Polygravity: Traffic Usage Accountability via Coarse-grained Measurements in Multi-tenant Data Centers

Hyunwook Baek
University of Utah

Cheng Jin
University of Minnesota

Guofei Jiang
NEC Labs America

Cristian Lumezanu
NEC Labs America

Jacobus Van der Merwe
University of Utah

Ning Xia
NEC Labs America

Qiang Xu
NEC Labs America

ABSTRACT

Network usage accountability is critical in helping operators and customers of multi-tenant data centers deal with concerns such as capacity planning, resource allocation, hotspot detection, link failure detection, and troubleshooting. However, the cost of measurements and instrumentation to achieve flow-level accountability is non-trivial. We propose Polygravity to determine tenant traffic usage via lightweight measurements in multi-tenant data centers. We adopt a tomogravity model widely used in ISP networks, and adapt it to a multi-tenant data center environment. By integrating datacenter-specific domain knowledge, sampling-based partial estimation and gravity-based internal sinks/sources estimation, Polygravity addresses two key challenges for adapting tomogravity to a data center environment: *sparse traffic matrices* and *internal traffic sinks/sources*. We conducted extensive evaluation of our approach using realistic data center workloads. Our results show that Polygravity can determine tenant IP flow usage with less than 1% average relative error for tenants with fine-grained domain knowledge. In addition, for tenants with coarse-grained domain knowledge and with partial host-based sampling, Polygravity reduces the relative error of sampling-based estimation by $\frac{1}{3}$.

CCS CONCEPTS

• **Networks → Network measurement**; **Network monitoring**; **Data center networks**; *Cloud computing*; *Network management*;

KEYWORDS

Traffic Matrix Estimation, Tomogravity, SNMP, Multi-tenancy

1 INTRODUCTION

Network traffic accountability allows network operators to break down network usage and map it to consumers such as servers, virtual machines, virtual networks or tenants. Fine-grained network accountability, such as determining the bandwidth of each *point-to-point* (i.e., host-to-host, or VM-to-VM) flow is critical to ensure reliable cloud performance and customer satisfaction. For example, when a network hotspot appears, operators wish to determine which consumer utilizes the network more than it should, or which consumers may suffer performance degradation due to the hotspot.

Network accountability is challenging in multi-tenant data centers, as today's network systems cannot provide a line-rate flow measurement for datacenter-scale traffic. Researchers have proposed two classes of solutions to realize datacenter-scale network monitoring. Fine-grained flow sampling solutions such as sFlow [27] or NetFlow [13] can monitor individual IP flows at network devices, thereby simplifying network accountability. However, proposed datacenter-scale flow sampling relies on intrusive instrumentation either on network devices [16, 17, 30, 31] or on end hosts [9, 14]. Furthermore, sampling flows pervasively in a whole data center with high network coverage is prohibitively expensive, particularly at aggregation and core switches. More recent approaches make use of software-defined networking to enable flow-level counters that can account for bandwidth usages at switches [22]. However, these approaches also suffer from scalability concerns (due to limitations in device memory and the number of flow rules that can be used for such measurements).

An alternative solution to reveal fine-grained flow usage relies on combining the network routing matrix with coarse-grained link-level measurements, *e.g.*, SNMP. Initially developed for ISP networks, *tomogravity* [36] was also applied to datacenters [20, 21, 23], albeit with less accurate results (*e.g.*, best case average relative error around 15%). The main challenge for accurate traffic estimation with tomogravity in a datacenter is that datacenter traffic, naively interpreted, might not exhibit the inherent structure needed for this approach. Specifically, tomogravity was designed with two assumptions, which hold for ISP networks, about the network traffic characteristics: 1) *all nodes proportionally contribute to overall traffic flows*; 2) *the network does not have internal sinks/sources*. In contrast to the first assumption, Kandula et al. [23] showed that the datacenter traffic matrix is sparse, and all nodes do not contribute to the traffic matrix. In addition, contrary to the second assumption, datacenter networks may contain a variety of internal sinks and sources. For instance, due to software-defined firewalls [3, 5], host machines or SDN devices may be heavy internal traffic sinks. Likewise, due to broadcast-based image distribution [19, 24, 26] and

SoCC '17, September 24–27, 2017, Santa Clara, CA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5028-0/17/09…$15.00
https://doi.org/10.1145/3127479.3129258

port mirroring for *Intrusion Detection* or *Real User Monitoring*, the internal switches may act as heavy traffic sources.

Our key insights are that, (i) data center administrators have access to readily available information about the contributing nodes, e.g., tenant-level virtual topology configuration or access control setups, which can be used to deal with fact that data center traffic flow is contributed by a limited number of nodes, and that (ii) noise due to internal sinks/sources can be effectively canceled out by integrating information about their behavior into the tomogravity model.

In this paper, we show that utilizing such cloud configuration *domain knowledge* is key to precise traffic estimation. We propose a novel method termed Polygravity, derived from the original tomogravity algorithm, to account for fine-grained flow usage in multi-tenant datacenters with heterogeneous domain knowledge. Polygravity performs significantly better than previous methods for data center network accountability: for tenants with fine-grained domain knowledge, Polygravity reduces the average relative error of estimating flow usage to less than 1%; for tenants with coarse-grained domain knowledge, with assistance of host-based partial sampling, Polygravity consistently reduces the relative error by $\frac{1}{3}$ compared to the relative error of the sampling-only solution.

To summarize, our contributions include:

- We identified domain knowledge integration as a key enabler to apply the gravity model to the sparse traffic matrix estimation problem, and systematically adapted tomogravity to the variation of datacenter infrastructure, making Polygravity generally applicable to other data centers (§3.2.2).
- We first identified the *'no internal sinks/sources'* assumption of tomogravity model does not hold in datacenter networks, and devised a *Inner Gravity Estimation* model for augmenting the domain of gravity models to internal sink/source nodes (§3.2.1).
- We designed Polygravity model to selectively integrate additional estimation models such as sampling for better estimation, especially in case the given domain knowledge for a tenant is coarse-grained (§3.2.3).
- To thoroughly evaluate the performance of Polygravity, we generate realistic datacenter traffic by tenant level traffic emulation based on previous datacenter traffic measurement studies [8, 12, 23] and simulating datacenters with different environmental setups (§4).

2 BACKGROUND AND RELATED WORK

Formulating a traffic matrix to represent resource allocation is a general approach in network traffic engineering. The data sources used to construct such a traffic matrix often include link counts from SNMP data, routing information, topology information, and so forth. In this section, we first introduce how we construct a traffic matrix to formulate the traffic usage accountability problem, and then we explore the related work with respect to solving traffic matrices, and the applicability in data centers.

Network Traffic Matrix Estimation: We formulate the problem of determining flow usage as follows:

$$\underset{\mathbf{t}}{\operatorname{argmin}}\ ||\mathbf{x} - \mathbf{A} \cdot \mathbf{t}|| \tag{1}$$

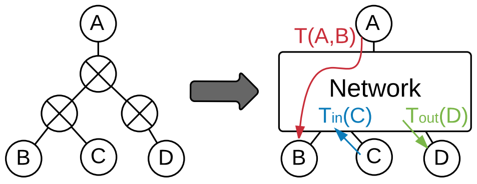


Figure 1: A view of gravity model to an example network

where $\mathbf{t}$ is the $n \times 1$ flow traffic vector[1] representing the traffic usage of each flow (n denotes the number of flows), $\mathbf{A}$ is then $m \times n$ routing path matrix revealing whether a flow traverses through each physical interface (m denotes the total number of physical links), and $\mathbf{x}$ is the $m \times 1$ link count vector.

Given that link counts $\mathbf{x}$ and $\mathbf{A}$ are commonly available, our goal is to determine the optimal solution $\mathbf{t} = \mathbf{t}_o$, which is the true traffic vector and naturally minimizes the term $||\mathbf{x} - \mathbf{A} \cdot \mathbf{t}||$. A straightforward approach is using quadratic programming; e.g., applying *Least Square Method* for Equation 1. However, in real-world data center networks, the number of flows is significantly larger than the number of links (i.e., $\mathbf{A}$ is a fat matrix as $m \ll n$). There can be multiple solutions that satisfy Equation 1. Thus, a particular solution $\mathbf{t}_p$ yielded by the least square is not necessarily the optimal solution $\mathbf{t}_o$.

Network tomography refers to the methodologies that infer this optimal traffic matrix $\mathbf{t}_o$ by using a limited number of measurements such as link counts $\mathbf{x}$. Vardi [33] adopted a Poissonian model and employed an iterative approach that uses the EM algorithm [15] to find approximate solutions, which was the first to put the idea of network tomography into practice. Yu et al. [11] investigated the time-varying nature of the sender-receiver traffic by fitting the basic independent and identically distributed (i.i.d.) model locally using a moving data window.

Tomogravity Model Given that there could be multiple solutions fitting Equation 1, it is intuitive to augment the traffic matrix with more external domain knowledge to constraint the search process for $\mathbf{t}_p$ closer towards the optimal solution $\mathbf{t}_o$.

In large-scale IP networks, Zhang et al [36] solved the network traffic matrix estimation problem – *determining the traffic matrix representing the volume that flows from every ingress point into the network and to every egress point out of the network* – by proposing *tomogravity* modeling, which consists of *gravity modeling* and *quadratic programming* for refinement.

A typical gravity model views a target network as a black-box surrounded by sites as shown in Figure 1, assuming that the network traffic from a site to another site is proportional to both the total traffic coming out from the source site and the total traffic coming into the destination site:

$$T(s,t) \propto T_{in}(s) \cdot T_{out}(t) \tag{2}$$

where $T(s,t)$ denotes the traffic from site s to site t, $T_{in}(s)$ denotes the total traffic going *into the network* from site s, and $T_{out}(t)$ denotes total traffic *going out from the network* to site t.

