tcptrace.org/

[7]TRace Plot Real-time (TRPR) - http://pf.itd.nrl.navy.mil/proteantools/trpr.html

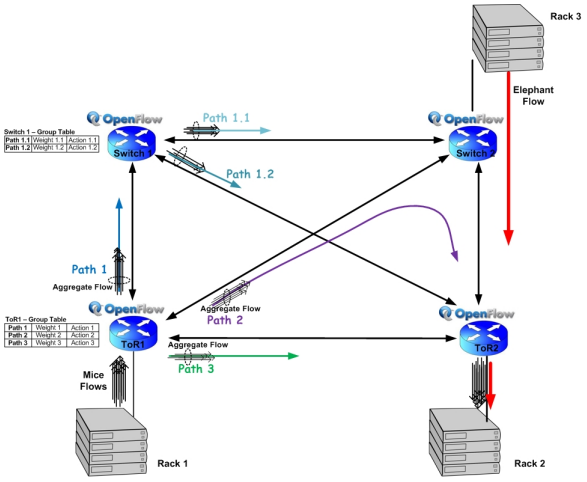

Fig. 5. Experimental Prototype Scenario

- Two OFSoftSwitch13 Topology - two softswitches are used between two hosts (e.g., Host A and Host B), and the controller. Host A pings Host B, the path taken by the first packet is Host A $\rightarrow$ Softswitch1 $\rightarrow$ Controller $\rightarrow$ Softswitch1 $\rightarrow$ Softswitch2 $\rightarrow$ Host B with a latency of 4.9ms including the messages exchanged between the switches and the controller. After the path setup the average RTT is around 1.52ms.

2) OFSoftSwitch13 Bandwidth Limitation:
For testing the bandwidth limitation imposed by the softswitch two topology scenarios were considered:

- One Path Topology - we created one path between two hosts (e.g., Host A and Host B) and generated traffic. We noticed that regardless of the number of hops between the two hosts, when using only one path the maximum achievable traffic rate is around 180Mbps.
- Multiple Paths Topology - we created multiple paths between two hosts (e.g., Host A and Host B) and generated traffic. We noticed that when more than one path is used the maximum achievable traffic rate is around 110Mbps.

C. Experimental Scenarios

To analyze the performance of the proposed OFLoad solution, we have considered the experimental prototype already described and the scenario illustrated in Figure 5, under various traffic load. This section presents testing results and analyzes the performance of the proposed algorithm under different heavily loaded network conditions. It is assumed that Rack 3 generates traffic towards Rack 2 to heavily load the link between Switch2 and ToR2. We name this traffic *elephant flow traffic* (e.g., video flows), however it can also represent the aggregated traffic load from Rack 3 towards Rack 2. To see if the mice flow distribution impacts the elephant flow traffic, Rack 1 generates mice flow traffic towards Rack 2 over three possible paths. It is assumed that the mean mouse flow duration is less than 2s, the inter-arrival time is between 400ms and 40us with all the mice flows pointing to the same destination ToR. The mice flow size is considered to be $<$ 100KB and the number of active mice flows at any given moment is between 1500 and 2000. These values were selected based on the motivation presented in Section II and based on the limitations of the experimental prototype, previously introduced.

TABLE I. EXPERIMENTAL SCENARIOS

Scenario	Elephant Flow Rate [Mbps]	Aggregate Mice Flow Rate [Mbps]
1	150	75
2	150	100
3	170	75
4	170	100

Four scenarios are considered, as listed in Table I. Scenario 1 and Scenario 2 consider the elephant flow traffic as 83% of the maximum link capacity. Because between Rack 3 and Rack 2 the traffic is assumed to be routed in a single-path manner, the maximum achievable rate is around 180Mbps, as previously discussed. Consequently, the elephant flow traffic rate is 150Mbps. The mice flow traffic is generated such that an average rate of 75Mbps for Scenario 1 and 100Mbps for Scenario 2 of the aggregated mice traffic is maintained. As between Rack 1 and Rack 2 there are three available paths considered, the maximum achievable rate is 110Mbps, as discussed in the experimental prototype limitations section. In Scenario 3 and Scenario 4 the elephant flow traffic is increased to 94% of the link capacity, reaching 170Mbps.

