Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

# JOTA: Joint optimization for the task assignment of sketch-based measurement

Zhiyang Su[a,*], Ting Wang[a], Mounir Hamdi[b]

[a] The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[b] The Hamad Bin Khalifa University, Doha, Qatar

A R T I C L E   I N F O

*Article history:*
Received 1 October 2015
Revised 3 January 2017
Accepted 14 February 2017
Available online 15 February 2017

*Keywords:*
Software defined networking
Network management
Network measurement

A B S T R A C T

Sketch-based measurement provides traffic data summary in a memory-efficient way with provable accuracy bound. Recent advances in software defined networking facilitate the development and implementation of sketch-based measurement applications. However, sketch-based measurement usually requires TCAMs and SRAMs which are precious resource in switch to match packet fields. The key challenge for sketch-based measurement is how to accept more concurrent measurement tasks with minimum resource usage. Existing proposals attempt to achieve this goal by exploring different task assignment algorithms. We argue that by sacrificing a small amount of accuracy, the resource usage can be decreased dramatically. In this paper, we propose JOTA, a novel system to improve the performance of the task assignment for sketch-based measurement. By considering both TCAM and SRAM capacities, we formulate the problem as a mixed integer nonlinear programming problem. Due to its high computational complexity, we divide the initial problem into two stages and develop a two-stage heuristic to efficiently produce task assignment. In particular, we present an algorithm which guarantees $(1+\beta)$ approximation ratio to solve the second stage task assignment. Extensive experiments with three different measurement tasks and real packet traces demonstrate that JOTA significantly reduces the resource usage and accepts 33% more tasks compared with the first fit approach.

## 1. Introduction

Timely and accurate network traffic measurement is crucial in network management. The network management tasks such as traffic engineering, load balancing and anomaly detection rely on the measurement results to take further actions. For example, traffic engineering finds better routes based on the real-time traffic monitoring results.

Recently, software defined networking (SDN) has emerged as an active research field which attracted considerable attention from both academia and industry. SDN enables the programmability of the network by decoupling the control plane and the data plane. Motivated by this, software defined measurement leverages the flexibility of SDN to implement many customized measurement tasks. Specifically, software defined measurement not only can dynamically tune measurement granularity, but also has the capability to accommodate many advanced streaming algorithms to implement complex measurement tasks such as sketch-based measurement [1]. Typically, software defined measurement enhances the network management in a variety of ways:

- High flexibility. The granularity of the measurement task can be dynamically changed by modifying the corresponding monitoring switches and rules. This property is extremely useful because it tunes the flow aggregation granularity according to real-time traffic. It also provides opportunities to trade off the measurement cost and accuracy in both spatial and temporal dimensions.
- Generic measurement task support. The programmability of SDN enables a uniform measurement framework to adopt various measurement tasks, such as packet-based measurement, flow-based measurement and even sketch-based measurement. Moreover, the logically centralized controller supports multiple measurement tasks running across the network in parallel.
- Low deployment cost. The deployment of NetFlow and sFlow usually requires switch hardware support. Extra measurement components such as data collector and analysis module are needed. In SDN, in order to set up measurement tasks, only the controller and the corresponding switches are involved: the controller installs the monitoring rules to the switches and

* Corresponding author.
*E-mail addresses:* zsuab@cse.ust.hk (Z. Su), twangah@cse.ust.hk (T. Wang), hamdi@cse.ust.hk (M. Hamdi).

http://dx.doi.org/10.1016/j.comcom.2017.02.009
0140-3664/© 2017 Elsevier B.V. All rights reserved.

maintains their states. Clearly, the deployment is light-weight and the measurement cost is relatively low.

However, software defined measurement brings both opportunities and challenges. Despite the high flexibility of software defined measurement, TCAMs and SRAMs are required to match the hash value of the packet header and to store counters, respectively. Unfortunately, TCAM is power-hungry and expensive to implement in switches. There are only thousands of TCAM entries in most of the commodity switches today [2–4]. SRAMs are comparatively cheaper, but they still have a limited size [1]. The accuracy of the upper layer monitoring applications are determined by the resources allocated to it [4,5]. Moreover, TCAMs are mainly used as forwarding components, which further limits the number of entries for the monitoring applications.

In this paper, we concentrate on optimizations for sketch-based measurement. Sketch-based measurement usually uses streaming algorithms to provide compact traffic summary [1,4,5]. Compared with traditional measurement tools such as NetFlow [6] and sFlow [7], sketch-based measurement is able to perform various measurements at relatively low memory usage with a required accuracy bound.

We observe that sketch-based measurement has two interesting properties. First, sketch-based measurement usually has provable accuracy bounds when given the allocated resource. The accuracy of measurement tasks can be estimated in advance. Second, less is more: the relation between the resource and accuracy is conform to the law of diminishing marginal utility [4,5]. For instance, for a probabilistic counting with stochastic averaging (PCSA) sketch [8], if the accuracy increases from 22% to 61%, only 3 bitmaps are needed; in contrast, if the accuracy raises from 90% to 95%, about 180 bitmaps are needed. These two observations motivate us to save a large amount of resource and accept more monitoring applications by slightly trading off the accuracy of some resource-hungry applications. Compared with our previous work COSTA [9], JOTA takes both TCAM and SRAM resource into account, and we introduce weight for each task to provide more assignment flexibility. Besides, we develop another novel heuristic to solve the multi-resource task assignment problem, which is much harder than the single-resource assignment problem. Our approach enables a possibility to use provisioning and optimization techniques to place tasks intelligently.

The primary contributions of this paper are listed below:

- We present JOTA, a novel system for sketch-based measurement, which achieves different levels of balance between the measurement application accuracy and the resource usage.
- We formulate the sketch-based software defined measurement task assignment as a mixed integer nonlinear programming problem which is proved to be NP-hard. Due to its high computational complexity, we decompose the problem and propose a near-optimal two-stage heuristic to generate the final task assignment with negligible performance loss.
- Extensive experimental results demonstrate that JOTA is efficient in terms of running time, acceptance ratio and accuracy. It significantly reduces the resource usage and raises the task acceptance ratio.

The rest of this paper is structured as follows. Section 2 introduces the background of sketch-based measurement. Section 3 describes the architecture of JOTA and formulates the task assignment problem in software defined measurement. Specifically, we decouple the problem into two phases and propose a novel heuristic to obtain near-optimal assignment. Section 4 elaborates on the performance of JOTA by real packet traces simulation. Finally, Section 5 summarizes related work and Section 6 concludes the paper.

## 2. Sketch-based measurement tasks

In this section, we detail the implementation of three typical software defined measurement tasks and present the methodology to estimate the resource usage by their accuracy bounds.

### 2.1. Packet sampling

Packet Sampling (PS) is a common task in network monitoring. To implement a power-of-two ratio sampling, we can leverage the wildcard rules in TCAMs. For example, for the following wildcard rules $R_1$: 0***, $R_2$: 00** and $R_3$: 000*, the probability that the hash of a packet matches $R_1$, $R_2$ and $R_3$ is $\frac{1}{2}$, $\frac{1}{2^2}$ and $\frac{1}{2^3}$, respectively. We can extend the power-of-two ratio sampling to a more general case. Consider any decimal probability $p$ can be converted to a binary floating point number. However, sometimes the binary floating point number is of an infinite length (e.g. $(0.33)_{10} = (0.0101\ldots)_2$). Thus, we have to trade off the truncated length and the accuracy. Given the fraction part of the binary floating point number $B = b_1 b_2 \ldots b_n$, we need $l(B) = \sum_{i=1}^{n} b_i$ TCAM entries to implement the arbitrary probability packet sampling. Obviously, $l(B)$ is bounded by $n$, which is determined by the expected accuracy. Assume each TCAM entry has $e$ bits, to construct the wildcard rules group, for each bit $b_i = 1$ in $B$, we create a rule following this pattern: set the leftmost $i-1$ bits to 1, the $i$th bit to 0, and wildcarded all other bits. For example, for $e = 8$, we create two rules for $(0.0101)_2$: $R_1$: 10******, $R_2$: 1110****. The probability that a packet hash matches any one of the rules is: $P = \sum_{b_i=1} 2^{-i} \approx p$. Given the expected accuracy and the sampling rate $p$, we calculate the binary floating point number of $p$ and truncate it by the expected accuracy. We obtain the required number of TCAM entries by $l(B)$ afterwards. Define the accuracy function as $f(T, S)$, where $T$ and $S$ are the allocated TCAMs and SRAMs respectively. The accuracy function is given by:

$$f_{PS}(T,S) = \sum_{i=1}^{T} 2^{-i} = 1 - \frac{1}{2^T} \tag{1}$$

### 2.2. Unique IP counting

Unique IP Counting (UIC) is another fundamental monitoring task in network managements. Many anomaly detection tasks such as denial-of-service attacks monitoring are implemented by UIC. In software defined measurement, we can employ PCSA sketch [8] to count the number of unique header field of packets in a memory-efficient way.

