HOLMES : HOLISTIC MICE-ELEPHANTS STOCHASTIC SCHEDULING IN DATA CENTER NETWORKS

Tingqiu Tim Yuan, Cong Xu, Tao Huang, Jian Li

Huawei Technologies

ABSTRACT

Dependability of Scheduling between latency-sensitive small data flows (a.k.a. mice) and throughput-oriented large ones (a.k.a. elephants) has become ever challenging with the proliferation of cloud based applications. In light of this mounting problem, this work proposes a novel flow control scheme, HOLMES (HOListic Mice-Elephants Stochastic), which offers a holistic view of global congestion awareness as well as a stochastic scheduler of mixed mice-elephants data flows in Data Center Networks (DCNs). Firstly, we theoretically prove the necessity for partitioning DCN paths into sub-networks using a stochastic model. Secondly, the HOLMES architecture is proposed, which adaptively partitions the available DCN paths into low-latency and high-throughput sub-networks via a global congestion-aware scheduling mechanism. Based on the stochastic power-of-two-choices policy, the HOLMES scheduling mechanism acquires only a subset of the global congestion information, while achieves close to optimal load balance on each end-to-end DCN path. We also formally prove the stability of HOLMES flow scheduling algorithm. Thirdly, extensive simulation validates the effectiveness and dependability of HOLMES with select DCN topologies. The proposal has been in test in an industry production environment.

1. INTRODUCTION

The wide adoption of diverse cloud-based applications and services exacerbates the challenges in design and operation of Data Center Networks (DCNs). Consider a typical mix of traffic patterns in cloud applications: mouse vs. elephant data flows. Mouse data flows (mice) are emails, web pages, data requests or any other short-lived data flows. Elephant data flows (elephants), on the other hand, are persistent data flows such as VM migrations, data migrations, MapReduce and other application flows that impact network bandwidth for minutes or hours if not longer. In multi-tenant mode, long-lasting elephant and short-lived mouse flows share on DCN paths [15]. According to the results shown in [7], the sizes of the numerous short-lived flows are usually less than 10KB, the majority of which tend to require very low latency. On the other hand, the long-lasting heavy DC flows are typically much larger than 100KB; although the number of these large flows is extremely small compared to that of the small flows. These elephant flows account for more than 80% of bandwidth in DCNs. The competing performance requirements of the two types of flows make it challenging to schedule them on shared paths. That is, the throughput-oriented large flows demand for high bandwidth, while the latency-sensitive small flows prefer minimal queuing delay. Note that, in hyper-scale data centers there are also many flows that are both high bandwidth and require low per packet latency. Such flows can be handled as an extension to the strategy of scheduling mice-elephants flows.

David C. Wyld et al. (Eds) : NeCoM, SEAS, SP, CMCA - 2018
pp. 01–21, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80901

Table 1 : Summary of Key Notations and Definitions

Notations	Definitions
b	Number of packets acknowledged by a received ACK
p	Probability that a packet in a flow is lost
$E[W]$	Expected average TCP window size
$L(w)$	Probability that a packet is lost when the window size is w
$W(t)$	Window size of a flow at time t
$Q(t)$	Queue length of an end-to-end path at time t
P_{SS}	Steady-state probability of the slow start period
P_{TD}	Steady-state probability of the convergence avoidance period
$B(p)$	Average data sending rate with the average probability of packet loss p
C	Bottleneck end-to-end link capacity of an end-to-end path
K	Marking threshold; when queue length exceeds this threshold, a congestion signal will be triggered
W_m	Maximum window size of a TCP flow
T_C	Duration of one TCP convergence avoidance period
T_S	Duration of one TCP slow start period
Q_{max}	Maximum queue size of an end-to-end path
N	Total amount of DC flows
D	Amplitude of oscillation in window size of a single flow
A	Amplitude of queue oscillations of an end-to-end path
$P_{M\text{-}E}$	Probability that the mouse flows affect the elephant flows
$P_{E\text{-}M}$	Probability that the elephant flows affect the mouse flows

To provide high bisection bandwidth, DCN topologies are often multi-rooted topologies, e.g. Fat-Tree, Leaf-Spine, characterized by a large degree of multipath. There are multiple routes between any two DCN endpoints [1, 2]. However, a critical issue in such network topologies is to design an efficient scheduling mechanism to balance the load among multiple available paths, while satisfying different application requirements defined in the Service Level Agreements (SLAs).

The de facto DCN flow scheduling scheme Equal Cost Multi Path (ECMP [3]) cannot meet such dynamic performance requirements in data centers. Hash collisions in ECMP can cause congestions, degrading throughput [4-6] as well as tail latency [7-9] of DCN flows. To balance the load between DCN switches and paths, stateful schedulers have been proposed, e.g. Conga [10], Hedera [4], etc. They monitor the congestion state of each path and direct flows to less congested paths, hence more robust to asymmetry network without control plane reconfigurations [11, 12]. Since maintaining global congestion information at scale is challenging, local congestion-aware or stochastic scheduling schemes are proposed, e.g. Expeditus [12], Drill [13], Hula [14], etc. Using simple or local information collection, these mechanisms are more efficient and applicable for complex DCN architectures, e.g. 3-tier Clos topologies. However, the majority of these scheduling mechanisms focus on balancing the loads of DCN according to the congestion information, without any consideration of cloud applications or data center traffic patterns.

Existing solutions to scheduling the mice-elephants hybrid DCN traffic fall into two categories. The first category deploys unified schedulers for both mice and elephants on shared paths, in spite of the competing performance requirements of the two. Based on the analysis of DC traffic patterns, these studies design novel scheduling algorithms or congestion signals [16, 17] and strike at the right balance between throughput and latency on shared DCN paths. The main challenge though is the interference between the elephant and mouse flows. The second category deploys network partition schemes that transfer the two types of flows over separate paths. [18] By isolating elephant and mouse flows, network partition solutions avoid the aforementioned interference. This is particular attractive

as hardware and system cost continues to drop. Nonetheless, new policies are required to adaptively partition the DCN paths, given dynamic DC traffic patterns and varied DC architectures.

This paper focuses on the second category, the network partition solutions. Using a stochastic performance model, we first theoretically prove that the interference between mice and elephants are inevitable under the unified scheduler, indicating that network partition is a more appropriate solution for handling such hybrid DC traffic. We then propose a novel scheduling scheme, HOLMES, for such hybrid traffic in datacenters. HOLMES architecture partitions a DCN into high-throughput and low-latency sub-networks, decouples the two competing scenarios and eliminates the interference in the hybrid traffic. HOLMES further deploys a stochastic and global congestion-aware load balancing algorithm that optimizes the performance of each sub-network. The stability of the HOLMES flow scheduling algorithm is also proved and validated in this paper.

The main contributions of this work are:

- We prove with a closed form the necessity of applying network partition solutions to mice-elephants DC traffic, showing that unified flow scheduling schemes or congestion control mechanisms are not optimal.
- We design HOLMES, an architecture that partitions all the DCN paths into sub-networks, isolating the paths of elephant and mouse flows. As a result, HOLMES eliminates the interference between the two. Moreover, an AI module of the HOLMES architecture enables machine learning techniques and offers a more comprehensive analysis of the DCN status for better scheduling.
- We propose a novel stochastic load-balancing algorithm for HOLEMS that is global congestion aware. Using the end-to-end congestion signal, the algorithm is agile to link or node failures in DCN. Moreover, it uses only limited global congestion information to improve the scheduling efficiency and sustain stability in all possible scenarios.
- We use an analytical model to evaluate the adaptability of HOLMES. We then confirm the finding with detailed simulations under varied DCN architectures and scheduling policies. These experimental results can further guide the hardware implementation of HOLMES.

The rest of this paper is organized as follows. Section II presents an overview of HOLMES. Section III and IV discuss in detail the HOLMES architecture and its load balancing mechanisms, respectively. Section V evaluates HOLMES with extensive simulation experiments. Section VI concludes the paper.

2. HOLMES OVERVIEW

HOLMES aims to satisfy the competing performance requirements of elephants and mice by partitioning a DCN into high-throughput and low-latency sub-networks. HOLMES proposes a stochastic and global congestion-aware mechanism to schedule the elephant and mouse flows in these separate sub-networks. This section presents an overview HOLMES.

A. Necessity of Network Partition

In order to lay out a solid foundation of HOLMES, we first theoretically prove the limitations of the unified scheduling schemes when handling hybrid DC traffic. The analytical model in turn promotes the need for deploying network partition solutions in the architecture design of flow scheduling and congestion control schemes.

