COSTA: Cross-layer Optimization for Sketch-based Software Defined Measurement Task Assignment

Zhiyang Su*, Ting Wang*, Mounir Hamdi*†
*Hong Kong University of Science and Technology, †Hamad Bin Khalifa University
{zsuab, twangah, hamdi}@cse.ust.hk

Abstract—**Sketch-based measurement provides traffic data summary in a memory-efficient way with provable accuracy bound. Recent advances in software defined networking (SDN) facilitate the development and implementation of sketch-based measurement applications. However, sketch-based measurement usually requires TCAMs 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 COSTA, a novel system to improve the performance of the task assignment for sketch-based measurement. We utilize the cross-layer information between the application and the task assignment layers to 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 produce task assignment efficiently. In particular, we present an algorithm which guarantees $(1+\alpha)$ approximation ratio to solve the second stage task assignment. Extensive experiments with three different measurement tasks and real packet traces demonstrate that COSTA significantly reduces the resource usage by up to 40% and accepts 30% more tasks.**

I. 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 attentions from both academia and industry. SDN enables the programmability of the network by decoupling the control plane and the data plane. Software defined measurement leverages the flexibility of SDN to implement many TCAM-based measurement tasks. Specifically, software defined measurement can accommodate many sketch-based streaming algorithms to provide compact traffic summary [1], [2]. Compared with traditional network measurement tools, software defined measurement is more flexible, generic and cost-effective.

However, software defined measurement brings both opportunities and challenges. Despite the high flexibility of sketch-based measurement tasks, 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 [3]. But the accuracy of the upper layer monitoring applications are determined by the allocated resource [2], [4]. Moreover, TCAMs are mainly used as forwarding components, which further limits the number of entries for monitoring applications.

We observe that software defined 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. For instance, for a probabilistic counting with stochastic averaging (PCSA) sketch [5], 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. We leverage the cross-layer information between the application and the management layers to optimize the final assignment intelligently.

The primary contributions of this paper are listed below:

- We present COSTA, a cross-layer optimized system for sketch-based measurement, which achieves different levels of balance between measurement application accuracy and resource usage.
- We formulate the sketch-based software defined measurement task assignment as a mixed integer nonlinear programming problem which has been shown to be NP-hard. Due to its high computational complexity, we develop a near-optimal two-stage heuristic to generate the final task assignment with negligible performance loss.
- Extensive experimental results demonstrate that COSTA is efficient in terms of running time, accept ratio and accuracy. It can reduce the resource usage by nearly 40% and raises the task accept ratio by up to 30%.

The rest of this paper is structured as follows. Section II describes the architecture of COSTA and formulates the cross-layer task assignment problem in software defined measurement. Section III elaborates on the performance of COSTA by real packet traces simulation. Finally, Section IV summarizes related work and Section V concludes the paper.

II. COSTA Design

In this section, we introduce the methodology to estimate the measurement resource usage and describe the architecture of COSTA. The formulation of the cross-layer task assignment problem is presented thereafter. Due to high computational complexity of the problem, a two-stage heuristic is proposed to efficiently generate the task assignment.

A. Resource Estimation of Sketch-based Measurement Tasks

Sketch-based measurement usually uses streaming algorithms to provide traffic summary. In this section, we detail the implementation of three typical tasks and present how to estimate the resource usage by their accuracy bounds.

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. Given the fraction part of the binary floating point number $B = b_1 b_2 \dots b_n$, we need $l(B) = \sum_{i=1}^{n} b_i$ TCAM entries to implement the arbitrary probability packet sampling [1]. 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 ith bit to 0, and wildcarded all other bits. 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 $T = l(B)$ afterwards. The accuracy function is given by:

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

2) Unique IP Counting: Unique IP Counting (UIC) is another fundamental monitoring task in network management, we can employ PCSA sketch [5] 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 $B = b_1 b_2 \dots b_l$ of a length l. All bits are initialized to zero in the beginning. In order to insert an element x, the hash value $h(x)$ is obtained by a uniform distributed hash function h. Then, we find the position of the rightmost "1" in $h(x)$ and set the bit of the same position of B 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 $l(B) = \min\{i | 0 < i \leq n \wedge b_i = 0\} - 1$. 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. The number of unique values is estimated as $m \cdot 2^{\frac{1}{m}\sum_{i=1}^{m} l(B_i)}/\phi$. The relative error of this estimation has been shown to be roughly $\phi/\sqrt{m}$ [5]. Assume the allocated SRAM size is S, the theoretical accuracy

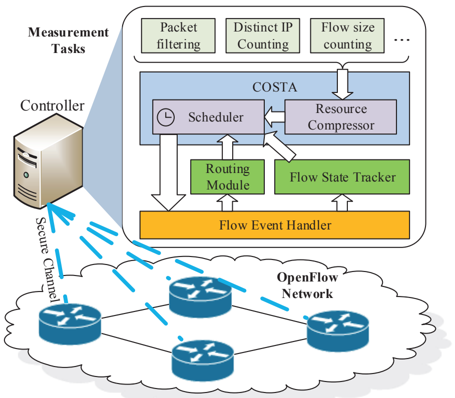

Fig. 1. COSTA architecture.

for PCSA sketch is $1 - \frac{\phi\sqrt{l(B)}}{\sqrt{S}}$. The PCSA sketch can be implemented by a group of wildcard rules as the PS task. The accuracy function is given by:

$$f_{UIC}(T) = 1 - \frac{\phi\sqrt{l(B)}}{\sqrt{S}} \tag{2}$$

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 a bloom filter and a CM sketch [6], where the former filters the target flows and the latter counts the flow size. Bloom filter can be implemented by TCAMs in switch. 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 α, the required number of TCAM entries can be obtained by $T = \lceil -\frac{nk}{m \ln(1-(1-\alpha)^{\frac{1}{k}})} \rceil$. 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 τ 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 τ, where $\delta_{i,i_t} = 1$ if $i = i_t$, otherwise $\delta_{i,i_t} = 0$. A 2-dimension array A with d rows is created to store the counters. Upon receiving a packet t, the following counters are updated: $A[j][h_j(i_t)] = A[j][h_j(i_t)] + c_t, j = 1, 2 \dots d$. The size of flow i up to τ 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:

$$f_{FSC}(T) = 1 - (1 - e^{-\frac{kn}{mT}})^k \tag{3}$$

These tasks are carefully selected: the packet sampling task is a typical TCAM-based monitoring application, the unique IP counting uses fixed number of TCAMs which cannot be compressed, and the flow size counting task is a mixed sketch-based measurement application. Although we only choose three typical tasks, COSTA is generic to be easily extended to a variety types of tasks by providing their theoretical accuracy functions.

Symbol	Definition
G	The undirected graph of the SDN network
V	The set of switches, $m = \|V\|$
E	The set of links
C_i	The number of TCAM entries in switch s_i
D	The set of tasks, $n = \|D\|$
L_j	The eligible set of t_j
T_j	The assigned number of TCAMs for t_j
f_j	Theoretical accuracy function for t_j
F_{ij}	The profit to assign t_j to s_i
ϕ	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
α	Error tolerance factor

TABLE I
SUMMARY OF NOTATIONS.

B. Architecture

Figure 1 illustrates the architecture of COSTA. Software defined measurement usually consists of three layers: the application layer, the management layer and the physical layer. COSTA mainly works in the management layer, while estimates the resource usage in the application layer via the task resource estimator. 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.