PCSA sketch hashes an element (e.g. packet header) into different bins of a bitmap with a power-of-two ratio and estimates the number of distinct values by the bitmap. The bitmap is defined as $B = b_1 b_2 \ldots b_l$ of a length $l$, where $b_i = \{0, 1\}, i = 1, 2 \ldots, l$. All bits are initialized to zero in the beginning. Define hash function $h(x)$ as the position of the rightmost "1" of the uniformed distributed hash value of $x$. In order to insert an element $x$, we set bit $b_{h(x)}$ to "1". Obviously, we have $P(h(x) = i) = 2^{-i}$. Finally, The number of unique values can be derived from the length of the uninterrupted block of ones in $B$, whose length $l(B)$ is defined as:

$$l(B) = \min\{i | 0 < i \le n \wedge b_i = 0\} - 1 \tag{2}$$

For a single bitmap, the number of distinct values $C(B)$ can be estimated as $C(B) = 2^{l(B)}/\phi$, where $\phi \approx 0.775351$ is a constant. To further reduce the relative error of the estimation, we use $m$ bitmaps to improve its quality. To add an element to the sketch, insert only one of the $m$ sketches randomly. Suppose the $m$ sketches are $B_1, B_2, \ldots, B_m$, the number of unique values $C(B_1, B_2, \ldots, B_m)$ is

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **R1:** | * | * | * | * | * | * | * | * | 1 | 0 | 0 | 0 | **4** |
| **R2:** | * | * | * | * | * | * | * | * | * | 1 | 0 | 0 | **3** |
| **R3:** | * | * | * | * | * | * | * | * | * | * | 1 | 0 | **2** |
| **R4:** | * | * | * | * | * | * | * | * | * | * | * | 1 | **1** |

**Fig. 1.** Example of wildcard rules group for a PCSA sketch.

estimated as:

$$C(B_1, B_2, \ldots, B_m) = m \cdot \frac{2^{\frac{1}{m}\sum_{i=1}^{m} l(B_i)}}{\phi} \tag{3}$$

The relative error of the PCSA sketch in (3) is proved to be roughly $\phi/\sqrt{m}$ [8]. Assume the allocated SRAM size is $S$, the theoretical accuracy for PCSA sketch is $1 - \frac{\phi\sqrt{l}}{\sqrt{S}}$. The PCSA sketch can be implemented by a group of wildcard rules. Suppose the upper bound number of unique values is $U$, the required number of TCAM entries $n$ is determined by $n = \lceil \log_2 U \rceil$. Each rule $R_i$ is constructed as follows: set the rightmost $i-1$ bits to “0”, the $i$th bit to “1”, and wildcarded the other bits. The probability of a hash that matches $R_i$ is $2^{-i}$. Fig. 1 is an example with the upper bound number of unique values 16. The priority is marked right next to each rule. Clearly, the wildcard group can match the rightmost “1” of the given hash value as they have descending priorities. It is worth noting that the required number of TCAMs for PCSA sketch cannot be compressed. Otherwise, the estimated unique values will overflow. The accuracy function is given by:

$$f_{UIC}(T, S) = 1 - \frac{\phi\sqrt{l(B)}}{\sqrt{S}} \tag{4}$$

### 2.3. Flow size counting

Flow Size Counting (FSC) is a monitoring task which counts the size of a set of flows. FSC can be implemented by bloom filter and CM sketch [10], where the former filters the target flows and the latter counts the flow size. Bloom filter can be implemented by TCAMs in switch. A single TCAM entry usually has a length of 72–576 bits today [11]. Due to the false positive rate and the limited bits length in TCAMs, target elements can be stored in multiple TCAM entries. The false positive rate of a bloom filter is $(1 - e^{-\frac{kn}{m}})^k$, where $n$ is the number of elements, $m$ is the length of the bitmap and $k$ is the number of hash functions. Given the expected accuracy $\alpha_e$, the required number of TCAM entries can be obtained by: $\lceil - \frac{nk}{m \ln(1-(1-\alpha_e)^{\frac{1}{k}})} \rceil$.

After filtering the target flows, we utilize CM sketch to summarize the filtered packet stream. Each element of the stream is a sequence $(i_t, c_t)$, where $i_t$ is the flow id of packet $t$, $c_t$ is the length of packet $t$. Let $\tau$ denote the query time. Then, $s_i(\tau) = \sum_{t=0}^{\tau} c_t \delta_{i,i_t}$ is the size of flow $i$ up to the current time $\tau$, where $\delta_{i,i_t} = 1$ if $i = i_t$, otherwise $\delta_{i,i_t} = 0$. Given the error rate $\epsilon$ and the failure probability $\delta$, we have $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$. A 2-dimension array $A$ with $d$ rows and $w$ columns is created to store the counters. The required SRAM size is determined by $\lceil \frac{e}{\epsilon} \rceil \cdot \lceil \ln \frac{1}{\delta} \rceil$. Upon receiving a packet $t$, the following counters are updated: $A[j][h_j(i_t)] = A[j][h_j(i_t)] + c_t, j = 1, 2 \ldots d$. The size of flow $i$ up to $\tau$ can be estimated as: $\hat{s}_i(\tau) = \min_j A[j][h_j(i_t)](\tau)$. The accuracy function of FSC is given by:

$$f_{FSC}(T, S) = (1 - (1 - e^{-\frac{kn}{mT}})^k) * (1 - \frac{e \cdot \lceil \ln \frac{1}{\delta} \rceil}{S}) \tag{5}$$

These tasks are carefully selected: the packet sampling task is a typical TCAM-based monitoring application, whose accuracy is affected by the allocated number of TCAM entries; the unique IP counting uses fixed number of TCAM entries, while its accuracy is determined by the allocated SRAM size; the flow size counting is a mixed sketch measurement task whose accuracy is related to the allocated TCAM and SRAM resource. Although we only choose three typical tasks, JOTA is generic to be easily extended to a variety types of tasks by providing their theoretical accuracy functions.

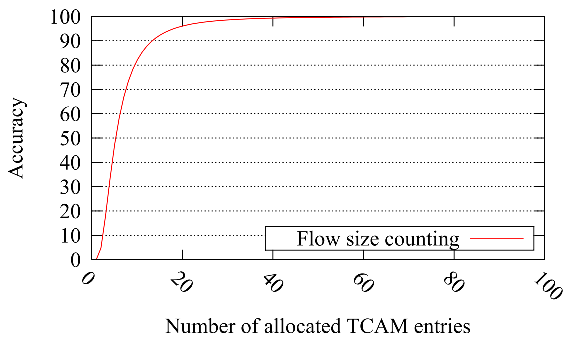

**Fig. 2.** Theoretical accuracy for a bloom filter as the allocated TCAMs increase. (200 target flows, 72 bits TCAM and 3 hash functions).

## 3. JOTA design

In this section, we explain the motivation of the joint task assignment and describe the architecture of JOTA. The formulation of the joint task assignment problem is presented thereafter. Due to its high computational complexity, a two-stage heuristic is proposed to efficiently generate the task assignment.