The main challenge that any unified scheme needs to address is the interference between the elephant and mouse flows. A typical scenario that mouse flows affect elephant flows is that a burst of mouse flows triggers a congestion signal, e.g. Quantized Congestion Notification (QCN), which in turn reduces the transmission rates of elephant flows in the next long period of time. Measurement results show that the short-lived flows in large-scale clusters exhibit significant traffic bursts. Theophilus Benson et al further explain that the traffic bursts in DCNs are caused by the huge amount of aggregated edge links as well as the concurrency of parallelized computing schemes, e.g. shuffle period in a MapReduce process. Experimental results also show that a momentary traffic burst can lead to short-lived congestion in a DCN. On the other hand, the effect of elephant flows on mouse flows is that elephant flows occupy the vast majority of the network buffers; hence the mouse flows will result in long queuing delay and flow completion time (FCT), a.k.a. the bufferbloat problem.

We calculate the probability that the above two scenarios would occur with the unified flow scheduling solutions and how they would affect network performance. Some key notations and definitions are illustrated in Table I. We then prove that fundamental limitations exist in the unified scheduling schemes when handling hybrid DC traffic. (Details of the proof is omitted in this paper due to page limit, however, we will post it after publication.) Therefore, network partition is a better solution that eliminates the interference by separating the mouse and elephant flows.

B. HOLMES Architecture with Network Partition

W. Wang et al [18] and W. Cheng have applied the network partition solutions to separate the large and small flows in DCN paths, and to avoid the interference between the two.

Similarly, HOLMES deploys the network partition solution, which partitions all the end-to-end DCN paths into two logical sub-networks to isolate the transmission of elephant and mouse flows. A centralized controller is deployed to map each link to the high throughput or the low latency sub-network. The traffic transmitted between each pair of edge switches may contain both elephant and mouse flows. Therefore, HOLMES architecture needs to ensure that at least one of the multiple paths between each two edge switches belongs to the high throughput sub-network, while at least another one belongs to the low latency sub-network too. The scales of the two subnetworks can be determined either statically or dynamically according to the architecture of the DC as well as the traffic patterns.

In addition to network partition, network statistics and analysis functions are provided by the HOLMES AI module. We discuss the detailed architecture in Section III.

C. Global Congestion-aware Load Balancing Algorithm based on Power of Two Choices Model

Another important component in HOLMES is its load balancing algorithm. Since local congestion-aware load balancing algorithms have been proved to react slowly to link failures [10, 14] and are prone to form congestion trees, we deploy global congestion-aware load balance in HOLMES. Inspired by the Power-of-Two-Choices model [43, 44], we further design a stochastic load-sensitive flow scheduling algorithm, which reduces the overhead of congestion information maintenance and improves the scheduling efficiency.

M. Mitzenmacher and S. Ghorbani et al have shown that tagging a single unit of memory with random sampling and identify the shortest output queue from the previous time slots guarantees stability [13, 43]. We extend a distributed implementation of this approach: When a packet arrives at an input port of an edge TOR switch, that TOR switch resolves the source and destination addresses of the packet.

It then assigns the packet to the least loaded of d randomly chosen end-to-end paths from all N (d << N) paths. The choice of the previous time slot is saved for the scheduling of the next time slot.

Section IV will discuss the detailed process and analyze the stability of this load balancing algorithm.

3. HOLMES ARCHITECTURE

Fig. 1 shows the HOLEMS architecture in three layers: AI layer, control layer and infrastructure layer. The HOLMES AI layer contains a cognitive computing cluster to implement the software AI module. The AI module collects the DCN state information from the DCN monitors, and applies the machine learning algorithms to generate some analysis results, e.g. networking tendency predictions, network outlier locations, etc. These learning results generated by the AI module provide a more comprehensive view of DCN behaviors. They will be then used for network partition and flow scheduling operations.

The HOLMES control layer is responsible for generating the network partition, congestion control and local balancing policies, based on the monitoring information as well as the learning results generated by the AI module. The policies generated in the SDN are decomposed into a series of finegrained partition and scheduling orders, which are transmitted to the DCN switches and end hosts for execution. Without deploying the HOLMES AI module, the functions provided by the control layer are the same as the traditional SDN controllers.

The HOLMES infrastructure layer executes the specific network partition as well as the flow scheduling operations. It is responsible for storing, forwarding and processing data packets. The detailed operation orders are transmitted and configured on the DCN switches. The DCN switches first map each link to the high throughput network or the low latency subnetwork, according to the network partition policies. When the elephant and mouse flows arrive at a DCN switch, their packets are scheduled to the pre-partitioned paths separately. This process is managed by the HOLMES control layer.

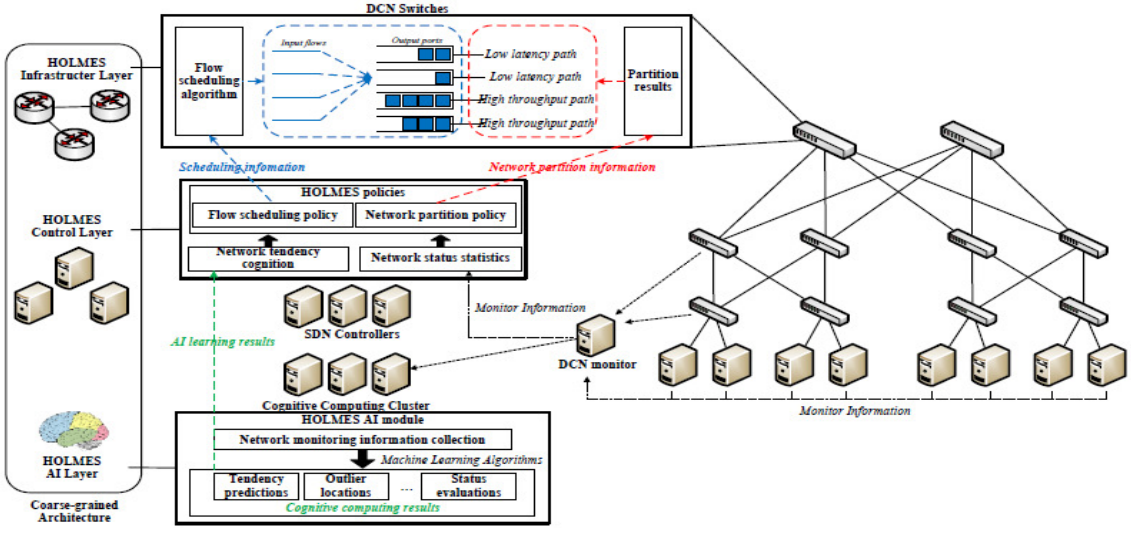

Fig. 1. HOLMES architecture: Based on the real-time monitor information, HOLMES AI module first analyzes the status or tendency of the DCN using machine learning models, and generates the learning results. Next, according to the analysis results, HOLMES control layer designs a network partition policy and a corresponding flow scheduling policy, and the policies are generated in the SDN controllers. Finally,

the network partition as well as the flow scheduling operations will be executed on the DCN switches or hosts, under the guidance of the SDN controllers.

A. HOLMES Network Partition

Both static and dynamic network partition mechanisms are enabled by HOLMES. When the scale of the DCN is small, HOLMES provides static network partition policies to DCN switches to avoid the frequent scale changes of the two subnetworks. It also makes the execution of the network partition policy more stable. On the other hand, when the scale of the DCN is large, HOLMES will deploy the dynamic network partition solution that dynamically adapts the scales of the two types of sub-networks to the traffic load variants. Moreover, the dynamic network partition solution utilizes the network resources more efficiently, especially when the arriving rate of DC traffic is slow.

Both static and dynamic network partition mechanisms have been studied [18]. One can either integrate these mechanisms in HOLMES or implement the appropriate network partition mechanism according to the scale of the DCN as well as the traffic patterns.

Both static and dynamic network partition mechanisms are enabled by HOLMES. When the scale of the DCN is small, HOLMES provides static network partition policies to DCN switches to avoid the frequent scale changes of the two subnetworks. It also makes the execution of the network partition policy more stable. On the other hand, when the scale of the DCN is large, HOLMES will deploy the dynamic network partition solution that dynamically adapts the scales of the two types of sub-networks to the traffic load variants. Moreover, the dynamic network partition solution utilizes the network resources more efficiently, especially when the arriving rate of DC traffic is slow.

Both static and dynamic network partition mechanisms have been studied [18]. One can either integrate these mechanisms in HOLMES or implement the appropriate network partition mechanism according to the scale of the DCN as well as the traffic patterns.

B. Application Scenarios for HOLMES Architecture

Compared with the commonly used SDN architectures, a prominent feature of HOLMES is the implementation of the AI module and its machine learning algorithms. Machine learning methods have been widely used in network management and DC scheduling policy generation operations. Those continuing learning and analysis results provide a comprehensive understanding of network features and behaviors, which benefits the designing of the corresponding network partition and flow scheduling policies. Therefore, the deep analysis and accurate AI prediction provided by the AI module enable the HOLMES architecture to perform more complex and intelligent operations.

