Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks

Daniel S. Marcon, Marinho P. Barcellos
Institute of Informatics – Federal University of Rio Grande do Sul, RS, Brazil
Email: {daniel.stefani, marinho}@inf.ufrgs.br

***Abstract*—Software-Defined Networking (SDN) can simplify traffic management in large-scale datacenter networks (DCNs). On one hand, it provides a robust method to address the challenge of performance interference (bandwidth sharing unfairness) in DCNs. On the other, its pragmatic implementation based on OpenFlow introduces scalability challenges, as it (a) adds latency for new flows (the controller must process hundreds of thousands of requests per second and install appropriate rules in switches); and (b) requires large flow tables in devices (DCNs can have more than 16 million distinct flows per second with different requirements and duration). To employ OpenFlow-based SDN in DCNs, recent work has proposed techniques that require hardware customization to keep up with the high dynamic traffic patterns of these networks. We make two key observations: providers do not need to control each flow individually (e.g., VM-to-VM), since they charge tenants based on the amount of resources consumed by applications; and congestion control in the intra-cloud network is expected to be proportional to the tenant's payment. Based on these insights, we introduce Predictor, a novel system for DCNs that enables fine-grained network management for providers, minimizes flow table size by controlling flows at application-layer and reduces flow setup time by proactively installing rules in switches. It also enables tenants to request and receive predictable network performance for both intra- and inter-application communication, with work-conserving bandwidth sharing. Evaluation results show that Predictor provides significant improvements against DevoFlow (reducing flow table size up to 87%) and offers predictable and guaranteed network performance for tenants.**

I. Introduction

The paradigm of Software-Defined Networking (SDN) has emerged as an efficient method that offers programability to dynamically manage and configure network resources. On one hand, SDN provides a robust method to address the challenge of performance interference [1], [2] in multi-tenant cloud datacenter networks (DCNs). While providers offer guaranteed computing resources, the network is shared in a best-effort manner among all tenants [3]. The lack of network guarantees (for both intra- and inter-application communication) results in unpredictable and poor overall application performance [4]. Several recent papers [1], [5]–[7] have proposed techniques to address this issue. Nonetheless, they do not provide fine-grained control over network resources for providers, as they require complex interactions among hundreds of thousands of hypervisors.

On the other hand, SDN, and pragmatic OpenFlow-based deployments, faces scalability challenges in DCNs (and, in general, in high-performance, dynamic networks) for the following reasons [8], [9]. First, the transition between the data and control planes whenever a new flow arrives at a switch[1] may add some delay (latency for communication between switches and the controller), and the high frequency at which flows arrive and demands change in DCNs hinders controller scalability. Second, the number of entries needed in flow tables of forwarding devices can be significantly higher than the amount of resources available in commodity switches, especially for large-scale DCNs [8], [10] (as such networks can have more than 16 million flows per second [11]).

Scaling the controller has been the main topic of some studies. They address this issue either by devolving control to the data plane [8], [12] or by developing a logically distributed controller [13]. The former approach adds several functionalities to switches and, thus, requires more complex, customized hardware, while the latter does not scale for large DCNs where communications occur between virtual machines (VMs) connected by different top-of-rack (ToR) switches.

In this paper, we make two *key observations*: (i) providers do not need to control each flow individually, since they charge tenants based on the amount of resources consumed by applications; and (ii) congestion control in the intra-cloud network is expected to be proportional to the tenant's payment [1], [14]. Therefore, we adopt a broader definition of flows, considering it at application-layer[2], and introduce Predictor, a novel system for Infrastructure-as-a-Service (IaaS) cloud datacenters. Predictor addresses the two aforementioned challenges (performance interference and scalability of OpenFlow-based deployments in DCNs), offering the following benefits.

First, it enables fine-grained network management for cloud providers by employing the SDN paradigm and allowing control of flows at application-layer. Predictor makes use of wildcards to reduce flow table size at switches and to allow providers to control traffic and gather statistics at application-layer for each link and device in the network.

Second, it enables tenants to request and receive a minimum guaranteed bandwidth for both intra- and inter-application communication, while allowing the use of more bandwidth when there is spare capacity (work-conservation). The controller proactively installs rules in switches at application allocation time to guarantee bandwidth for communication between VMs of the same application, and reactively sends rules for inter-application communication. On one hand, proactively installing rules at switches reduces flow setup time and management traffic in the network (since switches do not

[1]We use the terms "switches" and "forwarding devices" to refer to the same set of SDN-enabled network devices, that is, data plane devices that forward packets based on a set of flow rules.

[2]An application is represented by a set of VMs that consume computing and network resources (see Section III-A for further details).

request the controller assistance for each new flow). On the other hand, some flow table entries may take longer to expire (they are removed only when their respective applications conclude and are deallocated). Inter-application forwarding rules, in turn, are reactively installed in switches, because applications may not know all other applications they will communicate with in advance (when they are allocated) and intra-application traffic volume is expected to be higher [1].

Overall, the major contributions of this paper are:

- We propose a topology-agnostic strategy that takes advantage of SDN to provide fine-grained network management for IaaS providers (e.g., by controlling flows at link-level) and predictable performance with guaranteed bandwidth for tenants, without requiring customized hardware in switches.
- We present the design of Predictor, a novel system that implements the proposed strategy for large-scale DCNs.
- We show the benefits of Predictor, comparing it against the state-of-the-art controller for DCNs (DevoFlow [8]). Evaluation results show that Predictor can significantly reduce flow table size at forwarding devices (up to 87%) with small increase of controller load. Furthermore, Predictor offers predictable network performance with guaranteed bandwidth and work-conserving sharing.

