COMIC ALGORITHM: COMPRESSED SENSING MEETS COUNT-MIN APPLIED TO HEAVY HITTER DETECTION

Symeon Chouvardas, Moez Draief

Noah's Ark Lab, Huawei
Huawei France R&D,
Paris, France,
simos.chouvardas@gmail.com, moez.draief@huawei.com

ABSTRACT

In this paper, we are concerned with the problem of network monitoring and more specifically the task of Heavy Hitter (HH) detection and flow size estimation. We propose an algorithm, which is based on the interaction between two different entities, namely the controller and the switches. The controller, which has a centralized view of the network and has access to aggregate information, is performing an estimation of the flow sizes by exploiting an adaptive compressed sensing technique. On top of that, a properly modified sketch based algorithm that takes into account the information coming from the first stage is deployed on the switches. The theoretical properties of the algorithm are discussed and numerical experiments using both synthetic and real network data verify that the proposed algorithm outperforms the existing state of the art approach.

Index Terms— Count-Min, Compressed Sensing, SDN

1. INTRODUCTION

Software-Defined Networking (SDN) has radically transformed the landscape in several fields of networking, such as the network architecture of data centers, carrier networks, etc. The SDN architecture provides programmable data planes that can be configured from a centralized controller platform. This engenders a separation between control and forwarding planes, thus creating an opportunity to implement routing processes that are more efficient than classic ones: in fact, the controller can take real-time decisions at a (logically) centralized location using an accurate and global view of the network.

The task of *network monitoring* is of paramount importance in SDN as it plays an important role in crucial problems, such as: load balancing, network routing, flow splitting, capacity expansion, etc, e.g., [4, 5, 15, 10]. At the heart of network monitoring lies the acquisition and storage of accurate measurement of the traffic that passes through the switches of the network. Nevertheless, as the traffic increases, the number of packets and flows (aggregate of packets) that exist in the network, becomes huge and storing all the available information turns out to be impractical and in many scenarios infeasible. For that reason, various sampling and compression techniques have been utilized in SDN. The most typical one is the packet sampling approach, which samples one packet in a stream of N packets. This technique is also adopted in the widely used Cisco NetFlow protocol, [1], for flow measurement. Despite its popularity, the packet sampling approach has an important drawback; if the frequency of sampling is low, then we lose a lot of information, whereas if the frequency is high, we need high memory resources to store it. To that direction, more sophisticated monitoring techniques have been the focus of research for the past years, e.g., [7, 18]

One of the most important and challenging problems in network monitoring is that of Heavy Hitter (HH) or elephant flow detection. The term HH is a general term, which has been used in the data stream community, e.g., [6, 17, 9], and stands for " an element whose frequency in a data set is no smaller than a user–supplied threshold", [8]. The problem of HH identification lays important roles in tasks, such as, traffic accounting, traffic engineering, load balancing, anomaly detection. However, at the same time, it is challenging due to exploding traffic volume and the limited monitoring resources, e.g., memory, control plane capacity. In this paper, we are concerned with the problem of *accurate* and *timely* HH detection having at our disposal limited resources.

1.1. State of the Art

Several papers in the SDN related literature, have studied the problem of HH detection. For example, the paper of [11] proposes iSTAMP, which is a method located in the controller, which estimates HHs by using Comperessed Sensing and Multiarmed Bandits approaches. Sketch based techniques, which identify the HHs by implementing Count-Min sketches in the switches, have been proposed in [20, 14]. A TCAM-based software-defined measurement technique, has been developed in [16] where the resources are dynamically updated with respect to the traffic situation.