### 3.1. Motivation

Fig. 2 illustrates the theoretical accuracy for a flow size counting task as the allocated TCAMs increase. An interesting observation is that when the accuracy is above 90%, the required number of TCAM entries increases dramatically even with little increment of the accuracy. Intuitively, for high-accuracy measurement tasks, by sacrificing a little bit accuracy, the consumed resource would be decreased significantly. Combining this property with the provable accuracy bounds of the sketches, it is promising to design a software defined measurement task assignment system, which strikes a good balance between the accuracy and the consumed resource by leveraging the cross-layer information of the measurement tasks.

Resource allocation can be achieved by dynamic or static allocation technique. Dynamic allocation has the capability to adaptively tunes resource according to the traffic change. However, it usually uses simple heuristic to find a sub-optimal solution without sophisticated models and planning. In contrast, static allocation methods can leverage optimization techniques to obtain better assignment result by provisioning the resource usage. For the sketch-based measurement, the resource usage can be easily derived and most of the real-world tasks keep unchanged as time varies. Therefore, static resource allocation is promising in this scenario.

### 3.2. Overview

Theoretically, software define measurement can be divided into three layers as illustrated in Fig. 3. From the top to the bottom, the application layer includes monitoring applications, which accomplish monitoring tasks by the measurement API provided by the management layer. The management layer takes responsibilities to allocate resources and assign tasks to the underlying network. The management layer communicates with the physical layer by the API provided by the SDN controller.

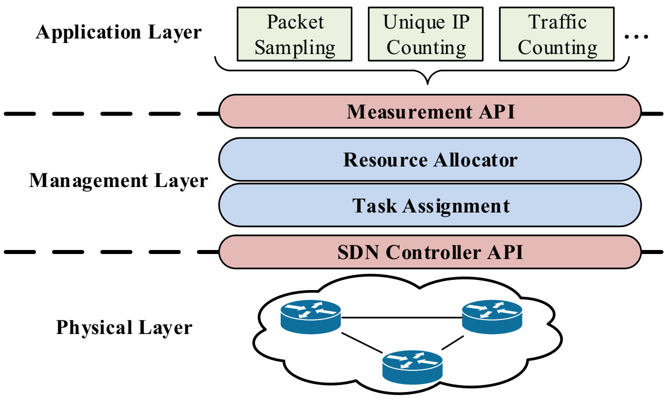

**Fig. 3.** Layers of software defined measurement.

**Table 1**
Summary of notations.

| Symbol | Definition |
|---|---|
| $G$ | The undirected graph of the SDN network |
| $V$ | The set of switches, $m = \|V\|$ |
| $E$ | The set of links |
| $C_i^T$ | The number of TCAM entries in switch $s_i$ |
| $C_i^S$ | The size of SRAM in switch $s_i$ |
| $D$ | The set of tasks, $n = \|D\|$ |
| $L_j$ | The eligible set of $t_j$ |
| $T_j$ | The allocated number of TCAMs for $t_j$ |
| $S_j$ | The allocated SRAM size for $t_j$ |
| $f_j$ | Theoretical accuracy function for $t_j$ |
| $F_{ij}$ | The profit to assign $t_j$ to $s_i$ |
| $\phi$ | A constant for PCSA sketch |
| $l$ | The length of each bitmap |
| $S$ | The allocated SRAM size for PCSA sketch |
| $U$ | The upper bound of the number of unique values |
| $a_j$ | The expected accuracy for $t_j$ |
| $p_j$ | User specified weight for $t_j$ |
| $\alpha$ | Error tolerance factor |

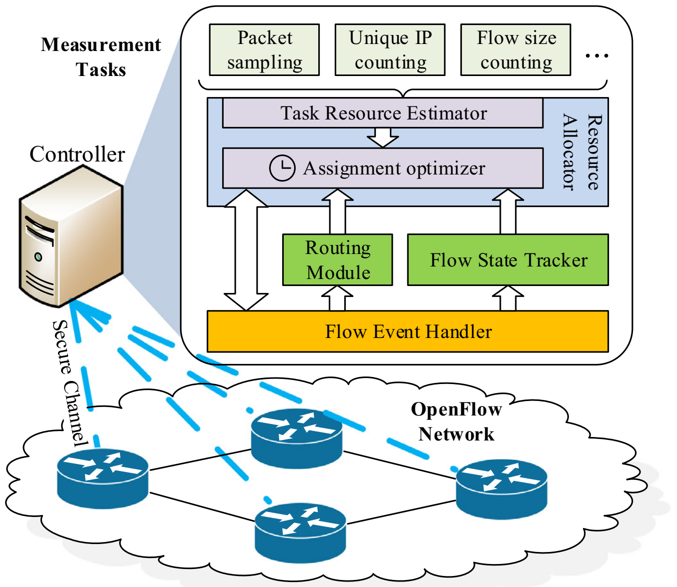

**Fig. 4.** JOTA architecture.

JOTA mainly works in the management layer, while estimates the resource usage in the application layer via the task resource estimator. The trade-off between resource usage and task accuracy can be achieved by the joint optimization across these two layers. Each task can be assigned to a set of switches which is called eligible set. For instance, in a multi-rooted tree topology, a unique IP counting task can be deployed in different root switches. The eligible sets could be specified according to user requirements. Fig. 4 illustrates the architecture of JOTA. The workflow is described as follows. The flow event handler receives and processes low-level network messages through OpenFlow protocol. It pushes flow arrival and removal notifications to the routing module and the flow state tracker. The routing module generates the forwarding path for each flow by the routing algorithm. The eligible switch set is generated by the routing module and the flow state tracker. Meanwhile, the task resource estimator computes the requested resource based on the user configuration and the task type. Finally, the scheduler synthesizes all the information and calculates the optimal task assignment. The assignment is distributed to underlying physical switches by the scheduler afterwards.

## 3.3. Problem formulation

Table 1 lists the notations for the task assignment problem. The SDN network is an undirected graph $G = (V, E)$, where $V = \{s_1, s_2, \ldots, s_m\}$ represents the set of switches with $m = |V|$, $E$ represents the set of links in the network. Each switch $s_i$ has $C_i^T$ TCAM entries and $C_i^S$ SRAMs. Let $D = \{t_1, t_2, \ldots, t_n\}$ denote the measurement task set. For each task $t_j$, let $L_j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ represent the eligible set for $t_j$, where $k$ is the number of the candidate switches. Variable $T_j$ denotes the assigned number of TCAM entries for $t_j$, $S_j$ denotes the allocated SRAM size for $t_j$, and $f_j(T_j, S_j)$ denotes the theoretical accuracy function for the corresponding sketch algorithm which is given $T_j$ TCAMs and $S_j$ SRAMs. $f_j$ is one of the three functions $f_{PS}$, $f_{UIC}$ or $f_{fsc}$. Network operator specifies an expected accuracy $a_j$ and a weight $p_j$ for task $t_j$. An error tolerance factor $\alpha$ which indicates the upper bound of loss in accuracy for these tasks is also given. We define the profit of a task as its theoretical accuracy. The objective is to maximize the total profits of all the accepted tasks. Thus, if $t_j$ is not assigned to a switch belongs to the eligible set, the profit $F$ is zero. Let variable $x_{ij}$ denote whether task $t_j$ is assigned to switch $s_i$, the cross-layer task assignment problem can be formulated as:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} F_{ij}(T_j, S_j) \cdot x_{ij} \tag{6}$$

$$\text{subject to: } \sum_{j=1}^{n} T_j x_{ij} \le C_i^T, \forall s_i \in V \tag{7}$$

$$\sum_{j=1}^{n} S_j x_{ij} \le C_i^S, \forall s_i \in V \tag{8}$$

$$\sum_{i=1}^{m} x_{ij} \le 1, \forall t_j \in D \tag{9}$$

$$f_j(T_j, S_j) \ge (a_j - \alpha) \cdot \sum_{i=1}^{m} x_{ij}, \forall t_j \in D \tag{10}$$

$$T_j \ge 0, \forall t_j \in D \tag{11}$$

$$S_j \ge 0, \forall t_j \in D \tag{12}$$

$$x_{ij} \in \{0, 1\}, \forall s_i \in V, \forall t_j \in D \tag{13}$$

where

$$F_{ij}(T_j, S_j) = \begin{cases} f_j(T_j, S_j) * p_j, & \text{if } s_i \in L_j; \\ 0, & \text{otherwise.} \end{cases}$$

$$f_{PS}(T, S) = \sum_{i=1}^{T} 2^{-i} = 1 - \frac{1}{2^T}$$

$$f_{UIC}(T, S) = 1 - \frac{\phi\sqrt{l}}{\sqrt{S}}, T = \lceil \log_2 U \rceil c$$

$$f_{FSC}(T, S) = 1 - (1 - e^{-\frac{kn}{mT}})^k * (1 - \frac{e \cdot \lceil \ln \frac{1}{\delta} \rceil}{S})$$

$$x_{ij} = \begin{cases} 1, & \text{if } t_j \text{ is assigned to switch } s_i; \\ 0, & \text{otherwise.} \end{cases}$$

