2017 IEEE 10th International Conference on Cloud Computing

Efficient Hierarchical Traffic Measurement in Software-Defined Datacenter Networks

Shiping Chen†, You Zhou‡, Shigang Chen‡

† School of Optical-Electrical and Computer Engineering,
University of Shanghai for Science and Technology, Shanghai, China
‡ Department of Computer and Information Science and Engineering,
University of Florida, Florida, USA

***Abstract*—Software-defined datacenters combine centralized resource management, software-defined networking, and virtualized infrastructure to meet diverse requirements of cloud computing. To fully realizing their capability in traffic engineering and flow-based bandwidth management, it is critical for the switches to measure network traffic for both individual flows between virtual machines and aggregate flows between clusters of physical or virtual machines. This paper proposes a novel hierarchical traffic measurement scheme for software-defined datacenter networks. It measures both aggregate flows and individual flows that are organized in a hierarchy with an arbitrary number of levels. The measurement is performed based on a new concept of hierarchical virtual counter arrays, which record each packet only once by updating a single counter, yet the sizes of all flows that the packet belongs to will be properly updated. We demonstrate that the new measurement scheme not only supports hierarchical traffic measurement with accuracy, but does so with memory efficiency, using a fewer number of counters than the number of flows.**

I. Introduction

Cloud computing systems are becoming increasingly large in scale. As a key direction of cloud platform development, software-defined datacenters combine centralized resource management and software-defined networking with diverse client requirements and IaaS (infrastructure as a service) provisioning [1], [2], [3]. These datacenters will offer great flexibility in virtualization of computing, storage and network resources. However, achieving both scalability and sophisticated service offerings places unprecedented challenges in resource management and real-time workload monitoring.

An efficient datacenter-wide monitoring function is essential to optimal resource management. Traditionally, each server measures its workload and report residual computing/storage resources to the controller for resource allocation. In software-defined datacenters, it is also important for the switches to measure network traffic, which provides critical information for bandwidth management and traffic engineering. This task is challenging when commodity SDN switches are used, which operate at 10 or 100 Gbps line speed with limited on-board resources. The focus of this paper is to investigate how to perform flow-level traffic measurement with high efficiency in both processing overhead and memory consumption.

Software-defined networks (SDN)[4][5][6] not only make management easier and more flexible, but also support network-wide traffic engineering. They quickly make their way into datacenters [7], [8]. Their ability of routing individual flows on different paths is important to quality-of-service provisioning and load balancing. But traffic engineering at the flow level requires the controller to know the flow rates. While the OpenFlow standard [9] supports per-flow traffic measurement with statistic counters in the flow table, it is however not scalable.

SDN switches typically have a few thousands of entries in their TCAM-based flow tables. Even the high-end Broadcom Trident2 chipset supports only 16K forwarding rules [6]. On the other hand, datacenter networks are experiencing more and more flows. For example, in a real datacenter with 1500 operational server clusters [10], the average arrival rate reaches 100K flows per second. Clearly, there are not enough entries in the flow tables to support per-flow routing/measurement [5], [11]. To address this problem, the prior work uses wildcard forwarding rules in the flow tables for aggregate routing, and performs per-flow traffic measurement in on-die cache memory such as SRAM. The controller will identify elephant flows or other flows of interest based on the policy or performance requirements, and reroute these flows on their individual paths.

There exist many memory-efficient traffic measurement algorithms designed to work in on-chip SRAM, which is small but larger than TCAM. Most of the algorithms (e.g., CountMin [12], Counter braids [13], [14], and Counter tree [15]) trade for memory efficiency with more processing overhead, updating multiple counters for each packet, with the exception of RCS [16], which updates one counter per packet. Moreover, these algorithms are designed in the context of Internet, where flows are assumed to be independent, i.e., a packet belongs to one and only one flow and packets of any two flows do not overlap. This is however not necessarily true for traffic management in software-defined datacenters that support network-resource virtualization. For example, an institute may want to deploy a large virtual campus network in the cloud, connecting multiple virtual sites (implemented on designated racks in

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.29

a datacenter). Work groups are deployed on virtual sites, and each group contains virtual machines that communicate with other virtual machines in the same or different sites/groups. Quality-of-service requirements are specified for the virtual connections between sites, between work groups, and for communications between VMs. To enforce the QoS requirements and perform traffic engineering such as re-routing, we need to monitor both aggregate flows between sites (work groups) and individual data (TCP) flows between VMs. These flows are not independent because an individual data flow between two VMs may belong to an aggregate flow between two work groups that contain the VMs, which in turn belongs to a high-level aggregate flow between sites that host the work groups. In this case, all flows form a hierarchical structure where an aggregate parent flow contains multiple individual flows or lower-level aggregate flows as children.

It is a burden for the controller to keep track of all individual flows in order to compute the traffic volume (size) of aggregate flows; in fact, this is often not feasible because many highly compact measurement algorithms [12], [13], [14], [15], [16] do not even include individual flow identifiers in their traffic synopsis (from which per-flow traffic information is extracted). Alternatively, we can require the switches to measure both individual flows and aggregate flows, which may however require each packet to be recorded multiple times (even for RCS [16]), one for each flow that the packet belongs to. This approach multiplies the per-packet processing overhead, which is highly undesirable because the traffic measurement function may become a throughput bottleneck or a limiting factor for the structure of user-defined flow hierarchies.