One typical application scenario for HOLMES architecture is the deployment of application driven networks (ADN), where a physical network is sliced into logically isolated sub-networks to serve different types of applications. Each network slice in ADN can deploy its own architecture and corresponding protocols, to satisfy the requirements of the applications it serves. The key operations when implementing ADN are: (1) Constructing an application abstraction model to formulate the resource requirements of the applications; (2) mapping the distinct properties of applications to respective network resources. It is shown that the complexity and performance of these operations can be improved when some application features are pre-known [8]. Hence, the HOLMES AI module benefits the analysis of application features as well as the prediction of resource requirements, which further alleviate the complexity of application abstractions and mapping operations. Moreover, the design and implementation of network slicing mechanisms can also be realized by the cooperation of the control layer and the infrastructure layer.

Algorithm 1 Flow scheduling policy in HOLMES

Input: Load condition of each end-to-end path at time t:
$\{load(1), load(2), load(3),\ldots\}$
Path number of the selected path at time t -1: $s^{(t-1)}$
Output ports of the TOR and aggregate switches on each path:
$\{\{P^A_{(1)}, P^T_{(1)}\}, \{P^A_{(2)}, P^T_{(2)}\}, \ldots\}$

Output: Path number of the selected path at time t -1: s
Output ports TOR and aggregate ports on the selected path:
$\{P^A_{(s)}, P^T_{(s)}\}$

Initialize: $m = 2$, $d = 1$;
Initialize: $load_{OPT} = load(s^{(t-1)})$;
Random select m end-to-end paths $\{Path(1), Path(2), \ldots, Path(m)\}$
Construct candidate set: $L \leftarrow \{Path(1), Path(2), \ldots, Path(m)\} \cup \{s^{(t-1)}\}$
for each path i $(1 \le i \le m+1)$ in the candidate set L **do**
  if $load(Path(i)) \le load_{OPT}$ **then**
    $load_{OPT} = load(Path(i))$;
end for
Assign value: s = Path number of the path with load $load_{OPT}$;
return $\{P^A_{(S)}, P^T_{(S)}\}$ and s

Similarly, HOLMES architecture is also applicable for some other intelligent or complex application scenarios, which demand a deep understanding of network or application features such as Internet of Vehicles (IoV), coflow scheduling and some other network architecture based on network slicing or network partitions. With the immense proliferation of complex and diverse cloud-based applications, we expect such architecture to be the development trend in the future.

C. Out-of-order vs. Efficiency

While elephants contribute to the majority volume of DCN traffic, mice account for 80% of the number of instances. Outof-order scheduling done at host side may sacrifice efficiency. In addition, compatibility with legacy hardware has to be ensured. Therefore, one may still want to consider deploy conventional, sometimes oblivious, ECMP scheduling for inorder scheduling of mice. However, we expect such overhead to diminish, while hardware and system cost continues to drop. As a result, HOLMES is the way to go.

4. HOLMES SCHEDULING ALGORITHMS

Once the hybrid DC traffic is separated into different subnetworks, scheduling algorithms affect the performance of each sub-network. In this section, we discuss the aforementioned global congestion-aware scheduling algorithm and prove the stability of its stochastic policy.

A. Necessity of Deploying Stochastic Scheduling Algorithms

Compared with the other state-aware flow scheduling algorithms, stochastic flow scheduling algorithms are more applicable for large-scale data centers according to the following reasons:

1) Simplification of computing complexity

One of the key factors that degrade the performance of the traditional ECMP mechanism is the lack of global congestion information. To overcome this limitation, a group of studies have designed new flow scheduling polices based on a global “macroscopic” view of the entire DCN, e.g. CONGA [10]. However, in large-scale and heavily loaded data centers, the global macroscopic load balancing

algorithms introduce unacceptable computing complexity to deal with the massive information, and the control loops in these scenarios are much slower than the duration of the majority of congestion incidents in data centers [13]. Therefore deploying the stochastic algorithms to achieve micro load balancing is a more viable solution. The micro load balancing solutions require only limited congestion information, which simplifies the computing complexity and enables instant reactions to load variations in large-scale data centers.

2) Optimization of storage complexity

In data centers, 8 and 16-way multipathing are common, while there is growing interest in multipathing as high as 32 or even 64. Specifically, with 40 servers in a rack, there will be 40 uplinks. Each flow can use a different subset of the 40 links, leading to 2^{40} possible subsets. Keeping the state of each path in this scenario requires unacceptable storage resources, which is difficult to be implemented. On the contrary, stochastic scheduling algorithms are effective to cope with the optimization of storage complexity, as well as the small number of register reads. Edsall *et al* [26] deploy the stochastic power of-two-choices hashing solution for balancing the loads of DC routers. The storage complexity of such a stochastic solution is logarithmically reduced.

3) Better adaptability for heterogeneous DCNs

A typical flow scheduling method in multi-rooted DCNs is equal traffic splitting based on hashing, as used in the traditional ECMP approach. However, the uniform hashing approach cannot achieve optimal load balance without the assumption of symmetric and fault-free topology [5, 10], which is not generally true in heterogeneous data centers. To provide better adaptability for heterogeneous DCNs, weighted traffic distribution methods have been widely adopted in the global macro load balancing solutions [11]. In order to correct the imbalance caused by the even distribution approach and enable fair resource allocation, the weighted approaches distribute the traffic among available paths in proportion to the available link capacity of each path. The global weighted traffic distribution solutions have shown good adaptability to dynamically changing network topologies. However, these solutions still need real-time state information collection of all the paths, which introduces additional computing and storage complexity.

Stochastic flow scheduling algorithms can reduce the computing and storage overhead of weighted traffic distribution mechanisms, while maintaining the adaptability to heterogeneous DCN topologies. Consider the stochastic Power of-Two-Choices: The algorithm only needs to record the states of the two randomly chosen paths; therefore, the storage and computing complexity are dramatically reduced. Moreover, the algorithm compares the load conditions of these two chosen paths, select the better one, hence performs a weighted operation in another form. Stochastic load balancing solutions have also been proved to be applicable for heterogeneous DCNs [13, 26]. Based on these justifications, we extend stochastic flow scheduling algorithms to our HOLMES mechanism.

B. Flow Scheduling Algorithm in HOLMES

We consider a stochastic scheduling policy, (d, m) policy: The input port chooses d random end-to-end paths out of all the possible paths. It finds the path with the minimum occupancy among all the d samples and m least loaded samples from the previous time slot. It then schedules the input packet to the selected end-to-end path.

Increasing the value of d and m to >>2 and >>1 will degrade the performance, since the large number of random samples makes it more likely to cause the burst of packet arrivals on the same path [13]. As

a result, we set m=1 and d=2 in our scheduling model. The detailed flow scheduling procedure is shown in Alg. 1.

Using global congestion information, the algorithm reacts rapidly to the link or node failures. Moreover, the limited information used in the algorithm improves the scheduling efficiency while avoids the traffic bursts on the same switch ports.

C. Stability Analysis of HOLMES's Scheduling Algorithm

We prove the stability of this stochastic global congestion aware scheduling algorithm in a two-tier Clos DCN topology. We abstract an end-to-end path in a Clos network (Fig. 2A) as a serial queuing system consists of a series of queues (Fig. 2B). As a result, the whole Clos DCN topology is abstracted as a queuing network. We then evaluate the performance of a specific leaf-to-spine DCN path using a stochastic queuing network model.

We focus on analyzing the stability of the scheduling process from when a packet arrives at a TOR switch to when the packet reaches the spine switch. The packet experiences two queuing processes, at the TOR and the aggregate switch port, respectively. The entire path from the TOR node to the spine node can also be modeled as a large queue.

Based on the results of [56] and with a similar method shown in [57], we prove that HOLMES's scheduling algorithm is stable for all uniform and nonuniform independent packet arrivals. Some key notations and definitions used in the scheduling model are illustrated in Table II.

We prove that the global (1, 1) policy is stable for all admissible parallel arrival process. We construct a Lyapunov function L as follows:

$$L(t) = \sum_{i=1}^{NN_A} (Q_i(t) - Q^*(t))^2 + \sum_{i=1}^{NN_A} Q_i^2(t)$$

To prove the algorithm is stable, we show that there is a negative expected single-step drift in the Lyapnuov function, i.e.,

$$E[L(t+1) - L(t) \mid L(t)] \le \varepsilon L(t) + k \quad (\varepsilon, k > 0)$$

We divide the Lyapunov function into two sub functions as:

$$L_1(t) = \sum_{i=1}^{NN_A} (Q_i(t) - Q^*(t))^2 \qquad L_2(t) = \sum_{i=1}^{NN_A} Q_i^2(t)$$

Based on the above formulation, we prove that there exists a negative expected single-step drift in the Lyapnuov function in each possible case. Therefore, the global (1, 1) policy is stable. Based on the (d, m) policy, the HOLMES's scheduling algorithm is also stable. (Details of the proof is omitted due to page limit; however, we will post it after publication.)

