HERCULES: Integrated Control Framework for Datacenter Traffic Management

Wonho Kim
Princeton University
Princeton, NJ, USA
Email: wonhokim@cs.princeton.edu

Puneet Sharma
HP Labs
Palo Alto, CA, USA
Email: puneet.sharma@hp.com

***Abstract*—The large scale and high performance requirements of Cloud computing pose many challenges to the data center network operators. These networks typically require high bisection bandwidth, strict performance isolation, and power-efficient operation. Recently, many researchers have proposed various network controller systems for programming networks, each of them to individually address only one of these challenges. In this paper, however, we show that running multiple controllers in a shared network fabric independently is not only inefficient but conflicting control decisions by different controllers can also lead to serious network performance degradation. We present HERCULES, an integrated control framework, to enable coexistence and operation of multiple controllers. As an initial proof of concept, we have integrated four existing controllers into HERCULES framework. Our evaluation of the integrated controller shows that it can enable multiple controllers to leverage each other and collectively achieve the multiple goals of the controllers simultaneously.**

I. INTRODUCTION

Large-scale datacenters are designed to consolidate diverse applications from multiple tenants onto a shared network fabric. To meet given SLAs while reducing management costs, such datacenters typically impose complex requirements on network operators. The network fabrics in datacenters need to provide high bisection bandwidth for bandwidth-intensive applications [1], [2], strict performance isolation for QoS sensitive flows [3], [4] and energy-efficient network operations [5], [6] for improving energy efficiency and proportionality of the networks.

In recent times, researchers have proposed a variety of systems that address these requirements [4], [7], [8], [9], [10], [11]. A common approach adopted by the existing systems is to introduce a logically centralized network controller into network management framework; which is feasible because datacenter networks are typically autonomous and under control of single management domain. These network controllers collect traffic statistics, manage state of network devices, and dynamically program/configure the devices to achieve particular goals such as high-bisection bandwidth, network resource slicing, and network energy efficiency.

Datacenter networks need to provide all of these benefits simultaneously. However, our experience of developing network controllers reveals that it is extremely difficult to collectively achieve the goals of multiple controllers. Simply, running multiple controllers independently has several limitations that make them unsuitable for production datacenters. An obvious problem is increased management complexity because of the redundant state maintained across multiple controllers. A more serious problem is that these controllers could make conflicting decisions due to the lack of information exchange amongst them. Our experiment scenarios demonstrate that running independent controllers can lead to serious performance collapse in the network.

To make network controllers efficiently deployable in datacenters, it is essential to enable the controllers to achieve coordinated control of the shared fabric. In this paper, we present the design and implementation of HERCULES, an integrated control framework upon which multiple controllers can be implemented without compromising on their original goals and benefits.

The main challenge is that existing controllers are designed based on radically different assumptions about capabilities of network elements. Some controllers assume intelligent network switches and unmodified endhosts [4], [12] while other controllers leverage control points in endhosts connected with commodity switches [7], [9]. The design decisions in those controllers are made to optimize for scalability, flow setup latency, and resource utilization. Therefore, extending one type of controller to include additional goal will not only require extensive re-implementation but might also compromise on the benefits of original controllers. This constraint also rules out existing control frameworks that depend on specific functions in network elements such as OpenFlow-enabled switches [13], [14].

HERCULES architecture extends the control of shared network infrastructure to endhosts in datacenters. We identify primitive endhost functions that existing endhost-based controllers need to access for their operation. HERCULES agents run in endhosts to provide remote interface to the defined functions. HERCULES central manager manages common network state that can be shared by multiple controllers. This eliminates the redundant state management across multiple controllers, and enables interactions between the controllers through the shared state. HERCULES also provides a notification mechanism so that the controllers can promptly adjust their control actions to certain updates from other controllers. Using HERCULES, we integrated four existing controllers that we developed for traffic management in datacenter. Our evaluation results show that the integrated controller can collectively achieve the goals of the controllers without performance degradation.

978-1-4673-0269-2/12/$31.00

The key contributions of our work are: (1) elucidating the coordination and conflict issues when running multiple independent controllers in a shared fabric, (2) design and implementation of an integrated control framework, and (3) experiments demonstrating the benefits of the integrated controller using datacenter topologies.

This paper is organized as follows: Section II presents the details of the existing controllers that are instantiated collectively using the HERCULES framework. The HERCULES architecture and functioning of its components are described in Section III. In Section IV we describe how the existing controllers are integrated together and share state information via HERCULES modules. The results from our evaluation study using an emulated testbed are discussed in Section V. This is followed by related work and conclusion in Sections VI and VII respectively.

II. BACKGROUND

Network controllers are management servers that provide dynamic configuration of network elements to achieve particular performance goals. For instance: (a) *multi-pathing controllers* [7], [9], [10] manage routing of flows in redundant topology for high-bisection bandwidth; (b) *slicing controllers* [4] configure QoS knobs embedded in network switches for performance isolation between tenants; (c) *large-flow controllers* [8], [15] detect and repin elephant flows for improving bandwidth utilization; and (d) *power controllers* [5] attempt to power off inactive network elements for saving network energy in datacenters.