This paper proposes a novel hierarchical traffic measurement scheme for software-defined datacenter networks. It measures both aggregate flows and individual flows in a hierarchy with an arbitrary number of levels. It records each packet only once by updating a single counter, yet the sizes of all flows that the packet belongs to will be properly updated. Our idea is to introduce hierarchically-constructed virtual counter arrays, each storing the size of one flow. The virtual arrays of individual flows select their counters from the virtual arrays of their parent aggregate flows. The virtual arrays of those aggregate flows select their counters from the virtual arrays of higher-level parent flows. The virtual arrays from the highest-level aggregate flows select their counters from a physical counter array, which can support a virtually unlimited number of flows due to counter sharing. When a switch receives a packet, it updates one counter that is shared by the virtual arrays of all flows that the packet belongs to, simultaneously updating information for all those flows. We derive the mathematical formula for estimating the size of any (individual or aggregate) flow from its virtual counter array. Our simulation shows that the new measurement scheme not only supports hierarchical traffic measurement with good accuracy, but does so with memory efficiency, using a fewer number of counters than the number of flows overall (thanks to counter sharing) — an important feature for implementing such a function with on-die SRAM memory of SDN switches.

II. Flow Modeling and Problem Statement

Consider a flow hierarchy consisting of k levels of flows to be measured. The set of flows at the ith level is denoted as S_i, $1 \leq i \leq k$, where the flows at the bottom kth level are individual flows and the flows at other levels are aggregate flows, each containing a subset of flows at the next level. Let $S_* = \bigcup_{i=0}^{k} S_i$. For an aggregate flow f_i at the ith level, $1 \leq i < k$, let $F(f_i)$ be the subset of $(i+1)$th-level flows that it contains. Flow f_i is called the *parent flow*, and flows in $F(f_i)$ are called *children flows*.

As an example, at each top-of-rack switch in a datacenter, we may want to measure the outbound traffic, including both the aggregate flows from this rack to other racks and the individual TCP flows from source addresses in the rack to destinations outside of the rack. In this case, we model the flows in a two-level hierarchy, where the first level is the set of aggregate flows, each contains all packets from the rack to another rack, and the second level is the set of TCP flows, each contains all packets from certain source address/port in the rack to certain destination address/port outside the rack. The identifier for each aggregate flow is the rack address, which may be the address prefix for the subnet assigned to the rack. The identifier for each TCP flow is the classical five-element tuple, including source address, source port, protocol ID for TCP, destination address and destination port. The identifiers can all be found from the packet header.

The problem is to measure the size of each flow in S_* in terms of number of packets or number of bytes; for simplicity, our technical discussion will focus on the former, but the proposed scheme can be easily modified for the latter. In order for the switch to keep up with the line speed, we should minimize both processing time and memory consumption. For hardware on-chip implementation, it is desired to reduce the number of memory accesses to the minimum of one counter update per packet, and reduce the number of counters to be fewer than the number of flows.

III. Hierarchical Virtual Counter Arrays

In this section, we propose a hierarchical traffic measurement scheme, called Hierarchical Virtual Counter Arrays (HVC). Below we first describe the construction of these arrays, then explain how to record packets in the virtual arrays, and derive the flow-size estimator based on the virtual arrays.

A. Virtual Counter Arrays for Flows at Different Levels

Each switch allocates an array C of m counters for the measurement function. The jth counter is denoted as $C[j]$, $1 \leq j \leq m$. For each first-level flow f_1, we construct a virtual counter array C_{f_1} by pseudo-randomly selecting s_1 counters from C, where s_1 is a pre-set parameter. Let $C_{f_1}[j]$

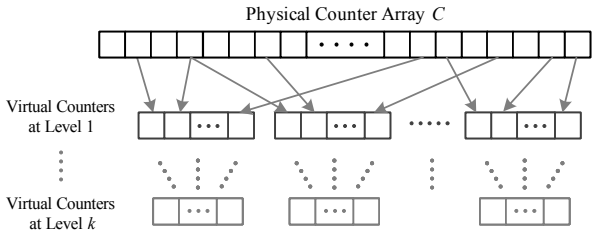


Fig. 1: An illustration of hierarchical virtual counters.

be the jth counter in C_{f_1}, $1 \le j \le s_1$. Its selection is given as follows:

$$C_{f_1}[j] = C[H_j(f_1)], \quad 1 \le j \le s_1, \tag{1}$$

where H_j, $1 \le j \le s_1$, is a hash function. It can be implemented from a master hash function H_* as follows:

$$H_j(x) = H_*(x \oplus H_*(j)), \tag{2}$$

where $\oplus$ is the XOR operator.

For each flow f_i at the ith level, $1 < i \le k$, we construct a virtual counter array C_{f_i} by pseudo-randomly selecting s_i counters from $C_{f_{i-1}}$, where f_{i-1} is the parent flow of f_i at the $(i-1)$th level, and $s_i < s_{i-1}$. The selection is given as follows.

$$C_{f_i}[j] = C_{f_{i-1}}[H_j(f_i)], \quad 1 \le j \le s_i. \tag{3}$$

Although multiple counters are used in the virtual array of each flow, those counters share the same physical counters in C, as illustrated in Fig. 1. In fact, from the same set of physical counters in C, we can construct a virtually unlimited number of virtual counter arrays. Our numerical evaluation will demonstrate that HVC can provide good estimation for the sizes of individual/aggregate flows with good accuracy when the total number m of counters is fewer than the number of flows.

B. Measurement at Switches

Each switch is informed by the controller of the number of levels in the flow hierarchy and the flow identifiers at all levels that can be extracted from the packet headers. Following an earlier example, the controller may assign each rack a 24-bit prefix for its subnet address and specify that at the first level, the aggregate flow — to which an arrival packet belongs — will be identified by the 24-bit prefix of the destination address in the IP header; at the second level, the TCP flow to which the packet belongs will be identified by the classical five-element tuple from the TCP/IP headers.