The formulation (6) is a Mixed Integer NonLinear Programming (MINLP) problem. Constraints (7) and (8) make sure that the total consumed resource will not exceed the capacity of each switch. Constraint (9) guarantees that each task will be assigned once to the switches. Constraint (10) prevents the task accuracy drops below the accuracy bound. Finally, (11) and (12) specify $T_j$ and $S_j$ are positive, and $x_{ij}$ is a binary variable. The formulation is known to be NP-hard [12]. In practice, it is more difficult to solve than the mixed integer programming and nonlinear programming problem. Owing to its high computational complexity, we propose a two-stage heuristic to generate task assignment efficiently.

### 3.4. Two-stage heuristic

In software defined measurement, there are two stages to deploy a measurement application: 1. estimate the resource usage in the application layer and 2. find the devices with enough available resource in the management layer to assign these applications. It is natural to decompose the initial problem into resource estimation stage and task assignment stage. It is also worth noting that by this partition, we divide the problem into a nonlinear programming problem and an integer programming problem. At the same time, the task accuracy information is maintained. After this transformation, we propose a two-stage heuristic to efficiently generate the task assignment with negligible performance loss. In the following section, we first describe the resource estimation stage which estimates and compresses the resource usage, then we detail the task assignment stage to place these tasks into the underlying switches.

#### 3.4.1. Resource estimation

Suppose all the tasks have a fixed demand of resource usage, then the only way to accept more tasks and increase the profit is to find a better placement of tasks, which is similar to a packing problem. However, benefit from the diminishing marginal utility property of sketch-based measurement, we argue that it is possible to compress the resource usage to accommodate more tasks. Hence, the objective of the resource compression problem is to maximize the sum of tasks' accuracy with the least total resource usage. The compression ratio $r$ can be specified by the network operator. The resource compression problem can be formulated as:

$$\max \sum_{j=1}^{n} f_j(T_j, S_j) \cdot p_j \tag{14}$$

$$\text{subject to: } \sum_{j=1}^{n} T_j \le r \cdot \sum_{i=1}^{m} C_i^T \tag{15}$$

$$\sum_{j=1}^{n} S_j \le r \cdot \sum_{i=1}^{m} C_i^S \tag{16}$$

$$f_j(T_j, S_j) \ge a_j - \alpha, \forall j \in N \tag{17}$$

$$T_j \ge 0, \forall t_j \in D \tag{18}$$

$$S_j \ge 0, \forall t_j \in D \tag{19}$$

where

$$f_{PS}(T, S) = \sum_{i=1}^{T} 2^{-i} = 1 - \frac{1}{2^T}$$

$$f_{UIC}(T, S) = 1 - \frac{\phi\sqrt{l}}{\sqrt{S}}, T = \lceil \log_2 U \rceil$$

$$f_{FSC}(T, S) = 1 - (1 - e^{-\frac{kn}{mT}})^k * (1 - \frac{e \cdot \lceil \ln \frac{1}{\delta} \rceil}{S})$$

$$x_{ij} = \begin{cases} 1, & \text{if } t_j \text{ is assigned to switch } s_i; \\ 0, & \text{otherwise.} \end{cases}$$

Since the resource compression problem is a nonlinear optimization problem, we propose an iterative numerical heuristic in Algorithm 1 to approximate the optimal result. The key insight behind the algorithm is that we always pick the most cost-effective task to reduce the resource usage, until the total resource usage is below the expected value. We introduce a parameter $g$ which is the granularity to trade off the accuracy. At each step, we sacrifice $\frac{\alpha}{g}$ accuracy and choose the most cost-effective task which reduces maximum resource usage. In order to find such task, we apply a priority queue to assist finding the task with the highest accuracy efficiently ($O(1)$ time complexity). The complexity of the algorithm is bounded by $O(ng\log n)$, where $n$ is the number of tasks and $g$ is the iteration granularity.

**Algorithm 1:** Resource compression algorithm.

```
Input: $T$: task set; $A$: task accuracy vector; $C^{TCAM}$: expected
       TCAM usage; $C^{SRAM}$: expected SRAM usage; $\alpha$: error
       tolerance factor; $g$: iteration granularity
Output: $optA$: Optimized task accuracy vector; $optT$:
        compressed TCAM usage; $optS$: compressed SRAM
        usage
1  $optA \leftarrow A, optT \leftarrow 0, optS \leftarrow 0, u \leftarrow \alpha/g$ ;
2  $Init(pq)$ ;                              // Init a priority queue
3  for $j \leftarrow 1$ to $n$ do
4      $Insert(pq, T[j], optA[j])$ ;
5      $(t, s) \leftarrow f_j^{-1}(optA[j])$ ;
6      $optT \leftarrow optT + t, optS \leftarrow optS + s$ ;
7  while $optT > C^{TCAM}$ or $optS > C^{SRAM}$ do
8      $m \leftarrow Pop(pq)$ ;   // Pop the task that has the maximum
       accuracy
9      $(t_{prev}, s_{prev}) \leftarrow f_j^{-1}(optA[m])$ ;
10     if $optA[m] - u > A[m]$ then
11         $optA[m] \leftarrow optA[m] - u$ ;
12     $Update(pq, m, optA[m])$ ;
13     $(t, s) \leftarrow f_j^{-1}(optA[m])$ ;
14     $optT \leftarrow optT - (t - t_{prev}), optS \leftarrow optS - (s - s_{prev})$
15 return $optA, optT, optS$
```

### 3.4.2. Task assignment

The key challenge of task assignment stage is to efficiently find a placement to maximize the number of accepted tasks. The assignment should satisfy the constraints that for each switch, the consumed TCAMs and SRAMs will not exceed the available TCAM entries and SRAMs respectively. Therefore, the task assignment problem can be formulated as an integer linear programming problem:

$$\max \sum_{i=1}^{m}\sum_{j=1}^{n} f_j(T_j, S_j)\cdot p_j \cdot x_{ij} \tag{20}$$

$$\text{subject to: } \sum_{j=1}^{n} T_j x_{ij} \le C_i^T, \forall s_i \in V \tag{21}$$

$$\sum_{j=1}^{n} S_j x_{ij} \le C_i^S, \forall s_i \in V \tag{22}$$

$$\sum_{i=1}^{m} x_{ij} \le 1, \forall t_j \in D \tag{23}$$

$$x_{ij} \in \{0,1\}, \forall s_i \in V, t_j \in D \tag{24}$$

where

$$x_{ij} = \begin{cases} 1, & \text{if } t_j \text{ is assigned to switch } s_i; \\ 0, & \text{otherwise.} \end{cases}$$

The problem formulation (20) is a Multi-Resource Generalized Assignment Problem (MRGAP) which is shown to be NP-hard [13]. It had been shown to be harder to solve than the generalized assignment problem (GAP) [12]. Due to its high computational complexity, heuristic algorithms are needed to find the assignment efficiently. A straightforward approach is a naive greedy algorithm. For each task, we select a switch belongs to its eligible set and has enough TCAM and SRAM resource to accept this task. We assign the task to this switch and update its available resource. Repeat such process until no more task can be accepted. We show this first fit algorithm in Algorithm 2, and we refer to it as “FF”.