Tomogravity uses gravity modeling as a way to obtain the initial traffic matrix model for an Autonomous System (AS). Zhang et al

[1] The $n \times 1$ traffic vector can be converted to $\sqrt{n} \times \sqrt{n}$ traffic matrix where $\sqrt{n}$ is the number of existing terminals. Thus, in this paper, we use *traffic matrix* and *traffic vector* interchangeably.

suggested two variations of the gravity model: - the *Simple Gravity Model* and the *Generalized Gravity Model*. The Simple Gravity Model estimates the traffic between two sites simply based on the network's total incoming or outgoing traffic. Though the Simple Gravity Model is helpful to figure out the overall traffic exchanges, this model is based on an assumption that all the sites exchange traffic evenly, which can over-simplify traffic behaviors of real-world networks. The Generalized Gravity Model is proposed to overcome the even-traffic restriction. They classified the sites - "Access Link" and "Peer Link" - and customized the gravity model for each combination of source and destination type according to the routing policy.

Though the gravity model is excellent for capturing overall traffic patterns, the results may not be consistent with the interior network traffic, since it does not take interior network traffic into account, i.e., the link counts in the network blackbox in Figure 1. Accordingly, the tomogravity model performs a second-step refinement on the initial results $\mathbf{t}_g$ obtained from gravity modeling by using the following quadratic programming:

$$\underset{\mathbf{t}_w}{\text{argmin}} \ ||\mathbf{x}_w - \mathbf{A} \cdot \mathbf{t}_w|| \tag{3}$$

where $\mathbf{x}_w = \mathbf{x} - \mathbf{A} \cdot \mathbf{t}_g$, i.e., the link error of the initial model $\mathbf{t}_g$.

Since a particular solution $\mathbf{t}_{w,p}$ yielded by Least Square method against Equation 3 has the smallest norm, the flow vector $\mathbf{t} = \mathbf{t}_g + \mathbf{t}_{w,p}$ becomes: first, the solution closest to $\mathbf{t}_g$ among the solutions minimize the objective function in Equation 1; second, the solution most consistent with the internal link counts on condition of Equation 1. After the second-step refinement, the final step of tomo-gravity modeling is replacing all negative values of $\mathbf{t}_g + \mathbf{t}_{w,p}$ to 0 and applying *Iterative Proportional Fitting* (IPF) [11] to minimizes the link error.

Tomogravity Model in Data Center Networks The tomogravity model works well for large-scale IP networks, but may not be applicable to modern data center networks. Kandula et al. investigated traffic characteristics in data centers, and observed the datacenter traffic matrix is sparse in comparison with ISP networks. For this reason, tomogravity model leads to a 60% median estimation error in their data center network. To improve the accuracy, Hu et al. [21] proposed to utilize additional information such as the ownership of virtual machine (VM) and shared jobs, achieving around 20% average relative error in the best case. In [20], the authors suggested to utilize SDN switches to additionally measure aggregated flows. However, for a real world deployment, it is prohibitive to exhaust available SDN rules only for improving traffic matrix estimation, considering the final improvement it achieved (around 15% best case average relative error). In contrast to these earlier efforts, Polygravity adopts domain knowledge of target datacenter in a holistic way, allowing highly fine-grained and flexible customization of the estimation model, which leads to be suitable for heterogeneous datacenter networks. In addition, our approach uniquely solves the internal sink/source problem which enables Polygravity to be highly tolerance under heavy internal sink/source noise.

3 METHODOLOGY

As mentioned in §2, an intuitive attempt to determine traffic usage accountability in modern data center networks is to form a traffic matrix to represent flows and solve the matrix using tomogravity model. However, traffic properties in multi-tenant data centers differ significantly from ISP networks in the following ways:

- There are tenant-isolation and function-virtualization factors in multi-tenant data center networks, e.g., VLAN, VXLAN, GRE tunneling, firewalls, etc. Such factors partition the network infrastructure and routing paths are accordingly segmented, which translates to highly a partitioned traffic matrix.
- Imbalanced incoming/outgoing traffic volume happens mainly due to various network level applications such as software-defined firewalls, broadcast-based VM image delivery, hot-standby middleboxes, port mirroring for IDS etc., leading to heavy "source" and "sink" spots [3, 5, 19, 24, 26].

Broadly speaking, we address these challenges by enhancing tomogravity model through three phases: augmenting the domain of gravity model with internal sinks/sources (in §3.2.1), redesigning gravity model to flexibly reflect network dynamics (in §3.2.2), and allowing integration of supplementary estimation models into gravity model (in §3.2.3).

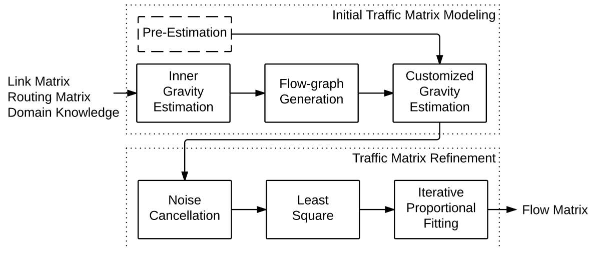
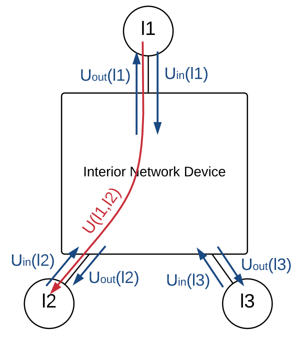


Figure 2: Polygravity Operation

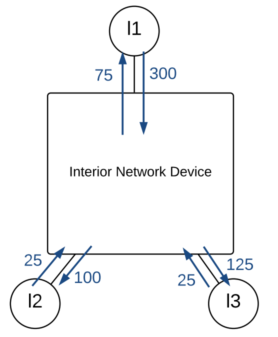
3.1 Overview

We name the full stack of our methodology *Polygravity* (from its *poly*-morphic nature and utilizing gravity model *multiple* times). Fig. 2 summarizes the complete steps to conduct Polygravity:

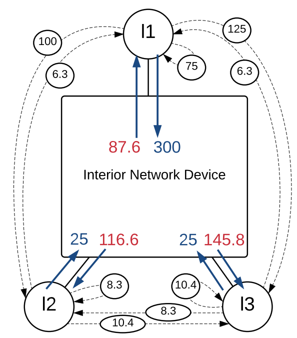
(1) For a given data set within a time window, compute the ingress and egress traffic (T_{in}, T_{out}) of interior interfaces by conducting *Inner Gravity Estimation*. Then, augment your gravity model's domain to include every interior interface that acts as non-negligible sink or source. (§3.2.1).
(2) Customize gravity model to fit your data center network by classifying the sites and constructing a flow graph. Since this step is mainly about processing the metadata of the target datacenter, automation of this step is feasible through network management tools. (§3.2.2 Steps 1 and 2).
(3) Apply the customized gravity model to augmented gravity domain of current data set and obtain the initial gravity model $\mathbf{t}_g$ (§3.2.2 Step 3).
(4) Conduct broadcast noise cancellation on the link matrix to minimize the impact of broadcast traffic (§3.3.2)
(5) Apply weighted Least Square Method, replace all negative values to zero and apply Iterative Proportional Fitting (§3.3.1).



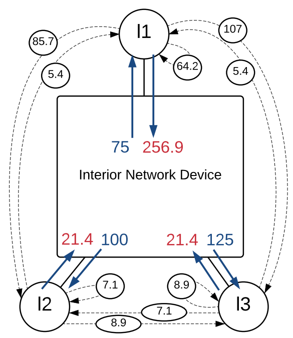
(a) Notations for Inner Gravity



(b) Example Link Traffic



(c) Inbound Inner Gravity Result



(d) Outbound Inner Gravity Result

Figure 3: Example Inner Gravity modeling for an interior network device acting as a sink

(b) shows the link counters for a device. Note that the device is acting as a sink because the total ingress traffic is larger than the total egress traffic; (c) shows the interface-to-interface traffic estimation based on the Inbound Inner Gravity and its resulting link traffic, which keeps the consistency of inbound link counters; likewise, (d) shows the results based on the Outbound Inner Gravity, that keeps the consistency of outbound link counters.

3.2 Constructing Initial Traffic Matrix

In this phase of traffic matrix construction, we propose two approaches: the *Augmenting Gravity Domain* and the *Network Infrastructure Adaptation* in accordance with the challenges mentioned above.

3.2.1 Augmented Gravity Domain. Previous network tomography methods [11, 20, 21, 32, 33, 36] assumed the interior network sinks and sources are negligible. However, multi-tenant data centers can seriously violate the assumption because these networks deploy various and complex techniques which make the interior network have heavy sinks (e.g., distributed firewall, virtual network host) or sources (e.g., multicast-based virtual machine image distribution, port mirroring for IDS). Consequently, the resulting traffic matrix elements are either overestimated or underestimated.

To expose the interior sources/sinks out of the black box, we perform a two-step approach: 1) estimating the amount of source/sink traffic of each interface of *interior network devices*[2] and 2) augmenting the classic sink/source domain to integrate the interfaces.

Though it is simple to determine if a network device is a source or a sink and how much it is (by comparing the total amount of inbound and outbound traffic), an interface-wise assessment is never easy. Even if we know the amount of inbound traffic consumed by a (sink) device, we cannot simply decide how much was consumed by each individual interface. One may try computing the numbers by setting a system of linear equations and finding a solution that minimize the error, but such a system is under-determined in most of cases so that it cannot give a single solution – if a device has k interfaces, then there can be k^2 variables but only $2k$ equations.