We now provide a brief description of various controllers that we integrate into HERCULES framework.

SPAIN and HBR: There is a fair amount of redundancy in the datacenter networks for protection against failure. This redundancy can also be exploited by multi-pathing controllers to increase the bisection bandwidth. SPAIN [9] and HBR [10] are two such multi-pathing controllers that have been recently proposed. These controllers pre-install a set of VLANs to be used for multi-pathing. SPAIN and HBR have smart algorithms to distribute active flows over the available VLANs. The decision to route a flow over a particular VLAN is dependent on the current traffic conditions in the network. SPAIN updates the weights associated with each VLAN at each edge switch based on the network traffic load. Similarly HBR changes the flow assignment to VLANs by remapping the flows to different hash buckets.

QoS controller: We proposed an OpenFlow based QoS controller [4] to automatically slice the network resources to meet the QoS requirements of the multiple tenants or applications sharing the common network fabric. The network operators can specify policies regarding the end-to-end QoS requirements for various tenants' and/or application flows. When new flows are detected by OpenFlow stubs in the edge switches, the QoS controller computes the resource provisioning (such as priority tags and rate limits) based on the flow's QoS requirements and the traffic load of already active flows in the network. The controller inserts flow-matching based rate limiters as well as proper setting of priority levels for the admitted flow at each switch along the path.

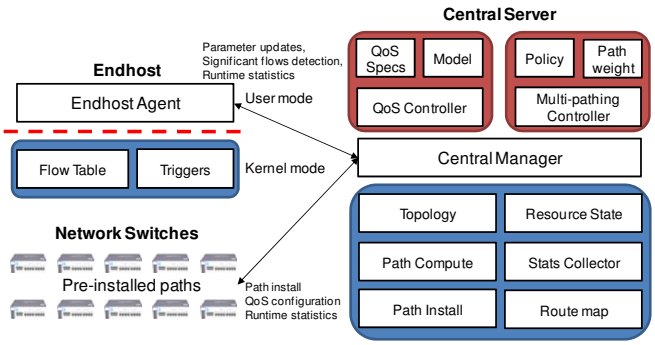


Fig. 1: HERCULES architecture. HERCULES consists of endhost agents and central manager. Endhost agent has flow table for routing of flows and trigger table for detection of significant flows. The central manager implements a set of modules that are shared by multiple network controllers.

Mahout controller: For effective utilization of datacenter fabric, it is important to detect elephant flows and distribute them smartly on different paths to avoid flow collisions. Hedera [15] shows that scheduling elephant flows can deliver up to 113% higher aggregate throughput than static hash-based ECMP scheme in datacenter testbed. We developed a large-flow controller, Mahout [8], that leverages endhosts for timely detection of elephant flows. Mahout installs a shim layer in endhosts that monitors socket send buffer. If the buffer crosses a certain threshold, the shim layer marks outgoing packets using an in-band signaling mechanism to notify the Mahout controller of the large flow. The Mahout controller can detect elephant flows two orders of magnitude faster with much lower overhead than polling-based approach used in Hedera.

While all these controllers have independent optimization/efficiency goals, they rely on the state information about the same network and also control the same network elements. Hence uncoordinated control of same fabric can have detrimental consequences.

III. HERCULES ARCHITECTURE

In this section we detail the design of HERCULES, an integrated network control framework that allows multiple controllers to co-exist without any major changes to controller code as well as without any adverse impact on the network performance. As shown in Figure 1, HERCULES consists of two primary components, namely, *endhost agent* and *central manager*. Each of these components implements common modules that are shared amongst various controllers for state and information exchange.

A. Endhost Agent

HERCULES endhost agent provides an interface to two primitive elements in endhost: *flow table* and *trigger table*. The flow table has rules for handling incoming and outgoing packets in the endhost. The trigger table contains rules to

detect significant flows that the controllers in the central manager need to be notified about. The flow table and the trigger table are implemented as kernel modules because their per-packet processing can be done most efficiently in the kernel mode.

Multi-pathing controllers use the flow table for spreading flows over multiple redundant paths for high-bisection bandwidth. The rules in the flow table are not limited to packet forwarding. Some controllers (VL2 [7], PortLand [12]) can use the flow table for converting logical address to physical address in packets. Slicing controllers can use the flow table for assigning specific paths and priority tags to QoS-sensitive flows. Likewise, large-flow controllers can insert rules to route detected elephant flows to the paths to reduce conflicts amongst the large flows in the network.

The flow/trigger tables are similar to OpenFlow protocol that defines an interface for managing rules in network switches. However, HERCULES endhost agent is able to support more powerful control actions than OpenFlow's packet forwarding rules because the implementation of the agents is not constrained by the capabilities of switch hardware. For example, the HERCULES flow table supports hash-based path selection and per-bucket statistics such as those described in [10]. Similarly, the HERCULES trigger table can be defined in a more flexible manner than the packet-header matching in OpenFlow. For instance, our large-flow controller uses the trigger table for installing trigger rules to detect flows based on the socket-buffer monitoring. This trigger can detect elephant flows an order of magnitude faster than periodic polling of statistics from switches. However such an endhost-based detection cannot be used with purely OpenFlow based controllers.