When a switch receives a packet, its extracts the identifiers of the flows at all levels that the packet belongs to, denoted as f_1, ..., f_k, where f_1 is the parent flow of f_2, which is in turn the parent flow of f_3, and so on. The switch then randomly chooses a counter from the virtual array C_{f_k} of flow f_k and increases that counter by one. More specifically, the switch generates a random number j in the range of $[1, s_k]$, where s_k is the size of a virtual array at the kth level. It updates $C_{f_k}[j] := C_{f_k}[j] + 1$, where $:=$ is the assignment operator. Because $C_{f_k}[j]$ is chosen from $C_{f_{k-1}}$ where f_{k-1} is the parent flow of f_k, we know that a counter in the virtual array of flow f_{k-1} is also increased by one. By the same token, a counter from the virtual array of any aggregate flow to which the packet belongs will automatically be increased by one. One counter update records the packet in the virtual arrays of all k flows, f_1, ..., f_k.

From (3), we know that $C_{f_k}[j]$ is in fact the following counter in $C_{f_{k-1}}$ at the $(k-1)$th level:

$$C_{f_k}[j] = C_{f_{k-1}}[H_j(f_k)] = C_{f_{k-1}}[H_*(f_k \oplus H_*(j))],$$

which is in turn the following counter in $C_{f_{k-2}}$ at the $(k-2)$th level:

$$C_{f_{k-1}}[H_*(f_k \oplus H_*(j))] = C_{f_{k-2}}[H_*(f_{k-1} \oplus H_*(H_*(f_k \oplus H_*(j))))].$$

Repeating the above reasoning, we have $C_{f_k}[j]$ is in fact the following counter in the physical array C:

$$C_{f_k}[j] = C[H_*(f_1 \oplus H_*(f_2 \oplus ...H_*(H_*(f_k \oplus H_*(j)))))].$$

Therefore, after receiving a packet, the actual operation that the switch performs is to increase the physical counter $C[H_*(f_1 \oplus H_*(f_2 \oplus ...H_*(H_*(f_k \oplus H_*(j)))))]$ by one, namely,

$$C[H_*(f_1 \oplus H_*(f_2 \oplus ...H_*(H_*(f_k \oplus H_*(j))))) \mathrel{+}= 1, \tag{4}$$

in the C style coding. It takes $(k+1)$ hash functions that can be pipelined for hardware implementation, and one counter update. For the earlier example of two-level flows which will be used in our simulations, the operation is

$$C[H_*(f_1 \oplus H_*(f_2 \oplus H_*(j)))] \mathrel{+}= 1, \tag{5}$$

which requires three hashes and one counter update. We stress that the virtual arrays of C_{f_1}, ..., C_{f_k} are not constructed at the switch. They are logical structures that are reflected in (4) and (5), which make sure that one counter update will show up in the virtual arrays of all k relevant flows during flow-size estimation when the controller needs this information.

To measure the flow size in terms of bytes, we change (4) as follow

$$C[H_*(f_1 \oplus H_*(f_2 \oplus ...H_*(H_*(f_k \oplus H_*(j))))) \mathrel{+}= \frac{l}{q}, \tag{6}$$

where l is the number of bytes in the packet and q is a normalizing integer constant for reducing the increment to the counter.

C. Estimation at Controller

Upon query from the controller, a switch will report its entire physical array C or a selected subset of counters as specified in the query. Whenever a counter overflows during measurement, the switch will report the counter value to the controller and reset the counter to zero. Therefore, the controller has all information needed to recover the proper

values of all counters at all switches. It may also choose to reset all counters to zeros for starting measurement anew.

During resource allocation, if the controller wants to know the size of a certain flow f_i at the ith level, it first locates the array C that carries the information. Consider the earlier example of two-level hierarchy. For a flow at the first level, we use the flow identifier (24-bit address prefix) to locate the top-of-rack switch and the use the counter array C from that switch for flow-size estimation. For a flow at the second level, we use the 24-bit prefix in the source address of the flow identifier to locate the switch. Below we describe the two cases for flow-size estimation.

First, consider a flow f_i at the ith level, where $1 < i \leq k$. The controller constructs the virtual array C_{f_i} of flow f_i and the virtual array $C_{f_{i-1}}$ of its parent flow f_{i-1}. Let n_{c_i} be the number of packets recorded in C_{f_i}, i.e., $n_{c_i} = \sum_{j=1}^{s_i} C_{f_i}[j]$. Let $n_{c_{i-1}}$ be the number of packets recorded in $C_{f_{i-1}}$, i.e., $n_{c_{i-1}} = \sum_{j=1}^{s_{i-1}} C_{f_{i-1}}[j]$. The array C_{f_i} records the size n_{f_i} of flow f_i and noise from other flows due to counter sharing. Let e be a random variable for the number of noise packets from other flows. We have

$$n_{c_i} = n_{f_i} + e. \tag{7}$$

The noise e can be measured statistically as follows: There are $n_{c_{i-1}} - n_{f_i}$ packets that do not belong to flow f_i but are recorded in $C_{f_{i-1}}$. Consider any of such packets. The probability for it to be recorded in C_{f_i} is $\frac{s_i}{s_{i-1}}$. If n_{f_i} is negligibly small when comparing with $n_{c_{i-1}}$, e approximately follows a binomial distribution:

$$e \sim Bino\big(n_{c_{i-1}} - n_{f_i}, \frac{s_i}{s_{i-1}}\big) \approx Bino\big(n_{c_{i-1}}, \frac{s_i}{s_{i-1}}\big). \tag{8}$$