As a way to model interior sink/source traffic, we applied gravity model on each individual device and extract the amount of sink/source traffic for each interface of the device, termed *Inner Gravity Estimation* (IGE). The underlying assumption of this approach is that *the more traffic leaves (comes) through an interface and the less traffic comes (leaves) through the other interfaces, the more likely the interface is a source (sink).*

Specifically, consider the following two simple gravity models:

$$U_I(l_i, l_j) = U_{in}(l_i) \cdot \frac{U_{out}(l_j)}{\sum_{l_k \in f(\gamma)} U_{out}(l_k)} \tag{4}$$

$$U_O(l_i, l_j) = U_{out}(l_j) \cdot \frac{U_{in}(l_i)}{\sum_{l_k \in f(\gamma)} U_{in}(l_k)} \tag{5}$$

where l_i denotes an interface, $U_{in}(l_i)$ (or $U_{out}(l_i)$) denotes the amount of traffic came into (or left from) the device through the interface l_i, and $f(\gamma)$ means the set of all interfaces of an interior network device γ. If we compute a gravity model for a network device only using Equation (4), because it divides every *inbound* traffic (U_{in}) proportionally, the resulting gravity model (U_I) preserves the consistency of the inbound traffic as Figure 3(c). Likewise, the gravity model based on equation (5) (U_O) keeps the consistency of outbound traffic (U_{out}) as illustrated in Figure 3(d). Let's call each of these gravity models *inbound* and *outbound inner gravity model.*

Then, IGE for each interior network device γ consists of the following steps:

(1) $\forall\, l_i \in f(\gamma)$, initialize $T_{in}(l_i) = T_{out}(l_i) = 0$.
(2) Compute the total amount of incoming traffic U_{in}^{total} and outgoing traffic U_{out}^{total}:
$$U_{in}^{total} = \sum_{l_i \in f(\gamma)} U_{in}(l_i), \quad U_{out}^{total} = \sum_{l_i \in f(\gamma)} U_{out}(l_i).$$
If $U_{in}^{total} > U_{out}^{total}$, the device is a sink; otherwise, if $U_{in}^{total} < U_{out}^{total}$, source.
(3) If the device is a sink, compute outbound inner gravity models U_O for e. Likewise, if source, compute inbound inner gravity model U_I.
(4) If sink, for each $l_i \in f(\gamma)$, compute the inconsistency of inbound traffic:
$$U_{in}^{incon}(l_i) = U_{in}(l_i) - \sum_{l_k \in f(\gamma)} U_O(l_i, l_k)$$
$$\text{If } U_{in}^{incon}(l_i) < 0, \; U_{in}^{incon}(l_i) = 0$$

[2]Here, an interior network device means a node that works as an intermediary device between the source and the destination. Thus, not only network routers and switches but also physical machines hosting VMs can be treated as interior network devices.

Likewise, if source, compute:

$$U_{out}^{incon}(l_i) = U_{out}(l_i) - \sum_{l_k \in f(\gamma)} U_I(l_k, l_i)$$

$$\text{If } U_{out}^{incon}(l_i) < 0,\ U_{out}^{incon}(l_i) = 0$$

(5) If sink, for each $l_i \in f(\gamma)$, if $U_{in}^{incon}(l_i) > 0$, the interface l_i is likely a sink for the network. So, distribute the total amount of sink traffic proportionally as:

$$T_{out}(l_i) = (U_{in}^{total} - U_{out}^{total}) \cdot \frac{U_{in}^{incon}(l_i)}{\sum_{l_k \in f(\gamma)} U_{in}^{incon}(l_k)}$$

Likewise, if source, distribute the total amount of source traffic proportionally as:

$$T_{in}(l_i) = (U_{out}^{total} - U_{in}^{total}) \cdot \frac{U_{out}^{incon}(l_j)}{\sum_{l_k \in f(\gamma)} U_{out}^{incon}(l_k)}$$

A more intuitive explanation of IGE is that, when an interface of a switch acts as a source, since the inbound inner gravity model on the switch cannot find further inbound traffic from other interfaces to supply the outbound traffic of the interface, it yields a traffic estimation that under-utilizes the outbound traffic of the interface; likewise, the outbound inner gravity model returns a traffic estimation that under-utilizes the inbound traffic of the interfaces acting as sinks.

Through conducting IGE over every interior network component, we can obtain a list of interior network sources and sinks as well as their approximate amount of traffic. Now, augment the domain of gravity model L (i.e., L is the set of all external nodes) to include these additional sources and sinks obtained through IGE:

$$L_{aug} \leftarrow L \cup \{l_i \mid max(T_{in}(l_i), T_{out}(l_i)) > \tau,$$
$$l_i \in f(\gamma),\ \forall \gamma \in \Gamma\}$$

where Γ is the set of every interior network element, and τ is a threshold of negligible sink/source size. We term this newly generated set L_{aug} '*Augmented Gravity Domain*'. Note that, with Augmented Gravity Domain, the total traffic comes into and goes out from the *blackbox network* become equal.

3.2.2 Network Infrastructure Adaptation. Due to variation of network infrastructure and diversity of traffic patterns, without appropriate tailoring the gravity model, direct application of the tomogravity model could yield poor estimation quality in reality [23]. Customizing the gravity model to fit to a specific network is a process of reflecting network domain knowledge to the model, so a single model can hardly fit to heterogeneous networks.

Through our experience of applying gravity model to various data centers, we have come up with a general approach for customizing gravity model for different types of networks. In this section, we present the general approach with an example cloud shown in Fig. 4(a).

Step 1: *Component Classification.* We first classify the network sites in the gravity domain L_{aug} (you can use the non-augmented domain L if interior sinks and sources are negligible). Classification of sites can simplify the process of cutting out unlikely existing flows from the gravity model in the Step 2. Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of all classes you defined, so $c_1 \cup c_2 \cup \ldots c_k = L_{aug}$ and $c_i \cap c_j = \phi$ if $i \neq j$. When classifying, the rule of thumb is that the more fine-grained the classification is, the sparser traffic matrix you may get. However, if no behavioral difference of two classes of sites is known, it is pointless to differentiate them. The six nodes

Algorithm 1 Compute Customized Gravity

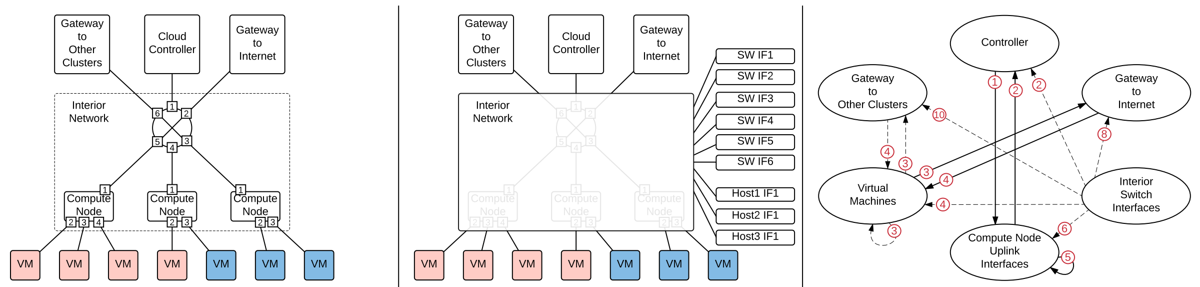
```
for each $l_i \in L_{aug}$ do                    ▷ Excluding pre-estimated amount
    $T_{in}(l_i) \leftarrow T_{in}(l_i) - \sum_{l_k \in L_{aug}} \hat{T}(l_i, l_k)$
    $T_{out}(l_i) \leftarrow T_{out}(l_i) - \sum_{l_k \in L_{aug}} \hat{T}(l_k, l_i)$
$E_{used} \leftarrow \phi$
$D \leftarrow \{(c, dir) \mid dir \in \{in, out\}, c \in C\}$          ▷ ordered set
Sort $D$ by an arbitrary order
for each $(l_i, l_j) \in L_{aug} \times L_{aug}$ do
    $T(l_i, l_j) \leftarrow 0$
for each $d = (c, dir) \in D$ do
    $E_{temp} \leftarrow \phi$
    if $dir = in$ then
        $E_{temp} \leftarrow \{e \mid e \in E, e \notin E_{used}, e = (c, c_x)\}$
        $L_k \leftarrow \{n \mid \forall(c, c_y) \in E_{used}, n \in c_y\}$
        $L_l \leftarrow \{n \mid \forall(c, c_z) \in E_{temp}, n \in c_z\}$
        for each $e \in E_{temp}$ do
            $c_x \leftarrow e[1]$                              ▷ $e = (c, c_x)$
            $\forall\, l_i \in c, \forall\, l_j \in c_x,$
            $T(l_i, l_j) \leftarrow \big(T_{in}(l_i) - \sum_{l_k \in L_k} T(l_i, l_k)\big) \cdot \frac{T_{out}(l_j) \cdot h(l_i, l_j)}{\sum_{l_l \in L_l} (T_{out}(l_l) \cdot h(l_i, l_l))}$
            if $T(l_i, l_j) < 0$ then $T(l_i, l_j) \leftarrow 0$
    else if $dir = out$ then
        $E_{temp} \leftarrow \{e \mid e \in E, e \notin E_{used}, e = (c_x, c)\}$
        $L_k \leftarrow \{n \mid \forall(c_y, c) \in E_{used}, n \in c_y\}$
        $L_l \leftarrow \{n \mid \forall(c_z, c) \in E_{temp}, n \in c_z\}$
        for each $e \in E_{temp}$ do
            $c_x \leftarrow e[0]$                              ▷ $e = (c_x, c)$
            $\forall\, l_i \in c_x, \forall\, l_j \in c,$
            $T(l_i, l_j) \leftarrow \big(T_{out}(l_j) - \sum_{l_k \in L_k} T(l_k, l_j)\big) \cdot \frac{T_{in}(l_i) \cdot h(l_i, l_j)}{\sum_{l_l \in L_l} (T_{in}(l_l) \cdot h(l_l, l_j))}$
            if $T(l_i, l_j) < 0$ then $T(l_i, l_j) \leftarrow 0$
    $E_{used} \leftarrow E_{used} \cup E_{temp}$
for each $(l_i, l_j) \in L_{aug} \times L_{aug}$ do
    $T(l_i, l_j) \leftarrow T(l_i, l_j) + \hat{T}(l_i, l_j)$
```

in Fig. 4(c) show an exemplary classification of the network sites in the example datacenter.