B. Central Manager

HERCULES central manager implements common modules shared by multiple network controllers. These modules have the information about physical topology, resource reservation state in network links, available paths between endhosts, and runtime statistics of the traffic in network. Network controllers run on top of the common modules. Each controller manages its own private modules. For instance, slicing controller needs QoS specs and performance models for performing admission control of new QoS flows. Multi-pathing controllers maintain path weights to control the distribution of traffic load over available paths. Similarly, large-flow controllers maintain a database of active elephant flows in the network.

The physical network topology is either given by network administrators or dynamically discovered using location discovery protocols as in PortLand. From the topology, the multi-pathing controller pre-computes available paths between endhosts. The identified paths need to be installed into network switches before they are used by endhosts. If the switches are OpenFlow-enabled, the central manager could install forwarding rules into the switches. In this work, however, we use VLAN-based path selection. The path installation module in the central manager merges paths into a set of VLAN trees and then remotely configures the VLAN settings in the network switches. The central manager also collects runtime statistics from endhost agents and switches and makes them available to its various controllers. These realtime statistics can be leveraged by different controllers in making informed decisions about flow placements etc. For instance, the multi-pathing controllers can use the runtime statistics to dynamically change the routing behaviors from the endhosts.

C. Network Fabric

In this paper, we assume that the target network fabric is composed of commodity-off-the-shelf (COTS) ethernet switches. From our experience of developing network controllers, we found that it is very difficult to add even a simple feature to the firmware in the network switches due to hardware constraints. Supporting inexpensive COTS switches improves the deployability of HERCULES framework in production datacenters because HERCULES does not require any modification in network switches. The only feature that HERCULES needs from given network fabric is VLAN support, which is standard feature in COTS ethernet switches. HERCULES uses VLANs to install routing paths in a given arbitrary topology with redundant paths. SPAIN shows that a small number of VLANs suffices to cover most types of datacenter topologies.

D. Coordinated Control of Shared Fabric

To collectively achieve the goals of multiple controllers, it is important to control network fabric in a coordinated manner at runtime. In HERCULES, controllers can interact with each other through *shared modules* and *update notifications* in the central manager.

HERCULES controllers share the common modules that the central manager provides. This eliminates the need to synchronize state across multiple controllers, and thus simplifies the implementation and the management of the controllers. In addition, the shared modules can greatly reduce the risk of making conflicting control decisions since the controllers have consistent views on the current network state. Administrators can configure access permission in each module. For example, in our implementation, only slicing controller can update the resource reservation state module. Other controllers have read-only access to the module for updating path weights (in multi-pathing controller) and placement of elephant flow (in large-flow controller).

In HERCULES, a controller can register itself for the notification of updates in certain modules. The central manager delivers the event notifications to the registered controller when any updates occur. In our implementation, the multi-pathing controller receives notification of the updates in the resource reservation state module. When the slicing controller reserves a fraction of bandwidth for a high-priority flow, the event is notified immediately to the multi-pathing controller. Then the multi-pathing controller reacts to the event by adjusting the weights of the affected paths. Our measurement results in Section V confirm that this prompt interaction is critical for avoiding undesirable performance degradation in network.

IV. Integrated Controller

In this section, we describe how we integrated four network controllers (SPAIN [9], HBR [10], QoS controller [4], and Mahout [8]) into the HERCULES central manager. We integrate these four controllers as a proof of concept. However, we believe that other similar network controllers will be easily added to the framework because HERCULES is not dependent on any specifics of the controllers.

We name the new controller *integrated controller*. The four controllers were developed independently for different performance goals. As a result, they were not designed to co-exist in a shared fabric. In the following subsections, we describe how mice flows, QoS flows, and elephant flows are handled in the integrated controller.

A. Multi-pathing of Mice Flows

Both SPAIN and HBR are multi-pathing controllers that share similar features. Either one of these two controllers can be selectively enabled in our integrated controller, but in this paper we focus our discussion on SPAIN for sake of brevity. In the integrated controller, mice flows are handled by the SPAIN controller. For each endhost, the SPAIN controller constructs a route map from the pre-computed paths stored in the central manager. The route map is a database that an endhost can query to find the VLAN IDs that reach a given destination. From the found VLAN IDs, HERCULES endhost agents select paths for outgoing flows based on ECMP-like hash computation.

The VLAN IDs in the route map are associated with *path weights* that define each VLAN's probability to be selected for an outgoing flow. The SPAIN controller initializes the weights in proportion to the available bandwidth on the paths. In network fabrics such as fat-tree [16], all VLANs have the same initial weights because every path has the same end-to-end bandwidth in the topology.

The SPAIN controller communicates the computed route map back to the central manager. Then, the controller installs the route map into each endhost's flow table. The kernel module in the endhost selects VLAN IDs for outgoing flows based on the destinations and the current path weights in the route map.