Hence, the expected number of noise packets in C_{f_i} is $E(e) = E(n_{c_{i-1}})\frac{s_i}{s_{i-1}}$. Taking expectation on both sides of (7), we have

$$\begin{aligned} E(n_{c_i}) &= n_{f_i} + E(e) = n_{f_i} + E(n_{c_{i-1}})\frac{s_i}{s_{i-1}}, \\ n_{f_i} &= E(n_{c_i}) - E(n_{c_{i-1}})\frac{s_i}{s_{i-1}}. \end{aligned} \tag{9}$$

Replacing $E(n_{c_i})$ with the instance value $\sum_{j=1}^{s_i} C_{f_i}[j]$, which is the observed number of packets recorded in C_{f_i}, and replacing $E(n_{c_{i-1}})$ with the instance value $\sum_{j=1}^{s_{i-1}} C_{f_{i-1}}[j]$, which is the observed number of packets recorded in $C_{f_{i-1}}$, we have an estimation $\hat{n}_{f_i}$ for n_{f_i} as follows.

$$\hat{n}_{f_i} = \sum_{j=1}^{s_i} C_{f_i}[j] - \frac{s_i}{s_{i-1}} \sum_{j=1}^{s_{i-1}} C_{f_{i-1}}[j]. \tag{10}$$

Next, we consider a flow f_1 at the first level. Following a similar process as above by replacing f_i with f_1 and replacing $C_{f_{i-1}}$ with C, we have the following estimation $\hat{n}_{f_1}$ for the size n_{f_1} of flow f_1:

$$\begin{aligned} \hat{n}_{f_1} &= \sum_{j=1}^{s_1} C_{f_1}[j] - \frac{s_1}{m} \sum_{j=1}^{m} C[j] \\ &= \sum_{j=1}^{s_1} C_{f_1}[j] - \frac{n s_1}{m}, \end{aligned} \tag{11}$$

where $n = \sum_{j=1}^{m} C[j]$ is the total number of packets recorded by the switch.

D. Estimation Accuracy

We now analysis relative bias and standard error of $\hat{n}_{f_i}$, $1 \leq i \leq k$.

1) Relative Bias: we first analyze the mean of n_{c_i}, which is the total number of packets recorded in C_{f_i}. From (7), we know that $n_{c_i} = n_{f_i} + e$. Combining with (8), we have

$$E(n_{c_i}) = E(n_{f_i}) + E(e) = n_{f_i} + E(n_{c_{i-1}})\frac{s_i}{s_{i-1}}. \tag{12}$$

Combining the above formula with (10), we can calculate the mean of $\hat{n}_{f_i}$:

$$\begin{aligned} E(\hat{n}_{f_i}) &= E\big(\sum_{j=1}^{s_i} C_{f_i}[j]\big) - \frac{s_i}{s_{i-1}} E\big(\sum_{j=1}^{s_{i-1}} C_{f_{i-1}}[j]\big) \\ &= E(n_{c_i}) - E(n_{c_{i-1}})\frac{s_i}{s_{i-1}} \\ &= n_{f_i} + E(n_{c_{i-1}})\frac{s_i}{s_{i-1}} - E(n_{c_{i-1}})\frac{s_i}{s_{i-1}} \\ &= n_{f_i}. \end{aligned} \tag{13}$$

The relative bias of $\hat{n}_{f_i}$, which is defined as $\frac{E(\hat{n}_{f_i})}{n_{f_i}} - 1$, is

$$Bias(\frac{\hat{n}_{f_i}}{n_{f_i}}) = \frac{E(\hat{n}_{f_i})}{n_{f_i}} - 1 = 0. \tag{14}$$

Therefore, our estimation of n_{f_i}, $1 \leq i \leq k$, is unbiased.

2) Standard Error: Consider an arbitrary counter in the virtual counter C_{f_i}. We use a random variable X_i for the value of the counter. Let F_i be the portion of X_i contributed from flow f_i, and E_i be the portion of X_i contributed from other flows. Clearly, $X_i = F_i + E_i$. Since each packet of f_i has the same probability $\frac{1}{s_i}$ to increase the value of the counter by one. Hence, F_i follows a binomial distribution: $F_i \sim Bino\big(n_{f_i}, \frac{1}{s_i}\big)$. Hence,

$$Var(F_i) = \frac{n_{f_i}}{s_i}(1 - \frac{1}{s_i}). \tag{15}$$

Now consider E_i with a hierarchical view. Each packet of another first-level flow has a probability of $\frac{1}{m}$ to increase the counter by one. The number of such packets is denoted by e_0. Each packet of another child flow of f_1 has a probability of $\frac{1}{s_1}$ to increase the counter by one, where f_1 is the first-level ancestor of f_i. The number of such packets is denoted by e_1. Repeating this process, we know that each packet of another child flow of $f_j, 1 \leq j \leq i-1$, has a probability of $\frac{1}{s_j}$ to increase the counter by one, where f_j is the jth level

ancestor of f_i. The number of such packets is denoted by e_j. Clearly, we have

$$E_i = \sum_{j=0}^{i-1} e_j. \tag{16}$$

Assume there is a large number of flows at each level, the size of each flow is negligible when comparing with the total size of its parent flow. We can approximately treat the packets independently. Hence, e_0 approximately follows a binomial distribution: $e_0 \sim Bino\big(n - n_{f_1}, \frac{1}{m}\big) \approx Bino\big(n, \frac{1}{m}\big)$ since $n_{f_1} \ll n$. Similarly, we know that $e_j, 1 \le j \le i-1$ approximately follows a binomial distribution: $e_j \sim Bino\big(n_{f_j}, \frac{1}{s_j}\big)$. Hence, we have

$$\begin{aligned} Var(e_0) &= \frac{n}{m}(1 - \frac{1}{m}), \\ Var(e_j) &= \frac{n_{f_j}}{s_j}(1 - \frac{1}{s_j}),\ 1 \le j \le i-1. \end{aligned} \tag{17}$$

