Traffic Estimation via Kalman Filtering under Partial Information in Software-Defined Networks

Alper Kamil Bozkurt and Gokcan Cantali
Dept. of Computer Engineering, Bogazici University
Istanbul, Turkey
alper.bozkurt@boun.edu.tr
gokcan.cantali@boun.edu.tr

Gürkan Gür
TETAM, Bogazici University
Dept. of Computer Engineering, Bogazici University
Istanbul, Turkey
gurgurka@boun.edu.tr

ABSTRACT

An accurate traffic matrix (TM) is essential for network design, management and optimization. Software-defined networking (SDN) provides flow level statistics with global centralized control which enables construction of more accurate traffic matrices. However, retrieving all the flow statistics can cause a very significant overhead in the system. In this work, we propose an inference framework which utilizes Kalman filtering to create an accurate and timely traffic matrix. In our scheme, only a small number of flow statistics are measured at a time, yet the estimate of the TM is highly accurate. Besides, we propose a switch selection strategy which aims to minimize the entropy of the estimate, that is to maximize the information obtained by the estimation. Using simulation-based experiments, we show that our framework provides a very accurate TM estimate compared to the one constructed by using all the flow statistics in the network.

CCS CONCEPTS

• **Networks → Network measurement**; **Network monitoring**; *Control path algorithms*; *Programmable networks*;

KEYWORDS

Network traffic estimation, network monitoring, Kalman filtering, software-defined networking

ACM Reference Format:
Alper Kamil Bozkurt and Gokcan Cantali and Gürkan Gür. 2017. Traffic Estimation via Kalman Filtering under Partial Information in Software-Defined Networks. In *AINTEC '17: AINTEC '17: Asian Internet Engineering Conference , November 20–22, 2017, Bangkok, Thailand.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3154970.3154977

1 INTRODUCTION

A traffic matrix (TM) stores all the flow rates for each pair of source and destination nodes in a network. Constructing a TM for a network is an active research topic since the emergence of data and communication networks. This interest mostly stems from the fact that an accurate TM is the key for solving many problems such as mitigating network congestion and anomaly detection. In addition, having the knowledge of an accurate network TM significantly eases the process of performance measurement and traffic engineering. For such reasons many researchers, especially network engineers, have a great interest in the methods of obtaining accurate TM information with reasonable overhead.

The Software-Defined Networking (SDN) paradigm [8] provides useful methods for obtaining more network information regarding TM. Using SDN, the number of packets matching specific rules can be stored in network devices. One such rule might consist of source IP, destination IP, source port, and destination port tuples. As a result, with such rules, specific flow attributes can be calculated or accurately estimated for a time interval. In SDN, switches can be queried for flow statistics, that is, the number of packets which have passed through the queried switch up to that time point. Moreover, matching rules can be added or removed by the controller via a protocol such as OpenFlow.

Even though SDN can greatly ease the process of obtaining flow information, a computational challenge still exists. To construct an accurate TM, one needs to observe nearly all network devices to obtain each flow information. However, querying so many switches may impose a communication and computation overhead and such approach is not scalable for large networks. Another method would be to add the necessary flow rules to a few number of switches and periodically query them to obtain statistics regarding TM. Nevertheless, such approach has its flaws too. The switches can only capture the flows passing through them and repetitively querying the same switches could disrupt their operation. Furthermore, the number of rules matching the flows is usually too large to be stored in valuable and limited Ternary Content Addressable Memory (TCAM). Given these conditions, a reasonable approach could be querying a small subset of switches which is changed over time without introducing new flow table rules.

In this paper, we propose a framework to infer the network traffic matrix using only the available partial information. Our approach utilizes Kalman filtering for constructing an accurate TM. We consider time-varying property of TMs and estimate the current TM by using our previous measurements. We also provide an entropy-based switch selection strategy for the measurement part. The goal of our selection strategy is to find the switch subset that maximizes the certainty of the TM computed by the Kalman filter. We show that our method estimates a TM from partial measurements accurately compared to the full measurement case.

The rest of the paper is organized as follows: Section 2 gives an overview of network state estimation techniques and approaches.