5. HOLMES PERFORMANCE EVALUATION

We evaluate HOLMES using simulation based on OMNET++. We construct a test-bed simulation platform to simulate the data transmission process in symmetric and asymmetric fat-tree DCNs. Similar to [10], a heavy-tailed distribution is used to generate DC flow of different sizes. The hosts of the DCN run TCP applications. The flow request rate of each TCP connection satisfies Poisson process.

By comparing the performance of the same DCN under different load balancing mechanisms, we validate HOLMES and analyze the influence of the network partition solution and end-to-end scheduling schemes on DCN performance. Moreover, we evaluate the adaptability of HOLMES's load balancing algorithm for heterogeneous architectures and discuss some technical issues in the hardware implementation of HOLMES.

A. Evaluation of HOLMES Network Partition

We evaluate the network partition policy of HOLMES in a scenario that hybrid elephant and mouse flows are scheduled in a same DCN with different scheduling schemes. The DCN topology deployed in this experiment is a Clos network with 2 and 4 leaf and spine switches respectively. We generate elephant and mouse flows to leaf switches with average sizes of 100KB and 1KB, respectively. The queue lengths of the switch ports are used as performance indicators.

Since the scale of the DCN in the simulation is not very large, HOLMES deploys the static policy that partitions the two sub-networks in advance. The buffer sizes of all the switch ports are set to be the same. When the buffer of a switch port is full, all the upcoming input packets to that port will be dropped. We compare HOLMES against two start-of-the-art unified load balancing schemes: CONGA [10] and queue length gradient based scheduling. Similar to the delay gradient based congestion control policy used in [16], we deploy the queue length gradient as the indicator and schedule the arrived packet to the port with the minimum length gradient.

Figs. 3A-3C show the queue length variation of the four ports of a spine switch under the three scheduling schemes. The Xaxis indicates the time period and the Y-axis denotes the queue length. We can see from Fig. 3A that using CONGA, the buffers of all the four ports are full after a period of time, indicating the throughput of the switch has been maximized, which benefits the transmission of the elephant flows. However, when a mouse flow arrives, all the packets in that flow have to wait for a long queuing time since all the output port are of heavy loads. Consequently, the latency of the mouse flow will increase and degrade the overall performance of the hybrid DC flows.

Similarly, as shown in Fig. 3B, the buffers of the four output ports are also almost full after a period of time using the length gradient based policy. The results indicate that the length gradient based load balancing policy still suffers from the interference between the elephant flows and the mouse flows.

Comparing Fig. 3A and Fig. 3B, we find that the load balancing condition of the gradient based scheme is a little worse than CONGA. The reason is that the gradient based scheme schedules the DC flows according to the changing trend but not the current state of the DCN. Finally, Fig. 3C shows that HOLMES has successfully isolated the elephant and mouse flows. Two of the ports have been partitioned to the low latency sub-network and used for transmitting the mouse flows. Fig. 3C shows that the buffers of the two ports are almost empty during the entire transmission procedure. Thus, packets in the mouse flows do not need to wait for additional queuing delays, and the low latency of the mouse flow is ensured. Moreover, the buffers of the other high throughput ports are also full filled, which satisfies the throughput requirements of elephant flows. Hence, by isolating the mixed traffic, HOLMES network partition policy successfully eliminated the interference of the elephant flows to the mouse flows.

The main shortcoming of the network partition solution is the inefficient use of network resources. Although the isolation of the hybrid traffic avoids the interactions of the elephant and mouse flows, the spared network resource in the low latency paths has not been fully used since the buffers of these paths are almost empty. An effective solution is to improve the buffer allocation by limiting the buffer

size of the low latency sub-network and assigning the spared buffers to the high throughput sub-network.

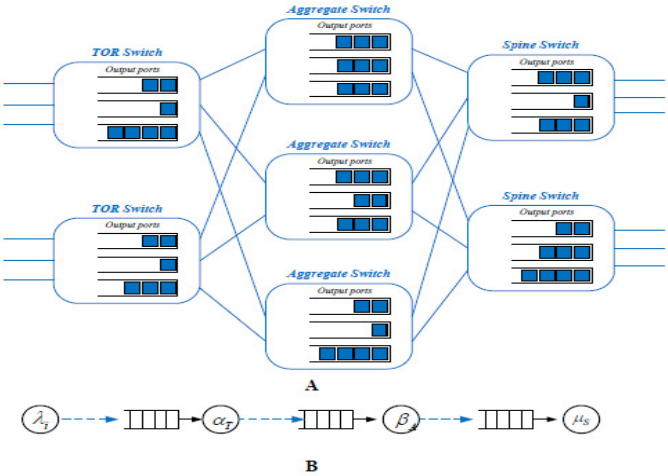

Fig. 2. Abstraction of a leaf-to-spine path in a Clos network (A) to a serial queuing system (B)

Table 2. Summary of Key Notations and Definitions in Scheduling Model

Notations	Definitions
λ_i	Average data arrival rate in the ith leaf-to-spine path
α_i	Average data processing rate of the TOR port in the ith leaf-to-spine path
w_i	Average data arrival rate of the aggregate port in the ith leaf-to-spine path
β_i	Average data processing rate of the aggregate port in the ith leaf-to-spine path
N	The number of the TOR and aggregate switches in a pod
N_A	The number of spine switches connected by each aggregate switch
$Q_k(t)$	The number of accumulated packets in the buffer of the kth leaf-to-spine path at time t
$\hat{Q}_i(t)$	The number of accumulated packets in the buffer of the end-to-end path chosen by the ith input port using HOLMES at time t
$Q^*(t)$	The number of the accumulated packets in the global least loaded leaf-to-spine path at time t
$Q_{iT}(t)$	The number of the accumulated packets on the TOR port of the ith leaf-to-spine path, at time t
$Q_{iA}(t)$	The number of the accumulated packets on the aggregate port of the ith leaf-to-spine path, at time t
$Q_T^*(t)$	The number of the accumulated packets on the TOR port of the global least loaded leaf-to-spine path at time t
$Q_A^*(t)$	The number of the accumulated packets on the aggregate port of the global least loaded leaf-to-spine path at time t

B. Stability Validation of HOLMES Scheduling Algorithm

We evaluate the stability of HOLMES flow scheduling algorithm. We simulate the scenario that DC traffic are scheduled in a DC with asymmetric network topology. We combine two different sized Clos networks, and construct an asymmetric DCN architecture. One of the Clos network consists of 2 leaf switches and 4 spine switches. The other is a Clos network with 5 and 4 leaf and spine switches,

respectively. 10 hosts are attached to each leaf switch. We concentrate on validating the stability of HOLMES flow scheduling algorithm, rather than the network partition mechanism in this scenario. Thus, we do not deploy the HOLMES network partition mechanism in the experiment and only execute the HOLMES flow scheduling algorithm.