B. Admission Control of QoS Flows

The integrated controller supports dynamic resource reservation for QoS-sensitive flows in the network. The QoS controller in the central manager takes QoS specifications that define QoS-sensitive flows and their high-level performance requirements. Unlike OpenFlow-based QoS controller, the integrated controller detects the QoS flows in the endhosts by inserting rules into the trigger tables using OpenFlow-like header matching.

Figure 2 presents the process of admission control and resource reservation in the integrated controller. (1) When a new QoS flow arrives in an endhost, the endhost agent triggers the QoS controller in the central manager. (2) The QoS controller finds the information about the available paths

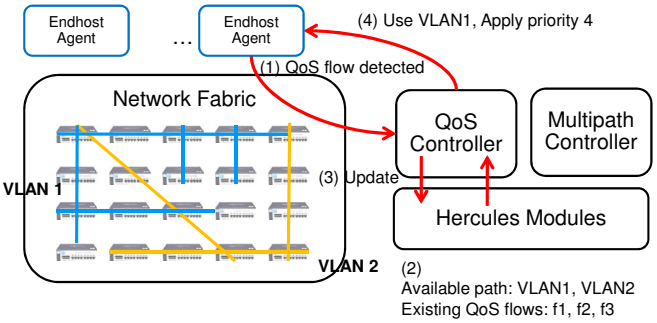


Fig. 2: Handling QoS flows in the integrated controller. In HERCULES framework, QoS flows are detected in the endhosts. Among pre-installed paths in network, the QoS controller finds a path that satisfies the detected flow's requirements. If such a path is found, the flow is admitted into network.

from the route map installed by the SPAIN controller. Then, the QoS controller examines the current resource reservation state of the available paths. (3) If the QoS controller can find a path that has sufficient unreserved bandwidth, it reserves a certain amount of the bandwidth on the path, and updates the reservation state in the central manager. (4) At this point, the flow is admitted into the network. The QoS controller installs a *forwarding* rule that assigns a specific path and priority tag to the subsequent packets from the admitted QoS flow. The rule sets VLAN ID and VLAN Priority Code Point (PCP) field in the outgoing packets.

Like in OpenFlow, the forwarding rule expires after an idle time set by the QoS controller. The endhost agent sends the expiration event to the QoS controller and removes the rule from the flow table. Upon receiving the notification, the QoS controller reclaims the bandwidth reserved for the flow, and updates the resource reservation state in the central manager. If there is a controller registered for the update event, the event is delivered to the controller for the inter-controller control actions. Such update events are key to informing other controllers that might be impacted by the change in resource availability in the network.

C. Adjustment of Path Weights

When a new QoS flow is admitted or an existing flow expires in network, the QoS controller updates the selected path's reservation state in the central manager. The QoS flows typically have higher priority than other mice flows, so the updates in the reservation state also mean the changes in the available bandwidth for mice flows on the affected paths. Therefore, the SPAIN controller needs to adjust the path weights on the available paths.

The update event is immediately delivered to the SPAIN controller using the event notification mechanism in the central manager. Upon receiving an update event, the SPAIN controller recomputes path weights and sends the updated weights to the affected endhosts.

Algorithm 1 Adjustment of Path Weights

```
Given:
    src, dst: source and destination
    V: set of VLANs that cover both src and dst

for each VLAN v ∈ V do
    path ← list of links from src to dst in v
    path_capacity ← ∞
    for each link l ∈ path do
        link_capacity ← l's unreserved bandwidth
        if link_capacity < path_capacity then
            path_capacity ← link_capacity
        end if
    end for
    path_weights[v] ← path_capacity
end for
return normalized path_weights
```

Algorithm 1 shows the pseudocode for adjusting the path weights in the integrated controller. The controller adjusts the weights in proportion to the bottleneck link's unreserved bandwidth in each path. If a link has a lower unreserved bandwidth, the integrated controller will route less flows to the paths crossing that link. This dynamic re-routing needs to be promptly done because even a single QoS flow can cause serious performance degradation for all mice flows in network as shown in our evaluation (Section V).

D. Detecting and Scheduling Elephant Flows

The Mahout controller leverages endhost triggers to detect elephant flows. If a TCP flow's socket buffer crosses a chosen threshold, the endhost signals the Mahout controller in the central manager using an in-band mechanism. In our implementation, we choose 100 KB for the threshold because more than 85% of flows in production datacenters [7] are less than 100 KB.

The Mahout controller assigns the detected elephant flow to a path that is expected to provide the maximum throughput to the flow. The calculation of the expected throughput on a path is based on the number of the existing elephant flows and the unreserved bandwidth on the path at a given time. We assume that mice flows do not affect the throughput of elephant flows because they are typically small, short-lived, and evenly distributed to all links by the SPAIN rules.