D. Results and Discussions

1) Reducing the Number of TCAM Entries:
In general the OpenFlow-based switch needs to maintain a rule in the forwarding table for each incoming flow. However doing this is very expensive as switch's memory is a scarce resource. The main idea behind OFLoad is to make use of the OpenFlow group option to reduce the number of TCAM entries in the OpenFlow switch. As the many-to-one traffic pattern is very common in datacenters for applications like MapReduce and web search, we propose to group the mice flows pointing to the same destination ToR. For example, assuming around 2000 active mice flows at a ToR switch pointing to the same destination ToR, the controller will group the mice flows based on the same destination ToR and will install a single rule matching the destination, instead of having a rule for each flow (e.g., 2000 rules in the TCAM entries). Moreover, by pointing to the group table, whenever changes in traffic distribution occur, a single explicit group message can update a set of flow entries avoiding sending an explicit message for each flow. In the experimental prototype as illustrated in Figure 5, the OpenFlow SoftSwitches makes use of the OpenFlow group option when routing the aggregated mice traffic from Rack 1 to Rack 2. For example, ToR1 has a group which points to three available paths of different weights, and Switch1 has a group which points to two available weighted paths. Without using the OpenFlow group option as we propose, the OpenFlow switch would have to install a rule for each

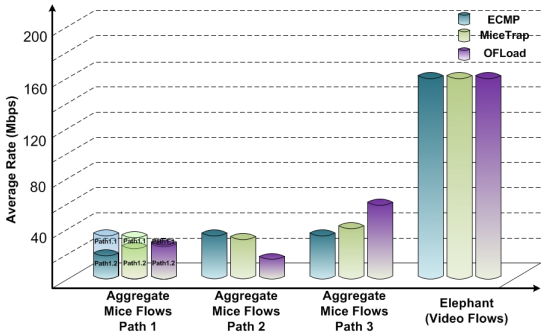

Fig. 6. Traffic Distribution on Paths for Scenario 1

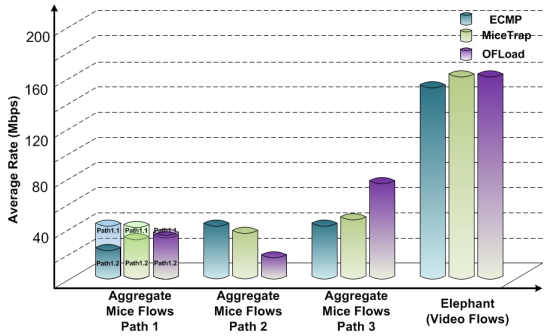

Fig. 7. Traffic Distribution on Paths for Scenario 2

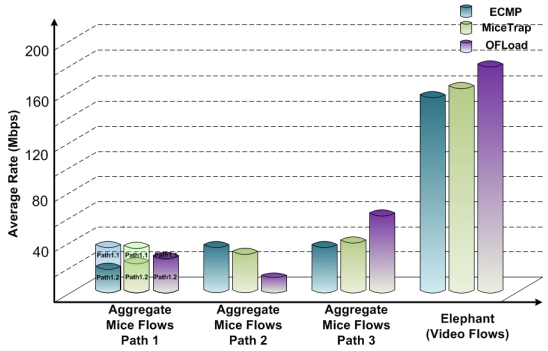

Fig. 8. Traffic Distribution on Paths for Scenario 3

incoming flow which may lead to an overload of the flow tables in the switches.