ACM ISBN 978-1-4503-5551-3/17/11…$15.00
https://doi.org/10.1145/3154970.3154977

In Section 3, we present the problem statement. Our model and methodology are explained in Section 4. In Section 5, we discuss some possible extensions and application areas of our method. Our experiment design and results are described in Section 6. Finally, Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORK

Traffic matrix (TM) estimation in operational networks is attracting a lot of research effort recently. The approaches for this task can be divided into two domains: i) traditional IP network methods and ii) SDN-based methods. In general, traditional methods make use of link loads to estimate the TM. Those loads can be retrieved via a protocol such as SNMP[2]. For modelling TM, traditional approaches define three different types of models as temporal, spatial and spatio-temporal[18]. In temporal models, the difference of flows between different time points are analyzed, such as in [14]. In spatial models, the focus is on the flow between source and destination [22]. In spatio-temporal models, both time varying and distance related properties of network traffic are considered [23], [21]. There are also machine-learning based methods for TM estimation, such as [12] which utilizes deep learning. Although the models defined in these studies are generally applicable, we can acquire much more accurate TMs by making use of SDN paradigm. Thus, we focus on the studies regarding TM estimation in SDN environment in the remaining part of this section. For more detailed information regarding traditional TM estimation methods, please refer to the study in [18].

A study where SDN is used to obtain traffic matrix is conducted by Jose *et al.* in [7]. They mainly focus on the trade-off between state accuracy and switch overhead in this work. They propose to sacrifice some of the accuracy in favor of less overhead. Another study regarding TM estimation in SDN environment is conducted by Tootoonchian et al. in [17]. The estimation system in that work is called OpenTM since it estimates the traffic matrix (TM) in OpenFlow networks. OpenTM uses a subset of switches for obtaining flow statistics. It proposes different approaches to choose the appropriate switches to query, such as *uniformly-random selection*, *non-uniform random selection*, and *least-loaded switch selection*. The results show that the strategies which give priority to the switches that are closer to the destination point perform much better compared to alternatives. On the other hand, querying based on the least-loaded switches increases the performance while causing worse state estimations.

A recent study for monitoring network utilization is FlowSense by Yu et al. [20]. FlowSense does not query the switches at predefined time intervals like OpenTM. Instead, it uses *PacketIn* and *FlowRemoved* messages to estimate link utilization. In OpenFlow, these messages are sent by the switches to the controller when a new flow starts and a flow expires, respectively. Although this approach enjoys high performance when the network is stable, it suffers in large networks. This is due to the fact that flows arrive at high rates in these systems, causing a large number of control packets which impose significant overhead.

Another study in network traffic measurement topic is conducted by Atary and Bremler-Barr in [1]. In that study, a framework called GRAMI (Granular RTT Measurement Infrastructure) is presented. GRAMI provides Round-Trip Time (RTT) measurements across any switches in the network. It uses pre-selected vintage points and turns them into monitoring points, which use active probing to measure the network traffic. Since probes are sent by such monitoring points, OpenFlow controller in the system is not used for the online monitoring function. Therefore, the overhead in the controller plane is highly reduced.

Gong et al. [5] focus on the problem of traffic matrix estimation in SDN under the network resource constraints such as TCAM entries and processing power. The authors suggest two novel strategies for generating rules that are to be installed on TCAM entries of SDN switches. One of these strategies is called Maximum Load Rule First (MLRF) which selects the rule with largest load, and creates a new rule that transfers approximately half the load of old rule to itself. The second strategy is called Large Flow First (LFF) which aims to measure the largest flows to achieve a better estimation of the traffic matrix. Based on the simulation results, these strategies seem to achieve feasible and accurate traffic estimation.

Another study regarding traffic matrix estimation using SDN is conducted by Polverini *et al.* in [13]. The study focuses on Internet Service provider (ISP) networks and uses SDN paradigm to propose a method for estimating TM in such networks. The authors make use of a concept named *flow spread*. Considering the solution space, the difference between the maximum and the minimum values of a flow is defined as the flow spread of that flow. Then, the method selects which flows are more important and should be queried based on that parameter. However, the study considers IE (Ingress-Egress) TM in the models. Thus, a method for estimating OD (Origin-Destination) TM in SDN environment is still an open problem.