**Algorithm 2:** First fit algorithm.

```
   Input: $T$: task set; $L$: eligible switch set for each task
   Output: $A$: a feasible task assignment
 1 $A \leftarrow \{-1, -1, \ldots, -1\}$ ;      // Init the assignment vector
 2 foreach $t \in T$ do
 3     foreach $l \in L_t$ do
 4         if $TCAM(s) \ge T_t$ and $SRAM(s) \ge S_t$ then
 5             $A[t] \leftarrow s$, $TCAM(s) \leftarrow TCAM(s) - T[t]$,
               $SRAM(s) \leftarrow SRAM(s) - S[t]$ ;
 6             break ;
 7 return $A$
```

Although the first fit algorithm is always able to provide a feasible solution, it has no performance guarantee. Also, the generated assignment has bad load balancing due to the algorithm prefers to assign tasks to the switches which appear first. Thus, the algorithm may lead to a sub-optimal solution. We try to improve the first fit algorithm by reducing the MRGAP to a GAP which has good approximation algorithms. We observe that the TCAM is more precious than the SRAM as it has a comparatively smaller size and harder to compress. Based on this observation, we develop a novel algorithm which relax the SRAM constraint and leverage a GAP approximation algorithm to provide a better assignment.

First, we ignore the SRAM constraint and generate the assignment only by the TCAM constraint. We attempt to improve the assignment by iteratively updating the profit and the assignment. Suppose we find an assignment for the first switch $s_1$, which is a 0–1 knapsack problem. Before processing the remaining switches, we update the profit function according to the following rules: if a task is assigned to $s_1$, the profit to assign this task to $s_i, i = 2, 3, \ldots, m$ is updated to $p_i - p_1$. Otherwise, no action needed. Then we find an assignment for the second switch $s_2$ and update the profit to assign the task to $s_i, i = 3, 4, \ldots, m$ if the task is assigned to $s_2$. Repeat this process until all the switches are processed. Currently the assignment only takes TCAM constraint into account. Namely, the assignment may violate the SRAM constraint which leads to an infeasible solution. Therefore, we perform constraint checking and remove the tasks which exceed the available SRAM resource in each switch from the assignment. Finally, we rerun the first fit algorithm to fill unassigned tasks into these avail spaces. Note that this process corresponds to an algorithm shown in Algorithm 3, and we refer to this approximation algorithm as “APX”. The rationale behind the algorithm is that the “marginal utility” to assign a task to a switch is adjusted by iteratively updating the profit from time to time. Moreover, this algorithm is proved to have a $(1+\beta)$ approximation ratio [14], where $\beta$ is the approximation ratio to solve the knapsack problem. Specifically, if we solve the knapsack problem optimally by dynamic programming, the final task assignment has an approximation ratio of 2; if a greedy algorithm is applied to solve the knapsack problem with an approximation ratio of 2, the final task assignment approximation ratio is 3.

**Algorithm 3:** Approximation algorithm.

```
   Input: $T$: task set; $L$: eligible switch set for each task; $v_{ij}$:
          profit to assign task $j$ to switch $i$
   Output: $A$: a feasible task assignment
 1 $A \leftarrow \{-1, -1, \ldots, -1\}$ ;      // Init the assignment vector
 2 for $j \leftarrow 1$ to $m$ do
 3     for $i \leftarrow 1$ to $n$ do
 4         if $A[i] = -1$ then
 5             $V_j[i] = v_{ij}$ ;
 6         else if $A[i] = k > 0$ then
 7             $V_j[i] = v_{ij} - v_{ik}$ ;
       // $Q$ stores the knapsack problem assignment
 8     $Q \leftarrow ApproximateKnapsack(V_j, L)$ ;
       // Iteratively update the GAP assignment by $Q$
 9     for $i \leftarrow 1$ to $n$ do
10         if $Q[i]! = -1$ then
11             $A[i] \leftarrow Q[i]$ ;
   // Perform constraint checking and fill unassigned
      tasks
12 for $j \leftarrow 1$ to $m$ do
13     $Init(TCAM(s_j)), Init(SRAM(s_j))$ ;
14 for $i \leftarrow 1$ to $n$ do
15     if $A[i] > 0$ and $SRAM(A[i]) - SRAM(t_i) >= 0$ then
16         $SRAM(A[i]) \leftarrow SRAM(A[i]) - SRAM(t_i)$,
           $TCAM(A[i]) \leftarrow TCAM(A[i]) - TCAM(t_i)$ ;
17 $FirstFit(T, L, A)$ ;
18 return $A$
```

**Table 2**
The performance comparison of the two-stage heuristic and the optimal solution for a small instance.

| | Profit | Running time (s) |
|---|---|---|
| Optimal | 2.884 | 33.25 |
| 1st stage | 2.861 | 0.27 |
| 2nd stage | 2.884 | 0.02 |

### *3.5. Implementation discussion*

The implementation of software defined measurement has been well studied [1,2,15]. Typically, these measurement implementations require hash functions, counters and wildcard rules, which can be implemented in today's commodity switches with slightly hardware modification. JOTA mainly works in the controller and it is a shim layer between the underlying network and measurement applications, which is on top of the prior sketch-based measurement and has no extra implementation requirements.

## 4. Evaluation

In this section, we evaluate and analyze the performance of JOTA comprehensively. We build a simulator written in Python and a GAP solver written in C++. All the experiments are conducted on a server with an Intel i7-4770 3.40 GHz processor and 32G memory. Both real-world topology and artificial generated graph are applied to evaluate the algorithms. We set similar parameters as the prior work [4,5]. Specifically, the number of TCAM entries and SRAM size for each switch are set to 1024 and 4096 KB respectively. The task accuracy is generated with a normal distribution (0.95, 0.2). We generate the source and destination hosts in a uniform random manner and use the shortest path as their forwarding paths. The weight of each task is uniformly generated within range (0, 1). The workload consists of three types of tasks as mentioned in Section 2. The accuracy experiments are conducted using real-world packet traces collected from a university data center network [16].

### *4.1. Two-stage heuristic performance*

To verify the performance of the two-stage heuristic, we compare the optimal solution and the heuristic in terms of the profit and the running time. We solve the MINLP by a widely-used MINLP solver OpenOpt [17]. We refer to this solution as optimal. The test instance consists of 4 switches and 3 tasks of different types as mentioned in Section 2. We use a compression ratio of 0.9 to reduce the resource usage. Table 2 lists the profit and the running time. Clearly, our two-stage heuristic generates the same result as the initial solution but 100 times faster than solving the initial problem. Note that even for such a simple and unpractical instance, the optimizer needs more than half a minute to solve the MINLP. The optimal algorithm is not scalable to practical instances. Moreover, the measurement task assignment usually involves dynamic flows and packets, it is crucial to generate a feasible solution within several seconds. On the other hand, the MINLP solver iteratively searches the solutions and we observe that the results are sensitive to the initial start point. Comparatively, the convergence of our two-stage heuristic is guaranteed and it provides a feasible solution all the time. In the following section, we will verify the performance of the two-stage heuristic on practical large-sized networks.

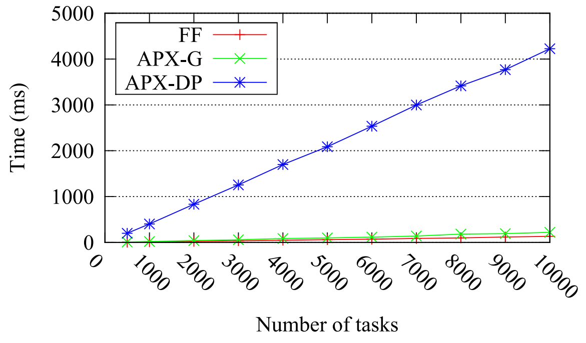

**Fig. 5.** Comparison of the total running time for different algorithms with resource compression (the compression ratio 0.7).