The paper is organized as follows. Section II provides the background on the cloud network sharing problem and the pros and cons of using SDN to solve it. Section III describes our strategy (and its implementation, Predictor), while Section IV presents its evaluation. Section V discusses the generality and limitations of our strategy, and Section VI examines related work. Finally, Section VII concludes the paper with final remarks and perspectives for future work.

II. Background

This section first examines the intra-cloud network sharing problem and, then, discusses the benefits and drawbacks of employing SDN on DCNs.

A. Cloud Datacenter Network Sharing

Current cloud providers (such as Amazon EC2) typically offer VMs with guaranteed computing resources. However, the underlying network is shared in a best-effort manner [7]. Measurement studies [4], [5] concluded that the network throughput achieved by VMs can vary by a factor of five or more. Such variability results in poor and unpredictable application performance, and tenants end up spending more money (since their applications take longer to finish) [5].

Popa et al. [14] elaborates on two main requirements for network sharing: i) bandwidth guarantees for both intra- and inter-application communication; and ii) work-conserving sharing to achieve high network utilization for providers. In particular, these two requirements present a trade-off: strict bandwidth guarantees may reduce utilization, since applications have variable network demands over time [6]; and a work-conserving approach means that, if there is residual bandwidth and some applications have demands, they should utilize it (even if the available bandwidth belongs to the guarantees of another application) [15]. Therefore, we employ SDN to enable fine-grained network management in order to develop a robust strategy to explore this trade-off.

B. Software-Defined Networking

SDN [16] seeks to decouple the control and data planes of the network. The pros and cons of using SDN to solve the network sharing problem in DCNs are discussed below.

Benefits. SDN (and OpenFlow) offers programability to dynamically configure and manage the entire network, enabling providers to offer a base level of network performance guarantees for tenants. In particular, it allows administrators to apply a wide-range of policies with little effort (without requiring device by device configuration), including bandwidth guarantees, routing and fault tolerance [8]. SDN also provides near-optimal traffic management, since the controller can request and receive information about network load on a low-level granularity (e.g., by device, link or specific flows traversing a link).

Drawbacks. This paradigm involves the control plane more frequently than traditional networking. The two main issues related to DCNs are flow setup time (the time taken to install a new flow rule in forwarding devices) and flow table size in switches.

Flow setup time is an important factor to be considered, as a new flow in the network is delayed at least two RTTs in forwarding devices (i.e., communication between the ASIC and the management CPU and between that CPU and the controller) [8], so the controller can install the appropriate rules at switches. Furthermore, when inserting high-priority rules at the Ternary Content-Addressable Memory (TCAM), switches must move down the table all other entries with lower priorities (which takes more time as flow table size increases). These delays may be impractical for latency-sensitive flows (adding even 1 ms of latency to these flows is intolerable [17]).

Flow tables, in turn, are a restricted resource in commodity switches, as TCAMs are typically expensive [8], [10]. Such devices usually have a limited number of entries available for OpenFlow rules, which may not be enough when considering that large-scale datacenter networks can have an elevated number of flows per second [11] (and approximately more than 1,500 new flows per second per rack [18]).

III. Predictor

Predictor implements our novel and low-overhead strategy to provide fine-grained management and predictable sharing inside the cloud DCN. It was designed taking four requirements into consideration: scalability, resiliency, predictable and guaranteed network performance and high network utilization. First, scalability is essential for the system to be realistic, as the network sharing strategy must scale to hundreds of thousands of VMs and to the heterogeneous workloads of cloud applications. Second, resiliency to churn at flow- and application-layer, as DCNs have high rate of new flows per second [11] and datacenters can have high rate of application allocation and deallocation [15], respectively. Third, predictable and guaranteed network performance in order to allow applications to maintain a certain level of performance even when the network is congested. Finally, high network utilization, so that all spare bandwidth can be used independently of the bandwidth guarantees assigned to VMs.

An overview of Predictor is shown in Figure 1. The system is composed of five components: Predictor controller, allocation module, application information base (AIB), network information base (NIB) and OpenFlow controller. They are discussed next.

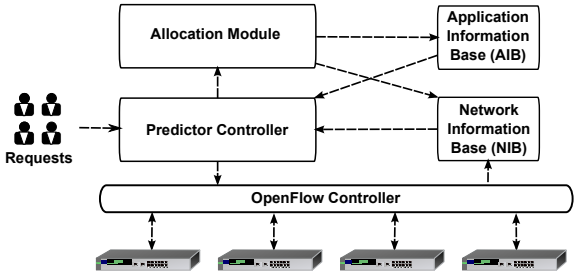


Fig. 1: Predictor system overview.

Predictor Controller. It receives both applications to be allocated (whose resources to be used are determined by the allocation module) and requests for inter-application communication (detailed in Section III-B). In case of an incoming application, it sends the request to the allocation module. Once the allocation is completed (or if the request is for inter-application communication), the Predictor controller generates and sends appropriate flow rules to the OpenFlow module. The OpenFlow module, then, updates the tables (of forwarding devices) that need to be modified.

The controller installs rules to identify flows at application-layer (more details in Section III-E). It can also take advantage of flow management at lower levels (for instance, by matching source and destination MAC and IP fields), since it uses the OpenFlow protocol. Nonetheless, given the amount of resources available in commodity switches and the number of flows that come and go in a small period of time, such low-level rules are expected to be kept to a minimum.