Algorithm 2 presents the pseudocode for the Mahout controller's dynamic scheduling of elephant flows. The eth represents the expected throughput for a given elephant flow on a VLAN path. The Mahout controller finds a path that can maximize eth among available VLAN paths from source to destination. Once such a path for the elephant flow is decided, the Mahout controller inserts a rule into the source's flow table to forward the elephant flow to the selected VLAN, $vlan\_to\_use$, which overrides the default SPAIN controller rules. The Mahout controller updates its database to reflect the placement of the new elephant flow in network. Like QoS flow rules, the Mahout controller rules expire after a certain idle time, which will signal the Mahout controller to update its database in the central manager. Similarly, the update events are delivered to the other affected controllers.

Algorithm 2 Scheduling of Elephant Flows

```
Given:
    src, dst: source and destination
    V: set of VLANs that cover both src and dst

max_eth ← −∞
vlan_to_use ← null
for each VLAN v ∈ V do
    path ← list of links from src to dst in v
    path_eth ← ∞
    for each link l ∈ path do
        elephants ← number of existing elephants in l
        link_capacity ← l's unreserved bandwidth
        eth ← link_capacity/(elephants + 1)
        if eth < path_eth then
            path_eth ← eth
        end if
    end for
    if path_eth > max_eth then
        max_eth ← path_eth
        vlan_to_use ← v
    end if
end for
return vlan_to_use
```

V. Evaluation

In this section, we evaluate HERCULES in an emulated network environment. Our goal is to examine how multiple network controllers interact in a shared fabric. The goal of this evaluation is not to collect performance results from hardware switches, instead to demonstrate the capabilities of HERCULES to integrate multiple controllers. Hence, we run our integrated controller in a virtualized network environment for more controlled experiments. Individual controllers should exhibit the same performance features that were studied independently because they retain their original architectures after the integration. Each controller's performance results in real testbeds are presented in previous works [4], [8], [9].

A. Emulation Experiment Setting

We use OpenFlowVMS framework [17] to create a virtualized fat-tree topology with 16 endhosts and 8 switches (Figure 3). The endhosts and switches run in separate virtual machines (KVM [18]) in a linux server machine. The VMs are connected using emulated ethernet links (VDE [19]). We set the link capacity to a very low value, 1 Mbps, to avoid any interference caused by CPU overhead in the server[1].

[1]Transferring data between VMs incurs significant CPU overhead because OpenFlowVMS uses non-hypervisor device drivers.

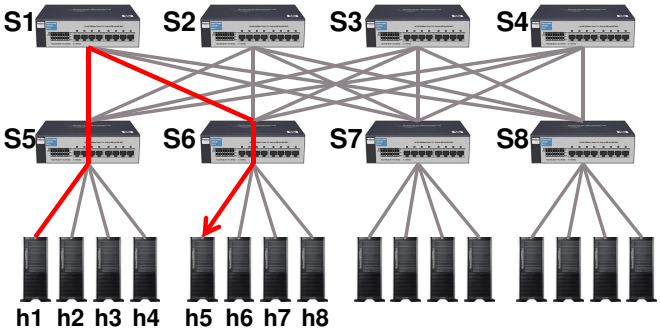


Fig. 3: Test topology for experiments. The two-level fat-tree topology is created using virtual machines connected by emulated ethernet links. The endhosts and switches run in separate VMs. To avoid CPU bottleneck, the link capacity is set to 1 Mbps.

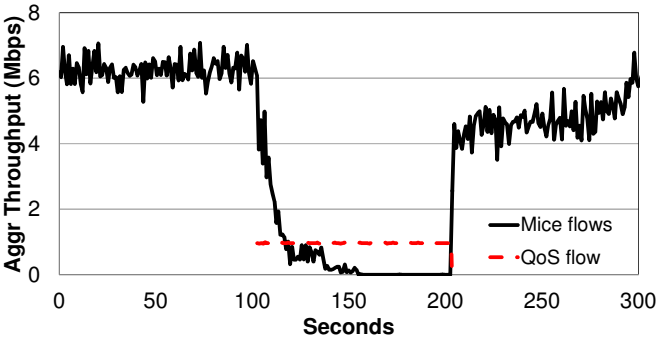


Fig. 4: Independent controllers. The SPAIN controller and the QoS controller run independently in the test topology. When a QoS flow is generated at 100s, the aggregate throughput of mice flows drops to zero because the endhost applications stall at the links reserved for the QoS flow.

We run the integrated controller in the HERCULES central manager. From the two-level fat-tree topology, the controller computes the available paths between the endhosts. The paths are merged into four VLAN trees that correspond to each of the top level switches ($S1$, $S2$, $S3$, and $S4$) in the topology. The computed route map is inserted into the flow table in each endhost. To generate mice flows, each endhost creates 10 threads that send small-sized data (from 1 KB to 20 KB) to randomly picked destinations.

B. Performance Collapse

We first run the SPAIN and the QoS controller independently in the network. At 0s in Figure 4, all endhosts started generating mice flows to random destinations. The aggregate throughput of the mice flows is around 6 Mbps. The network links exhibit almost the same utilization of around 30% during the experiment as the small flows are distributed over all available links.