Step 2: *Flow Graph.* We build a directed graph $G = (C, E)$ which describes the traffic relationships between classes. Here, we add edges according to our domain knowledge about the target network. An edge $e = (c_i, c_j)$ represents existence of network flows from sites in class c_i to sites in class c_j. An edge e can be either unconditional or conditional: if unconditional, the edge indicates there exists a flow for every pair (l_x, l_y), where $l_x \in c_i, l_y \in c_j$; if conditional, it means there exists a flow for some pairs. Such condition can be defined using a condition function $h(l_x, l_y)$:

$$h(l_x, l_y) = \begin{cases} 1, & \text{if the flow } l_x \text{ to } l_y \text{ exists} \\ 0, & \text{otherwise (including } l_x = l_y) \end{cases}$$

Note that the condition function always returns 0 for the flows that do not belong to any edges in the flow graph. Also, for the unconditional edges, the condition function always returns 1 except $l_x = l_y$.



(a) Example Cloud Network **(b) Augmented Gravity Domain** **(c) Customized Gravity Model**

Figure 4: An example walk-through of gravity model customization for a cloud network

(a) an example cloud network consists of one controller, two network gateway nodes, and three compute nodes hosting total seven VMs. The left four VMs and the right three VMs belong different virtual networks; (b) the result of gravity domain augmentation. The administrator assumed the compute nodes do not directly talk with VMs through network so she excluded compute nodes' (virtual) interfaces toward VMs from the augmentation; (c) a flow graph of a customized gravity model. The solid lines are unconditional edges, and the dotted lines are conditional edges. The numbers on the edges show the index of the set that each edge belongs to in the ordered set D of Algorithm 1.

Fig. 4(c) shows a flow graph for the cloud network in Fig. 4(a). Here, the administrator may define several condition functions for the conditional edges. For example, since the traffic between two VMs can exist only if the two belong to the same virtual network, the administrator can define the condition function as:

$$\text{if } l_x, l_y \in c_{vm},$$

$$h(l_x, l_y) = \begin{cases} 1, \text{if } l_x \text{ and } l_y \text{ belong to the same} \\ \quad \text{virtual network and } l_x \neq l_y \\ 0, \text{otherwise} \end{cases}$$

where c_{vm} denotes a class for VMs.

Step 3: *Gravity Model Computation.* According to the flow graph G, we can systematically compute a custom gravity model as described in Algorithm 1. An intuitive explanation is that the algorithm iterates over each class and distributes either T_{in} or T_{out} to the connected edges. $\hat{T}$ appearing in line 1-3 and 30-31 is a pre-estimated traffic model, which is explained in §3.2.3.

Since either Line 18 or Line 27 will be executed at most $|L_{aug}|^2$ times and each line has $O(|L_{aug}|)$ of complexity in worse cast, the worst case complexity of the algorithm is $O(|L_{aug}|^3)$. However, the actual computation is substantially smaller because the flows that do not belong to any edge will be set to 0 without going through the computation.

After Step 3, we can obtain the gravity model for every network flow (i.e. $T(l_i, l_j)\ {}^\forall l_i \in L_{aug},\ {}^\forall l_j \in L_{aug}$). We can express the resulting gravity model in a column vector as $\mathbf{t}_g = (t_1, t_2, \ldots, t_{n'})^T$ where $n' = |L_{aug}|^2$.

3.2.3 Reflecting Pre-estimation. The customized gravity model is based on the assumption that the each network node accesses a limited number of peers, and such an access model is known to the datacenter operators. However, operators do not always have access to such information – e.g., cloud tenants may define a virtual topology in a very coarse-grained level or a virtual topology itself is dynamically changing at the application level such as Apache Hadoop and Storm.

As a remedy to this practical challenge, Polygravity allows operators to utilize supplementary estimation models to alternatively make up the limitation of gravity model. For example, a random n-out-of-N sampling technique [37] fits well for this purpose – the random packet sampling technique captures the heavy flows in a sparse traffic matrix well, but it unreliably estimates smaller size flows and generally performs bad for dense traffic matrices, which exactly counter-balances the cons and pros of gravity model. In addition, n-out-of-N sampling technique is widely implemented as an industry standard such as sFlow [27] and NetFlow [13] and readily deployable in datacenters even without special hardware supports.

As described in Algorithm 1 line 1-3 and 30-31, Polygravity takes pre-estimated traffic into account by first assigning the traffic to the pre-estimated model and then running the gravity model for the left-over traffic. Depending on the quality of the pre-estimation, however, the naive application of pre-estimated traffic may not help or even harm the quality of estimation. Thus, we suggest an adaptive application of pre-estimated traffic to Polygravity. Especially for sampling, we may utilize *coefficient of variation*(CV) of a sampled flow as an approximate indicator of its accuracy. Since our purpose of utilizing sampling is sketching the overall portions of heavy flows in the sparse traffic matrix (which gravity model cannot capture without domain knowledge), it is reasonable to preserve the proportion of each flow in whole. Therefore, rather than adjusting individual flow by its CV, we adjust the entire pre-estimated traffic according to the mean of CVs ($\bar{CV}$). To be specific, we regard $\bar{CV} < 0.5$ sufficiently precise to take the pre-estimation as a whole, and $\bar{CV} > 1.5$ overly imprecise to reflect, and proportionally decrease the portion of pre-estimation when $\bar{CV}$ is in between.

Compared to the case of utilizing sampling alone for estimating the traffic of an entire datacenter, sampling with Polygravity enables datacenter operators to selectively deploy sampling. This is a significant benefit that leads to saving resources from processing and storing samples throughout the datacenter.

3.3 Estimation Refinement

Though our customized gravity model captures the overall traffic pattern, it does not take the links in the *black-box* in Figure 1 into account. With only the gravity model, the resulting flow estimation might show high error rate for interior network links. In this section we describe how Polygravity refines the initial gravity model solution.

3.3.1 Quadratic Programming. The classic tomogravity method refines the the initial gravity model result through Least Square Method (LSM) and Iterative Proportional Fitting (IPF) as introduced in §2. The refinement step of Polygravity basically identical to that of tomogravity. I.e., Polygravity first applies a weighted Least Square Method:

$$\begin{aligned} \underset{\mathbf{t}}{\operatorname{argmin}} \; & ||(\mathbf{t} - \mathbf{t_g})./\mathbf{w}|| \\ \text{where} \quad & ||\mathbf{A} \cdot \mathbf{t} - \mathbf{x}|| = 0 \end{aligned} \tag{6}$$

where $\mathbf{t_g}$ is the gravity model solution we got from §3.2, $\mathbf{w}$ is a weighting factor and ./ is an element-wise division operator. Since $\mathbf{A}$ is a fat matrix, there exist multiple solutions that satisfies $||\mathbf{A} \cdot \mathbf{t} - \mathbf{x}|| = 0$, and the goal is finding a solution that is closest to the the gravity model solution. Since the least square solution $\mathbf{t}$ can contain negative values, we first change all the negative values to zeros and apply Iterative Proportional Fitting [11] until the link error reaches to a certain threshold. For the weighting factor $\mathbf{w}$, we followed the original tomogravity's approach that uses the square root of the gravity model ($\mathbf{w} = \sqrt{\mathbf{t_g}}$).

3.3.2 Broadcast Noise Cancellation. One issue of using Augmented Gravity Domain is that it increases the number of flows to consider in quadratic programming, which naturally increases computation time. It is reasonable to take such interior network flows into account if the flows are important and expected to exist (such as flows from compute nodes to Controller in Fig. 4(c) or filtered traffic by firewall). However, if some flows are not worthwhile to refine at the expense of additional computation time, we can remove them from the refinement steps.

Interior flows introduced by broadcasting traffic can be considered as such "flows less worthwhile to refine" – in traffic matrix use cases such as hotspot detection, link failure detection, resource allocation, and capacity planning, the main input is point-to-point traffic matrix but not precise background noise traffic. In addition, the interior switch interfaces (i.e., the interfaces augmented due to broadcasting) introduce a significant number of flows compared with the other sites – a single interior switch interface can introduce nearly $|L|$ additional flows because the destination of flows from a switch can hardly be limited within some sites (e.g., the interior switch interface class in Fig. 4(c) had edges to all the other classes in the flow graph, and the condition function for interior switch interfaces may merely check if the interface is toward the peer or not). Therefore, by removing the interior flows attributed to broadcast traffic, we may significantly reduce the computation cost. Let's term these flows as *noise flows.*

Since the noise flows are actually existing, simply excluding them from the flow matrix $\mathbf{t}$, will harm the accuracy of the estimation result. To minimize the impact of noise flows, we can use the initial gravity estimation result of noise flows. In other words, we can 'cancel out' the traffic possibly attributed by noise flows from the link matrix $\mathbf{x}$ by computing:

$$\mathbf{x}_{nc} = \mathbf{x} - \mathbf{A}_{(br)} \cdot \mathbf{t}_{g(br)} \tag{7}$$

where $\mathbf{A}_{(br)}$ is a routing matrix of the noise flows, $\mathbf{t}_{g(br)}$ is a flow vector for the noise flows, and $\mathbf{x}_{nc}$ is the 'noise-canceled' link matrix.

4 EVALUATION

To evaluate Polygravity, we derived a synthetic data set based on measurement results from earlier studies and then used this data to compare Polygravity against a number of earlier traffic estimation approaches: In §4.1, we first describe our methodology for realistic synthetic data generation. In §4.2, we use the generated data sets to validate Polygravity by comparing it with different earlier approaches. In §4.3, we show the impact of interior source/sink and validate IGE and the noise cancellation technique. Finally, in §4.4, we evaluate the scalability of Polygravity by measuring the changes of accuracy and running time of Polygravity as the target datacenter scales up.