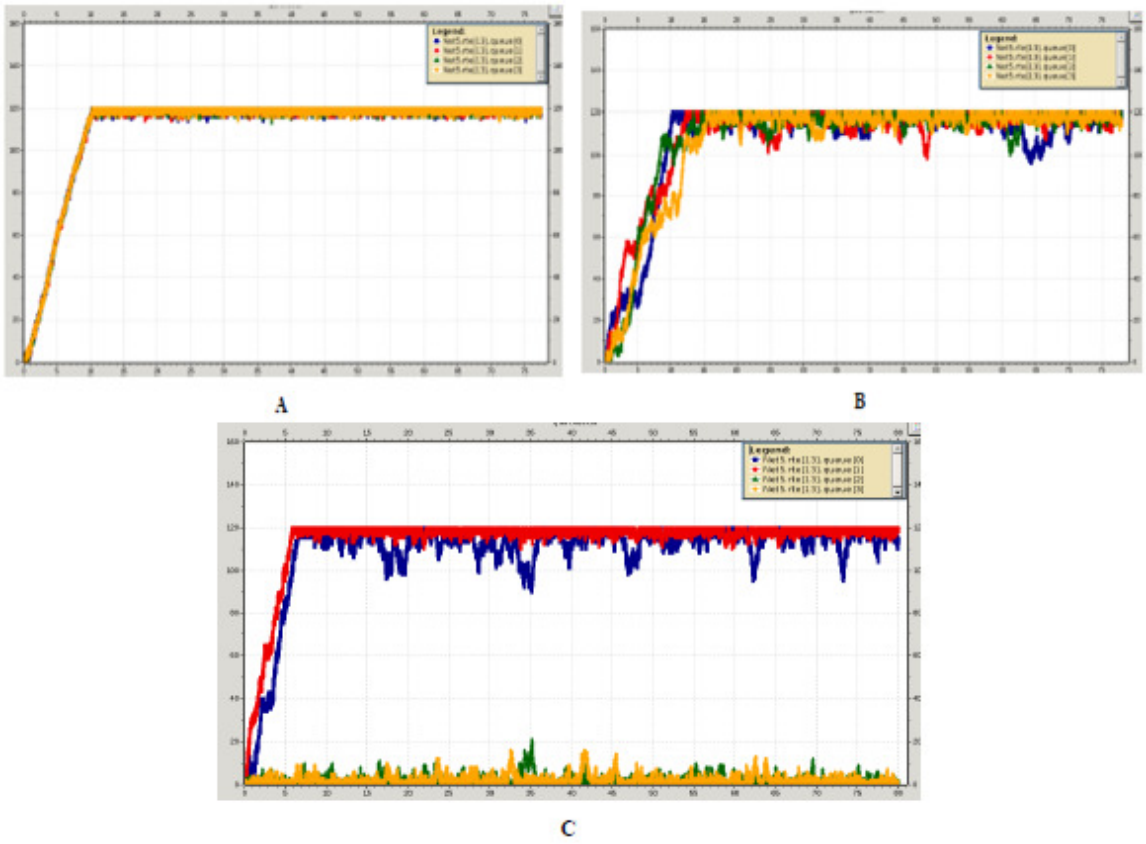

Fig. 3. Queue length changing trends of a DCN spine node's ports under policies CONGA (A), length gradient based policy (B) and HOLMES (C)

CONGA [10] and DRILL [13] are two load balancing solutions proven to be stable. Therefore, we compare them against HOLMES. All DC traffic is scheduled with the granularity of packet, and we focus on analyzing the stability of the three scheduling algorithms. When using the Power of Two Choices selections, we uniformly set d=2 and m=1. Similarly to the previous experiments, we deploy queue length as the load balance indicator to evaluate the overall load balancing condition of the DCN.

Figs. 4A-4C show the queue length changing trend of a specific leaf switch's ports, under load balancing policies CONGA, DRILL and HOLMES. We can see from Fig. 4A that the queue length changing trends of all the ports in a leaf switch are almost overlapped under CONGA, indicating that the queue lengths of all the switch ports are almost the same at each time unit. Therefore, the load balancing condition under CONGA is optimal among all the three mechanisms, since CONGA makes each scheduling decision based on the global congestion information. Without considering the time used for obtaining congestion information, CONGA obtains the global optimal load balancing result.

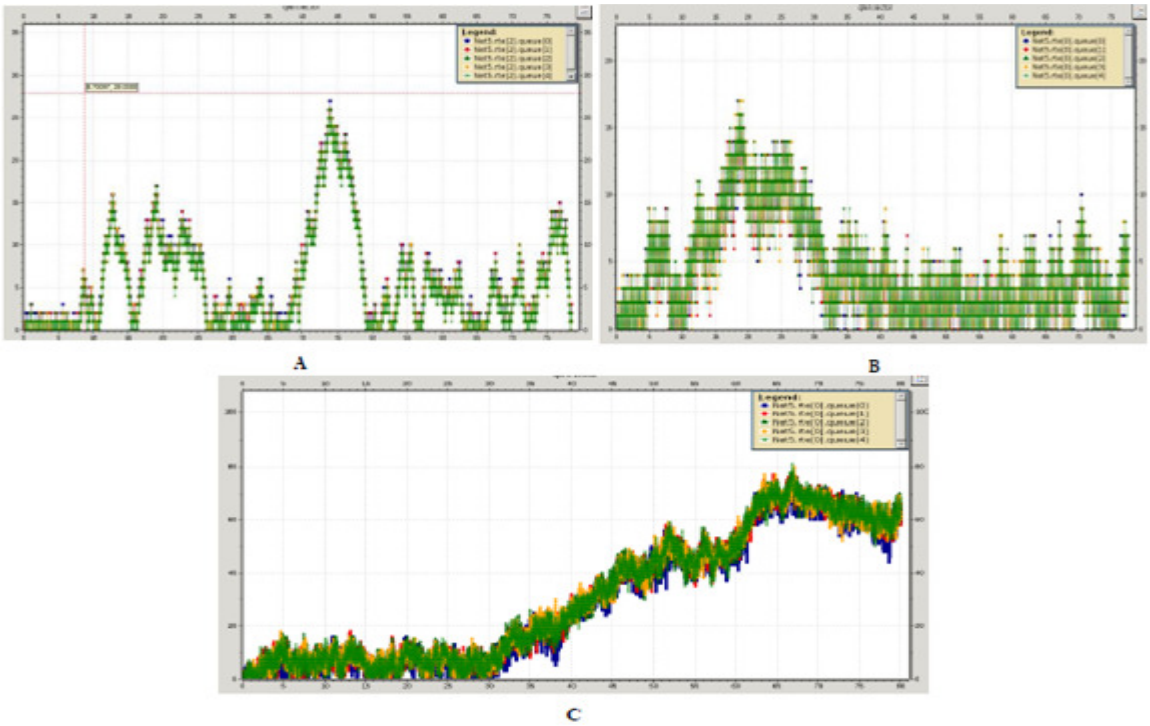

Fig. 4. Queue length changing trends of a DCN leaf node's ports under load balancing policies CONGA (A), DRILL (B) and HOLMES (C)

Fig. 4B shows the queue length changing trends of the same leaf switch ports under DRILL. Different with the former results, we find fluctuations in the queue length changing curve. In other words, the length difference of the longest queue and the shortest queue is clear. The reason is that the use of (d, m) policy in DRILL reduces the scale of the solution space, and the local optimal solutions affect the load balancing condition of the DCN.

Fig. 4C shows the queue length changing trends of the same leaf switch ports under HOLMES scheduling algorithm. The fluctuations also exist in the curve of Fig. 4C, where the amplitude of the fluctuation is more obvious. This phenomenon is also caused by the use of (d, m) policy. Compared with DRILL, the global (d, m) policy used in HOLMES further limits the solution space, and exacerbates the fluctuations. However, although the fluctuations are more obvious when executing HOLMES flow scheduling algorithm, we can also find an upper bound (about 10 packets) of the fluctuation amplitude, indicating that the length difference of the shortest and the longest queue is not infinite in HOLMES. Hence, our HOLMES flow scheduling algorithm is stable during the whole scheduling period. Moreover, limiting the solution exploration space reduces the time used to obtain the congestion information and make HOLMES more efficient and applicable for large-scale data centers.

C. Adaptability for Heterogeneous

As discussed earlier, both the stochastic flow scheduling algorithm and the weighted traffic splitting solutions show good adaptability of heterogeneous congestion states. We now evaluate the adaptability of the two solutions.

We first theoretically compare the approximate adaptability of the two solutions. Consider a simple leaf-spine DCN topology with N paths available between two racks. The loads of the N paths (number of packets accumulated in the buffers) are unevenly distributed. We compare the adaptability of the two mechanisms under the linear distribution of path loads.

For analytical tractability, we assume that the amounts of packets accumulated in the buffer of each path are: 0, 1, 2…, N-1 and the buffer size of each path is also set to N. Next, we calculate the probabilities that one arrived packet is scheduled to the least loaded path under the (2, 1) policy as well as the weighted traffic splitting policy.

The probability P1 that a packet is scheduled to the least loaded path under the (d, m) policy (d =2, m =1) can be calculated as:

$$P_1 = C_{N-1}^{d-1} / C_N^d = \frac{d}{N} = \frac{2}{N} \quad (d = 2)$$

When deploying the weighted traffic splitting solution, the probability that one packet is assigned to a specific path is proportion to the available capacity of that path. Then, the probability that the packet chooses the least loaded path is:

$$P_2 = \frac{N}{N + (N-1) + (N-2) + \ldots + 1} = N / C_{N+1}^2 = \frac{2}{N+1}$$

When the DCN scale is huge, the value of N is large enough, and then the value of P1 and P2 are almost equal. Thus, the adaptability of the (2, 1) policy and the weighted traffic splitting policy for heterogeneous are almost the same when the path loads satisfy linear distributions. However, compared with the weighted solution that maintains the load status of all the N paths, the storage complexity is dramatically reduced when using the stochastic flow scheduling mechanisms.