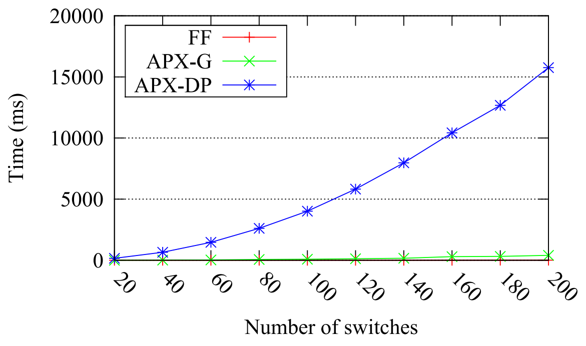

**Fig. 6.** The running time of different algorithms as the number of switches increases.

### *4.2. Running time*

To evaluate the efficiency of our algorithms, we use Erdős-Rényi graph [18–20] ($p = 0.05$) which is a widely-used random graph in network research as the topology. Fig. 5 shows the running time as the number of tasks varies from 500 to 10,000. As introduced before, we propose a first fit algorithm and a $(1 + \beta)$-approximation algorithm to generate task assignments, where $\beta$ is the approximation ratio for the algorithm to solve the knapsack problem. It is known that the greedy algorithm to solve the knapsack problem has an approximation ratio of 2, and the dynamic programming (DP) algorithm is able to obtain the optimal solution with an approximation ratio of 1 [12]. Thus, we use “APX-G” and “APX-DP” to represent the 3-approximation and 2-approximation algorithm respectively. The running time for all the algorithms is almost linear in the number of tasks. Clearly, the FF and the APX-G algorithms are several hundred times faster than the APX-DP algorithm, although the latter provides a tighter bound for the result. In practice, the FF and the APX-G algorithms are preferable since the measurement tasks are changing from time to time. These two algorithms are timely and scalable because they generate the assignment within 90ms even for a huge number of tasks.

We show the running time of varying the switch number from 20 to 200 in Fig. 6. The number of tasks is in proportion to the number of switches with an overload ratio of 100. The figure indicates that the running time for the FF and the APX-G is almost linear in the size of the network. Even for a network with 200 switches and tens of thousands of tasks, they can produce a feasible assignment within 0.5s.

To explore the overhead of the resource compression, we show in Fig. 7 the running time of the resource compression and the task assignment time of APX-G as the number of tasks varies. There is a steady increment for the resource compression and the total running time. In particular, the resource compression time occu-

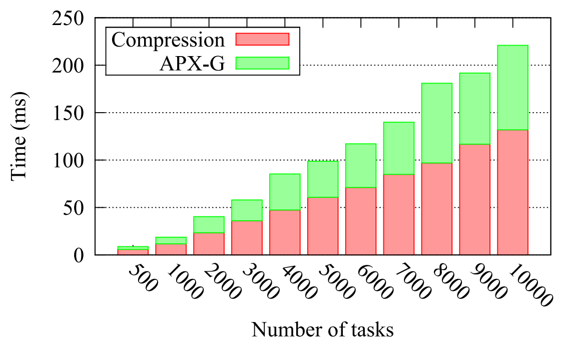

**Fig. 7.** The running time for the resource compression and the task assignment.

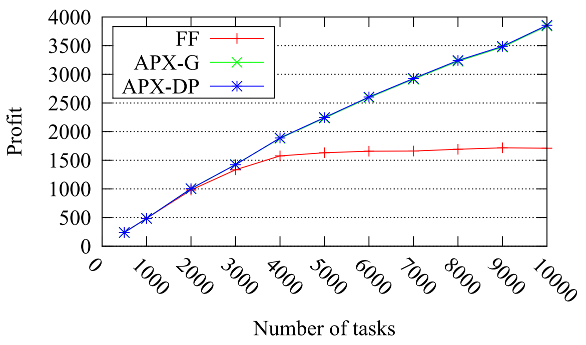

**Fig. 8.** Comparison of the total profit for different algorithms with resource compression (ratio 0.7).

pies 60% of the total running time on average. Fortunately, the total running time is still acceptable since they are less than 0.25 s. We will show in the following section that it is worth doing the resource compression because it significantly reduces the total resource usage and improves the acceptance ratio. Besides, this overhead is acceptable because the extra running time is less than 140ms even for 10,000 tasks. It is also worth noting that the resource compression has no effect on the running time of the second stage, which is a nice property owing to the division of the initial problem.

## 4.3. Resource compression performance

Fig. 8 shows the total profit of the proposed algorithms with resource compression. When the number of tasks is less than 3000, the profit increases linearly as there are space for new coming tasks. However, the APX-G and the APX-DP are superior to the FF when the switches are overloaded. The profit always increases for these two algorithms while the FF performs not well as the number of tasks increases. The performance of the two approximation algorithms has negligible difference. Consider the running time and the performance, the APX-G outperforms the other two algorithms because it strikes a good time-performance trade-off and produces the near-optimal result very efficiently.

We examine the performance of the resource compression algorithm in Figs. 9 and 10. The error tolerance factor $\alpha$ is set to 0.05 which allows up to 5% accuracy loss. As the compression ratio decreases, the total consumed number of TCAM entries is decreased while maintain the tasks' accuracy lower bounds. However, when the compression ratio drops below 0.5, the average task accuracy reaches the bottom. Therefore, the resource usage cannot be compressed anymore. Otherwise, the accuracy of the tasks cannot be guaranteed. Our resource compression algorithm works well to reduce the TCAM and the SRAM usage by up to 49% and 99% respectively. Notice that the required SRAM usage drops sharply as the

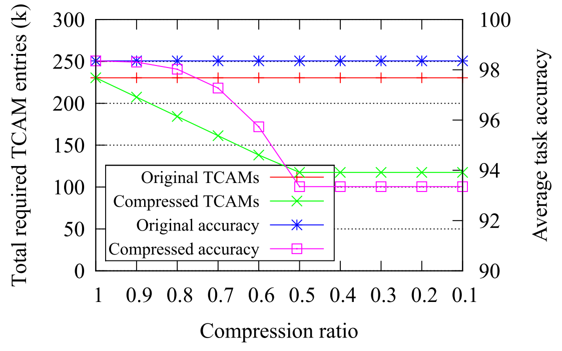

**Fig. 9.** The performance of resource compression of the TCAM with 5,000 tasks.

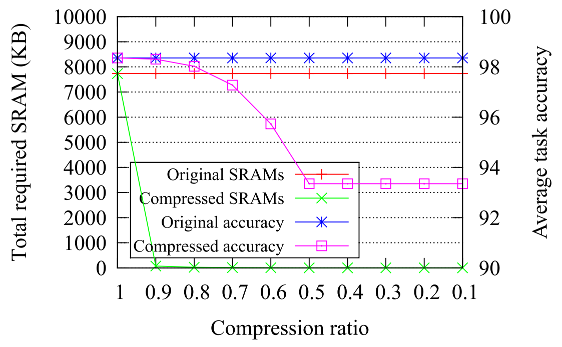

**Fig. 10.** The performance of resource compression of the SRAM with 5,000 tasks.

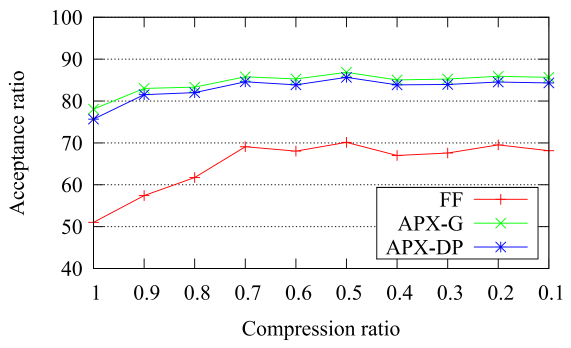

**Fig. 11.** The acceptance ratio of varying the compression ratio.

accuracy decreases. Therefore, Algorithm 3 leverage this property to relax the SRAM constraint. In the following section, we study the relation between the compression ratio and the task acceptance ratio.