1.2. Summary of our Contribution

In this paper we propose an algorithm for network monitoring and in particular for HH detection and flow size estimation. The algorithm is based on the interaction between two different components, the controller and the switches and it comprises two steps. In the first one, the controller, which has a centralized view of the network and has access to aggregate information, is performing an estimation of the flow sizes based on an adaptive compressed sensing technique on a small subset of the data, which arrive in a streaming fashion. In the second stage, we employ a properly modified sketch based algorithm, which takes into account the information coming from the first stage. In brief, the flows, which are identified as HHs on the previous step are monitored directly, instead of being put in the sketch, whereas the rest are normally inserted in the Count-Min sketch. Experiments using both synthetic and real data validate that the proposed algorithm outperforms existing sketching approaches in terms

Symeon Chouvardas is currently a Research Associate at Capital Fund Management, 23 rue de l'Université, 75007 Paris

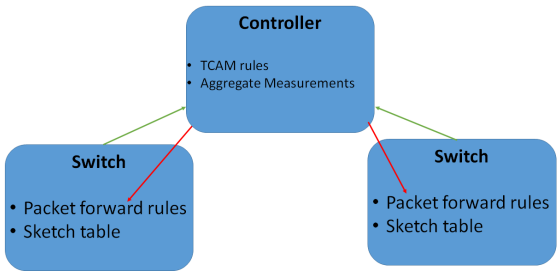

Fig. 1. System architecture under consideration

of HH identification as well as HH estimation. Finally, the algorithm is provably better than the ordinary Count-Min, since our theoretical study suggests that it leads to a tighter error bound

2. SYSTEM ARCHITECTURE

Figure (1) shows the general architecture of the system under consideration where the following entities coexist:

- **packets:** are the smallest units carrying data with respect to specific protocols, e.g., TCP/IP; packets contain parts of the message/file.
- **flows:** correspond to aggregates of packets from a specific source to a specific destination. Typical examples of flows include, youtube videos, data transfers, etc.
- **centralized controller:** an entity which has a view of aggregate information on the links of the network. The controller has a limited budget of TCAM (Ternary Content–Addressable Memory) entries that can be allocated. A TCAM entry is installed on the controller and is associated to one or more switches of the network. Each TCAM entry can be dedicated to aggregate specific flows, e.g., to aggregate all flows that have a specific prefix, or can measure explicitly a specific flow of the network.
- **switches**: receive packets as an input and their task is to process and forward them to the destination device. The routing of the packets corresponding to specific flows is determined by the TCAM installed on them.

We consider the set $\mathcal{S}$ to be a steam of items, which correspond here to flows, from a universe of n elements. To be more specific, at each instance, say t, a packet which belongs to the item (flow) k arrives, and its size will be denoted by $a_k(t)$. The total size of flow k, denoted by x_k that has crossed the switch during a certain time-window $[0, T]$ is denoted by $x_k = \sum_{t\in[0,T]} a_k(t)$, where T is our time horizon.

2.1. Count-Min Sketch

Count-Min (CM) algorithm has been originally proposed in [9] as a low complexity approach to estimate the number of occurrences of events of the same type over a stream of data. Count-Min accomplishes such task via a single pass over the whole data stream. Naturally, Count-Min can be employed to estimate the number of packets belonging to each flow passing through a network switch over a certain time window (i.e., the flow rate).

Let us now describe the details of Count-Min algorithm. Count-Min relies on a counter table CM of dimension $d\times w$ that is updated each time a packet (or a bunch of packets belonging to the same flow) is examined. Typically $dw \ll n$, where n is the number of all possible flows. This implies that the memory utilized by Count-Min is much smaller than the memory required to store an exhaustive counter vector with one element exclusively allocated for each flow. Let $h_1, \ldots, h_d : \{1, \ldots, n\} \to \{1, \ldots, w\}$ be d hash functions chosen randomly from a pair-wise independent family. When $a_k(t)$ consecutive packets belonging to flow $k \in \{1, \ldots, n\}$ are seen passing through the switch at time t, the counters c associated to the CM table elements $(1, h_1(k)), \ldots, (d, h_d(k))$ are incremented by $a_k(t)$, i.e., $c(i, h_i(k)) \leftarrow c(i, h_i(k)) + a_k(t), \quad i = 1, \ldots, d$. The frequency f_k is estimated by Count-Min as $\widehat{x}_k = \min_{1\le i\le d} c(i, h_i(k))$. It is already clear that $\widehat{x}_k$ is an overestimation of the real value x_k, since it suffers from hash collisions from other flows. The following Theorem clarifies the memory/accuracy trade-off underlying Count-Min.