Allocation Module. This component is responsible for allocating incoming applications at the cloud platform, according to available resources. It receives requests from the Predictor controller, determines the set of resources to be allocated for each new request and updates the AIB and NIB. We detail the resource allocation logic in Section III-C.

Application Information Base (AIB). It keeps detailed information regarding each allocated application, including its identifier (ID), VM-to-server mapping, IP addresses, bandwidth guarantees, network weight (for work-conserving sharing), links being used and other applications it communicates with. It provides information for the Predictor controller to compute flow rules that need to be installed in switches.

Network Information Base (NIB). It is composed of a database of resources, including hosts, switches, links and their capabilities (such as rate-limiters, and link capacity and latency). In general, it keeps information about computing and network state, which are received from the OpenFlow controller (current state) and the allocation module (resources used for newly allocated applications). The Predictor controller uses information stored in the NIB to map logical actions (e.g., intra- or inter-application communication) into the physical network. While AIB maintains information at application granularity, NIB keeps information at network layer. The design of NIB was inspired by Onix [16] and PANE [19].

OpenFlow Controller. It is responsible for communication to/from forwarding devices and Open vSwitches at hypervisors, in order to update network state and get information from the network. It receives information from the Predictor controller to modify flow tables in forwarding devices and updates the NIB upon getting information from the network.

We first describe in detail application requests (Section III-A) and how intra- and inter-application communication is handled (Section III-B), and use such information for the allocation of applications in the datacenter (Section III-C). Then, we present how the system provides work-conserving network sharing (Section III-D) and how key functionalities are implemented (Section III-E).

A. Application Requests

Tenants request applications using the hose-model (similarly to prior work [1], [5]–[7]), in order to capture the semantics of the guarantees being offered. In this model, all VMs of an application are connected to a non-blocking virtual switch through dedicated bidirectional links. Each application a is represented by its resource demand and network weight, formally defined as $< N_a, B_a, w_a >$, with the terms being defined as follows: $N_a \in \mathbb{N}^*$ specifies the number of VMs; $B_a \in \mathbb{R}^+$ represents the bandwidth guarantees required by each VM; and $w_a \in [0, 1]$ indicates the network weight. In particular, the network weight enables residual bandwidth (unallocated, or reserved bandwidth for an application and not currently being used) to be proportionally shared among applications with more demands than their guarantees. Therefore, the total amount of bandwidth available for a VM of application a at a given period of time, following the hose model, is denoted by $B_a + spare(s, v_a)$, where $spare(s, v_a)$ identifies the share of spare bandwidth assigned to VM v of application a located at server s:

$$spare(s, v_a) = \frac{w_a}{\sum_{v \uparrow V_s | v \in V_s} w_v} * SpareCapacity \quad (1)$$

where V_s denotes the set of all co-resident VMs (i.e., VMs placed at server s), $v \uparrow V_s \mid v \in V_s$ represents the subset of VMs at server s that need to use more bandwidth than their guarantees and $SpareCapacity$ indicates the residual capacity of the link that connects the physical server to the ToR switch.

Two assumptions are made for the sake of explanation (these are not limitations), as follows. First, we abstract away non-network resources and consider all VMs with the same amount of CPU, memory and storage. Second, we consider that all VMs of a given application receive the same bandwidth guarantees.

B. Bandwidth Guarantees

Predictor provides bandwidth guarantees for both intra- and inter-application communication. We discuss each one next.

Intra-application network guarantees. Typically, this type of communication represents most of the traffic in DCNs [1]. Thus, Predictor allocates and ensures bandwidth guarantees at application allocation time by proactively installing flow rules and rate-limiters in the network through

OpenFlow[3]. Each VM of a given application a is assigned a bidirectional rate of B_a (as detailed in Section III-A). Limiting the communication between VMs located in the same rack is straightforward, since it can be done locally by the Open vSwitch at each hypervisor.

In contrast, for inter-rack communication, bandwidth must be guaranteed throughout the network, along the path used for such communication. Predictor provides guarantees for this traffic by employing the concept of *VM clusters* (set of VMs of the same application located in the same rack). To illustrate this concept, consider a simplified scenario where a given application a has two clusters: $c_{a,1}$ and $c_{a,2}$. Since each VM of a cannot send or receive data at a rate higher than B_a, traffic between the pair of clusters $c_{a,1}$ and $c_{a,2}$ is limited by the smallest cluster: $rate_{c_{a,1},c_{a,2}} = \min(|c_{a,1}|, |c_{a,2}|) * B_a$, where $rate_{c_{a,1},c_{a,2}}$ represents the calculated bandwidth for communication between clusters $c_{a,1}$ and $c_{a,2}$, and $|c_{a,i}|$ denotes the number of VMs in cluster i of application a. In this case, $rate_{c_{a,1},c_{a,2}}$ is guaranteed along the path used for communication between these two clusters by rate-limiters configured in forwarding devices through OpenFlow.

Inter-application communication. Providing static hose guarantees for this type of communication does not scale for DCNs [1], as bandwidth guarantees for each VM would need to be enforced for communication with all other VMs in the network. Furthermore, tenants cannot be expected to know before allocation all other applications and services that their applications will communicate with. Instead, Predictor dynamically sets up guarantees for inter-application communication according to the needs of applications and residual bandwidth in the network. The Predictor controller provides two ways of establishing guarantees for communication between VMs of distinct applications and services, as follows.

Reacting to new flows in the network. When a VM needs to exchange data with one or more VMs of another application, it can simply send packets to those VMs. The hypervisor (through its Open vSwitch) of the server hosting the source VM receives such packets and, since they do not match any rule, sends them to the OpenFlow controller. The Predictor controller will, then, be called to determine the rules needed by the new flows, and the set of rules will be installed along the paths in the network.