The packets represented by e_j, $1 \le j < i$, are different and independent. Therefore, we have

$$\begin{aligned} Var(E_i) &= Var(\sum_{j=0}^{i-1} e_j) = \sum_{j=0}^{i-1} Var(e_j) \\ &= \frac{n}{m}(1 - \frac{1}{m}) + \sum_{j=1}^{i-1} \frac{n_{f_j}}{s_j}(1 - \frac{1}{s_j}). \end{aligned} \tag{18}$$

Since F_i and E_i are independent. We have

$$\begin{aligned} Var(X_i) &= Var(F_i) + Var(E_i) \\ &= \frac{n}{m}(1 - \frac{1}{m}) + \sum_{j=1}^{i} \frac{n_{f_j}}{s_j}(1 - \frac{1}{s_j}). \end{aligned} \tag{19}$$

In (10), $C_{f_{i-1}}[j], 1 \le j \le s_{i-1}$, are independent samples of X_{i-1}. Recall that the s_j counters in C_{f_i} are selected from $C_{f_{i-1}}$. When $i > 1$, from (10), the variance of $n_{f_{i-1}}$ is

$$\begin{aligned} &Var(\hat{n}_{f_i}) = Var(\sum_{j=1}^{s_i} C_{f_i}[j] - \frac{s_i}{s_{i-1}} \sum_{j=1}^{s_{i-1}} C_{f_{i-1}}[j]) \\ =& s_i\big(1 - \frac{s_i}{s_{i-1}}\big)^2 Var(X_i) + (s_{i-1} - s_i)\big(\frac{s_i}{s_{i-1}}\big)^2 Var(X_{i-1}), \end{aligned} \tag{20}$$

where $Var(X_i)$ and $Var(X_{i-1})$ can be calculated through (19). Next, we consider a flow f_1 at the first level. Following a similar process as above, we have the following variance of $\hat{n}_{f_1}$:

$$Var(\hat{n}_{f_1}) = s_1 Var(X_1). \tag{21}$$

The standard (relative) error, which is the standard deviation divided by the actual flow size, is given below:

$$SE(\frac{\hat{n}_{f_i}}{n_{f_i}}) = \frac{\sqrt{Var(\hat{n}_{f_i})}}{n_{f_i}}. \tag{22}$$

IV. Performance Evaluation

We perform simulations to evaluate the performance of HVC under different memory availability and system parameter settings, and compare it with RCS [16] which has the smallest per-packet processing overhead among the prior work.

We consider an SDN datacenter, where 1000 racks are turned on, actively serving client workload, which is assigned by the controller about evenly to the servers in the racks for load balancing. Suppose each rack contains 16 servers to host VMs, which communicate with one another via virtual infrastructure deployments, emulating campus/institution networks. The controller requires workload information from each server and each VM, as well as the network traffic matrix, both at the flow level and at the rack level, in order to support complex performance optimizations. Our focus is on network traffic estimation.

In our simulations, each top-of-rack switch is required to monitor both first-level aggregate flows from the rack to all other racks and second-level TCP flows from sources in the rack to destinations outside of the rack. There are 999 aggregate flows, each going to a different rack. The number of TCP flows contained in each aggregate flow is randomly chosen from a range $[500, 1500]$ with an average of 1000. The TCP flow sizes follow a Zipf distribution based on the characteristics of the traffic trace on our campus network.

A. Estimation Accuracy

The first set of simulations is to evaluate the accuracy of the proposed HVC scheme under different memory availability. The simulation results are shown in Fig. 2-4, where the memory allocated by a switch for traffic measurement is 0.5MB in the leftmost plot, 1MB in the middle plot, and 2MB in the rightmost plot. The counter size is set to 10 bits. Because the average number of TCP flows per switch is 1 million, the number of physical counters per TCP flow is 0.4 in the leftmost plot, 0.8 in the middle plot, and 1.6 in the rightmost plot. If we take the aggregate flows into consideration, the average number of counters per flow will be slightly lower. We set $s_1 = 10,000$ and $s_2 = 200$; we will study their impact in later simulations.

Fig. 2 shows the estimation results for first-level aggregate flows f_1, where each point represents a flow, with the x-coordinate being the actual flow size n_{f_1} and the y-coordinate being the estimated size $\hat{n}_{f_1}$. The equality line $y = x$ is also shown for reference. The closer a point is to the equality line, the more accurate the estimation is. As is expected, the estimation accuracy improves with more memory, which is shown with points clustered closer towards the line. Similar observation is made in Fig. 3, where each point represents a second-level TCP flow f_2.

The four plots in Fig. 4 show (a) the relative bias in the estimations for the first-level flows, (b) the corresponding standard error, (c) the relative bias in the estimations for the second-level flows, and (d) the corresponding standard error,

where the horizontal axis is the actual flow size. We can see that as the flow size increases, the relative bias quickly drops towards zero and the standard error also decreases rapidly, even when the average number of physical counters is less than 1. It is practically valuable to have high accuracy for large flows because they are the important flows in traffic engineering. Although the standard (relative) error of small flows is higher in the plots, their absolute error will remain relatively small.