C. Problem Formulation

Table I 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 TCAM entries. Let $D = \{t_1, t_2, \ldots, t_n\}$ denote the measurement task set. In software defined measurement, a task usually can be assigned to a set of switches which is named eligible 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, $f_j(T_j)$ denotes the theoretical accuracy function for the corresponding sketch algorithm which is given T_j TCAMs. f_j is one of the three functions f_{PS}, f_{UIC} or f_{fsc}. Network operator specifies an expected accuracy a_j for task t_j. An error tolerance factor α 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:

$$\begin{aligned}
\max \; & \sum_{i=1}^{m}\sum_{j=1}^{n} F_{ij}(T_j)\cdot x_{ij} \\
\text{subject to:} \; & \sum_{j=1}^{n} T_j x_{ij} \le C_i, \forall s_i \in V \\
& \sum_{i=1}^{m} x_{ij} \le 1, \forall t_j \in D \\
& f_j(T_j) \ge (a_j - \alpha)\cdot \sum_{i=1}^{m} x_{ij}, \forall t_j \in D \\
& T_j \ge 0, \forall t_j \in D \\
& x_{ij} \in \{0,1\}, \forall s_i \in V, \forall t_j \in D
\end{aligned} \tag{4}$$

where

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

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

The formulation (4) is a mixed nonlinear integer programming problem, which is known to be NP-hard [7]. 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 the task assignment efficiently.

D. 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 cross-layer task accuracy information is maintained. After this transformation, we propose a two-stage heuristic to efficiently generate the task assignment with negligible performance loss.

1) Resource Estimation: Suppose all the tasks require a fixed number of TCAM entries, 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 task accuracy at minimum resource usage. The compression ratio r can be obtained from the requirement of task assignment which is the second stage of the heuristic. The

Algorithm 1 Resource Compression Algorithm

```
function RESOURCECOMPRESSION($T, A, C, \epsilon, g$)
    ▷ $T$: task set; $A$: expected accuracy vector; $C$: expected resource usage;
    ▷ $\alpha$: error tolerance factor; $g$: iteration granularity
    $optA \leftarrow A, optC \leftarrow 0, u \leftarrow \epsilon/g$
    $Init(pq)$                                        ▷ Initialize a priority queue
    for $j \leftarrow 1$ to $n$ do
        $Insert(pq, T[j], A[j])$
        $optC \leftarrow optC + f_j^{-1}(A[j])$
    end for
    while $optC > C$ do
        $t \leftarrow Pop(pq)$                   ▷ Pop the task with the maximum accuracy
        $r \leftarrow f_t^{-1}(optA[t]), optA[t] \leftarrow optA[t] - u$
        if $optA[t] < A[t] - \alpha$ then        ▷ The task accuracy is below the lower bound
            $optA[t] \leftarrow optA[t] + u$; continue
        end if
        $Update(pq, t, opt[t])$
        $optC \leftarrow optC - (r - f_t^{-1}(optA[t]))$
    end while
    return $optA, optC$
end function
```

resource compression problem can be formulated as:

$$\begin{aligned} &\max \sum_{j=1}^{n} f_j(T_j) \\ &\text{subject to:} \sum_{j=1}^{n} T_j \leq r \cdot \sum_{i=1}^{n} C_i \\ &f_j(T_j) \geq a_j - \alpha, \forall j \in N \\ &T_j \geq 0, \forall t_j \in D \end{aligned} \tag{5}$$

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 priority queues to assist finding the task with the highest accuracy efficiently. 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.

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 will not exceed the available number of TCAM entries. Therefore, the task assignment problem can be formulated as an integer linear programming problem:

$$\begin{aligned} &\max \sum_{i=1}^{m} \sum_{j=1}^{n} f_j(T_j) \cdot x_{ij} \\ &\text{subject to:} \sum_{j=1}^{n} T_j x_{ij} \leq C_i, \forall s_i \in V \\ &\sum_{i=1}^{m} x_{ij} \leq 1, \forall t_j \in D \\ &x_{ij} \in \{0, 1\}, \forall s_i \in V, t_j \in D \end{aligned} \tag{6}$$

The problem formulation (6) is a variant of generalized assignment problem (GAP) which is shown to be NP-hard [7]. Therefore, heuristic algorithm is needed to efficiently find an assignment. A straightforward approach is a naive greedy algorithm. For each task, we select a switch belongs to its eligible set and has enough TCAM entries to accept this task. We assign the task to this switch and update its available resource in the management module. We refer to this approach as "first fit" algorithm (FF).