Receiving communication requests from applications. Prior to initiating the communication with VMs belonging to other applications, the source VM can send a request to the Predictor controller for communication with VMs from other application(s). This request is composed of the set of destination VMs, the bandwidth needed and the expected amount of time the communication will last. Upon receiving the request, the Predictor controller verifies residual resources in the network, sends a reply, and, in case there are enough available resources, generates and installs the appropriate set of rules for this communication. This approach is similar to providing an API for applications to request network resources, which is employed by PANE [19].

[3]While Predictor may overprovision bandwidth at the moment applications are allocated, it does not waste bandwidth because of its work-conserving strategy (explained in Section III-D). Without overprovisioning bandwidth at first, it would not be feasible to provide bandwidth guarantees for applications (as DCNs are typically oversubscribed).

C. Resource Allocation

The allocation process is responsible for performing admission control and mapping application requests in the datacenter infrastructure. A valid allocation must satisfy two requirements: computing and network resource availability [20]. For simplicity, we discuss Predictor and its allocation component in the context of traditional tree-based topologies implemented in current datacenters [7].

We design a location-aware heuristic to efficiently allocate tenant applications in cloud platforms. The key principle is minimizing bandwidth for intra-application communication (thus allocating VMs of the same application as close as possible to each other), since this type of communication generates most of the traffic in DCNs (as discussed before).

Algorithm 1 allocates one application at a time, as requests are received. First, it searches for the best placement in the infrastructure for the incoming application via dynamic programming. To this end, three data structures are defined and dynamically initialized for each request: i) set R_a stores subgraphs with enough computing resources for application a; ii) V_s^a stores the total number of VMs of application a the s-rooted subgraph can hold; and iii) C_s^a stores the number of VM clusters that can be formed in subgraph s. The algorithm traverses the topology starting at rack level, and up to the core, and determines subgraphs with enough available resources to allocate the incoming request (lines 2 - 12).

Algorithm 1: Location-aware algorithm.

Input : Physical infrastructure P, Application a
Output: Success/Failure code $allocated$

1 $R_a \leftarrow \varnothing$;
2 **foreach** *level l of P* **do**
3 **if** l == *1* **then** // Top-of-Rack switches
4 **foreach** *ToR r* **do**
5 $V_r^a \leftarrow$ number of available VMs in the rack;
6 $C_r^a \leftarrow 1$;
7 **if** $V_r^a \geq N_a$ **then** $R_a \leftarrow R_a \cup \{r\}$;
8 **else** // Aggregation and core switches
9 **foreach** *Switch s at level l* **do**
10 $V_s^a \leftarrow \sum_{w \in \{\text{set of directly connected switches at level } l-1\}} V_w^a$;
11 $C_s^a \leftarrow \sum_{w \in \{\text{set of directly connected switches at level } l-1\}} C_w^a$;
12 **if** $V_s^a \geq N_a$ **then** $R_a \leftarrow R_a \cup \{s\}$;
13 $allocated \leftarrow$ failure code;
14 **while** *Application a not allocated and R_a not empty* **do**
15 $r_a \leftarrow$ Select subgraph from R_a;
16 $allocated \leftarrow$ Allocation of VMs and bandwidth for application a at r_a;
17 **return** $allocated$;

After verifying the physical infrastructure and determining possible placements, the algorithm selects one subgraph r_a at a time to allocate the application (lines 13 - 16). The selection of a candidate subgraph takes into account the number of VM clusters. Therefore, the selected subgraph is the one with the minimum number of VM clusters, so that VMs of the same application are allocated close to each other, reducing the amount of bandwidth needed for intra-application communication (recall that the network often represents the bottleneck when compared to computing resources). When a subgraph is selected, the algorithm allocates the application, reserving bandwidth for communication between its VMs as presented in Section III-B.

Finally, the algorithm returns a success code if application a was allocated or a failure code otherwise (line 17). Applications that could not be allocated upon arrival are discarded, similarly to Amazon EC2 admission control.

D. Work-Conserving Rate Enforcement

Predictor provides bandwidth guarantees with work-conserving sharing. This is because only enforcing guarantees through static provisioning results in underutilization and fragmentation [7], while only work-conserving sharing does not provide strict guarantees for tenants [1]. Therefore, in addition to ensuring a base level of guaranteed rate, Predictor can proportionally share available bandwidth among applications with more demands than their guarantees, as defined in Equation 1.

We design an algorithm to periodically set the allowed rate for each co-resident VM on a server. In order to provide smooth interaction with TCP, we follow ElasticSwitch [7] and execute the work-conserving algorithm between periods of time one order of magnitude larger than the network round-trip time (RTT), i.e., 10 ms instead of 1 ms.

Algorithm 2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time [21]). It receives as input the list of VMs (V_s) hosted on the server (s) and their current demands (which are determined by monitoring VM socket buffers, similarly to Mahout [22]). First, the rate of each VM with demands less or equal than its bandwidth guarantees (represented by $v \downarrow V_s \mid v \in V_s$) is calculated and enforced (lines 1 - 2). Then, the algorithm calculates the residual bandwidth of the link connecting the server to the ToR switch (line 3). The residual bandwidth is calculated by subtracting from the link capacity the guarantees of VMs with higher demands than their guarantees (represented by $v \uparrow V_s \mid v \in V_s$) and the rate of VMs with less or equal demands than their guarantees.

Algorithm 2: Work-conserving rate allocation.