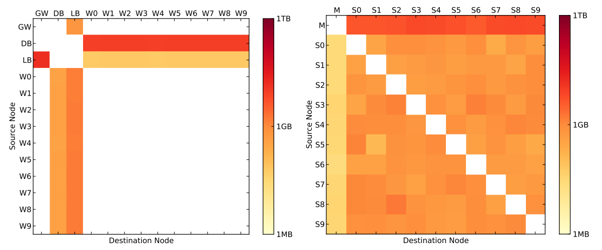
4.1 Synthetic Data Generation

Our original target through this work is estimating tenant-level network traffic of our public multi-tenant datacenters [4, 29]. For validation of our algorithm against these datacenters, we should have the complete sets of SNMP data $\mathbf{x}$, routing matrices $\mathbf{A}$ and flow level data $\mathbf{t}$ from the datacenters as well as the domain knowledge. Similar to [36], though we could collect the SNMP data and the routing matrices from the target datacenters, the complete flow level data collection was not available due to the privacy policy and the vendor implementation of flow collection. For the datacenter [29], we could obtain flow samples, but the maximum sampling rate was too low compared to the amount of datacenter traffic to apply the same technique as [36] and to generate realistic data sets.

As a solution to this problem, we first generated sets of tenant-level traffic in Emulab testbed [34] with a realistic setup based on previous measurement studies [8, 12, 23] and collected complete sets of flow data from them. We then synthesized the flow data into simulated multi-tenant datacenters, and computed a consistent SNMP data set from the routing matrix of the simulated datacenter and the synthesized flow data. In §4.1.1 and §4.1.2, we describe the detailed methodology to generate realistic tenant traffic for web service style tenants and map-reduce style tenants, the most typical types of tenants in clouds. In §4.1.3, we describe how we synthesize the data and simulate datacenter traffic.

4.1.1 Web Service Tenant. A three-tier architecture [6] is a typical setup for web services. For a realistic web service tenant traffic generation, we first deployed a three-tier web application consisting of a front-end load-balancer, 10 *Wordpress* middle-tier nodes and a back-end MySQL database. In this setup, the load-balancer receives requests from clients through a gateway and forwards each of them to a Wordpress node in a round-robin fashion. The Wordpress nodes interact with the back-end database to process the clients' requests.

Next, we generated workload based on a real-world multi-tenant datacenter traffic pattern introduced by Benson et al. [8] – the



(a) Web Service Tenant (b) Map-Reduce Tenant

Figure 5: Traffic Matrices of Tenants (a) Each `GW`, `DB`, `LB` and `W` stands for Gateway, Database, Load-balancer and Wordpress node. (b) Each `M` and `S` means Master and Slave.

authors observed the distributions of 1) *number of active* `flow`s[3], 2) `flow` *interarrival times* and 3) `flow` *lengths* in multi-tenant datacenters form lognormal distributions. Since these three variables are dependent, if we control any two to follow lognormal distributions, the other naturally forms a lognormal distribution. Specifically, we picked the traffic pattern observed at a specific switch in [8] (named PRV2$_3$) where nearly 85% of observed traffic was HTTP (i.e., web service traffic), and implemented a workload generator which mimics web service clients by randomly generating `flow`s with `flow` interarrival times and `flow` lengths following lognormal distributions close to the patterns of the PRV2$_3$.

We ran the workload generator for around 70 minutes and obtained 14 traffic matrices at 5 minute interval. Figure 5(a) visualizes the resulting traffic matrix over the entire period. As we can see from the figure, in this web application, Wordpress nodes do not communicate with each other. In addition, since the client requests are load-balanced in a round-robin fashion, the overall traffic is evenly distributed over the Wordpress nodes. On average, the total size of flows for 5-min interval is 8.2 GBytes (min: 6.1GB, max: 8.5GB).

4.1.2 *Map-Reduce Tenant.*

We deployed a Hadoop cluster consisting of 1 master node and 10 slave nodes for map-reduce style tenant traffic generation. For realistic workload generation, we batch processed 100 jobs with the distribution of per-job input size close to that of one of CloudEra customer (CC-e) and Facebook (FB-2010) introduced by Chen et al. [12], which follows lognormal distribution with logarithmic mean= 16.166 (≃10MB). Surprisingly, this generated a traffic pattern very similar to that of Microsoft Datacenter [23], which is mainly used for Map-Reduce jobs. Figure 6 shows the resulting distribution of duration of `flow`s in one of the slave nodes.

It took around 80 minutes to finish the all jobs, so we could collect total 16 traffic matrices at 5 minute interval. Figure 5(b) visualizes the traffic matrix of the map-reduce style tenant we generated. Different from the web service tenant, all map-reduce nodes exchange traffic with each other and the sizes of exchanged traffic are erratic. On average, the total size of traffic for 5-min interval is 3.1 GBytes but the variance was large (min: 0.1GB, max: 14.9GB).

[3]In [8] and [23], the term `flow` is used to mean *a continuous sequence of packets* from a source to a destination, which differs from the term flow we have used – point-to-point flow. For clear distinction, we notate `flow` only if it means '*a continuous sequence of packets having the same five tuples*'.

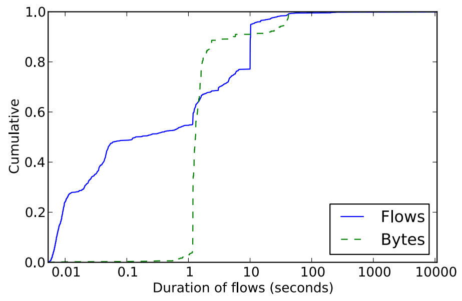


Figure 6: Cumulative distribution of `flows`

The `flows` are observed at the Slave 0 node while running the Map-Reduce workload emulating the CloudEra and Facebook jobs [12]. Other slave nodes showed the identical pattern. More than 70% of the `flows` last less than ten seconds. Also, 50% of the bytes are in `flows` lasting less than 1.3s, which could be shifted to 25s similar to the Microsoft Datacenter [23] when we changed the lognormal mean of the job size to 1GBytes.

4.1.3 *Synthesis into Datacenter.*

Now we can simulate any virtual datacenter with multiple tenants each of which may have each traffic matrix we measured. For the evaluation of general performance of Polygravity (§4.2) and its noise cancellation (§4.3), we created a virtual datacenter consisting of 60 virtual machines, 20 hosts, 4 edge switches, 2 aggregation switches and 1 core switch and 1 gateway to the Internet[4]. For the scalability evaluation (§4.4), we gradually increase the scale of the data centers as shown in the Table 3.

In general, cloud providers use specific virtual machine placement policies for various purposes. To answer the question 'Does virtual machine placement policy influence the performance of estimation algorithms?', we tested two contrasting placement policies: *affinity* policy – *'placing a VM topologically close to the other VMs belonging to the same tenant'* – and *anti-affinity* policy– *'placing a VM in a host that does not host any VMs belonging to the same tenant'*. Simply speaking, affinity policy is preferable for improving the network throughput among the VMs, and anti-affinity policy is better for failure resistance.

For web service style tenants, we also investigated the influence of tenant-side configuration of security groups (or virtual topology). As explained in §4.1.1, the web service uses a three-tier architecture where each node communicates only with specific peers. In a cloud environment, a cloud tenant can setup his virtual topology and/or security groups to reflect such communication pattern (e.g., for dividing broadcast domains or enhancing security). We especially tested three possible security group setup – flat, tiered and point-to-point as illustrated in Figure 7. The flat security group setup simply allows every possible communication among the VMs and the external gateway, which makes the system vulnerable to attack since every single node is reachable from outside as well as each other. The tiered setup distinguishes VMs by their type, groups them together and limiting communication by defining the accessible groups for each group. The point-to-point setup strictly

[4]For the aggregated switches for achieving large bandwidth (e.g., Fat-tree structure with ECMP), our model regards aggregated group of switches as one switch. Likewise, any aggregated group of links is regarded as one link.

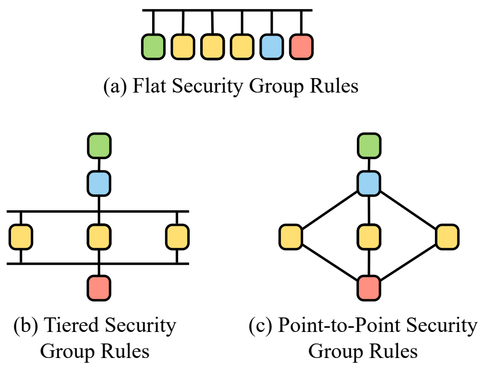

(a) Flat Security Group Rules
(b) Tiered Security Group Rules
(c) Point-to-Point Security Group Rules

Figure 7: Different levels of access control (aka security group in cloud) setups for web service. Each green, yellow, blue and red node refers Gateway, Load-balancer, Wordpress Node and Database.

limits unnecessary communication, which is most robust against propagation of attacks. Note that, since the cloud provider can access metadata describing tenants' virtual topology and security group setup, the cloud provider can readily reflect this information on the condition function $h(l_x, l_y)$ of Polygravity.

In the map-reduce environment, since all nodes may communicate with each other according to the job assignment by the master node, it is not easy for cloud providers to capture application-specific traffic patterns without special introspection or support from the tenants. As an alternative, we applied pre-estimation models introduced in §3.2.3 exclusively for map-reduce style tenants using host-based n-out-of-N packet sampling with various sampling rates.

4.2 Performance

To evaluate the performance of Polygravity, we conducted evaluations with various algorithms against various combinations of tenant styles and security group setups. The evaluated algorithms include tomogravity [36] and ATME-PB [21] as well as Polygravity. Tomogravity is identical to the combination of simple gravity and LSM/IPF. ATME-PB refines the gravity model by excluding traffic across tenants and applies Non-negative Least Square (NNLS) method after scaling down the initial gravity result as much as 20%. Thus, the initial model of ATME-PB is identical to customized gravity model of Polygravity with flat security group setup when we can ignore the internal sink/source traffic. For a fair comparison, we additionally tested the NNLS in the way ATME-PB suggests. For each combination of tenant styles, security group setups and placement policies, we have generated 10 different data sets.