Next, we consider another scenario that the load distribution of the DCN paths shows stronger heterogeneity. Suppose the path loads satisfy exponential distribution and the amounts of spared buffers in N paths are:

$$N, \frac{N}{2}, \frac{N}{4}, \frac{N}{2^3}, \ldots, \frac{N}{2^{N-1}}$$

Using the weighted traffic splitting policy, the probability that a packet chooses the least loaded path is:

$$P_2 = \frac{N}{N + \frac{N}{2} + \frac{N}{4} + \frac{N}{2^3} + \ldots + \frac{N}{2^{N-1}}} = \frac{N}{2N - \frac{N}{2^{N-1}}} = \frac{1}{2 - \frac{1}{2^{N-1}}}$$

On the other hand, when deploying the (d, m) policy, the probability becomes:

$$P_1 = C_{N-1}^{d-1} / C_N^d = \frac{d}{N}$$

When $d = N/2$, we get:

$$P_2 \approx P_1 = \frac{1}{2}$$

Therefore, when the load conditions of the DC paths are heavily heterogeneous, the (d, m) policy also needs to maintain plenty of load status information to keep its adaptability as good as the weighted

traffic splitting solutions. The stochastic scheduling mechanism does not show obvious advantages in this scenario.

The evaluations of the two approaches have also been made by some other researchers. Key et al [96] propose a coordinated solution that periodically samples again for new paths via random selection of stepping stone routers at the higher level, shifts the traffic load to the paths with the lowest loss rates (or lowest ECN rates, largest delays), then equates the marginal utility of its aggregate data rate to the loss rate on these "best" paths at the lower level. This approach can be considered as an improvement of the traditional weighted traffic splitting solutions, since it considers the real-time changes of the load states. Moreover, Key et al show that the approach outperforms the Power of Two Choices policy, and does almost as well as if each user saw the global list of the resources.

We simulate the execution process of the coordinate approach as well as the Power-of-Two-Choices algorithm on a same switch. Fig. 5 shows the changing trend of the overall switch load as the modeling factor (d) increases.

The load distribution of the switch ports is initialized exponentially in this experiment. We see from Fig. 6 that, as the value of d increases, the load condition of the switch is improved under the power of two choices policy ((d, m) policy); since a larger value of d increases the probability of choosing the lightest loaded output port. As we increase the value of d from 2 to 5, the power of two choices policy performs almost as well as the theoretical load optimal policy (d = 10), which validates our previous modeling results. On the contrary, when using the coordinated approach, the switch attains optimal performance when the value of d is small (d = 2) and the load state of the switch is almost as good as the theoretical load optimal policy. This simulation result is in accordance with the analysis in the literature. However, as the value of d increases, the load state of the switch becomes worse: when assigning d = 10, the load balancing condition of the switch under the coordinated policy is even worse than the (2, 1) policy.

Although the stochastic flow scheduling outperforms the weighted traffic splitting solution in most cases, it still has some limitations. The weighted traffic splitting solution maintains the load status of all the paths. It dynamically adjusts the value of each weight according to the current load status of each path (the static weight configuration has proven to be not applicable in [10]). However, when deploying the (d, m) policy, the value of d and m are constant after the initializations. Thus, when the values are not appropriately assigned, (d, m) policy will not perform as well as the weighted traffic splitting solutions. Hence, the HOLMES AI module is responsible for analyzing the overall heterogeneity degree of a DCN, and guiding the flow scheduling algorithm to set appropriate values of the algorithm factors (d and m). The detailed design and implementation of the HOLMES AI module is our future work.

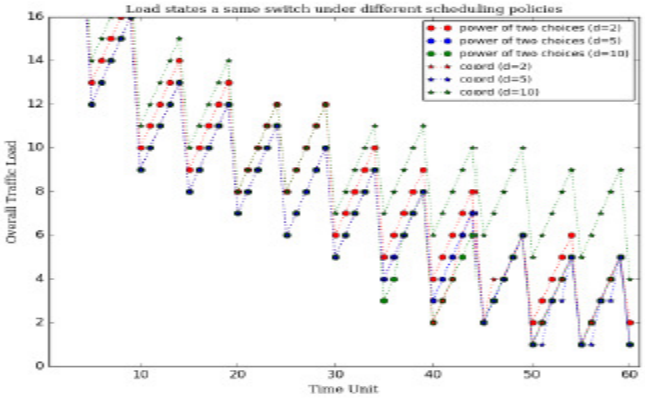

Fig. 5. Changing trend of a switch's overall traffic load under different policies

D. Technical Challenges in Hardware Implementations

Some technical challenges need to be considered to implement HOLMES in real-world data centers. We now summarize these challenges in hardware implementations.

1) Handling the packet reordering

The flow scheduling algorithm in HOLMES can be implemented with different data granularities: per packet scheduling, per flow scheduling or some intermediate data sizes, e.g. flowcell [24], flowlet [10], etc. When using the TCP transmission protocol and implementing the per packet (or flowcell) scheduling, some studies have shown that this finegrained traffic splitting techniques cause packet reordering and lead to severe TCP throughput degradations [23]. Therefore, the packet reordering problem needs to be considered when implementing the fine-grained HOLMES traffic scheduling algorithm. A viable solution is to deploy the JUGGLER network stack in data center traffic transmissions. JUGGLER exploits the small packet delays in datacenter networks and the inherent traffic bursts to eliminate the negative effects of packet reordering while keeping state for only a small number of flows at any given time. This practical reordering network stack is lightweight and can handle the packet reordering efficiently.

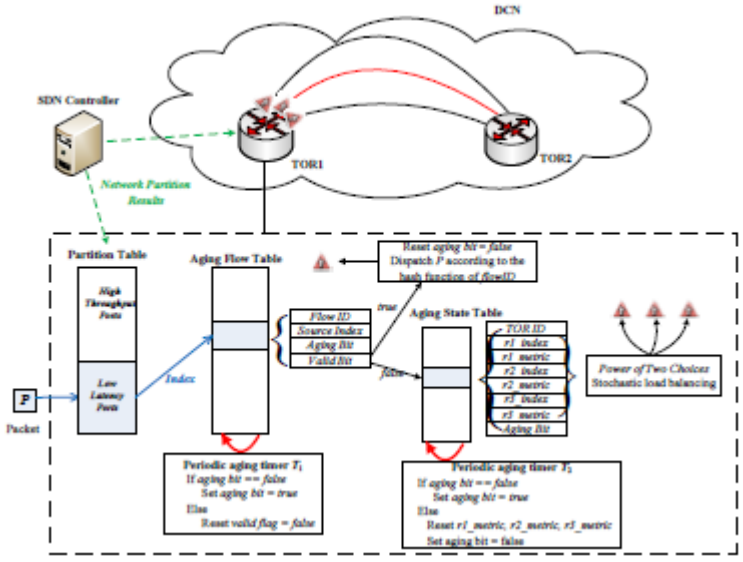

Fig. 6. Overview of HOLMES forwarding and state tables: A new flow table entry is set up by applying the (2, 1) policy in the state table. The period timer of each table is triggered every time period T1 and T2 to age out inactive flows and update the load status of each candidate end-to-end path.

2) Design of DCN forwarding and state tables

The design of the forwarding and state tables is also a noteworthy challenge. An appropriate approach should cope with the small time budget as well as the small register usage. We now propose a viable design to implement the per-flow scheduling algorithms of HOLMES

As shown in Fig. 6, a TOR switch maintains a flow table anda state table. The two tables work together to execute the load mbalancing policy attained from the SDN controller. Specifically,when a packet arrives, its flowID is hashed to map the packet to a flow table entry. If the table entry is valid, the packet is dispatched to the path indicated by the stored hash applied to the packet's flow ID. On

the contrary, if the packet's flowID is not maintained in the flow table, the TOR switch will look up the destination TOR ID in the state table. After that, the (d, m) policy is applied to compare the load states of the three candidate end-to-end paths to the destination TOR (r1_metric, r2 metric and r3_metric), and assign the packet to the optimal path. Two of the three end-to-end paths are randomly selected. The third one is the optimal path from the last selection. Finally, the flow ID and the hash of the chosen path will be inserted into the flow table.

The information of each table needs to be updated periodically to keep the real-time status of the traffic and paths. Thus, we associate an aging bit with each table entry. The aging bit of the flow table is responsible for marking inactive or idle flows: when a packet's flow ID maps the information in the forwarding table, the aging bit is cleared to indicate that the flow entry is active. A timer process visits every table entry every aging timeout T1. When the timer process visits a table entry, it either turns on the aging bit or invalidates the entry if the aging bit is already on. In other words, T1 is the timeout threshold to age out inactive flows, which is proportional to the size of the scheduling unit e.g. per-flow, per-flowcell, etc. If the packet's flow ID is not maintained in the flow table, the TOR switch will execute the (d, m) policy on the state table. Thus, T1 can also be considered as the time period to trigger the execution of the HOLMES scheduling algorithm. On the other hand, another timer process runs together with the aging bit of the state table to update the load status of each candidate end-to-end path. The timeout threshold to age out the old status information in the state table is set to T2. To ensure that the latest load state information is used when executing the (d, m) policy, the value of T1 and T2 should satisfy: $T1 \geq T2$. Moreover, in most cases, the global congestion control signals deployed in the flow scheduling algorithms are the feedback signals from the receivers of the end-to-end paths. Thus, we further get: $T1 \geq T2 \geq RTT$. Key et al have suggested that the policy that periodically sampling a random path and retaining the best paths may perform well.