Algorithm 2 Approximation Algorithm

```
function APPROXIMATE($T, L$)
    $A \leftarrow \{-1, -1, \ldots, -1\}$                ▷ Initialize the assignment array
    for $j \leftarrow 1$ to $m$ do
        for $i \leftarrow 1$ to $n$ do
            if $A[i] = -1$ then
                $V_j[i] = v_{ij}$
            else if $A[i] = k$ then
                $V_j[i] = v_{ij} - v_{ik}$
            end if
        end for
        ▷ $Q$ stores the knapsack problem assignment
        $Q \leftarrow$ APPROXIMATEKNAPSACK($V_j, L$)
        ▷ Iteratively update the GAP assignment by $Q$
        for $i \leftarrow 1$ to $n$ do
            if $Q[i]! = -1$ then
                $A[i] \leftarrow Q[i]$
            end if
        end for
    end for
    return $A$
end function
```

Although the first fit algorithm can provide a feasible solution for GAP, it has no performance guarantee. Also, the generated assignment has bad load balancing and leads to sub-optimal solution. We try to improve the first fit algorithm by iteratively updating the profits and assignments. Suppose we find the assignment for the first switch s_1, which is a 0-1 knapsack problem. Before processing the remaining switches s_2 to s_m, 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 the 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. Note that this process corresponds to an algorithm shown in Algorithm 2. 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, the approximation of this algorithm is shown to be $(1 + \alpha)$ [8], where α is the approximation ratio to solve the knapsack problem.

III. Evaluation

In this section, we evaluate and analyze the performance of COSTA from different aspects. 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. We use Erdős-Rényi graph [9] ($p = 0.05$) which is a widely-used random graph in network research as the network topology. The graph consists of 100 switches. We set the number of TCAM entries for each switch as 1024 which is the same as the prior work [4], [2]. The task accuracy is generated with a normal distribution $(0.95, 0.2)$.

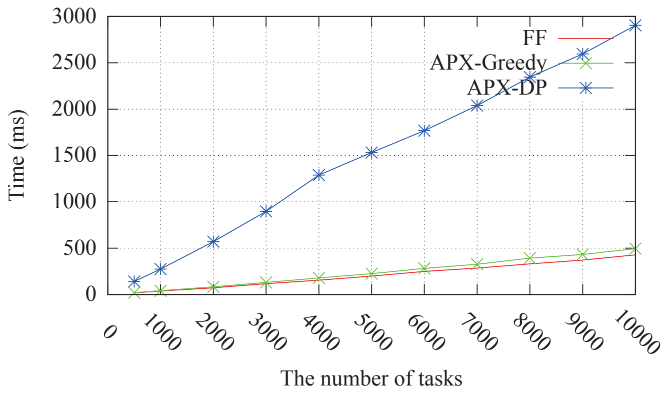

Fig. 2. Comparison of the running time for different algorithms with resource compression (the compression ratio 0.7).

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

TABLE II
THE PERFORMANCE COMPARISON OF THE TWO-STAGE HEURISTIC AND THE OPTIMAL SOLUTION FOR A SMALL INSTANCE.

We generate the source and destination hosts in a uniform random manner and use the shortest path as their forwarding paths. The workload consists of three types of tasks with the same proportion (33.3%).

A. 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 MNILP by a widely-used MNILP solver OpenOpt [10]. We refer to this solution as optimal. The test instance consists of 4 switches and 3 different types of tasks as mentioned in Section II. We use a compression ratio of 0.9 to reduce the resource usage. Table II 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 this is only a simple instance, it is not scalable to solve the MNILP in a short time. However, 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 MNILP solver searches the solutions iteratively and may lead to a sub-optimal result. In our experiments, 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.