For comparison, in Fig. 5, we show the relative bias and the standard error of RCS [16], which also updates a single counter to record a packet in a flow. However, because it does not have hierarchical design, it has to update two counters per packet in the simulations because each packet belongs to two flows, one TCP and one aggregate. Despite doubling the processing overhead, in general, RCS's accuracy is slightly lower than HVC's in Fig. 4.

B. Impact of Value s_1 on HVC

The second set of simulations evaluate the impact of s_1 (i.e., number of counters in first-level virtual counter arrays) on estimation accuracy. The memory size is set to 1MB and the value of s_2 is kept at 200. We repeat the simulations with $s_1 = 2500$, 5000, 20000, respectively. The estimation results for the first-level aggregate flows are shown by the first three plots in Fig. 6, respectively; the fourth plot shows their standard errors, together with that of $s_1 = 10000$ used in the simulations reported earlier. The results for the second-level TCP flows are shown in Fig. 7. Interestingly, as s_1 increases, the accuracy decreases at the first level but increases at the second level. The reason is that when the size of a first-level virtual array is larger, it carries more noise due to counter sharing, which degrades the first-level estimation, but in the meantime it offers more counters for the second-level children flows and thus improves their estimation.

C. Impact of Value s_2 on HVC

The third set of simulations evaluate the impact of s_2 (i.e., number of counters in second-level virtual counter arrays) on estimation accuracy. The memory size is set to 1MB and the value of s_1 is kept at 1000. We repeat the simulations with $s_2 = 50$, 100, 400, respectively. The estimation results for the first-level aggregate flows are shown by the first three plots in Fig. 8, respectively; the fourth plot shows their standard errors, together with that of $s_2 = 200$ used in the simulations reported earlier. The results for the second-level TCP flows are shown in Fig. 9. The observation is opposite to the previous subsection: as s_2 increases, the accuracy improves at the first level but degrades at the second level. The reason is that when the size of a second-level virtual array is larger, it carries more noise for second-level estimation, but causes more sharing at the first level, making the noise there more uniformly distributed, which helps noise removal.

V. Conclusion

This paper proposes a hierarchical traffic measurement scheme to measure aggregate flows and individual flows together in software-defined datacenter networks. The key idea is the introduction of virtual counter arrays that replicate the containment relationship among the flows. In addition, it achieves memory efficiency by letting virtual arrays to share counters. We mathematically derive the formulas for estimating the size of any (individual or aggregate) flow from its virtual counter array, and formally analyze the estimation accuracy. Our simulation demonstrates that the new scheme provides good measurement accuracy in tight memory space at a small per-packet processing overhead.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under grants 61472256, the Innovation Program of Shanghai Municipal Education Commission (Program Number 12zz137), and the first-class discipline construction project of Shanghai under grant S1201YLXK. It is also supported in part by the National Science Foundation of US under grant STC-1562485.

References

[1] "The Software-Defined Data Center," *http://www.vmware.com/software-defined-datacenter/*.

[2] C. S. Li, B. L. Brech, S. Crowder, D. M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software Defined Environments: An Introduction," *IBM Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.

[3] Yaser Jararweh, Mahmoud Al-Ayyoub, Elhadj Benkhelifa, Mladen Vouk, Andy Rindos, et al., "Software Defined Cloud: Survey, System and Evaluation," *Future Generation Computer Systems*, vol. 58, pp. 56–74, 2016.

[4] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*. ACM, 2011, vol. 41, pp. 254–265.

[6] Reuven Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, et al., "On the effect of forwarding table size on sdn network utilization," in *Proc. IEEE INFOCOM*. IEEE, 2014, pp. 1734–1742.

[7] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella, "Towards an Elastic Distributed SDN Controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013.

[8] Shao-Heng Wang, P. P. W. Huang, C. H. P. Wen, and L. C. Wang, "EQVMP: Energy-efficient and QoS-aware Virtual Machine Placement for Software Defined Datacenter Networks," *Proc. of International Conference on Information Networking*, 2014.

[9] Open Networking Foundation, "OpenFlow Switch Specification version 1.3.4," 2014.

[10] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," *Proc. of ACM Conference on Internet Measurement*, 2009.

[11] Hongli Xu, He Huang, Shigang Chen, and Gongming Zhao, "Scalable Software-Defined Networking through Hybrid Switching," *Proc. of IEEE INFOCOM*, 2017.

[12] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the Count-Min sketch and its applications," *Proc. of LATIN*, 2004.

[13] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," *Proc. of ACM SIGMETRICS*, June 2008.

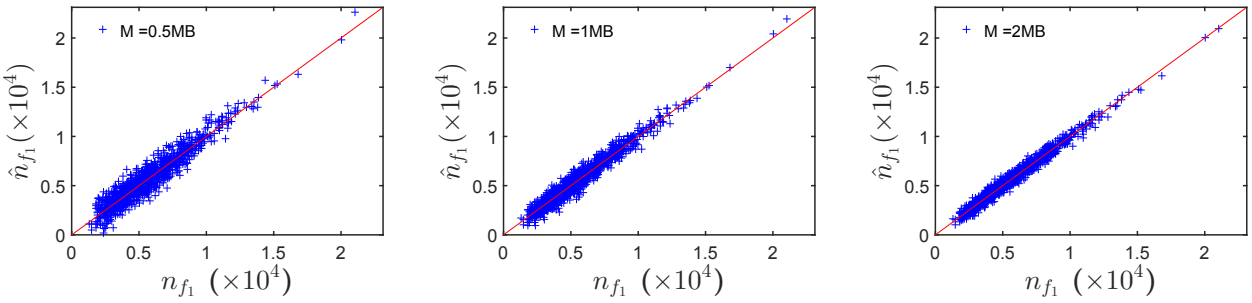


Fig. 2: Estimation results of first-level flows wwith 0.5MB, 1MB, and 2MB memory, respectively.