At 100s in Figure 4, we introduced a QoS flow from $h1$ to $h5$. The QoS flow was defined in the QoS specifications in

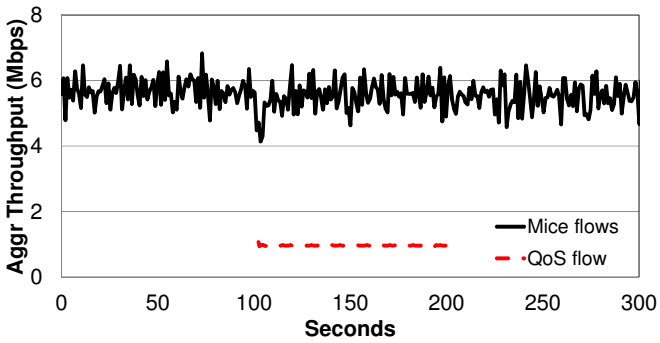


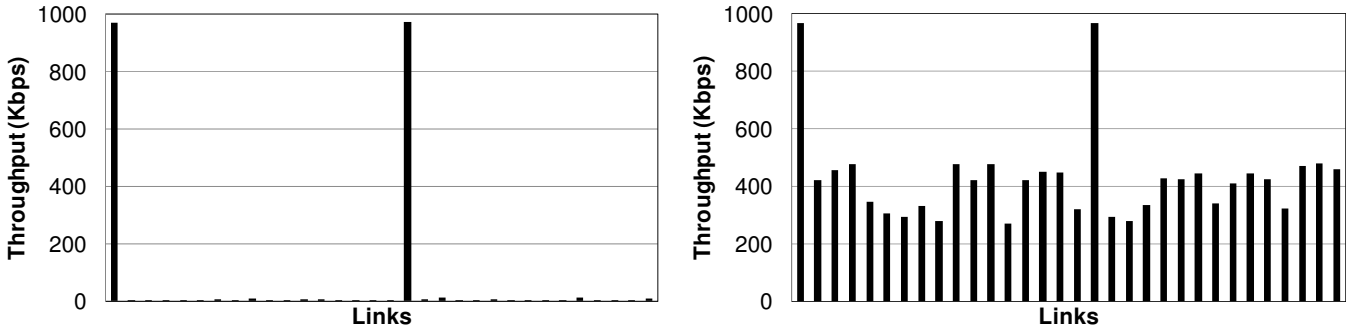
Fig. 5: The integrated controller. The SPAIN controller adjusts its paths weights immediately after a QoS flow is admitted into network at 100s. As a result, the aggregate throughput of mice flows is not affected by the QoS flow.

the central manager. When $h1$ detects the arrival of the new QoS flow, it signals the QoS controller. The QoS controller examines the available paths between $h1$ and $h5$ (VLAN1, VLAN2, VLAN3, and VLAN4) and assigns one of the paths, VLAN1, to the flow in $h1$'s flow table. The QoS flow generates high priority data at 1 Mbps rate, thus consuming all bandwidth on the path.

One might think that other mice flows would not be significantly affected by the QoS flow because (1) only 2 out of 32 (6%) uni-directional links are reserved, (2) multi-pathing works well, and (3) the mice flows are very short-lived. However, the aggregate throughput of the mice flows sharply decreases and reaches almost zero at 150s. Even after the QoS flow is finished at 200s, it takes more than 100s to recover the original throughput.

This *performance collapse* happens because endhost thread stalls when it attempts to send a flow to either of the two reserved links ($S5 \rightarrow S1, S1 \rightarrow S6$). An endhost application cannot know - or control - what paths its flows traverse. As a result, the application waits until its call of *send()* function is timed out though there are alternative paths to use for the flow in the network. We observed the same performance degradation with HBR, but omitted the results due to space constraints.

This implies that multiple controllers operating independently could cause serious performance problems. Each network controller is optimized for a specific goal, but not designed to co-exist with other controllers in a shared fabric. The SPAIN controller aims to utilize every available link in the network for high-bisection bandwidth, and the QoS controller attempts to reserve network resources on demand for high priority QoS-sensitive flows. Without shared modules and notification mechanisms, it is very difficult for a controller to know about the control actions by other coexisting controllers. In this section, we demonstrated a worst-case performance scenario with two independent controllers, but similar coordination issues could arise if different controllers run in isolation.



(a) Independent controllers

(b) The integrated controller

Fig. 6: Per-link throughput while a QoS flow is active. Each vertical bar represents a link's average throughput. With controllers running in isolation, most links are idle except for the two links used by the QoS flow because endhost applications stall at the paths crossing the reserved links. With the integrated controller, the mice flows are routed to alternative paths. As a result, all available links are utilized.