Theorem 1. *[9] For each flow k, the estimated frequency is an overestimation of the real frequency, i.e., $\widehat{x}_k \ge x_k$. Moreover, if $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \lceil \log \frac{1}{\delta} \rceil$ then the estimate $\widehat{x}_k$ is such that*

$$\widehat{x}_k \le x_k + \epsilon \|\boldsymbol{x}\|_1 \qquad k = 1, \ldots, n. \tag{1}$$

with probability at least $1 - \delta$.

2.2. Compressed Sensing for HH detection

As we have already mentioned, the controller (Fig. 1) has access to aggregate information. In particular, each TCAM entry determines which flows will be aggregated; the controller observes the outcome of this aggregation. Considering that the total number of TCAM entries equals to p, this can be equivalently written as:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \tag{2}$$

where each row of the matrix $\boldsymbol{A}$ corresponds to a TCAM rule indicating *which flows to aggregate*. Usually in our context, the matrix $\boldsymbol{A}$ is considered to be a *binary* matrix. Choosing the aggregation matrix gives us a lot of flexibility and, in practice, it can improve the estimation results. It is worth pointing out, that a system similar to (2), also rises if instead of the TCAM related matrix we insert the routing matrix; in that case, $\boldsymbol{y}$ is the load at the links of the network.[1]

Let us now shed some light on why this problem can be cast as a compressed sensing one. In practical scenarios, the size of the heavy hitters is significantly larger than the size of the rest flows whereas at the same time their number is significantly smaller. Recalling (2) and considering that there are K HHs on the item list, which belong to the set $\mathcal{S}$, we have:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_{\mathcal{S}} + \boldsymbol{A}\boldsymbol{x}_{\mathcal{S}^c} \approx \boldsymbol{A}\boldsymbol{x}_{\mathcal{S}}, \tag{3}$$

where $\boldsymbol{x}_{\mathcal{S}}$ denotes the vector, which consists of the HHs and has zero elsewhere and $\boldsymbol{x}_{\mathcal{S}^c}$ is the vector in which the HHs are set equal to 0. In a nutshell, this approximation yields that the HHs dominate the outcome of aggregation. Since the number of observations is smaller than the unknown entries of the vector to be estimated, the system (2) cannot be solved accurately using conventional techniques, such as the least squares method. On the contrary, compressed sensing techniques capitalize on the fact that the vector to be estimated is sparse, i.e., comprises a small number of non zero coefficients and

[1] The estimation of the flow sizes by solving the linear system (2), where the input matrix corresponds to the routing matrix, is a topic known as network tomography and has been extensively studied over the past decades, e.g., [19, 13]

Algorithm 1: Compressive Adaptive Sense and Search (CASS)

Input: number of non–zeros k.

Initialize: $l_0 = \min\{4k, n\}$, $s_0 = \log \frac{n}{l_0} + 1$,

$\mathcal{L} = \{1, 2, \ldots, l_0\}$

for $s = 1$ *to* s_0 **do**

  for $l \in \mathcal{L}$ **do**

    $\boldsymbol{a}_{s,l} = \mathbf{1}_{\mathcal{J}_{\log(l_0)-1+s,l}}$;

    sense: $y_{s,l} = \boldsymbol{a}_{s,l}^T \boldsymbol{x}$

  sort: $l_1,\ l_2, \ldots, l_{|\mathcal{L}|}$ s.t. ;

  $|y_{s,l_1}| \geq |y_{s,l_2}| \ldots \geq |y_{s,l_{|\mathcal{L}|}}|$;

  if $s \neq s_0$ **then**

    $\mathcal{L} = \{2l_1 - 1, 2l_1, 2l_2 - 1, \ldots, 2l_k - 1, 2l_k\}$;

  else

    $\hat{\mathcal{S}} = \{l_1, l_2, \ldots, l_k\}$;

    $\hat{x}_i = y_{s,i},\ \forall i \in \hat{\mathcal{S}}$

Output: $\hat{\mathcal{S}},\ \hat{x}_i$

Algorithm 2: CoMiC algorithm for HH detection

Input: observation ratio for CASS: α, number of HHs k, Count-Min table dimensions: $d,\ w$.