2) Load-Balancing and Multipath Routing:
To validate the benefits of the proposed OFLoad in terms of load-balancing, we compared its performance against that of ECMP, which is widely used in the commodity switches, and MiceTrap, another weighted routing algorithm in the four considered scenarios. In case of ECMP, the traffic is distributed equally on the multiple paths, so the weights are equal. In case of MiceTrap, the weights are computed using a simple additive function, as introduced in [31]. For both MiceTrap and OFLoad the weights are computed based on link utilization, and thus their computation depends on the already existing traffic in the network. This means that there will be two sets of weights, one set for Scenario 1 and Scenario 2, where the elephant flow rate is 150Mbps and another set for Scenario 3 and Scenario 4, where the elephant flow rate is 170Mbps. The two sets of path weight values are listed in Table II.

TABLE II. PATH WEIGHTS

Path	Scenario 1-2		Scenario 3-4	
	MiceTrap	OFLoad	MiceTrap	OFLoad
Path 1	0.33	0.250	0.33	0.250
Path 1.1	0.33	0.100	0.33	0.040
Path 1.2	0.66	0.900	0.66	0.960
Path 2	0.28	0.076	0.28	0.030
Path 3	0.39	0.673	0.39	0.720

The results for Scenario 1 are presented in Figure 6. In this scenario the average elephant flow rate is 150Mbps and the average aggregate mice flow is 75Mbps. In this situation the elephant flow is not impacted by the aggregate mice traffic. However looking at the traffic distribution among paths, we can see that in comparison with ECMP and MiceTrap, OFLoad, the new proposed algorithm, penalizes the loaded paths (Path 1.1 and Path 2) more heavily by giving them a poor weight. This results in less traffic transferred on the loaded paths and more on the unloaded paths, such as Path 3 and Path1.2, eventually increasing the overall transport performance.

In Scenario 2 we increase the aggregated mice traffic to 100Mbps. The results of this scenario are illustrated in Figure 7. It can be seen that in case of ECMP the increase in the aggregated mice traffic impacts the elephant flow traffic rate, which experiences a 6% decrease. Both MiceTrap and OFLoad achieve good traffic distribution among the paths and do not affect the elephant flow traffic.

Figure 8 illustrates the results for Scenario 3. In this scenario the elephant traffic is increased to 170Mbps such that the link between Switch2 and ToR2 became heavily loaded (up to 94% of the link capacity). The aggregated mice flow rate is kept at 75Mbps. As it can be seen, the elephant flow rate presents a decrease of 14% when using ECMP, and a 8% decrease when using MiceTrap. However when using OFLoad, the elephant flow rate is not affected at all by the aggregated mice flow traffic. This is because OFLoad is using a multiplicative function when computing the path weights and the heavily loaded paths are penalized more than when using a simple additive function, as in case of MiceTrap. This results in much less traffic routed through the loaded paths and the consequent OFLoad improvement in transport performance.

In Scenario 4, we maintain the 170Mbps rate for the elephant flow, and we increase the aggregated mice flow rate to 100Mbps, to study the behavior of the mechanism in heavily loaded conditions. It can be seen how the elephant flow rate goes down by 17% when ECMP is employed, and by 12% when using MiceTrap. However in case of OFLoad, which is based on a multiplicative weighted routing algorithm, the elephant flow rate is not impacted by the extra traffic. Most of the traffic is distributed on the unloaded paths such as Path 3 and Path 1.2 and this contribute decisively to the evident OFLoad usage benefit.

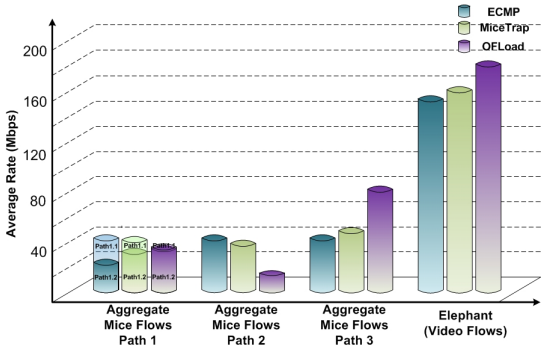

Fig. 9. Traffic Distribution on Paths for Scenario 4

E. Conceptual Comparison to Other Solutions

This section provides a conceptual comparison of the proposed OFLoad solution to other solutions from the literature introduced previously. Table III provides a summary of the existing solutions highlighting their main findings.