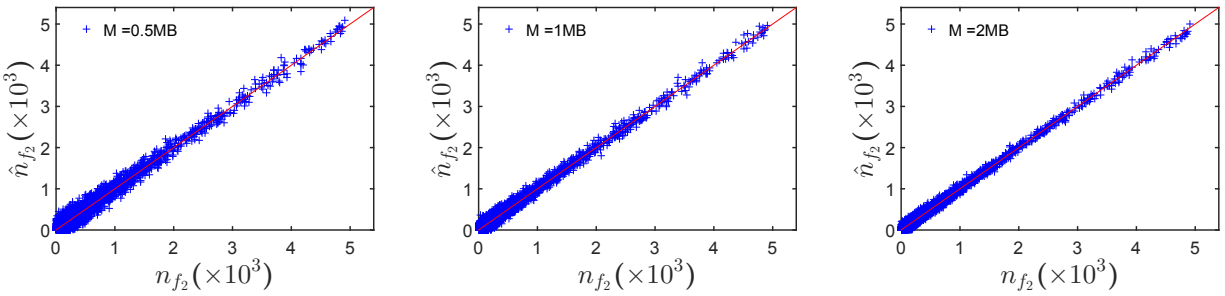
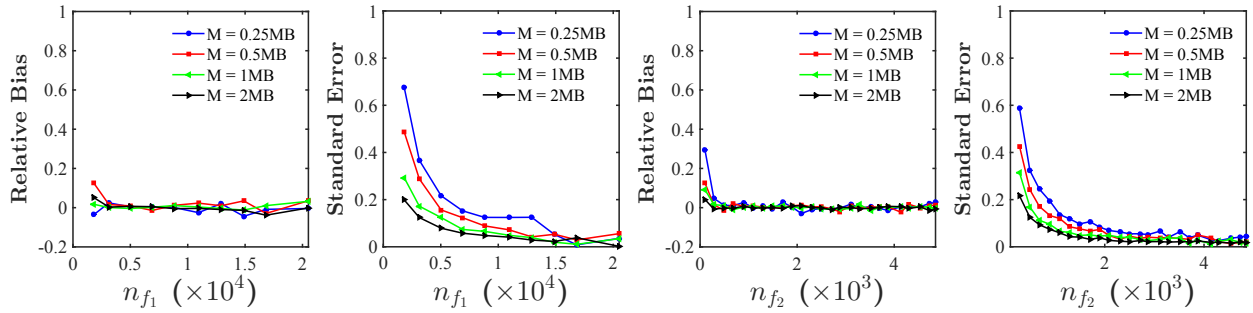
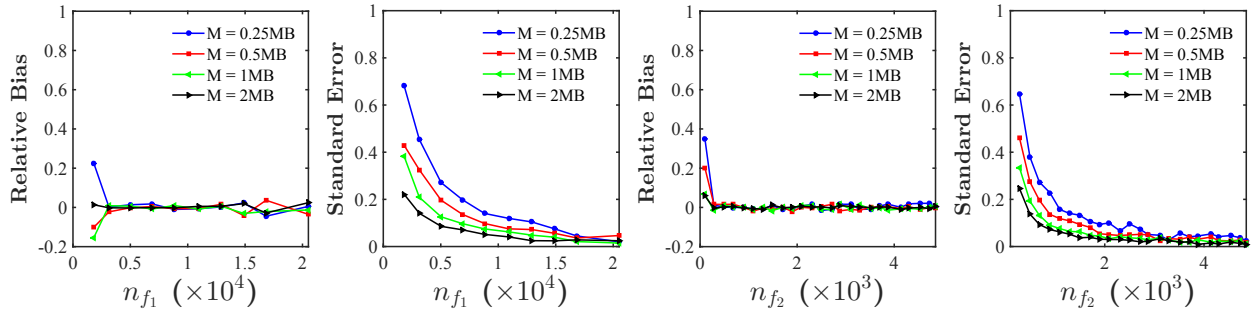


Fig. 3: Estimation results of second-level flows with 0.5MB, 1MB, and 2MB memory, respectively.



(a) Relative Bias of $\hat{n}_{f_1}$ (b) Standard Error of $\hat{n}_{f_1}$ (c) Relative Bias of $\hat{n}_{f_2}$ (d) Standard Error of $\hat{n}_{f_2}$

Fig. 4: Accuracy of first-level estimation $\hat{n}_{f_1}$ and second-level estimation $\hat{n}_{f_2}$ under HVC.



(a) Relative Bias of $\hat{n}_{f_1}$ (b) Standard Error of $\hat{n}_{f_1}$ (c) Relative Bias of $\hat{n}_{f_2}$ (d) Standard Error of $\hat{n}_{f_2}$

Fig. 5: Accuracy of first-level estimation $\hat{n}_{f_1}$ and second-level estimation $\hat{n}_{f_2}$ under RCS.

[14] Y. Lu and B. Prabhakar, "Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture," *Proc. of IEEE INFOCOM*, April 2009.