Input : VM set V_s, Current demand of VMs $demand$
Output: Rate $nRate$ for each co-resident VM

1 **foreach** $v \downarrow V_s \mid v \in V_s$ **do**
2 $\quad nRate[v] \leftarrow \min(B_v, demand[v])$;
3 $residual \leftarrow LinkCapacity - \left(\sum_{v \uparrow V_s} B_v + \sum_{v \downarrow V_s} nRate[v]\right)$;
4 **foreach** $v \uparrow V_s \mid v \in V_s$ **do**
5 $\quad nRate[v] \leftarrow$
$\quad B_v + \min\left(demand[v] - B_v, \left(\frac{w_v}{\sum_{v \uparrow V_s} w_v} \times residual\right)\right)$;
6 **return** $nRate$;

The last step establishes the bandwidth for VMs with higher demands than their guarantees (line 4 - 5). The rate is determined by adding the guarantees (B_v) and the minimum bandwidth between i) the difference of the current demand ($demand[v]$) and the guarantees (B_v); and ii) the proportional share of residual bandwidth the VM would be able to receive according to its weight w_v. By calculating the minimum bandwidth between these two values, Predictor guarantees that VMs will not receive more bandwidth than they need (which would waste network resources). With this approach, the algorithm can adapt to the significant variable communication demands of applications.

In summary, if the demand of a VM exceeds its guaranteed rate, data can be sent and received at least at that guaranteed rate. Otherwise, if it doesn't, the unutilized bandwidth will be shared among co-resident VMs whose traffic demands exceed their guarantees.

E. Key Implementation Aspects

Predictor relies on two key aspects: i) identifying flows at application-layer; and ii) providing network guarantees and dynamically enforcing rates for VMs. Figure 2 shows how our proof-of-concept implementation of Predictor handles these aspects, which are discussed next.

First, to perform application-layer flow identification, Predictor utilizes Multiprotocol Label Switching (MPLS). More specifically, applications are identified in OpenFlow rules through the label field in the MPLS header. The MPLS label is composed of 20 bits, which allows Predictor to identify 1,048,576 different applications. The complete operation of identifying and routing packets at application-layer works as follows. For each packet received from the source VM, the Open vSwitch (controlled via the OpenFlow protocol) in the hypervisor pushes a MPLS header (four bytes) with the application ID of the source VM in the label field. Subsequent switches in the network use both MPLS label and IP destination address (which may be wildcarded, depending on the possibilities of routing) matching fields to choose the correct output port to forward incoming packets. When packets arrive at the destination hypervisor, the Open vSwitch pops the MPLS header and forwards the packet to the correct VM.

Second, Predictor runs a local controller at hypervisor-level of each server in order to rate-limit VMs. More precisely, the local controller installs the appropriate rules at the Open vSwitch to dynamically set the allowed rate for each hosted VM. Such rate is calculated by Algorithm 2, discussed in Section III-D. Note that Predictor also reduces rate-limiting overhead when compared to previous schemes (e.g., Hadrian [1], CloudMirror [2] and ElasticSwitch [7]), as it only rate-limits the source VM while other schemes rate-limit each pair of source-destination VMs.

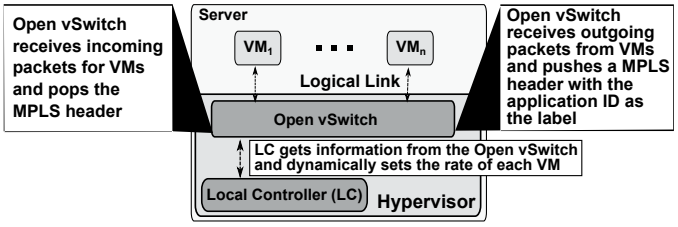


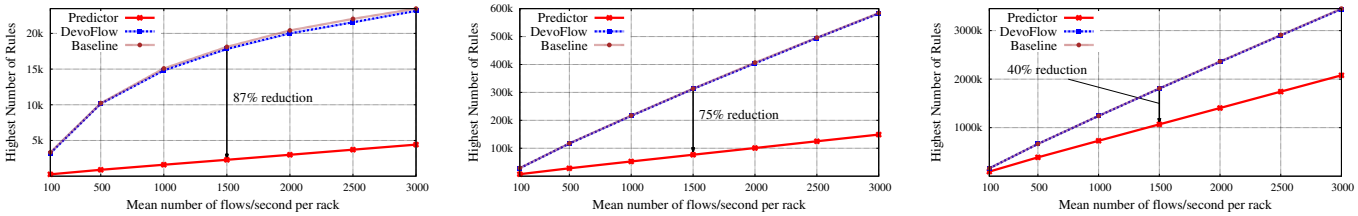
Fig. 2: Server-level implementation of Predictor features.

IV. Evaluation

We focus on showing that Predictor (i) can scale to large DCNs; (ii) provides both predictable network performance (with bandwidth guarantees) and work-conserving sharing; and (iii) outperforms existing schemes for DCNs (the baseline SDN/OpenFlow controller and DevoFlow [8]).

A. Setup

Environment. We have implemented a simulator that models an IaaS multi-tenant datacenter. The network is defined as a tree-like topology, similar to current DCNs. It is composed of a three-tier topology with 16,000 servers at level 0; each server has 4 VM slots, resulting in a total amount of 64,000 available VMs. Every 40 machines form a rack, and every 20 ToRs are connected to an aggregation switch. Finally, all



(a) At hypervisors. (b) At ToRs. (c) At aggregation switches.