Initialize: $l_0 = \min\{4k, n\}$, $s_0 = \log \frac{n}{l_0} + 1$

$\mathcal{L} = \{1, 2, \ldots, l_0\}, d, w$

while $i \leq N\alpha$ **do**

  run CaSS with k as an input to

return $\hat{\mathcal{S}}$

while $i \geq N * \alpha$ **do**

  if $k \notin \hat{\mathcal{A}}$ **then**

    Add item k to CM table

  else

    Add it in the queue monitor it directly

Output: Indices and size of HHs

can produce very accurate estimates. A typical formulation of the CS problem can be cast as follows: $\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \lambda\|\boldsymbol{x}\|_1$, where λ is a properly defined regularization parameter and the ℓ_1 norm promotes sparsity.

Turning our focus back to our problem, since we can choose the rows of the input matrix $\boldsymbol{A}$ (by assigning TCAM rules) then it is better to choose the rows in a sequential fashion, i.e., fix a row, observe the output of the system and then decide on the next one, instead of using an a-priori chosen fixed matrix. This is the main idea of the *adaptive compressed sensing* problem, e.g., [12]. The unknown vector of flows is measured through p linear projections of the form:

$$y_i = \boldsymbol{a}_i^T \boldsymbol{x},\ i = 1, \ldots, p \tag{4}$$

An adaptive compressed sensing algorithm, which is named Compressive Adaptive Sense and Search (CASS), has been proposed in [12]. This algorithm lends itself nicely to our framework, since it considers solving the compressed sensing problem in the setup where the non-zero unknown entries are strictly positive, which is the case here. The algorithm is summarized in Table 1.

The algorithm takes as an input the number of non-zeros of $\boldsymbol{x}$, which corresponds here to the number of HHs. It is worth mentioning, that it is not necessary to know this number *exactly*, but a rough estimate of it instead. As we will see in the simulations section the algorithm is robust to overestimates of the actual number of HHs. The TCAM rules, which are assigned sequentially, take the form of vectors of length n comprising ones and zeros depending on whether the respective flows will be aggregated or not. We define the following sets: $\mathcal{J}_{k,l} = \left\{\frac{(l-1)n}{2^k} + 1, \ldots, \frac{ln}{2^k}\right\}, k = 0, 1, \ldots, \log n,\ l = 1, \ldots, 2^k$. For a fixed k, initially the unknown vector is measured with l_0 TCAM entries, each with support over a single dyadic subinterval. Afterwards, the largest k measurements in absolute value are selected; these define the support. At the next steps, the supports of the sensing vectors corresponding to the k largest measurements are bisected, giving $2k$ support sets. These $2k$ support sets define the support of the sensing vectors on step $s = 2$. The procedure continues sequentially taking $2k$ measurements on each step and bisecting the support of the k largest measurements. At the final step, the k largest elements together with their values are returned. For more details the reader is referred to [12].

3. THE COMIC ALGORITHM

So far techniques that have been proposed in the literature try to estimate the sizes of the flows and/or the HHs by developing algorithms which exploit information coming from either the controller or the switches. Nevertheless, these two units can collaborate and one can pass information to the other. Here, we develop an algorithm that can facilitate this interaction between the controller and the switches. Our algorithm processes in two steps. In the first step the controller gathers aggregate information coming from the TCAM rules. Then, the CASS algorithm is applied and we obtain a first estimate of the HHs; the set of identified HHs, will be denoted by $\widehat{\mathcal{S}}$. The size of the data that will be used in this step is determined by a user defined parameter $\alpha \in [0, 1]$; in the CASS step αN packets are utilized, where N is the total number of packets. This is the set of *potential HHs*. After the estimation of $\widehat{\mathcal{S}}$ we continue by inserting information into our Count-Min sketches. Nevertheless, a flow is inserted in the sketch only if it doesn't belong to the set $\widehat{\mathcal{S}}$. On the contrary, items that belong to $\widehat{\mathcal{S}}$ are inserted to a special queue of size k. In this queue, there is an cell per flow so that no collisions exist and the size of the potential HHs is directly measured. The steps of the CoMiC algorithm are detailed in Table 2.