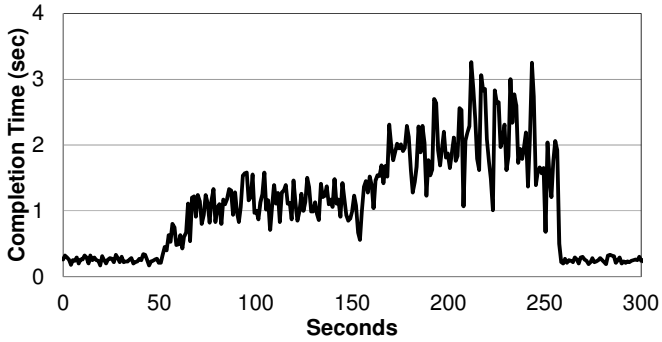


Fig. 7: Flow completion time. Two QoS flows are generated at $50s$ and $150s$, consuming 0.9 Mbps bandwidth on the paths. Although the links are not fully reserved for the QoS flows, mice flows experience increased completion times due to the congestion in the links until the integrated controller is enabled at $250s$.

C. Weight Adjustment in Multi-pathing Controller

Next, we run the integrated controller and generate the same workload. As shown in Figure 5, the QoS flow does not affect the performance of the mice flows in the network. The multi-pathing controller adjusts its path weights immediately after the QoS flow is admitted. The controller sets the weights to zero for the paths crossing the two fully reserved links ($S5 \rightarrow S1, S1 \rightarrow S6$) so that endhosts avoid sending flows through the reserved paths.

Figure 6 presents per-link throughput while the QoS flow is active with/without the integrated controller. There are 32 uni-directional links between eight switches in the test topology. When the controllers run independently, most links were idle except for the two links used by the QoS flow. With the integrated controller enabled, the SPAIN controller adjusts the weights for the affected paths immediately after the QoS flow is admitted. As a result, the mice flows are routed to alternative paths.

One might think that this performance problem occurs only when the high-priority QoS flow takes all bandwidth on its path. And that such a full reservation may be rare in real networks. Hence, we repeat the experiment with a slightly different setting. We generate a QoS flow ($h1$ to $h5$) at $50s$ in Figure 7. Now the peak rate of the flow is set to 0.9 Mbps, not 1 Mbps. The aggregate throughput of the mice flows still decreases again, but does not reach zero because there is available bandwidth for the mice flows in the reserved links. However, the mice flows show increased completion times due to the congestion in the links. We generate another QoS flow ($h2$ to $h6$) at $150s$. The QoS controller assigns the second QoS flow to VLAN2 because VLAN1 is occupied by the first QoS flow. With the second QoS flow added, the completion times were almost doubled. With the integrated controller enabled, the mice flows showed the same performance as when there is no QoS flow in the network.

Flow	Src	Dst	VLAN	Path
$q1$	h1	h5	1	h1-S5-S1-S6-h5
$e1$	h2	h6	2	h2-S5-S2-S6-h6
$e2$	h3	h7	3	h3-S5-S3-S6-h7
$e3$	h4	h6	2	h4-S5-S2-S6-h6

TABLE I: Scheduling of elephant flows in the integrated controller. $q1$ is a QoS flow and $e1$, $e2$, and $e3$ are elephant flows. The elephant flows are detected by the triggers in the endhosts.

D. Scheduling of Elephant Flows

In this section, we examine the scheduling of elephant flows in the integrated controller. The Mahout controller is integrated into the central manager to manage elephant flows. The controller installs large-flow detection rules in each endhost's trigger table. We use 100 KB as the threshold for considering a flow as an elephant because most flows (greater than 85% of flows) are less than 100 KB in production datacenters.

We generate a mixture of mice flows, QoS flows and elephant flows in the test topology. The mice flows are generated by each endhost as before. A QoS flow, $q1$, is generated from $h1$ to $h5$, and three elephant flows are introduced into the

network one by one. Those large flows are long-running TCP flows that we generate using iperf utility. We start an elephant TCP flow $e1$ from $h2$ to $h6$. Likewise, we generate $e2$ ($h3$ to $h7$) and $e3$ ($h4$ to $h6$) with $100s$ interval in between.

Table I presents the paths that the integrated controller assigned to the generated test flows. The QoS controller assigns $q1$ to VLAN1 since there is no active QoS flow on the path. When $e1$ is detected, the Mahout controller knows that the link ($S5 \rightarrow S1$) in VLAN1 is reserved for $q1$. Therefore, $e1$ is assigned to an alternative path in VLAN2 to deliver the maximum expected throughput to the flow. Likewise, $e2$ is assigned to VLAN3 to avoid the collision with $e1$. For $e3$, the link ($S6 \rightarrow$h6) is already shared by $e1$. Using an unused VLAN (e.g., VLAN4) would not help $e3$ avoid the collision. As a result, $e3$ is assigned to VLAN2. The mice flows and the QoS flow exhibited the same throughputs as in Figure 5 while the elephant flows consumed remaining bandwidth in the assigned paths.

The evaluation scenarios we selected clearly demonstrate that HERCULES framework achieves the desired goal. The integrated controller enables multiple independent controllers to be integrated together for coordinated control decisions. The QoS-sensitive flows are routed to the reserved paths, the elephant flows avoid long-term collisions, and the mice flows do not suffer from the congestion caused by the significant flows.