B. Efficiency

To explore the overhead of the resource compression, we show the running time in Figure 2 as the number of tasks varies. Clearly, the running time for all the algorithms is almost linear in the number of tasks. In practice, the FF and the APX-Greedy algorithms are preferable since the measurement tasks are changing from time to time. These two algorithms are scalable because they generate the assignment within 0.5s even for a huge number of tasks.

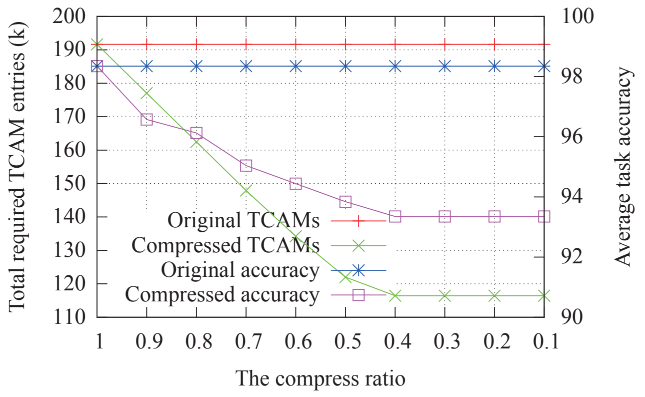

Fig. 3. The performance of resource compression with 8,000 tasks.

C. Effectiveness of Resource Compression

We examine the performance of the resource compression algorithm in Figure 3. The error tolerance factor α 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 meets the expected task accuracy. However, when the compression ratio drops below 0.4, 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 is able to reduce the resource usage by up to 40%.

D. Accept Ratio

We define the task accept ratio as the number of accepted tasks over the total number of tasks. We explore how the resource compression ratio affects the accept ratio in Figure 4, which has the same parameters as in Figure 3. We clearly observe that the accept ratio for all the algorithms raises as the resource compression ratio decreases. Specifically, after the resource compression, the FF raises the accept ratio by roughly 30%. The reason is that the FF utilizes a greedy strategy instead of iteratively updating the assignment, resource compression stage greatly helps COSTA to accept more tasks as there are more available resource.

In Figure 5 we show the relation between the number of tasks and the accept ratio with a resource compression ratio of 0.7. We clearly observe that the APX-Greedy and the APX-DP perform uniformly better than the FF as the number of tasks increases. The APX-Greedy and the APX-DP accepts 25% 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.

E. Task Accuracy

We use packet traces collected from a university data center [11] to verify the task accuracy in the measurement application layer. We case study the accuracy of an accepted unique IP counting task in Figure 6. The figure shows the actual and the measured number of unique IPs for a six minute traces. The required accuracy is 90%. The query interval for the number of unique IPs is ten seconds. Clearly, the measured number of unique IPs is closely follow the actual number of

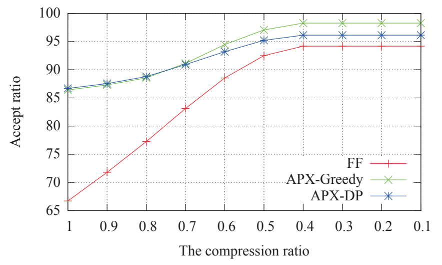

Fig. 4. The accept ratio of varying the compression ratio.

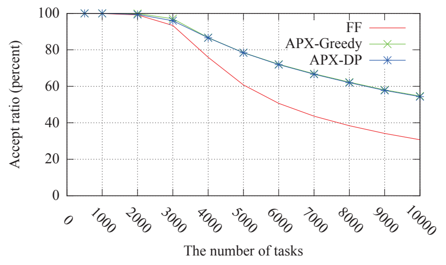

Fig. 5. The accept ratio of varying the number of tasks (the compression ratio 0.7).