As metrics to compare the performances of different algorithms, we reused the Root Mean Square Error (RMSE) and the Root Mean Square Relative Error (RMSRE) used by [36]. RMSE and RMSRE are defined as below:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \qquad \text{RMSRE}(T) = \sqrt{\frac{1}{n_T}\sum_{\substack{i=1\\x_i>T}}^{n}\left(\frac{\hat{x}_i - x_i}{x_i}\right)^2}$$

Table 1: Performance of various algorithms
RMSREs are computed on the largest 75% of the entire traffic. RMSEs are in Mbps. The fractions next to algorithm names refer sampling rates.

(a) A datacenter running 5 web service style tenants

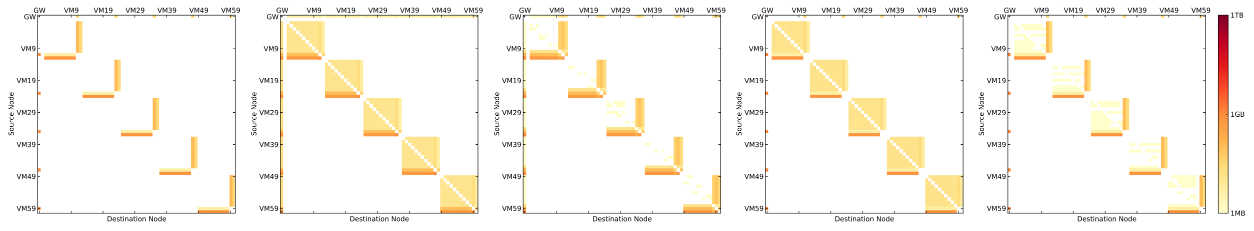
Algorithms	Traffic Matrix Errors			
	Affinity		Anti-Affinity	
	RMSE	RMSRE	RMSE	RMSRE
Tomogravity	682.93	70.52%	703.83	83.83%
ATME-PB	568.91	44.26%	532.98	45.83%
Polygravity-Flat	485.93	36.06%	480.77	34.77%
Polygravity-Tiered	0.29	0.01%	1.66	0.11%
Polygravity-Point-to-Point	0.24	0.00%	1.24	0.08%

(b) A datacenter running 5 map-reduce style tenants

Algorithms	Traffic Matrix Errors			
	Affinity		Anti-Affinity	
	RMSE	RMSRE	RMSE	RMSRE
Tomogravity	143.10	58.64%	181.23	73.28%
ATME-PB	139.33	55.94%	144.08	59.10%
Polygravity	119.09	49.21%	127.46	51.92%
Sampling-$\frac{1}{1000}$	89.81	48.39%	93.72	46.62%
Polygravity-$\frac{1}{1000}$	58.76	30.14%	64.68	30.42%
Sampling-$\frac{1}{300}$	43.35	21.14%	44.09	20.96%
Polygravity-$\frac{1}{300}$	36.49	15.93%	37.48	16.46%
Sampling-$\frac{1}{100}$	24.36	13.27%	25.44	12.66%
Polygravity-$\frac{1}{100}$	19.56	9.90%	21.35	10.13%

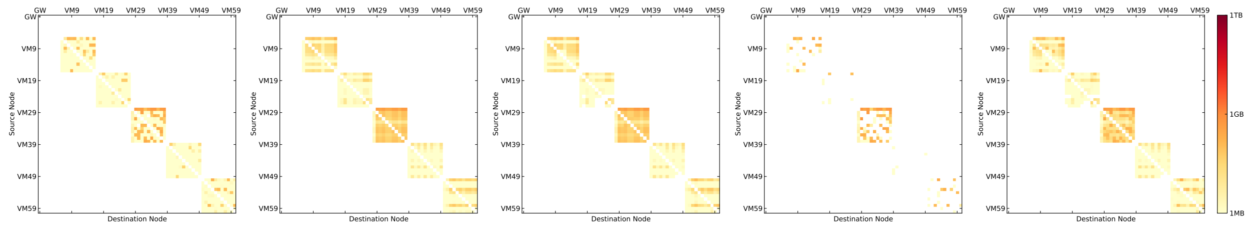
where $\hat{x}_i$ is the estimation of flow i, x_i is the ground truth and n_T is the number of flows greater than the threshold T. Especially, we adjusted T to cover the largest flows comprising 75% of the entire traffic as [36].

General Performance: in Table 1(a), we present the resulting traffic matrix errors against datacenters running 5 web service style tenants with different placement policies. As we can see from the result, for Polygravity starts to show significant performance improvement once we applied the domain knowledge about the 'three-tiered security group setup'. What then resulted in the *tipping point* from the 'flat' to the 'tiered'? For an intuitive explanation consider Fig. 8 which visualizes traffic matrices. If we compare the initial estimation results of both (Fig. 8(b) and (d)), we can see the both of the domain knowledge runs yield similar results except for the lines along the gateway nodes (the top line and the leftmost line) and the lines from load balancers to the Wordpresss nodes (the second bottom lines in each tenant square) in (b). In other words, the condition function of 'tiered' case prevents the gravity model from distributing traffic to unlikely existing flows along the lines but that of 'flat' doesn't. This may make big difference in terms of the position in the least square's constraint sub-space (i.e., their closest solutions on the constraint subspace are far from each other) and the difference can be signified as they go through the refinement. A



(a) Ground Truth (b) Gravity w/ Flat (c) Polygravity w/ Flat (d) Gravity w/ Tiered (e) Polygravity w/ Tiered

Figure 8: Sample traffic matrices in a datacenter with 5 web service tenants and the *affinity* placement policy. (b) and (d) are the initial traffic matrices for (c) and (e).



(a) Ground Truth (b) Gravity w/ Flat (c) Polygravity w/ Flat (d) Sampling 1/1000 (e) Polygravity 1/1000

Figure 9: Sample traffic matrices in a datacenter with 5 map-reduce tenants and the *affinity* placement policy. (b) is the initial traffic matrices for (c), and (d) is the pre-estimation model for (e).

similar pattern of performance improvement was observed in the study of Zhang et al. [36] – they could almost double the accuracy of tomogravity by applying *domain knowledge* about *access links* and *peer links* (and, presumably, especially by excluding traffic among peer links). We argue that the key for performance improvement of traffic estimation is finding such tipping point domain knowledge.

Finding such domain knowledge is not always obvious, as the case for map-reduce style tenants. In a map-reduce cluster, every node communicates with every other node and the traffic among them are not necessarily evenly distributed as shown in Fig. 9(a). Since the domain knowledge we used for the map-reduce style tenant is 'flat' saying 'every node can communicate each other and the traffic is distributed proportional to each of their input and output traffic', Polygravity hardly captures the erratic patterns of the distribution of the traffic as Fig. 9(c). However, when an supplementary model for traffic matrix estimation (such as sampling) is available, Polygravity could effectively integrate it into the model and improve the estimation result as shown in Fig. 9(d) and (e). Especially for n-out-of-N sampling, we could observe 25%-35% reduction of sampling-based estimation's relative error through Polygravity. Table 1(b) summarizes the results.

As we can see from the both Table 1(a) and (b), the placement policy did not show significance influence on the performance of Polygravity though the Affinity policy showed slightly better performances in general.

Heterogeneous Tenants: in a real-world cloud environment, the types of tenants are unlikely to be the same over the data center. Therefore, we can naturally ask if Polygravity can still perform well if the target datacenter hosts heterogeneous tenants with different granularity of domain knowledge. To answer to this question, we have conducted another experiment with a datacenter hosting 2 map-reduce style tenants and 3 web service style tenants with two different placement policies (for the web service style tenants, we used 'tiered' security group setup). The Table 2 shows the result.

Table 2: Performance of Polygravity with heterogeneous types of tenants.

Each MR and WS stands for a map-reduce style tenant and a web service style tenant. RMSEs are in Mbps. RMSREs are computed on the largest 75% of the traffic. For the per-tenant performance, RMSEs and RMSREs are computed based on the traffic of each tenant.

Tenant	Traffic Matrix Error			
	Affinity		Anti-Affinity	
	RMSE	RMSRE	RMSE	RMSRE
Total	113.59	21.19%	127.80	23.39%
MR1	537.36	46.13%	532.92	45.57%
MR2	374.82	58.51%	508.50	52.42%
WS1	3.57	0.16%	7.44	0.15%
WS2	3.81	0.15%	7.75	0.16%
WS3	3.92	0.08%	23.37	0.56%

First of all, we can notice that Polygravity still showed good estimation performance *for the tenants who offer more fine-grained domain knowledge*. This property of Polygravity has a significant implication, especially for practical utilization of Polygravity in multi-tenant environment – the quality of estimation can be *isolated* for each tenant. Also, if a cloud user wants better quality of estimation service, she can help the cloud provider by offering more fine-grained domain knowledge such as carefully set security groups or application-level metadata.

Another noticeable feature is that the case with affinity policy generally showed better performance for the tenants with fine-grained domain knowledge. We can understand the difference to be the result of the LSM – if the flows from different tenants are less *entangled* at the link level, the least square computation does not need to adjust the flows of a tenant in a large due to the flows of another tenant, so it can minimize the noise from the tenants with coarse-grained domain knowledge. This can be verified from the initial gravity results of the two placement policies, where the difference in the error was negligible.

4.3 IGE and Noise Cancellation

To evaluate the performance of the inner gravity estimation (IGE) and the broadcast noise cancellation (N/C) of Polygravity, we injected various tenant-level multicast traffic into a datacenter with 5 web service tenants and compared the performance of Polygravity with and without IGE and noise cancellation. Specifically, we varied three different factors of noise traffic – *the total size of interior source traffic* (`SourceSize`); *the number of destination VMs for each multicast* (`NumDsts`); and *the number of multicast generating tenants* (`NumTenants`). While varying each factor, we fixed other factors at their default values (`SourceSize=32GB`, `NumDsts=10`, `NumTenants=5`). Each experiment was repeated 10 times with different data sets, both in a datacenter with affinity policy and in another with anti-affinity policy.