## 4.4. Acceptance ratio

The bridge between the two stages of JOTA is the resource compression ratio. We define the task acceptance ratio as the number of accepted tasks over the total number of tasks. We use a real topology “Interroute” which is taken from a public network topology repository the Internet topology zoo [21]. The network consists of 110 switches, which is a medium-sized Europe backbone network. To further explore how the resource compression ratio affects the acceptance ratio, we analyze their relation in Fig. 11.

We clearly observe that the acceptance ratio for all the algorithms raises as the resource compression ratio decreases. Specifically, after the resource compression, the FF raises the acceptance ratio by roughly 20%. The reason is that the FF utilizes a greedy strategy instead of iteratively updating the assignment, resource compression stage greatly helps JOTA to accept more tasks as there

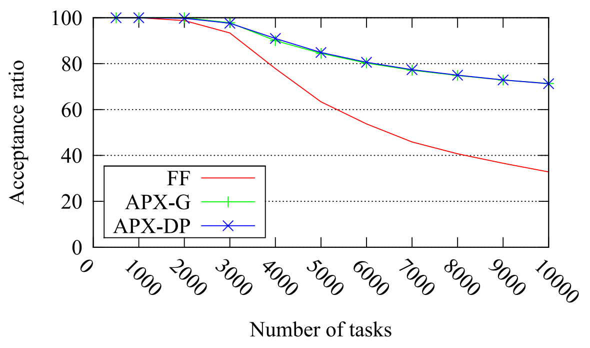

**Fig. 12.** The acceptance ratio of varying the number of tasks (the compression ratio 0.7).

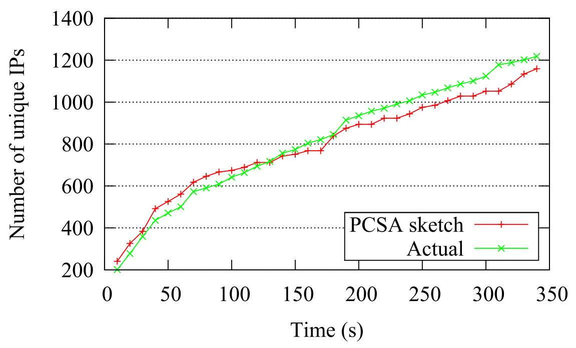

**Fig. 13.** The result of a unique IP counting task (with 11 TCAMs).

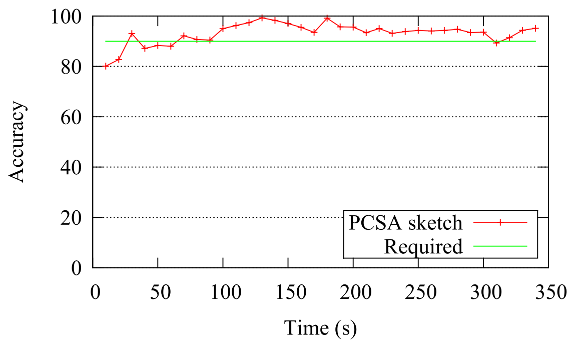

**Fig. 14.** The accuracy of a unique IP counting task.

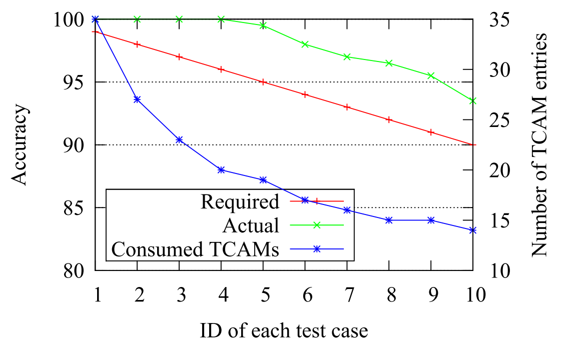

**Fig. 15.** The actual accuracy of a flow size counting task for different required accuracy.

are more available resource. Comparatively, the performance gain for the two approximation algorithms is not as much as the FF, but we still get 10% of acceptance ratio increment. Note that almost all the tasks have been accepted as the acceptance ratio is raised to 85%, the resource compression significantly improve the performance of JOTA. However, when the compression ratio is below 0.7, there is no noticeable performance gain for all the algorithms as the underlying network leaves no room to accommodate more tasks.

In Fig. 12 we show the relation between the number of tasks and the acceptance ratio with a resource compression ratio of 0.7. We clearly observe that the APX-G and the APX-DP perform uniformly better than the FF as the number of tasks increases. This is conform to the profit trend in Fig. 8. The APX-G and the APX-DP accepts 33% more tasks than the FF when the number of tasks is 10,000. The result also illustrates that the proposed approximation algorithms are not sensitive to the increasing number of tasks. Although the acceptance ratio decreases as there is no available resource in the network, the total profit of JOTA keeps increase gradually.

### 4.5. Task accuracy

We use packet traces collected from a university data center [16] to verify the task accuracy in the measurement application layer. We case study the accuracy of an accepted unique IP counting task and a flow size counting task.

The accuracy for unique IP counting task is defined as the difference between the measured and the actual number of unique IPs over the ground truth at one specific moment. Fig. 13 shows the actual and the measured number of unique IPs for a six minute traces. As introduced in Section 2, the unique IP counting task uses a fixed number of TCAMs which cannot be compressed. The required accuracy is 90%. The query interval for the number of unique IPs is ten seconds. Clearly, the measured number of unique IPs closely follows the actual number of unique IPs. The required accuracy and the actual accuracy are depicted in Fig. 14. Obviously, the accuracy is better than the required accuracy in most of the time. The average accuracy is 93.1% which is above the required accuracy.

We also examine the accuracy of a flow size counting task as the required accuracy decreases from 99% to 90%. A six minute traces with 22,000 flows are used in our experiment. The number of target flows are 200. The accuracy is defined as the number of correctly measured flows over the total number of flows. The result is shown in Fig. 15. The consumed number of TCAMs are also plotted. We observe that the actual accuracy is always above the required accuracy. This is conform to the theoretical false positive rate bound of the bloom filter. Also, by relaxing the accuracy from 99% to 96%, the consumed number of TCAMs drops from 35 to 20, which reduces 43% of the resource usage for this task.

In summary, we verify that our two-stage heuristic provides near-optimal solution and it is scalable to large scale networks. The APX-G algorithm strikes a good trade-off between the running time and the performance. Our resource compression algorithm is quite effective and greatly helps the FF algorithm to increase the acceptance ratio. It assists the approximation algorithms to accept more tasks as well. The case studies of the two typical measurement applications demonstrate that the task assignment generated by JOTA guarantees the expected accuracy.

## 5. Related work

Software defined measurement has been extensively studied recently. ProgME [22] enabled flexible flow counting by introducing the concept of flow set which is an arbitrary set of flows for different applications. OpenSketch [1] proposed a three-stage pipeline to implement a variety of sketch-based streaming algorithms for network monitoring. DCM [15] designed a two-stage bloom filters as the data plane to achieve memory-efficient flow measurement. Furthermore, MeasuRouting [23] rethinks the measurement framework to route the traffic according to the measurement requirements. Besides, sketch techniques were widely used in different issues of data mining field, such as tracking frequent items [24,25] and topic discovery [26]. These work proposed various frameworks to improve the implementation of measurement applications. Our work is inspired by these proposals and built on top of the OpenSketch architecture.

Based on existing software defined measurement frameworks, many systems are proposed to implement cost-effective measurement applications. OpenTM [27] presented a traffic matrix estimation system in the OpenFlow-based network. It explored the tradeoff between the switch workload and the accuracy by comparing different query strategies. Following work [28,29] globally optimized the overhead of statistics collection. PayLess [30] proposed APIs to collect the flow statistics using adaptive polling frequency. In order to further reduce the measurement cost, FlowSense [31] employed passive measurement techniques to infer the network utilization with zero measurement cost. The implementation of secure monitoring in SDN are studied as well [32,33]. On the other hand, sampling is widely used to reduce the overhead of software defined measurement. CSAMP [34] maximized the monitoring flow coverage by consistent sampling. DevoFlow [2] proposed a sampling-based method to improve the performance of statistics collection. Moreover, a sampling extension for monitoring applications is presented in [35]. OpenSample [36] attempted to reduce the control loop for active measurement in SDN by a protocol-aware sampling. OpenWatch [37] proposed a prediction based dynamic adjustment scheme for anomaly detections, it tuned the measurement granularity in both the spatial and the temporal dimensions. Besides, low-latency software defined monitoring is also explored [36,38,39]. These work mainly focused on the design of cost-effective measurement applications. In contrast, JOTA works as a middle-layer to optimize the placement of these applications.