The reasoning behind the algorithm is the following. We are interested in estimating the size of HH flows as accurately as possible. From a practical perspective this is important because HHs need a special treatment, e.g., alteration in their routing decisions, assignment of specific TCAM rules for them, etc. From a theoretical perspective, as it can be seen from (1), the HHs dominate the term $\|\boldsymbol{x}\|_1$ and if one discards them from the sketch, as we do here by putting them in the queue, then the upper bound of the error can be significantly reduced. In order to discard them, we first use the CASS algorithm on a subset of our data so as to estimate the set of potential HHs. Roughly speaking, this gives us a prior knowledge about the flows and when we construct our sketch we take this knowledge into account. The performance improvement that comes from this prior knowledge compared to the vanilla sketching algorithm, will be justified in the next sections both theoretically and practically.

Remark 1. *It is worth pointing out that there is an underlying assumption in our proposed algorithm. In particular, we assume that during the first stage, i.e., the stage of collection and aggregation, the set of HHs is the same with the set of HHs at the second stage,*

i.e., the stage where we build our sketch. Experiments using real data validate that this assumption holds in practice

Remark 2. *The a-priori knowledge, that comes from the use of the CaSS algorithm on a subset of our data, is beneficial for our algorithm. To be more specific, under certain assumption with respect to the flow size it can be shown that Denoting by $\hat{x}_i$ the estimate of the i–th item, it holds that:*

$$x_i \leq \hat{x}_i \leq x_i + \epsilon^{'}\|\boldsymbol{x}\|_1, \tag{5}$$

where $\epsilon^{'}$ is strictly smaller than the ϵ occurring by the Count-Min algorithm

4. NUMERICAL RESULTS

In this section, we will evaluate the performance of the CoMiC algorithm using both synthetic and real–world data. We compare the proposed algorithm, with the vanilla Count-Min algorithm.

4.1. Synthetic Data

In this set of experiments, the data are generated with respect to the power law distribution. This distribution approximates reasonably well the internet traffic, e.g., [3]. The algorithms will be tested in scenarios with different distributions, which stem from different parametrizations. More specifically, the probability for a packet belonging to flow i is $\frac{1}{(i+q)^{\beta}}$, where q gives a starting point and β is a parameter that determines the tail of the distribution. We will conduct different experiments for different values of β and q. In total, there are 10000 packets belonging to 100 flows. We set ϕ equal to 0.02. Regarding the Count-Min sketch, there are 5 hash functions, each of length 11, hence a total size of 55. It is obvious that the compression rate equals to $\frac{55}{100} \approx \frac{1}{2}$. Regarding the CoMiC algorithm we have the following. The portion of data related parameter equals to $\alpha = 0.3$ and we use the CASS having as an input a value of k equal to $2K$, i.e., twice the size of the true HHs.

In the first experiment, we set $q = 1$, $\beta = 1$; the histogram of the flow sizes is give in Fig. 2. This distribution approximates well the network traffic, yet HH detection using it becomes more challenging, because there are many flows, the size of which is smaller yet comparable to that of the HHs. So if there are collisions between these flows in the same bucket then these will be falsely reported as HHs. We present the following metric: $\frac{\#\text{EstimatedHHs}}{\#\text{HHs}}$, i.e., the number of identified HHs divided by the actual number of HHs using the CoMiC algorithm and the Count-Min. We also present the distance of the estimated HHs from the actual ones, i.e., $\|\widehat{\boldsymbol{x}}_{\mathcal{S}} - \boldsymbol{x}_{\mathcal{S}}\|$.

1. It can be readily observed that the Count-Min fails to identify

Metric	CoMiC	CM
$\#\text{EstimatedHHs}/\#\text{HHs}$	1.55	10.89
$\|\widehat{\boldsymbol{x}}_{\mathcal{S}} - \boldsymbol{x}_{\mathcal{S}}\|$	0	1063

Table 1. Results for the first experiment

and estimate the size of the HHs, due to the shape of the flow size distribution. On the contrary, using the CoMiC algorithm the number of false positives is significantly reduced and the size of the correctly identified HHs is perfectly estimated.