`SourceSize`: Figure 10 (a1) and (b1) show the changes of relative errors as the size of each multicast flow exponentially increases, from 728KB (introducing 32MB of traffic from interior sources, and additional link level traffic as much as 0.03% of the non-noise traffic) to 1GB (introducing 32GB of traffic from interior sources, and additional link level traffic as much as 25% of the non-noise traffic). As we can see from the figures, IGE and N/C consistently suppress the error due to broadcast. Especially when the size of broadcast traffic was large (greater than 1% of the non-broadcast traffic) in the datacenter with affinity policy, it consistently decreased the relative errors of non-noise flows by $\frac{2}{3}$ (the gap between two lines in Fig. 10(a1)). Interestingly, in the datacenter with anti-affinity policy, the vanilla Polygravity (i.e., Polygravity without IGE and N/C) showed better noise-tolerance, so IGE and N/C could help to decrease the RMSRE only by $\frac{1}{2}$ to $\frac{1}{3}$.

`NumDsts`: Even if the total size of the interior sources is the same, the distribution of the interior sources can vary depending on the form of multicast. For example, though both a *2MB multicast flow destined to two nodes* and a *1MB multicast flow destined to three nodes* will introduce 2MB of traffic from interior sources, the first one actually generates 2MB of traffic from a single interior source and the second one introduces 1MB of traffic from each of two interior sources. We changed the distribution of interior sources by changing the number of destinations of each multicast flow while keeping the total size of interior source traffic to be 32GB, and observed the performance of IGE and N/C. Fig. 10 (a2) and (b2) show the results. In a nutshell, the more widespread the interior sources are, the more noise-tolerant Polygravity was, but the performance improvement by IGE and N/C was consistent regardless of the distribution: 30%-60% smaller RMSRE with affinity policy and 10%-20% smaller with anti-affinity policy. Similar to the case of varying `SourceSize`, the vanilla Polygravity showed better noise-tolerance with anti-affinity policy, which decreased the contribution of IGE and N/C.

Table 3: The counts of datacenter components as the datacenter scales up. Each of `EdgeSW` and `AggSW` refers to the number of edge switches and that of aggregation switches.

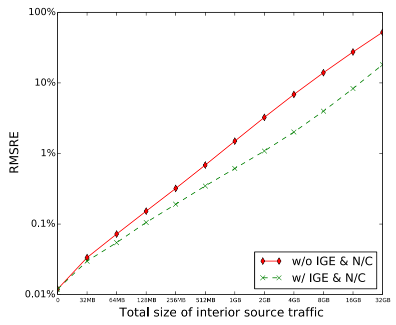
Scale	Flows	Links	VMs	Hosts	EdgeSW	AggSW
1	930	44	30	10	2	1
2	3660	87	60	20	4	2
3	8190	130	90	30	6	3
4	14520	173	120	40	8	4
5	22650	216	150	50	10	5
6	32580	259	180	60	12	6
7	44310	302	210	70	14	7
8	57840	345	240	80	16	8
9	73170	388	270	90	18	9
10	90300	431	300	100	20	10

`NumTenants`: Another way to change the distribution of the interior sources is changing the number of multicast flows. This time, we changed the number of multicast flows by limiting the number of tenants generating the multicast flows (the number of destinations and the total size of interior source traffic are fixed). For the affinity case, the result was similar to the case of varying `NumDsts`. However, in the anti-affinity case, the vanilla Polygravity showed stronger noise-tolerance, even out-performing the Polygravity with IGE & N/C for some cases (`numTenants=1` and 2).

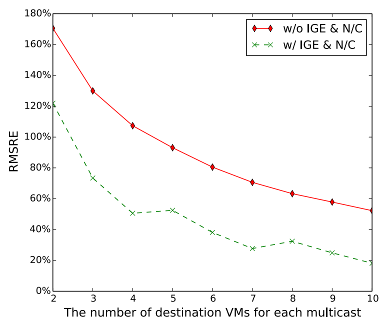
In summary, we could observe that IGE and N/C consistently improved the performance of Polygravity when there exist widely-distributed broadcast noises. This can be explained by the nature of the gravity models in Polygravity, which assumes the interior sinks/sources are proportionally distributed over the network interfaces and the sinks/sources are proportionally mapped with other network ends. In addition, the vanilla Polygravity showed stronger noise-tolerance under anti-affinity policy, though there was still some room for improvement by IGE and N/C. Presumably, this is because the noise traffic could be more evenly scattered across all flows in the affinity case, which decreases the *root mean square* value.

4.4 Scalability

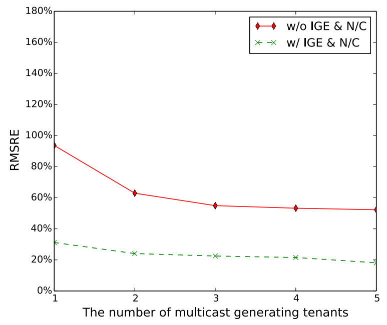
To evaluate the scalability of Polygravity, we have gradually scaled up the target datacenter as shown in Table 3 and measured the computation time and accuracy for both Polygravity and tomogravity. Figure 11 shows the changes of computation time as the target datacenter scales up. As we can imagine, the computation time of LSM was almost identical for both (Polygravity showed slightly larger due to the augmented gravity domain). For the initial gravity modeling, Polygravity's case takes obviously longer than tomogravity's simple gravity model. The most dramatic difference between the two algorithms was shown in the IPF step. This is because tomogravity's estimation result was generally too far from the ground truth, which leads excessive number of iteration at the IPF step. As we can see from the figure, the main scalability bottleneck for Polygravity is LSM (Singular Value Decomposition, to be specific), which is essentially the same for tomogravity in



(a1) varying SourceSize (in log scale)

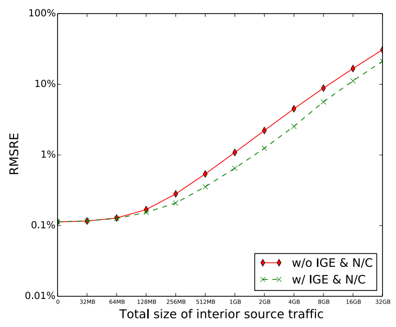


(a2) varying NumDsts

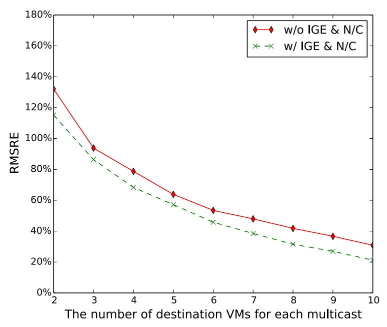


(a3) varying NumTenants

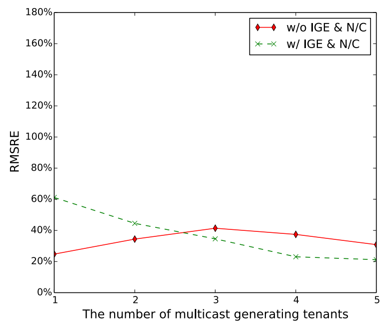
(a) Change of RMSRE of non-noise flows in a datacenter with *affinity* placement policy as noise changes



(b1) varying SourceSize (in log scale)



(b2) varying NumDsts



(b3) varying NumTenants

(b) Change of RMSRE of non-noise flows in a datacenter with *anti-affinity* placement policy as noise changes

Figure 10: RMSRE of Polygravity with and without IGE and broadcast noise cancellation (N/C). Each datapoint is based on 10 different data sets. RMSREs are computed on the largest 75% of the traffic (Note: both axes of (a1) and (b1) are in log scale).

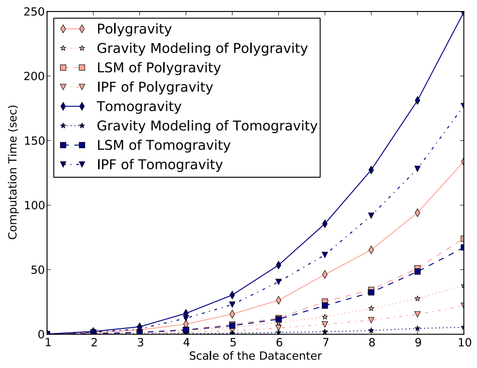


Figure 11: The computation time of tomogravity and Polygravity as the target datacenter scales up.

ISP network. We may reduce the actual computation time of SVD through two approaches – first, by making use of distributed SVD algorithms with more computation resources; second, by reducing the size of matrix ($m \times n$). As future work, we are exploring an effective way to reduce n by grouping uninteresting flows without sacrificing the accuracy for other flows.

Regarding the accuracy for the algorithms, regardless of scale, both of the algorithms showed highly consistent results – 77% to 82% of RMSRE for tomogravity, and 0.12% to 0.14% for Polygravity.

Table 4: Measurement Error Tolerance of Polygravity. RMSEs are in Mbps. RMSREs are computed on the largest 75% of the traffic.

Noise Level	traffic matrix errors		link errors	
	RMSE	RMSRE	RMSE	RMSRE
noise free	0.29	0.01%	1.56	0.02%
$\sigma = 0.01$	18.45	1.72%	186.70	1.62%
$\sigma = 0.02$	29.98	3.05%	363.02	2.93%
$\sigma = 0.04$	49.65	5.46%	715.03	4.60%

4.5 Measurement Error Tolerance

Measurement error in SNMP counters is very common, so one might have a practical concern regarding the impact of this measurement error on the performance of Polygravity. Zhang et al. [36] showed that tomogravity has robust error-tolerance against this type of measurement error by inducing errors to the tomogravity model. To see if Polygravity inherits this feature from tomogravity, we conducted the same experiment.

To be specific, we first generated an error term ϵ:

$$\epsilon = \mathbf{x} * N(0, \sigma), \tag{8}$$

where $*$ is an element-wise multiplication operator and $N(0, \sigma)$ is a vector with random entries following a normal distribution with mean 0 and standard deviation σ. We then induced the error ϵ to the link vector:

$$\mathbf{x}_{err} = \mathbf{x} + \epsilon \tag{9}$$

We ensured the non-negativity of $\mathbf{x}_{err}$ by changing negative values in $\mathbf{x}_{err}$ to 0. Table 4 shows the performance of Polygravity under different levels of noise. Here, we used a datacenter hosting five web service style tenants with affinity placement policy, and each experiment was repeated over 10 different data sets. The result was identical to that of the tomogravity – the measurement level errors proportionally degrade the accuracy of Polygravity.