[15] M. Chen and S. Chen, "Counter Tree: A Scalable Counter Architecture

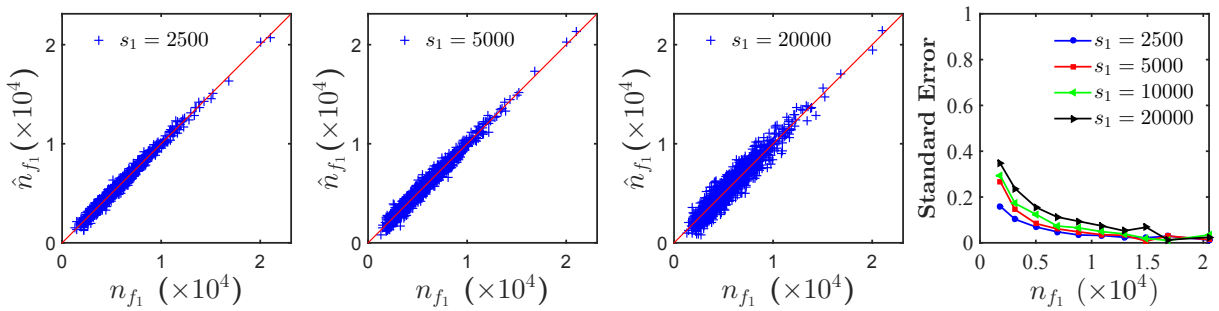


Fig. 6: Estimation results of first-level flows under different value of s_1 with 1MB memory.

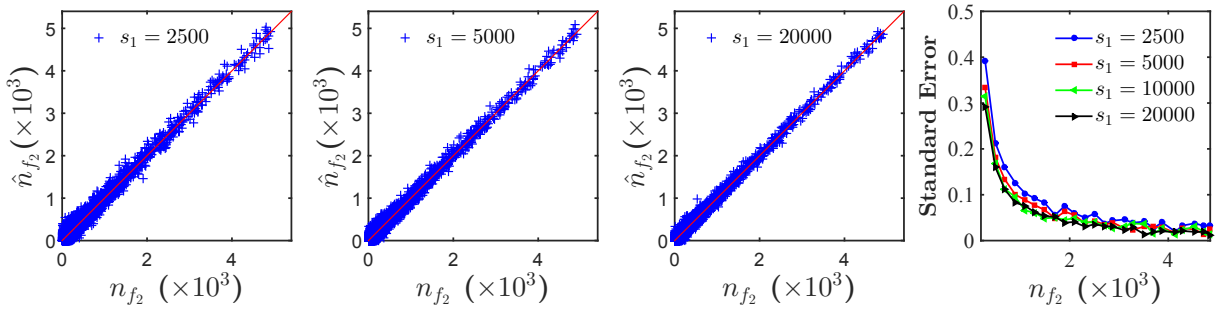


Fig. 7: Estimation results of second-level flows under different value of s_1 with 1MB memory.

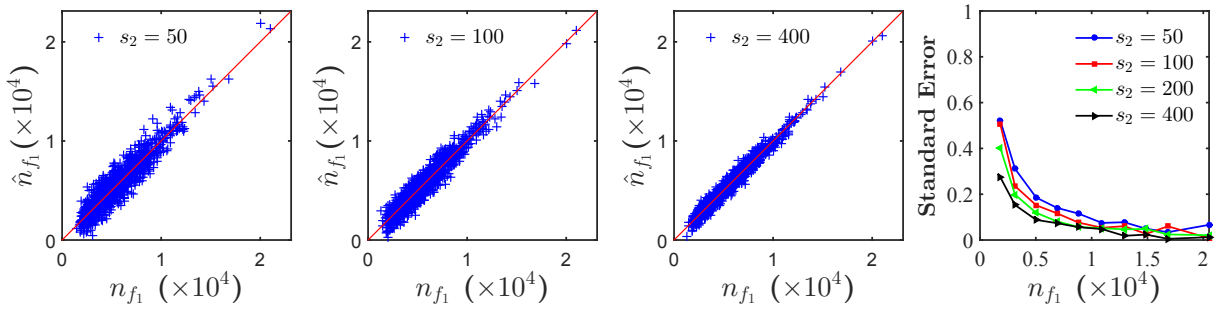


Fig. 8: Estimation results of first-level flows under different values of s_2 with 1MB memory.

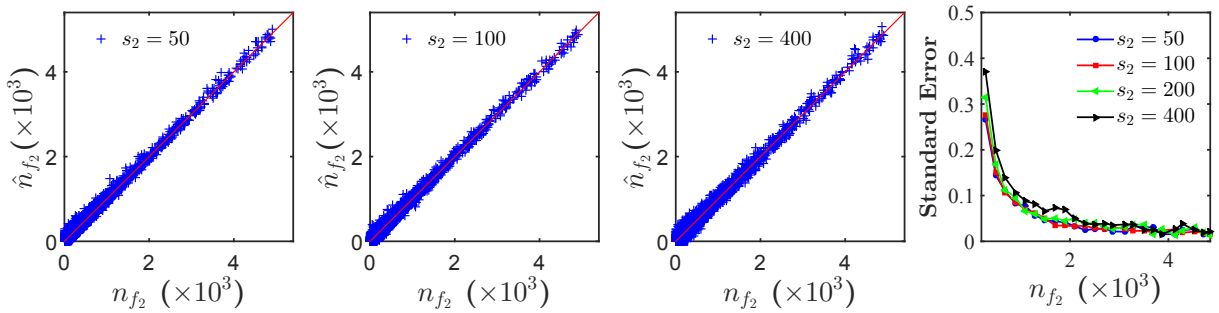


Fig. 9: Estimation results of second-level flows under different values of s_2 with 1MB memory.

for Per-Flow Traffic Measurement," *Proc. of IEEE ICNP*, November 2015.

[16] T. Li, S. Chen, and Y. Ling, "Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing," *Accepted by IEEE INFOCOM (Review Scores: 5/5/6)*, 2011.