Fig. 3: Highest number of rules at forwarding devices.

aggregation switches are connected to a core switch. The capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 50 Gbps for aggregation-core links. Therefore, the default oversubscription of the network is 4.

Workload. The workload is composed of incoming application requests (to be allocated in the cloud platform) arriving over time. In particular, we consider an heterogeneous set of applications, including MapReduce and Web Services. As defined in Section III-A, each application a is represented as a tuple $< N_a, B_a, w_a >$. Given the lack of proper traces for DCNs, the workload was generated in line with related work [1], [6]. N_a is exponentially distributed around a mean of 49 VMs (following measurements from prior work [15]). B_a was generated by reverse engineering the traces used by Benson et al. [11] and Kandula et al. [18]. More specifically, we used their measurements related to inter-arrival flow-time and flow-size at servers to generate and simulate network demands of applications. Of all traffic, 20% of flows are destined to other applications [1]. We pick the destination of each flow by first determining whether it is an intra- or inter-application flow, and then uniformly select a destination. The weight w_a is uniformly distributed in the interval $[0, 1]$.

B. Results

We begin by examining the scalability of employing Predictor on large DCNs. To this end, we verify controller load as well as the number of rules in flow tables, as these are typically the factors that restrict scalability the most [7], [9]. We compare Predictor against the baseline SDN/OpenFlow controller and the state-of-the-art controller for DCNs (DevoFlow [8]). In the baseline scheme, switches forward to the controller packets that do not match any rule in the flow table (we consider the default behavior of OpenFlow versions 1.3 and 1.4 upon a table-miss event). The controller, then, responds with the appropriate set of rules specifically designed to handle the new flow. DevoFlow, in turn, considers flows at the same granularity than the baseline. However, forwarding devices, through the use of more powerful hardware and template rules, generate rules for small flows (without involving the controller) and forward only packets of large flows to the controller. Therefore, switches have similar number of rules when compared to the baseline, but controller load is significantly reduced.

Reduced number of flow table entries. Figure 3 shows how the offered load (in new flows per second per rack) affects the occupancy of flow tables. More precisely, the plots show in Figures 3(a), 3(b) and 3(c), respectively, the largest number of entries required at any hypervisor, ToR and aggregation switch for a given mean rate of new flows at each rack (in core devices, results are not shown, as Predictor provides negligible gains).

In all three plots, we see that the mean number of arriving flows during an experiment affects directly the number of rules needed in devices. Results in Figure 3(a) present a logarithmic behavior according to the mean number of new flows, while in Figures 3(b) and 3(c), a linear one. These results are explained as follows: (i) the number of different flows that pass through ToR and aggregation switches is large and may quickly increase due to the elevated number of end-hosts (VMs) and arriving flows in the network; and (ii) in hypervisors, the number of rules (and distinct flows) is smaller (with a reduced growth when increasing the number of new flows in the network) because of the limited number of VMs hosted at each physical server.

Overall, the increase of the total number of flows requires more rules for the correct operation of the network (according to the needs of tenants) and enables richer communication patterns (representative of cloud datacenters [1]). Additionally, the number of rules for the baseline and DevoFlow is similar because (i) they consider flows at the same granularity; and (ii) the same default timeout for rules was adopted.

Because Predictor manages flows at application-layer and also wildcards the destination address in rules when possible (as explained in Section III-E, when the routing process was described), it significantly reduces the number of rules needed in forwarding devices, especially for realistic numbers of flows in large-scale DCNs (i.e., higher than 1500 new flows per second per rack [7]). In fact, Predictor reduces the number of rules up to 87% at hypervisors, 75% at ToRs and 40% at aggregation devices. In core devices, the reduction is negligible (around 1%), because (a) a high number of flows does not need to traverse core links to reach their destinations, thus baseline and DevoFlow (which handle flows at VM-to-VM granularity) do not install many rules at core devices, while Predictor installs application-layer rules at these devices; and (b) Predictor proactively installs rules for intra-application traffic (while other schemes install rules reactively).

In summary, Predictor reduces the number of rules required in forwarding devices, which can (i) minimize the amount of resources occupied by rules in TCAMs of forwarding devices (TCAM is a very expensive resource [10]); (ii) improve hypervisor performance (as measured by LaCurts et al. [23]); and (iii) minimize the time needed to install high-priority rules in TCAMs, since all lower-priority rules in the table must be moved down to perform this operation.

Small controller load. As DCNs typically have a high

number of flows per second, it is important that the controller is able to efficiently handle flow setups. Figure 4 shows the flow request rate the controller needs to be able to process in each case. As expected, the number of messages sent to the controller increases according to the mean number of new flows per rack, because the controller must set up network paths and allocate resources for these new flows (i.e., flows without matching rules in forwarding devices). The baseline imposes a higher load to its controller than DevoFlow (recall that DevoFlow only requires controller intervention to install rules for large flows, at the cost of more powerful hardware at forwarding devices). Predictor, in turn, proactively installs application-layer rules for intra-application communication at allocation time and reactively sends rules for inter-application traffic upon receiving communication requests, significantly reducing the number of flow requests when compared to the baseline (around 77%).

In contrast, Predictor requires more flow request messages than DevoFlow (around 22% more, considering the total number of flows in the network), most of which are from inter-application traffic. Nonetheless, Predictor controller load can be reduced by allowing tenants to specify at allocation time some (or all) other applications and services that their applications will communicate with (at the cost of some burden, as tenants will need more knowledge of their applications).