VI. RELATED WORK

The advent of cloud computing has not only led to increase in the size of datacenter networks, it has put additional demand on the networks to support additional functionality such as performance isolation and energy efficient operations. Driven by the complexity and scalability considerations, there has been a recent push towards *Software Defined Networking (SDN)*. *SDNs* are comprised of two key components, a network of programmable elements and a network controller to program and manage the network of programmable elements in the context of 4D project [20], [21]. *OpenFlow* is one of the primary efforts in terms of making the network elements more programmable and providing API to modify and manage the switch flow tables and other elements. A variety of OpenFlow controllers such as NOX [13] and BEACON [22] are being used by the SDN research community. While these controllers can be extended to implement different modules such as load-balancing, multi-pathing etc., these controllers are limited to the OpenFlow APIs and require a network of OpenFlow devices. Our HERCULES framework has been designed to support OpenFlow based controllers as well as the controllers for COTS network devices.

Prior research has developed independent controllers for each of the various requirements and constraints on the data center networks. SPAIN [9] and HBR [10] are two multi-pathing network controllers that have been proposed to achieve high-bisection bandwidth. Both of these controllers can run on COTS switches and rely on end-point support for flow path-selection. On the other hand, an automated OpenFlow based QoS controller to create virtual network slices for providing isolation between and performance guarantees to multiple tenants (and applications) on the same converged network fabric was presented earlier [4]. In order to reduce operational costs we had earlier leveraged an OpenFlow based power controller called ElasticTree [5] to control the network topology and routes for reducing energy consumption.

The performance and overheads of the network controllers have to scale with the size of the datacenter networks. The scalability issues are further exacerbated in case of fine-grained flow level control enabled by OpenFlow. Onix [23] is a distributed control platform with support for basic state dissemination primitives to implement OpenFlow controllers as a distributed system with global network view. Devoflow [24] delegates some control decisions to the switches by pre-populating them with rules to overcome the bandwidth bottleneck between the controller and the switches. These scalability enhancements to controller design are orthogonal and complimentary to the goals of this paper.

VII. CONCLUSION

Controller based networks are becoming more and more prevalent as the scale and dynamics of the today's networks grow. In particular, cloud computing puts increasing demands on provisioning high-bisection bandwidth in the data center networks for reduced costs as well as impress the need for rapid dynamic provisioning for multiple tenants. Hence, network operators need to consider deployment of network controllers for traffic management in large-scale datacenters. In this paper, we focused on the problem of running multiple network controllers in a shared fabric. We showed that running controllers in isolation can lead to serious performance degradation as well as increased management complexity. We presented the design and implementation of HERCULES which is an integrated control framework for network controllers. We demonstrated the flexibility and extensibility of HERCULES by integrating four existing controllers developed earlier for different goals. These four controllers were integrated into the framework without any significant modification to their implementation. Our initial experimentation with the integrated controller shows that it can enable multiple controllers to leverage each other and collectively achieve the goals of the controllers. In order to better study the scalability and deployability of HERCULES, we will be deploying it on the *OpenCirrus* [25] cloud research platform at HP Labs. As more controllers are integrated using the HERCULES framework, it will enable us to further analyze the controller requirements.

VIII. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. We also thank Sujata Banerjee, Andy Curtis, Jeongkeun Lee, Sung-Ju Lee, Jeff Mogul, Jayaram Mudigonda, Mike Schlansker, Jean Tourrilhes, and Praveen Yalagandula for help with controller integration and beneficial discussions.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of USENIX OSDI*, 2004.

[2] Hadoop MapReduce., http://hadoop.apache.org/mapreduce/.

[3] FCoE (Fibre Channel over Ethernet), http://www.fcoe.com/.

[4] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," in *Proceedings of USENIX INM/WREN*, 2010.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proceedings of USENIX NSDI*, 2010.

[6] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proceedings of USENIX NSDI*, 2008.

[7] 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 *Proceedings of ACM SIGCOMM*, 2009.

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

[9] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies," in *Proceedings of USENIX NSDI*, 2010.

[10] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble routing for datacenter networks," in *Proceedings of ACM/IEEE ANCS*, 2010.

[11] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," in *Proceedings of ACM SIGCOMM*, 2010.

[12] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of ACM SIGCOMM*, 2009.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM Computer Communication Review*, 2008.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM Computer Communication Review*, 2008.

[15] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *Proceedings of USENIX NSDI*, 2010.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of ACM SIGCOMM*, 2008.

[17] OpenFlowVMS, http://www.openflow.org/wk/index.php/OpenFlowVMS/.

[18] Kernel Based Virtual Machine, http://www.linux-kvm.org/.

[19] Virtual Distributed Ethernet, http://vde.sourceforge.net/.

[20] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, 2005.

[21] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *Proceedings of ACM HotNets*, 2004.

[22] Beacon Openflow Controller, http://www.openflowhub.org/display/Beacon/Beacon+Home.

[23] 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 *Proceedings of USENIX OSDI*, 2010.

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

[25] Open Cirrus(TM): The HP/Intel/Yahoo Open Cloud Computing Research Testbed. http://opencirrus.org/.