Looking at the OpenFlow-based flow management solutions for datacenter networks, such as DevoFlow [16], [17], Hedera [18] and Mahout [19], they all propose a method to differentiate between mice and elephant flows. However, the proposed traffic management/flow-scheduling schemes cope with elephant flows only while the mice flows have not been taken into account as much. The controller should have a global traffic view to make efficient routing decision. For example, Hedera uses ECMP for short-lived flows and the OpenFlow controller handles the elephant flows (> 100MB) only. The authors in [28] showed that Hedera performs comparable to ECMP for a traffic matrix in which most of the contending traffic is carried by flows with less than 100MB of data. Hercules [20] assumes that mice flows do not affect the throughput of the elephant flows because they are typically small and evenly distributed over all links. However, this cannot guarantee good load balancing as the mice flows could be spread among links that might be already loaded. This could lead to waste of bandwidth, unbalanced link utilization and hot-spot links, critical degradation of the application performance and latency. The proposed OFLoad in turn, groups the mice flows and routes them as an aggregate using a weighted multi-path routing algorithm in order to overcome the limitations of ECMP and achieve load balancing.

Moreover, ECMP, Hedera, DevoFlow, Mahout, etc. rely on static flow-level traffic splitting by selecting a path for a flow based on the current network conditions. However, as the network conditions change over time, the statically-selected path may not be optimal anymore.

Compared to OFLoad, other load balancing solutions like LABERIO [24], FlowBender [25], Presto [26], Niagara [27], or Expeditus [30] they either do not look into differentiating between the mice and elephant flows or they try to move the load balancing functionality at the network edge limiting the global view of the controller. Breaking the flows into sub-flows is another option adopted by some of the solutions. However, this could actually lead to instability, low throughput and out-of-order packet delivery.

V. Conclusions and Future Works

This paper argues and demonstrates that in case of many-to-one datacenter traffic patterns, a large number of mice flows which are very common in case of MapReduce or web search applications, have the potential to negatively impact the elephant flow traffic throughput. It is known that the traffic throughput for elephant flows is critical to ensure the good performance of the overall applications as more than 80% of the cumulative traffic volume within the datacenter is carried by the elephant flows. Consequently the mice flows need not to be ignored, as their aggregate effect is sufficient to degrade other applications' performance. This paper proposed OFLoad, a novel OpenFlow-based two-stage design framework for data center networks. OFLoad makes use of the OpenFlow group option to aggregate the mice flows based on the destination rack. The aggregated mice flow is then routed using a new proposed multiplicative weighted multi-path routing algorithm which achieves good load-balancing.

We built an experimental prototype to investigate the benefits of using OpenFlow-based grouping option and to analyze the performance of OFLoad when compared to ECMP and MiceTrap, in terms of load-balancing. Their performance was analyzed under various traffic load network conditions scenarios. The results show that in heavily loaded network conditions, for example Scenario 4, employing ECMP could lead to 17% decrease in the elephant traffic throughput. Athough MiceTrap has a 30% increase in elephant traffic throughput when compared to ECMP, by using OFLoad the elephant flow traffic throughput is not impacted at all. This is because of the routing algorithm employed by OFLoad in which an innovative multiplicative function is used to penalize heavily the loaded paths, by giving them poor value weights. This results in less traffic sent on the loaded paths and favorising the unloaded paths.

Furthermore, unlike the basic behavior of the OpenFlow switch requiring to install a rule for each incoming flow and may lead to an overload of the forwarding tables in the switches, OFLoad makes use of the OpenFlow group option to aggregate the mice flows, thus reducing the forwarding table size significantly.