In general, the Predictor controller is aware of all traffic in the network, while the DevoFlow controller has knowledge of only large flows (approximately 50% of the total traffic [11]), to perform fine-grained management in the network.

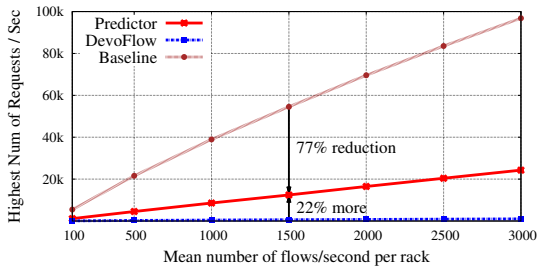
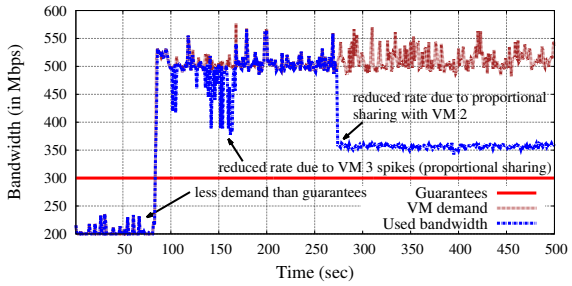


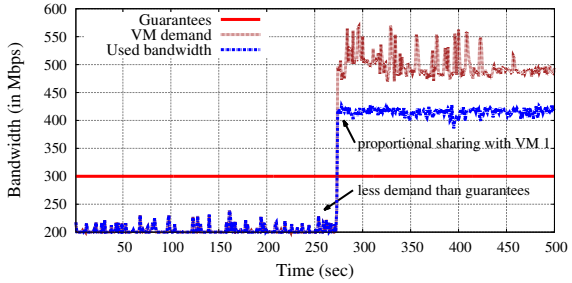
Fig. 4: Controller load.

After verifying the feasibility of employing Predictor on DCNs, we turn our focus to the challenge of bandwidth sharing unfairness. In particular, we show that Predictor can provide minimum bandwidth guarantees with work-conserving resource sharing under worst-case scenarios, achieving both predictability for tenants and high utilization for providers.

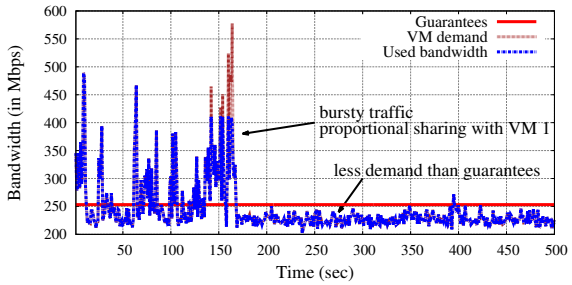
Minimum bandwidth guarantees for VMs. We define it as follows: the VM rate should be (a) at least the guaranteed rate if the demand is equal or higher than the guarantees; or (b) equal to the demand if it is lower than the guarantees. To illustrate this point, we show, in Figure 5, the set of VMs (in this case, three VMs from different applications) allocated on a given server during a predefined time period of an experiment. Note that VM1 [Figure 5(a)] and VM2 [Figure 5(b)] have similar guarantees, but receive different rates ("used bandwidth"). This happens because they have different network weights, and the rate is calculated according to their demands, bandwidth guarantees, network weight and residual bandwidth. In summary, we see that Predictor provides minimum bandwidth guarantees for VMs, since the actual rate of each VM is always equal or higher than the minimum between the application demand and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.



(a) VM 1.



(b) VM 2.



(c) VM 3.

Fig. 5: Bandwidth allocation for VMs on a given server during a predefined period of time.

Work-conserving network sharing on servers. It means that bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among other VMs with more demands than their guarantees (according to the weights of each application). Figure 6 shows the aggregate bandwidth[4] on the server holding the set of VMs in Figure 5. In these two figures, we verify that Predictor provides work-conserving sharing in the network, as VMs can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high network utilization.

In general, Predictor provides significant improvements

[4]Note that Predictor considers bandwidth guarantees when allocating VMs (i.e., it does not take into account temporal demands). Therefore, even though the sum of temporal demands of all VMs allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these VMs will not exceed the link capacity.

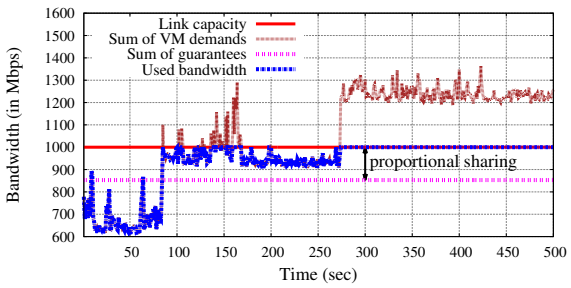


Fig. 6: Work-conserving sharing on a given server.

over DevoFlow, as it allows high utilization and fine-grained management in the network for providers and network predictability with guarantees for tenants and their applications. As a side effect, Predictor has higher controller load than DevoFlow (the cost of providing fine-grained management in the network).

V. Discussion

Application-layer flow identification. In our proof-of-concept implementation, Predictor identifies flows at application-layer through an MPLS label (application ID with 20 bits). Therefore, it needs an MPLS header in each packet (adding four bytes of overhead). In practice, when considering the matching fields defined by OpenFlow, application-layer flows could also be identified by utilizing IEEE standard 802.1ad (Q-in-Q) with double VLAN tagging. The advantage of double tagging is higher number of IDs available (24 bits), while the drawback is an overhead of eight bytes (two VLAN headers) per packet.