5 DISCUSSION

Polygravity is designed for continuous traffic matrix monitoring in a multi-tenant datacenter. In this section, we discuss some practical concerns for deploying Polygravity.

Sampling and Other Estimation Techniques: Polygravity is not meant to replace all benefits of sampling. For instance, as we saw in Figure 9(d), sampling alone shows great performance for the elephant flow detection. However, to increase its coverage, we must increase the sampling rate by some orders of magnitude, as we saw in Table 1(b). An advantage of Polygravity over the sampling-only approach is that we can selectively apply sampling techniques. For example, when a cloud administrator wants to estimate the entire traffic matrix of the data center with smaller relative error, the administrator can apply a high sampling rate to just the nodes with coarse-grained domain knowledge.

This advantage of Polygravity is applicable to other estimation and/or measurement techniques. For instance, when a cloud administrator wants a precise traffic matrix but domain knowledge for certain flows is imprecise, he may set the SDN rules for the flows, collect precise flow counts for them, and simply reflect the measurement result as a pre-estimation model of Polygravity. This feature of Polygravity allows cloud administrators to flexibly deploy any hybrid solution.

Interior Sinks/Sources: Considering Section 4.3, one might wonder if there exists such a high volume of traffic from interior sinks and sources in a datacenter. As a practical example, in the CloudLab datacenters [29], the virtual machine and bare metal images are transferred to the distributed host machines through Frisbee [19], which internally uses switch level multicast to reduce the amount of traffic in the data center. Through our experience of traffic analysis on one of the CloudLab clusters, we could observe large traffic from interior sources due to Frisbee, especially when there were many VMs starting off. Tenant-level multicast is another source of such interior source traffic. Unless a VM itself changes multicast to unicast, a multicast flow unavoidably induces a branching of the flow to multiple destinations, either at switch devices or at host machines (e.g., OpenStack Neutron ML2 [1]).

Regarding the interior sinks, the most practical example is packet loss due to a cloud's security group (distributed firewalls) [3, 5]. When we estimate VM-to-VM traffic, host machines work as intermediary network devices. However, if a flow from one VM to another is dropped by a security group rule, it does not go through the virtual interface between the destination VM and its host machine. In this case, the flow cannot be mapped to the VM-to-VM flow, but to a flow between the source VM and the destination VM's host machine. This makes the host machine seem to be both an intermediary device and an end host, which is an interior sink in our model. Note that this type of noise flow does not need to be noise-canceled, since the number of such flows is very small in comparison to the number of possibly existing multicast flows.

Usability: As mentioned earlier, an accurate end-to-end traffic matrix can provide a great deal of help in many different network problems: traffic engineering [7], virtual machine scheduling [25], network design [28], capacity planning and failure detection [18]. However, traffic matrix estimation may not be suitable for some other types of problems. For instance, Polygravity can be overkill for cloud billing, because we do not need to know the size of every end-to-end flow but rather the aggregated amount of traffic (e.g., aggregated amount of traffic in-out/to-from a datacenter for each tenant in AWS [2]). Likewise, Polygravity may be insufficient to problems that requires finer-grained level of traffic information (e.g., protocol, size and interarrival time of packets for datacenter security analysis [10, 35]). In addition, if the target problem is sensitive to the accuracy of the estimation result, the administrator needs to carefully compare *the expected accuracy* of the Polygravity model in the target network and *the required accuracy for the problem.* Since precise comparison of these two accuracies is still an open problem, as future work, we plan to apply Polygravity to some of aforementioned network problems to answer to the question as well as to see the impact of our solution.

6 CONCLUSION

In this paper, we present Polygravity, a traffic matrix estimation algorithm for a multi-tenant datacenter via coarse-grained link level measurement data and using different types of domain knowledge.

Through our evaluation, we show Polygravity can estimate traffic matrix with less than 1% average relative error with fine-grained domain knowledge. In addition, when provided domain knowledge is coarse-grained, Polygravity's estimation has $\frac{1}{3}$ smaller relative error with assistance of sampling than sampling-only approach. Polygravity is especially suitable for a multi-tenant environment since it can show relatively clean performance isolation for each tenant with heterogeneous domain knowledge. Our scalability evaluation showed Polygravity consistently performs well regardless the scale of target data center.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their valuable comments, and our shepherd Theophilus Benson for his guidance and encouragement. This work was initiated when the student was an intern at NEC Labs America and was supported in part by the National Science Foundation under grant number 1314945.

REFERENCES

[1] 2014. Openstack Cloud Administrator Guide. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/5/pdf/Cloud_Administrator_Guide/Red_Hat_Enterprise_Linux_OpenStack_Platform-5-Cloud_Administrator_Guide-en-US.pdf. (2014).
[2] 2017. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. (2017).

[3] 2017. AWS Security Groups. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html. (2017).
[4] 2017. NEC Cloud Solutions. http://www.nec.com/en/global/solutions/cloud. (2017).
[5] 2017. OpenStack Security Groups. https://wiki.openstack.org/wiki/Neutron/SecurityGroups. (2017).
[6] 2017. Three-tier Architecture. https://en.wikipedia.org/wiki/Multitier_architecture. (2017).
[7] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks.. In *NSDI*, Vol. 10. 19–19.
[8] Theophilus Benson, Aditya Akella, and David a. Maltz. 2010. Network traffic characteristics of data centers in the wild. *Proceedings of the 10th ACM SIGCOMM* (2010), 267. https://doi.org/10.1145/1879141.1879175
[9] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM, 8.
[10] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 129–138.
[11] Jin Cao, Drew Davis, Scott Vander Wiel, and Bin Yu. 2000. Time-Varying Network Tomography: Router Link Data. *J. Amer. Statist. Assoc.* 95, 452 (2000), 1063–1075. https://doi.org/10.1080/01621459.2000.10474303
[12] Yanpei Chen, Sara Alspaugh, and Randy Katz. 2012. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1802–1813.
[13] Benoit Claise. 2004. Cisco systems NetFlow services export version 9. (2004).
[14] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. 2011. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM, 2011 Proceedings IEEE*. IEEE, 1629–1637.
[15] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.
[16] Luca Deri, Ellie Chou, Zach Cherian, Kedar Karmarkar, and Mike Patterson. 2011. Increasing data center network visibility with cisco netflow-lite. In *Network and Service Management (CNSM), 2011 7th International Conference on*. IEEE, 1–6.
[17] Cristian Estan, Ken Keys, David Moore, and George Varghese. 2004. Building a better NetFlow. In *ACM SIGCOMM Computer Communication Review*, Vol. 34. ACM, 245–256.
[18] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 350–361.
[19] Mike Hibler, Leigh Stoller, Jay Lepreau, Robert Ricci, and Chad Barb. 2003. Fast, Scalable Disk Imaging with Frisbee. In *USENIX Annual Technical Conference, General Track*. 283–296.
[20] Zhiming Hu and Jun Luo. 2015. Cracking network monitoring in DCNs with SDN. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 199–207.
[21] Zhiming Hu, Yan Qiao, and Jun Luo. 2015. ATME: Accurate Traffic Matrix Estimation in both Public and Private Datacenter Networks. *IEEE Transactions on Cloud Computing* PP, 99 (2015), 1–1. https://doi.org/10.1109/TCC.2015.2481383
[22] Cheng Jin, Cristian Lumezanu, Qiang Xu, Hesham Mekky, Zhi-Li Zhang, and Guofei Jiang. 2017. Magneto: Unified Fine-grained Path Control in Legacy and OpenFlow Hybrid Networks. In *SOSR*.
[23] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The Nature of Datacenter Traffic: Measurements & Analysis. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09* (2009), 202. https://doi.org/10.1145/1644893.1644918
[24] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. 2009. SnowFlock: rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 1–12.
[25] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. 2010. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In *2010 Proceedings IEEE INFOCOM*. IEEE, 1–9. https://doi.org/10.1109/INFCOM.2010.5461930
[26] Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. 2011. Going back and forth: Efficient multideployment and multisnapshotting on clouds. In *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 147–158.
[27] Peter Phaal, Sonia Panchen, and Neil McKee. 2001. *InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks.* Technical Report.
[28] Michael Poss and Christian Raack. 2011. Affine Recourse for the Robust Network Design Problem: Between Static and Dynamic Routing.. In *INOC*. Springer, 150–155.
[29] Robert Ricci and Eric Eide. 2014. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. 39, 6 (2014), 36–38.
[30] Vyas Sekar, Michael K Reiter, Walter Willinger, Hui Zhang, Ramana Rao Kompella, and David G Andersen. 2008. cSamp: A System for Network-Wide Flow Monitoring. In *NSDI*, Vol. 8. 233–246.
[31] Vyas Sekar, Michael K Reiter, and Hui Zhang. 2010. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 328–341.
[32] Claudia Tebaldi and Mike West. 1998. Bayesian inference on network traffic using link count data. *J. Amer. Statist. Assoc.* 93, 442 (1998), 557–573.
[33] Yehuda Vardi, Author Y Vardi, Source Journal, American Statistical, and No Mar. 1996. Network tomography: Estimating source-destination traffic intensities from link data. *J. Amer. Statist. Assoc.* 91, 433 (1996), 365–377.
[34] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.* (2002).
[35] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. 2013. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 199–208.
[36] Yin Zhang, Matthew Roughan, Nick Duffield, and Albert Greenberg. 2003. Fast accurate computation of large-scale IP traffic matrices from link loads. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (2003), 206. https://doi.org/10.1145/885651.781053
[37] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. 2009. RFC 5475: Sampling and Filtering Techniques for IP Packet Selection. (2009).