As Software Defined Networks is still maturing there are many questions left to be answered. For example, the choice between an overlay SDN solution vs. an underlay SDN solution might worth investigating. With the overlay SDN the traffic is tunnelled over the existing physical network without any visibility into the paths the flows are taking. Whereas, underlay SDN makes use of global information knowledge to directly manipulate the network components and route the traffic flows on specific paths. However, selecting between overlay or underlay SDN is not trivial as it has implications in terms of complexity, monitoring, troubleshooting, security, performance management, SLA compliance, etc. As the research community would benefit from such a study, it is included in our future works.

TABLE III. CONCEPTUAL COMPARISON WITH EXISTING APPROACHES

Mechanism	Application	Topology	Workload	Findings
DevoFlow [16], [17]	OpenFlow-based flow management solution for datacenter networks	three-level Clos and two-dimensional HyperX topologies	MapReduce job (shuffle phase) and Microsoft Research measurements of a 1500-server cluster	Differentiates between mice and elephant flows by using a threshold at the edge switch for the transferred bytes. The DevoFlow controller maintains visibility of the elephant flows only.
Hedera [18]	OpenFlow-based flow management solution for datacenter networks	fat-tree network topology	synthetic workloads: stride, staggered probability, random, random bijection shuffle	Differentiates between mice and elephant flows at the edge switches. Makes use of placement algorithms to compute good paths for the elephant flows only.
Mahout [19]	OpenFlow-based flow management solution for datacenter networks	three-level Clos topology	MapReduce job (shuffle phase)	Differentiates between mice and elephant flows at the end-host. The controller routes the elephant flows only on the least congested paths.
Hercules [20]	traffic management solution for datacenter networks	two-level fat-tree network topology	synthetic traffic	It integrates four existing controllers including Mahout [19]. It enables multiple controllers to leverage each other and achieve the goals collectively.
LABERIO [24]	OpenFlow-based load balancing scheme	fully populated fat-tree network topology	uniform, semi-uniform, hot spot	In a hot-spot scenario LABERIO reduces with up to 13% the transmission time when compared to round robin and provides similar performance for uniform and semi-uniform scenarios.
FlowBender [25]	distributed end-host driven load balancing scheme	fat-tree network topology	all-to-all, storage-type	In a real setup, FlowBender achieves up to 40% reduction in the flow completion tail latencies for large flows compared to ECMP. The simulation results show that FlowBender achieves similar performance to other schemes from the literature.
Presto [26]	software-based edge driven load balancing	Clos network topology	synthetic workload: shuffle, stride(8); random, random bijection	For shuffle workload Presto achieves similar performance as ECMP. Whereas for non-shuffle workloads Presto improves by $38\% - 72\%$ upon ECMP in terms of elephant flow throughput.
Niagara [27]	SDN-based traffic splitting solution	basic campus network and symmetric and asymmetric datacenter topologies	synthetic VIP traffic and real datacenter traces	For symmetric topologies Niagara performs better than ECMP but offers similar performance to MicroTE. Niagara incurs $> 20\%$ imbalance for 15% time frames. For asymmetric topologies Niagara performs slightly worse than MicroTE for $< 2\%$ imbalance.
Expeditus [30]	distributed congestion-aware load balancing protocol	three-level Clos topology	web search and data mining workloads	For mice flows, Expeditus outperforms ECMP by up to 45% in tail flow completion times and by up to 38% in mean flow completion time for elephant flows in 3-tier Clos networks.
OFLoad	OpenFlow-based load balancing mechanism	clique topology	many-to-one workload	OFLoad aggregates the mice flows and routes them in a weighted multi-path manner such that less traffic is sent on the loaded paths and the unloaded paths are favourised. Thus, the elephant flows throughput is protected whereas by using ECMP the elephant flow rate could go down by up to 17%. Moreover, OFLoad reduces the forwarding table size significantly.