Dynamic rate allocation with feedback from the network. The designed work-conserving algorithm does not take into account the network feedback provided by the OpenFlow module. This design choice was deliberately made; we aim at reducing the amount of management traffic in the network, since DCNs are typically oversubscribed networks with scarce resources [6]. Nonetheless, the algorithm can be easily extended to consider feedback, which can further help controlling the bandwidth used by flows traversing congested links.

Application ID management. Predictor controller assigns IDs for applications (in order to identify flows at application-layer) upon allocation and releases IDs upon deallocation. Therefore, ID management is straightforward, as Predictor has full control over which IDs are in use at each period of time.

VM migration. Ideally, VM migration should be transparent to tenants. Predictor can maintain IPs and guarantee connectivity of VMs upon migration by readily installing new flow rules (and, if needed, removing old rules) in the appropriate forwarding devices.

Failures. Link and forwarding device failure impacts the traffic using the failed resource. By providing a logically centralized control (with real-time network state gathered by the OpenFlow module), Predictor can quickly detect failures and react accordingly, for instance, by rerouting the affected traffic (i.e., installing alternative paths).

VI. Related Work

Researchers have proposed several schemes to enable the use of the SDN paradigm in DCNs and to improve management and network sharing among tenants. Proposals related to Predictor can be divided into three classes, as follows.

OpenFlow controllers. DevoFlow [8] and DIFANE [12] propose to devolve control to the data plane. The first one introduces new mechanisms to detect elephant flows and make routing decisions at forwarding devices, while the second keeps all packets in the data plane. These schemes, however, require more complex, customized hardware at forwarding devices. Kandoo [13] provides a logically distributed control plane for large-scale networks. Nonetheless, it presents scalability issues when communication occurs between VMs located at different racks (a common case in DCNs).

Distributed rate-limiting. CloudMirror [2], Oktopus [5], Proteus [6] and Choreo [23] provide strict bandwidth guarantees for tenants by isolating applications in virtual networks. Despite their benefits, these approaches are not work-conserving and address only intra-application communication. ElasticSwitch [7] and EyeQ [24] attempt to provide strict bandwidth guarantees with work-conservation. However, they focus only on intra-application communication, and EyeQ cannot provide guarantees upon core-link congestion [25]. Finally, Hadrian [1] introduces a strategy that considers inter-application communication, but (i) it requires a larger, custom packet header (hindering its deployment); and (ii) its switches must dynamically perform rate calculation (and enforce such rate) for each flow in the network.

Flow table management. FasTrak [26] seeks to reduce hypervisor overhead by managing ToR devices and hypervisors as a unified set, moving rules back and forth. Despite the benefits, it (i) requires generic routing encapsulation (GRE) between the source and destination ToRs (20 bytes of data overhead per packet); (ii) may add latency for flows when moving their respective rules between hypervisors and ToRs; and (iii) cannot precisely rate-limit VMs (as their rate must be dynamically calculated by both ToRs and hypervisors according to the rules at each one of them).

VII. Final Remarks

We have introduced Predictor, a system that takes advantage of the SDN paradigm to provide fine-grained network management and predictable bandwidth sharing among applications in DCNs. To achieve this goal, Predictor addresses the scalability challenges of OpenFlow: i) it minimizes flow setup time by proactively installing rules for intra-application communication at allocation time (since this type of communication represents most of the traffic in DCNs); and ii) it reduces the number of flow rules in forwarding devices by managing flows at application-layer. Evaluation results show the benefits of Predictor, which can scale to large networks and provide predictable network performance. In future work, we will address Predictor's placement in DCNs, as controller placement is an important challenge of SDNs [27].

Acknowledgements

This work has been supported by MCTI/CNPq/Universal (Project Phoenix, 460322/2014-1).

REFERENCES

[1] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *USENIX NSDI*, 2013.

[2] J. Lee, Y. Turner, M. Lee, L. Popa, J.-M. Kang, S. Banerjee, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, 2014.

[3] J. Guo, F. Liu, X. Huang, J. C. Lui, M. Hu, Q. Gao, and H. Jin, "On Efficient Bandwidth Allocation for Traffic Variability in Datacenters," in *IEEE INFOCOM*, 2014.

[4] R. Shea, F. Wang, H. Wang, and J. Liu, "A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environment," in *IEEE INFOCOM*, 2014.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[6] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *ACM SIGCOMM*, 2012.

[7] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing," in *ACM SIGCOMM*, 2013.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM*, 2011.

[9] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," vol. PP, no. 99, 2014, pp. 1–29.

[10] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *IEEE INFOCOM*, 2014.

[11] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[12] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *ACM SIGCOMM*, 2010.

[13] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, 2012.

[14] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: sharing the network in cloud computing," in *ACM SIGCOMM*, 2012.

[15] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *USENIX OSDI*, 2010.

[17] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM IMC*, 2009.

[19] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," in *ACM SIGCOMM*, 2013.

[20] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, "Hierarchical network-aware placement of service oriented applications in clouds," in *IEEE/IFIP NOMS*, 2014.

[21] D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

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

[23] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: Network-aware task placement for cloud applications," in *ACM IMC*, 2013.

[24] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: practical network performance isolation at the edge," in *USENIX NSDI*, 2013.

[25] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 2139–2147.

[26] R. Niranjan Mysore, G. Porter, and A. Vahdat, "FasTrak: Enabling Express Lanes in Multi-tenant Data Centers," in *ACM CoNEXT*, 2013.

[27] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *IFIP/IEEE IM*, 2013.