The periodically sampling of path congestion states in the state table makes the real-time collection of status information becomes a technical challenge. The state collection operations should not introduce obvious transmission overheads and performance penalties. Especially in the TCP incast scenarios where multiple source nodes transmit traffic at the same time to a common destination node, the state collection operations introduce additional traffic and are prone to cause DCN throughput collapse. A viable solution for collecting the real-time congestion status is deploying RepSYN as the signal to detect the load conditions of the multiple paths, as shown in [33]: before transmitting data among multi-rooted paths, multiple TCP connections are established; however, traffic is only transmitted using the first established connection and the other connections are ended immediately. The delay experienced by an SYN reflects the latest congestion condition of the corresponding path, and thus the congestion states can be collected. Moreover, this solution only replicates SYN packets to probe the network, which does not aggravate the TCP incast in a DCN.

The state table only needs to periodically maintain the congestion states of two randomly chosen paths and the congestion-optimal path in the latest time unit. Compared with some other congestion-aware solutions e.g., CONGA [10], RepFlow [32], the storage complexity has been dramatically optimized. In order to make the scheduling results of the (d, m) policy more effective, we choose the disjoint end-to-end paths (paths with different intermediate aggregate or spine switches) to avoid the scenario that the same local congestions is shared by multiple end-to-end paths. This implementation is applicable for more complex multi-tier Leaf-Spine topologies or asymmetric DCN topologies.

3) Dealing with the stale information

When implementing HOLMES scheduling algorithm with packet granularity, the transmission latency of a packet is so small that the information refresh rate in the state table cannot catch up with. Correspondingly, the load balancing algorithm has to use the stale information to make the scheduling

decisions. Mitzenmacher et al have pointed out that the delayed information leads to a herd behavior of the scheduling results: data will herd toward a previously light loaded path for much longer time than it takes to fulfill the path. Thus, another technical challenge is to deal with the stale information used in the load balancing algorithms.

To further discuss the effects of the stale information on the execution results of different scheduling policies, we design another experiment that evaluates the load balancing status of a same TOR switch under different state table timers. We simulate a scenario that traffic arrives at a TOR switch with a fixed rate (1 packet/time unit). The TOR switch contains 10 output ports with fixed data transmission rate (1 packet/ 8 time units). Two different timers are deployed to periodically update the information of the TOR state table (state information is updated every 5 or 20 time units respectively). We focus on evaluating the load balancing status of the same TOR switch under three different flow scheduling approaches in this experiment: (2, 1) policy, (5, 1) policy, as well as the deterministic policy (d =N=10 in the definition of the (d, m) policy).

Figs. 7A, 7C and 7E show the changing trend of the TOR load balancing states under the deterministic policy, (2, 1) policy and (5, 1) policy respectively, when the state table is updated every 5 time units. As comparisons, Fig. 7B, 7D and 7F show the performance of the three policies with the state table updated every 20 time units. We analyze the effects of the stale information age to different scheduling policies.

Figs. 7A and 7B show the execution results of the deterministic scheduling policy when the refresh period of the state table is set to 5 or 20 time units respectively. We find from the two figures that, although the deterministic policy provides optimal performance using the perfect run-time state information, it suffers from the effects of the stale information seriously. We see from Fig. 7A that even if the update period is set short (5 time units), the load states of the TOR ports show obvious herd phenomena: traffic will herd toward a previously light loaded port first; the next refresh of the state table will put the port high up the load list and the port will become light loaded again; then the path will be heavy loaded again after several time periods as the next herd sees that it is light loaded. From Fig. 7A we see that each TOR port's load condition shows obvious periodically changing trend (heavy loaded, light loaded, heavy loaded, and so on), and this herd phenomenon is more obvious as the age of the stale state information increases. As shown in Fig. 7B, the amplitude of the fluctuation of each port's load curve is impressively obvious compared with Fig. 7A: the maximum load of a port has increased from 5 packets to more than 15 packets. Compared with the other two approaches (shown in Fig. 7D and 7F), the load balancing condition under the deterministic policy becomes the worst as the age of the stale information increases. Thus, adaptability of the deterministic policy to the stale information is poor.

Theoretically, the load balancing condition of the TOR switch under (5, 1) policy is in second place among all the three approaches, when deploying the perfect run-time state information. Clearly, deploying a larger value of d in the (d, m) policy increases the probability of choosing the light loaded output port. However, compare Fig. 7E with Fig. 7C, we find that the TOR's load balancing condition under (5, 1) policy (shown in Fig. 7E) does not outperforms the (2, 1) policy (shown in Fig. 7C) when using the short aged stale information. Moreover, compared with Fig.7D and 7F, we find that as the age of the stale information increases, (2, 1) policy gradually outperforms (5, 1) policy. The experimental results reflect an issue that compared with the probability of choosing the optimal path, the adaptability for herd behavior becomes the dominant factor for performance optimization.

Comparing Fig. 7C with 7D, we see that increasing the age of the stale information does not affect the execution results of (2, 1) policy: the maximum load of a TOR port is still maintained less than 5 packets when the information update period increases from 5 time units to 20 time units. Comparing

Figs. 7B, 7D and 7F, we also find that (2, 1) policy performs best among all the three approaches when using the long aged stale information. The experimental results indicate that (2, 1) policy provides optimal adaptability for HOLMES with the stale information, especially when the age of the information is large. The main reason is that (2, 1) policy combines the advantages of both worlds: It deploys real load information to schedule traffic, yet rejects the herd behavior much more strongly than the other two approaches.

Besides adjusting the value of modeling factors, e.g. d, m, another applicable solution is to deploy the machine learning methods to predict the current load state based on the historical stale information. Fox et al [95] deploy queue length as the metric for load balancing and notice rapid oscillations in queue lengths, caused by the stale information. To overcome this limitation, they further change the manager stub to keep a running estimate of the change in queue lengths between successive reports and show that these estimations are sufficient to eliminate the oscillations. Dahlin [98] also estimates the job arrival rates and the age of the stale information, and proposes corresponding algorithms to balance the traffic load in a distributed system. Similar approaches can also be deployed in HOLMES, which estimate the current load states of different paths by analyzing the successive historical state information. This function will be realized in the HOLMES AI module, and will be another one of our future works.

Overall, the simulation experimental results validate the modeling results in Section IV. They show that HOLMES load balancing algorithm is stable and adaptable in heterogeneous DCNs.

6. CONCLUSION

This paper presents HOLMES, a novel DCN flow scheduling scheme, which tackles mixed (mice vs. elephants) data center traffic. Using a stochastic performance model, we first prove the necessity of isolating mice and elephants with a closed form. We then present the HOLMES architecture that partitions a DCN into high-throughput and low-latency sub-networks. We further design a stochastic and global congestion-aware load balancing algorithm that schedules the corresponding DC traffic to each sub-network. Simulation results show that HOLMES network partition policy can successfully eliminate the interference between the mouse and elephant data flows. Finally, we prove that HOLMES flow scheduling algorithm is stable and scalable for large-scale data centers.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” In ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63-74.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, P. Patel, and S. Sengupta. “VL2: A Scalable and Flexible Data Center Network,” In ACM SIGCOMM Computer Communication Review, 2009, 39(4): 51-62.

[3] C. Hopps. Analysis of an Equal-Cost Multi-Path algorithm. RFC 2992, Network Working Group, 2000.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” In Proc. USENIX NSDI, 2010, Vol. 10, pp. 19-19.

[5] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, “Per-packet load-balanced, low-latency routing for Clos-based data center networks,” In Proc. ACM CoNEXT, 2013, pp. 49-60.

[6] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, “On the impact of packet spraying in data center networks,” In Proc. IEEE INFOCOM, 2013, pp. 2013-2018.

[7] 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 Computer Communication Review, 2010, 40(4): 63-74.

[8] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. M. B. Prabhakar, and S. Shenker, “pFabric: Minimal near-optimal datacenter transport,” In ACM SIGCOMM Computer Communication Review, 2013, 43(4): 435-446.

[9] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, “DeTail: Reducing the flow completion time tail in datacenter networks,” In ACM SIGCOMM Computer Communication Review, 2012, 42(4): 139-150.

[10] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," In ACM SIGCOMM Computer Communication Review, 2014, 44(4): 503-514.

[11] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," In In Proc. ACM EuroSys, 2014.