REFERENCES

[1] Cisco Systems Inc., "Cisco visual networking index: Forecast and methodology, 2014–2019," 2015.

[2] ——, "Cisco global cloud index: Forecast and methodology, 2014–2019," 2015.

[3] 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 conference*. ACM, 2009, pp. 202–208.

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

[5] W. Wu, Y. Chen, R. Durairajan, D. Kim, A. Anand, and A. Akella, "Adaptive data transmission in the cloud," in *IWQoS, 2013 Proceedings IEEE/ACM International Symposium on Quality of Service*. IEEE, 2013.

[6] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[7] J. Moy, "OSPF Version 2," RFC 2178 (Draft Standard), Internet Engineering Task Force, July 1997, obsoleted by RFC 2328. [Online]. Available: http://www.ietf.org/rfc/rfc2178.txt

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[10] J. S. Turner, "A proposed architecture for the geni backbone platform," in *Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on*. IEEE, 2006, pp. 1–10.

[11] V. Maglaris and C. Cervelló-Pastor, "With evolution for revolution: Managing federica for future internet research," *IEEE Communications Magazine*, p. 3, 2009.

[12] H. Harai, "Designing new-generation network: Overview of akari architecture design," in *Asia Communications and Photonics Conference and Exhibition*. Optical Society of America, 2009.

[13] A. Köpsel and H. Woesner, "Ofelia–pan-european test facility for openflow experimentation," in *Towards a Service-Based Internet*. Springer, 2011, pp. 311–312.

[14] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for openflow," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 6606–6610.

[15] S. Song, H. Park, B. Y. Choi, T. Choi, and H. Zhu, "Control path management framework for enhancing software-defined network (sdn) reliability," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, 2017.

[16] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: Cost-effective flow management for high performance enterprise networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 1.

[17] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41,

no. 4, pp. 254–265, Aug. 2011.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[19] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1629–1637.

[20] W. Kim and P. Sharma, "Hercules: Integrated control framework for datacenter traffic management," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 70–78.

[21] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies." in *NSDI*, 2010, pp. 265–280.

[22] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble routing for datacenter networks," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2010, p. 23.

[23] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," *Proc. INM/WREN*, vol. 10, pp. 1–1, 2010.

[24] H. Long, Y. Shen, M. Guo, and F. Tang, "Laberio: Dynamic load-balanced routing in openflow-enabled networks," in *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, March 2013, pp. 290–297.

[25] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, December 2014, pp. 149–159.

[26] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, August 2015, pp. 465–478.

[27] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2015, pp. 6:1–6:13.

[28] 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*. ACM, 2011, p. 8.

[29] P. Wang and H. Xu, "Expeditus: Distributed load balancing with global congestion information in data center networks," in *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2014, pp. 1–3.

[30] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong, "Expeditus: Congestion-aware load balancing in clos data center networks," in *ACM Symposium on Cloud Computing (SoCC)*, October 2016, pp. 442–455.

[31] R. Trestian, G.-M. Muntean, and K. Katrinis, "Micetrap: Scalable traffic engineering of datacenter mice flows using openflow," in *IM, 2013 Proceedings IEEE*. IEEE, 2013.

[32] K. Lan and J. Heidemann, "On the correlation of internet flow characteristics," Technical Report ISI-TR-574, USC/ISI, Tech. Rep., 2003.

[33] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a electrical/ architecture for modular," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339–350, 2011.

[34] N. Brownlee and K. Claffy, "Understanding streams: Dragonflies and tortoises," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 110–117, 2002.

[35] Q. T. Nguyen-Vuong, Y. Ghamri-Doudane, and N. Agoulmine, "On utility models for access network selection in wireless heterogeneous networks," in *IEEE Network Operations and Management Symposium (NOMS)*, April 2008, pp. 144–151.