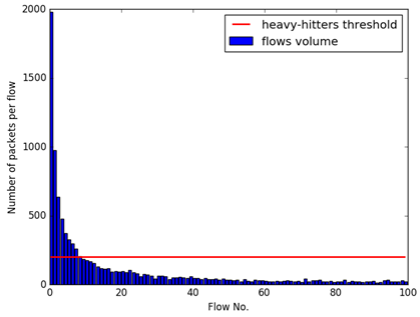

Fig. 2. Flow size histogram for the first experiment

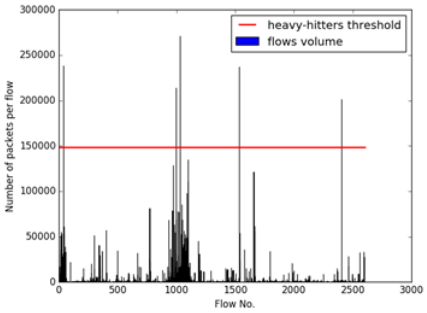

Fig. 3. Flow sizes for the Caida data

4.2. Real Data

In this section we are going to evaluate the performance of the algorithms on a real data scenario. In particular, we will apply our algorithm on the CAIDA UCSD Anonymized Passive OC48 Internet Traces Dataset, [2]. In this scenario a total number of 2606 flows is considered. The flow sizes are illustrated in Fig. 3. The dimension of the sketch table in both the algorithms equals to 20×50 so we have in total 1000 buckets, leading to a compression rate $\approx 2/5$. For the CoMiC algorithm we fix $\alpha = 0.3$ and the number of HHs utilized in the first step equal to $2K$. Finally, ϕ is set equal to 0.02, leading to $K = 5$. The vector of sizes of the actual HHs equals to $\boldsymbol{x}_{\mathcal{S}} = [271060,\ 238489,\ 237636,\ 214125,\ 201536]^T$ and the estimated one given by the Count-Min is: $\widehat{\boldsymbol{x}}_{\mathcal{S}} = [315512,\ 285488,\ 270763,\ 240524,\ 236395]^T$ and the estimated values from the CoMiC algorithm are: $\widehat{\boldsymbol{x}}_{\mathcal{S}} = [282682,\ 238489,\ 249977,\ 214125,\ 201536]^T$. The results are presented in Table 2. It can be seen that in this experiment

Metric	CoMiC	CM
$\#\text{EstimatedHHs}/\#\text{HHs}$	1.6	1.8
$\|\widehat{\boldsymbol{x}}_{\mathcal{S}} - \boldsymbol{x}_{\mathcal{S}}\|$	84819	16952

Table 2. Results for the Caida experiment

the CoMiC algorithm slightly outperforms the Count-Min one in terms of false positives. Moreover, the proposed scheme leads to a significantly improved estimation error. Nevertheless, it should be mentioned that compared to the previous examples, in which the estimation error was zero, here it takes a relatively high value.

5. REFERENCES

[1] www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html.

[2] http://www.caida.org/data/passive/passive_oc48_dataset.xml.

[3] L. A. Adamic and B. A. Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 8. ACM, 2011.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[7] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 464–475. VLDB Endowment, 2003.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] G. Dittmann and A. Herkersdorf. Network processor load balancing for high-speed links. In *Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, volume 735. July, 2002.

[11] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma. Intelligent SDN based traffic (de) Aggregation and Measurement Paradigm (iSTAMP). In *INFOCOM, 2014 Proceedings IEEE*, pages 934–942. IEEE, 2014.

[12] M. L. Malloy and R. D. Nowak. Near-optimal adaptive compressed sensing. *IEEE Transactions on Information Theory*, 60(7):4001–4012, 2014.

[13] M. Mardani, G. Mateos, and G. B. Giannakis. Dynamic anomalography: Tracking network anomalies via sparsity and low rank. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):50–66, 2013.

[14] R. G. A. V. Masoud Moshref, Minlan Yu. SCREAM: Sketch Resource Allocation for Software-defined Measurement. In *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, December 2015.

[15] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris. Global optimization for hash-based splitting. 2016.

[16] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. DREAM: Dynamic resource allocation for software-defined measurement. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 419–430. ACM, 2014.

[17] S. Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[18] S. Shin and G. Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[19] Y. Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American statistical association*, 91(433):365–377, 1996.

[20] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, 2013.