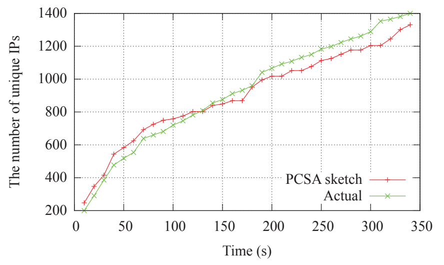

Fig. 6. The result of a unique IP counting task (with 11 TCAM entries).

unique IPs. The average accuracy is 93.1% which is above the required accuracy.

In summary, we verify that our two-stage heuristic provides near-optimal solution and it is scalable to large scale networks. The APX-Greedy 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 accept 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 COSTA guarantees the expected accuracy.

IV. Related Work

Resource allocation for software defined measurement has been extensively studied recently. OpenSketch [1] proposed a three-stage pipeline to implement a variety of sketch-based streaming algorithms for network monitoring. Detailed analysis of the trade-off between the accuracy and the resource usage was presented in [2]. Furthermore, DREAM [4] dynamically allocate resource to TCAM-based measurement tasks across multiple switches. COSTA is orthogonal to these existing proposals as we concentrate on the task assignment of sketch-based software defined measurement. In particular, we propose a novel algorithm to shrink measurement resource usage to accommodate more tasks.

This paper is also inspired by the design of low-cost high-accuracy software defined measurement frameworks. OpenTM [12] explored different query strategies to collect flow statistics from the switches. FlowCover [13] optimized the bandwidth consumption of flow statistics collection. PayLess [14] proposed APIs to collect the flow statistics using adaptive polling frequency. Furthermore, FlowSense [15] employed passive measurement techniques to infer the network utilization with zero measurement cost. OpenWatch [16] proposed a prediction-based dynamic adjustment scheme for anomaly detections. Our work is built on top of these proposals and globally optimize the measurement task assignment across the application and the management layers.

V. Conclusion and Future Work

In this paper, we present COSTA, a novel cross-layer task assignment system for software defined measurement. 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 nonlinear integer programming problem. Due to its high computational complexity, we propose a two-stage heuristic to efficiently produce near-optimal task assignment. Specifically, the algorithm compresses the resource usage in the first stage and generates task assignments with an approximation ratio of $(1+\alpha)$ in the second stage. The performance of COSTA is verified by simulations with different types of tasks and real-world packet traces. Experimental results demonstrate that COSTA significantly reduces the resource usage and increases the task accept ratio by up to 30% with negligible loss in accuracy.

Acknowledgments

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

References

[1] M. Yu, L. Jose, and R. Miao, “Software defined traffic measurement with OpenSketch,” in *NSDI*, 2013.
[2] M. Moshref, M. Yu, and R. Govindan, “Resource/accuracy tradeoffs in software-defined measurement,” in *HotSDN*, 2013.
[3] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: scaling flow management for high-performance networks,” in *SIGCOMM*, 2011.
[4] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, “DREAM: Dynamic resource allocation for software-defined measurement,” in *SIGCOMM*, 2014.
[5] P. Flajolet and G. Nigel Martin, “Probabilistic counting algorithms for data base applications,” *Journal of Computer and System Sciences*, 1985.
[6] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *J. Algorithms*, 2005.
[7] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.
[8] R. Cohen, L. Katzir, and D. Raz, “An efficient approximation for the generalized assignment problem,” *Inf. Process. Lett.*, 2006.
[9] B. Bollobás, *Random graphs*. Academic Press, 1985.
[10] “Openopt,” http://www.openopt.org/.
[11] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in *IMC*, 2010.
[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, “OpenTM: traffic matrix estimator for OpenFlow networks,” in *PAM*, 2010.
[13] Z. Su, T. Wang, Y. Xia, and M. Hamdi, “Flowcover: Low-cost flow monitoring scheme in software defined networks,” in *GLOBECOM*, 2014.
[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, “PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks,” in *NOMS*, 2014.
[15] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, “FlowSense: monitoring network utilization with zero measurement cost,” in *PAM*, 2013.
[16] Y. Zhang, “An adaptive flow counting method for anomaly detection in sdn,” in *CoNEXT*, 2013.