Resource allocation for software defined measurement has attracted significant attention from many researchers. L. Jose et al. detected hierarchical heavy hitters by changing the measurement rules in the switches [40]. Detailed analysis of the trade-off between the accuracy and the resource usage was presented in [5]. Since the case study only focused on the single switch measurement task, DREAM [4] extended the work to dynamically allocate resource across multiple switches. A recent work SCREAM [41] used a variation of the dynamic allocation algorithm to allocate SRAM resource to sketch-based measurement tasks. DREAM and SCREAM are the most related work to JOTA. However, DREAM focused on TCAM-based measurement, while SCREAM concentrated on memory-hungry sketch-based measurement. Besides, our previous work [9] optimized sketch-based measurement by considering only TCAM resource allocation as well. JOTA is orthogonal to these approaches since it takes both TCAM and SRAM resource constraints into account. Furthermore, after we revisit the task assignment process, we introduce a novel resource compression stage to globally optimize the task assignment.

## 6. Conclusions and future work

In this paper, we present JOTA, a novel sketch-based measurement task assignment system for software defined networks. We explore the relation between the accuracy and the allocated resource for various sketch-based measurement tasks. By considering both the estimation of resource usage in the application layer and the available resource in task assignment layer, we formulate the task assignment problem as a mixed integer nonlinear programming problem. Due to its high computational complexity, we decompose it into the resource compression stage and the task assignment stage. A two-stage heuristic is proposed to efficiently produce task assignment. Specifically, JOTA compresses the resource usage in the first stage and relaxes one constraint to transform the problem to an MRGAP in the second stage. It leverages a $(1+\beta)$ approximation algorithm to generate task assignment thereafter. The performance of JOTA is verified by simulations with different types of tasks and real-world packet traces. The experimental results demonstrate that JOTA significantly reduces the resource usage and increases the task acceptance ratio with negligible loss in accuracy. Currently, we only support assigning task to one switch. In the future, we plan to extend our model to accommodate multiple switches assignment with performance guarantee. We also would like to develop an intelligent algorithm to find the optimal compression ratio without user intervention.

## Acknowledgments

This research is supported by HKUST Research Grants Council (RGC) 613113.

## References

[1] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with OpenSketch, NSDI, 2013.
[2] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: scaling flow management for high-performance networks, SIGCOMM, 2011.
[3] K. Kannan, S. Banerjee, Compact tcam: Flow entry compaction in tcam for power aware sdn, in: International Conference on Distributed Computing and Networking, 2013, pp. 439–444.
[4] M. Moshref, M. Yu, R. Govindan, A. Vahdat, DREAM: Dynamic resource allocation for software-defined measurement, SIGCOMM, 2014.
[5] M. Moshref, M. Yu, R. Govindan, Resource/accuracy tradeoffs in software-defined measurement, HotSDN, 2013.
[6] NetFlow, http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.
[7] M. Wang, B. Li, Z. Li, sFlow: Towards resource-efficient and agile service federation in service overlay networks, ICDCS, 2004.
[8] P. Flajolet, G. Nigel Martin, Probabilistic counting algorithms for data base applications, J. Comput. Syst. Sci. 31 (2) (1985) 182–209.
[9] Z. Su, T. Wang, M. Hamd, COSTA: Cross-layer optimization for sketch-based software defined measurement task assignment, in: IWQoS, 2015, pp. 183–188.
[10] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, J. Algorithms 55 (1) (2005) 58–75.
[11] A. Goel, P. Gupta, Small subset queries and bloom filters using ternary associative memories, with applications, SIGMETRICS, 2010.
[12] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Inc., 1990.
[13] B. Gavish, H. Pirkul, Algorithms for the multi-resource generalized assignment problem, Manage. Sci. 37 (6) (1991) 695–713.
[14] R. Cohen, L. Katzir, D. Raz, An efficient approximation for the generalized assignment problem, Inf. Process. Lett. 100 (4) (2006) 162–166.
[15] Y. Yu, Q. Chen, X. Li, Distributed collaborative monitoring in software defined networks, HotSDN, 2014.
[16] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, IMC, 2010.
[17] Openopt, http://www.openopt.org/.
[18] B. Bollobás, Random Graphs, Academic Press, 1985.
[19] A. Jamakovic, P. Van Mieghem, On the robustness of complex networks by using the algebraic connectivity, in: International Conference on Research in Networking, 2008, pp. 183–194.
[20] R. Van Der Hofstad, Random graphs and complex networks.
[21] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, JSAC 29 (9) (2011) 1765–1775.
[22] L. Yuan, C.-N. Chuah, P. Mohapatra, ProgME: Towards programmable network measurement, ToN, 2011.
[23] S. Raza, G. Huang, C.-N. Chuah, S. Seetharaman, J.P. Singh, Measurouting: A framework for routing assisted traffic monitoring, ToN, 2012.
[24] Y. Tong, X. Zhang, L. Chen, Tracking frequent items over distributed probabilistic data, World Wide Web 19 (4) (2016) 579–604.
[25] Y. Tong, L. Chen, Y. Cheng, P.S. Yu, Mining frequent itemsets over uncertain databases, Proc. VLDB Endowment 5 (11) (2012) 1650–1661.
[26] Y. Tong, C.C. Cao, L. Chen, Tcs: efficient topic discovery over crowd-oriented service data, in: SIGKDD, 2014, pp. 861–870.
[27] A. Tootoonchian, M. Ghobadi, Y. Ganjali, OpenTM: traffic matrix estimator for OpenFlow networks, PAM, 2010.
[28] Z. Su, T. Wang, Y. Xia, M. Hamdi, FlowCover: Low-cost flow monitoring scheme in software defined networks, GLOBECOM, 2014.
[29] Z. Su, T. Wang, Y. Xia, M. Hamdi, CeMon: A cost-effective flow monitoring system in software defined networks, Comput. Netw. 92 (2015) 101–115.

[30] S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba, PayLess: A low cost network monitoring framework for software defined networks, NOMS, 2014.
[31] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, FlowSense: Monitoring network utilization with zero measurement cost, PAM, 2013.
[32] S. Shin, G. Gu, Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?), ICNP, 2012.
[33] A. Zaalouk, R. Khondoker, R. Marx, K. Bayarou, Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions, NOMS, 2014.
[34] V. Sekar, M.K. Reiter, W. Willinger, H. Zhang, R.R. Kompella, D.G. Andersen, CSAMP: a system for network-wide flow monitoring, NSDI, 2008.
[35] S. Shirali-Shahreza, Y. Ganjali, FleXam: flexible sampling extension for monitoring and security applications in openflow, HotSDN, 2013.
[36] S. Junho, T.T. Kwon, C. Dixon, W. Felter, J. Carter, Opensample: A low-latency, sampling-based measurement platform for commodity sdn, ICDCS, 2014.
[37] Y. Zhang, An adaptive flow counting method for anomaly detection in sdn, CoNEXT, 2013.
[38] sflow-rt, http://inmon.com/products/sFlow-RT.php.
[39] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, R. Fonseca, Planck: millisecond-scale monitoring and control for commodity networks, SIGCOMM, 2014.
[40] L. Jose, M. Yu, J. Rexford, Online measurement of large traffic aggregates on commodity switches, HotICE, 2011.
[41] M. Moshref, M. Yu, R. Govindan, A. Vahdat, Scream: Sketch resource allocation for software-defined measurement, CoNEXT, 2015.