[12] P. Wang, H. Xu, Z. Niu and D. Han, "Expeditus: Distributed congestion-aware load balancing in clos data center networks," In Proc. ACM CoNEXT on Student Workshop, 2014, pp. 1-3.

[13] S. Ghorbani, B. Godfrey, Y. Ganjali and A. Firoozshahian, "Micro load balancing in data centers with DRILL," In Proc. ACM HotNets, 2015, pp. 17.

[14] N. Katta, M. Hira, C. Kim, A. Sivaraman and J. Rexford, "HULA: Scalable load balancing using programmable data planes," In Proc. ACM SOSR, 2016.

[15] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," In IEEE Transactions on Services Computing, 2010, 3(4): 266–278.

[16] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," In ACM SIGCOMM Computer Communication Review, 2015, 45(4): 537-550.

[17] D. A. Hayes and G. Armitage, "Revisiting TCP congestion control using delay gradients," In Proc. IFIP Networking, 2011.

[18] W. Wang, Y. Sun, K. Salamatian and Z. Li, "Adaptive path isolation for elephant and mouse flows by exploiting path diversity in datacenters," In IEEE Transactions on Network and Service Management, 2016, 13(1): 5-18.

[19] V. Liu, D. Halperin, A. Krishnamurthy and T. Anderson, "F10: a fault-tolerant engineered network," In Proc. USENIX OSDI, 2010.

[20] J. Kim, W. J. Dally, S. Scott and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," In ACM SIGARCH Computer Architecture News, 2007, 35(2): 126-137.

[21] J. H. Ahn, N. Binkert, A. Davis, M. McLaren and R. S. Schreiber, "HyperX: topology, routing and packaging of efficient large-scale networks," In Proc. ACM SC, 2009, p. 41.

[22] J. Kim, W. J. Dally, S. Scott and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," In ACM SIGARCH Computer Architecture News, 2008, 36(3): 77-88.

[23] A. Dixit, P. Prakash and R. R. Kompella, "On the Efficacy of Fine-Grained Traffic Splitting Protocols in Data Center Networks," In ACM SIGCOMM Computer Communication Review, 2011, 41(4): 430-431.

[24] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," In ACM SIGCOMM Computer Communication Review, 2015, 45(4): 465-478.

[25] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," In Proc. ACM CoNEXT, 2013, pp. 151-162

[26] T. Edsall, A. Fingerhut, T. Lam, R. Pan, and G. Varghese, "Flame: Efficient an robust hardware load balancing for data center routers," [Department of Computer Science and Engineering], University of California, San Diego, 2012.

[27] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," In ACM SIGCOMM Computer Communication Review, 2013, 43(4): 15-26.

[28] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," In ACM SIGCOMM Computer Communication Review, 2013, 43(4): 3-14.

[29] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," In Proc. ACM CoNEXT, 2011, pp. 8:1-8:12.

[30] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," In ACM SIGCOMM Computer Communication Review, 2005, 35(4): 253-264.

[31] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," In Proc. IEEE INFOCOM, 2001, pp. 1300-1309.

[32] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," In Proc. IEEE INFOCOM, 2014, pp. 1581-1589.

[33] S. Liu, W. Bai, H. Xu, K. Chen and Z. Cai "RepNet: Cutting tail latency in data center networks with flow replication," In Computer Science, 2014.

[34] R. Gallager, "A minimum delay routing algorithm using distributed computation," In IEEE Transactions on Communications, 1977, 25:73-85.

[35] N. Michael and A. Tang, "Halo: Hop-by-hop adaptive link-state optimal routing," In IEEE/ACM Transactions on Networking, 2015, 23(6): 1862-1875.

[36] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," In ACM SIGCOMM Computer Communication Review, 2007, 37(2): 51-62.

[37] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," In Proc. ACM STOC, 1981, pp. 263-277.

[38] R. Zhang and N. McKeown, "Designing a fault-tolerant network using valiant load balancing," In Proc. IEEE INFOCOM, 2008.

[39] R. Zhang and N. McKeown, "Designing a predictable internet backbone with valiant load-balancing," In Proc. IEEE/ACM IWQoS, 2005, pp. 178-192.

[40] D. Wischik, C. Raiciu, A. Greenhalgh , and M. Handley, “Design, implementation and evaluation of congestion control for multipath TCP,” In Proc. USENIX NSDI, 2011, Vol. 11, pp. 8-8.
[41] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, “Improving datacenter performance and robustness with multipath tcp,” In ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63-74.
[42] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, “Fastpass: A centralized zero-queue datacenter network,” In ACM SIGCOMM Computer Communication Review, 2014, 44(4): 307-318.
[43] M. Mizenmacher, “The power of two choices in randomized load balancing,” In IEEE Transactions on Parallel and Distributed Systems, 2001, 12(10): 1094-1104.
[44] Y. T. He and D. G. Down, “Limited choice and locality considerations for load balancing,” In Performance Evaluation, 2008, 65(9): 670-687.

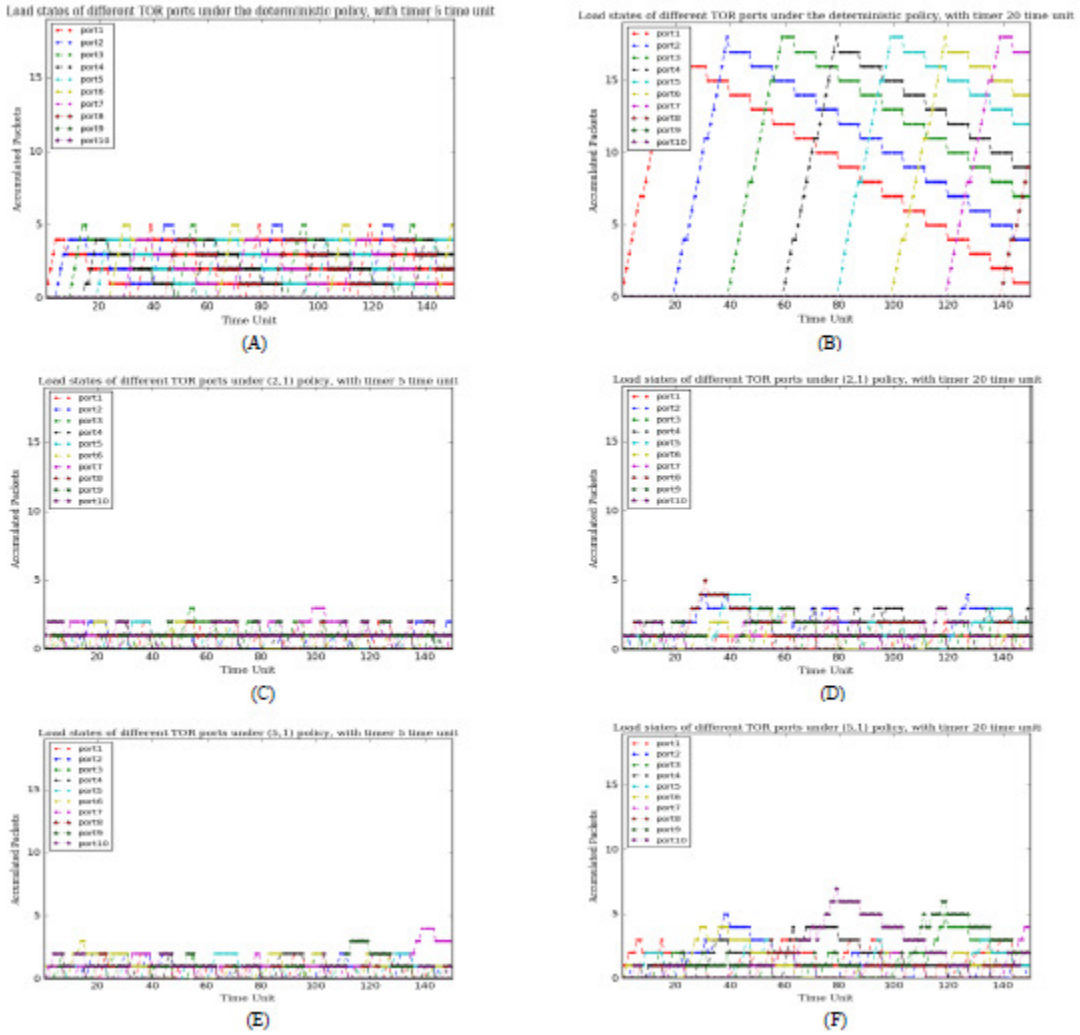

Fig. 7. Load states changing trend of 10 TOR ports under deterministic policy (A, B), (2, 1) policy (C, D) and (5, 1) policy (E, F) with different state table timer (T2=5, T2=20) respectively. The 10 ports are located on the same TOR switch.