3 PROBLEM STATEMENT

In this section, we construct the traffic estimation problem in a mathematical formulation for laying out the purpose of the proposed framework. Consider a network with n hosts and m switches. Let $\mathbf{x}$ be the vector representation of the TM where x^i is the rate of OD flow i. An OD flow is defined as the traffic between a pair of source and destination hosts. The length of $\mathbf{x}$ is the number of possible OD flows which is $c = n(n-1)$[1]. Let r be the total number of the rules in all flow tables and $\mathbf{y}$ denote the vector of all flow statistics corresponding to these rules. The idea here is the amount of change in the flow statistics vector $\mathbf{y}$ in a certain time interval can be described by superpositions of the elements of the TM vector $\mathbf{x}$.

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \Delta t \mathbf{A} \mathbf{x}_t \tag{1}$$

For the sake of simplicity, we henceforth assume Δt is fixed and equal to 1. We also assume that flows are not bifurcated, that is, a single flow between a source node and destination node passes through only one single path. The matrix $\mathbf{A}$ can be constructed by using the routing information and the rules in the flow tables, which are available via software-defined networking. The element A^{ij} is 1 if the OD flow j passes through the switch that the rule i belongs to and the rule i matches the flow, and 0 otherwise. We

[1]We consider a single aggregate macroflow between end-points and do not discern different microflows.

can see $\mathbf{A}$ and $\mathbf{y}$ can be easily expanded by port statistics by adding imaginary rules matching input ports of the flows.

Only a small number of elements of $\mathbf{y}$ can be measured at a time since measuring all the elements $\mathbf{y}$ is most likely to introduce impractical overhead. Let $\mathbf{z}$ be the vector of measured flow statistics calculated by the following formula,

$$\mathbf{z}_t = \mathbf{M}_t \mathbf{y}_t + \epsilon_t \tag{2}$$

Here ϵ_t is the measurement error and $\mathbf{M}$ is a binary matrix which is to say, it has a single entry of 1 in each row and each column, and 0s elsewhere. In a full measurement, $\mathbf{M}$ becomes a $r \times r$ permutation matrix. Our first goal is to compute the joint posterior distribution, $p(\mathbf{x}_t, \mathbf{y}_t | ,\mathbf{z}_{1:t}, \mathbf{M}_{1:t})$ of the flow statistics and the underlying OD flows given all the previous measurements. Here we use the notation $a_{1:t} := \{a_1, \ldots, a_t\}$.

Now suppose we are given a set of measurement matrices $\mathbb{M}$. At each time step we are supposed to use one of the matrices in this set for measurement. Our second goal is to identify the matrix $\mathbf{M}^* \in \mathbb{M}$ to be used in the next measurement which makes the estimate the most certain. To put into a more formal manner, we want to find $\mathbf{M}_{t+1}$ which minimizes the entropy of $p(\mathbf{x}_{t+1}, \mathbf{y}_{t+1} | \mathbf{z}_{1:t}, \mathbf{M}_{1:t}, \mathbf{M}_{t+1})$,

$$\mathbf{M}^* = \underset{\mathbf{M}_{t+1}}{\operatorname{argmin}} \left\{ h(p(\mathbf{x}_{t+1}, \mathbf{y}_{t+1} | \mathbf{z}_{1:t}, \mathbf{M}_{1:t}, \mathbf{M}_{t+1})) \right\} \tag{3}$$

where $h(\cdot)$ denotes the entropy of a probability density function.

4 METHODOLOGY

Our methodology can be analyzed in three main parts: i) the system model which is used to model the network traffic and the measurements, ii) the Kalman filtering approach for traffic estimation, iii) the switch selection strategy which improves our estimations.

4.1 System Model

We model the temporal evolution of the OD flows as a hidden Markov process. That is, given the current OD flows $\mathbf{x}_t$, the future OD flows $\mathbf{x}_{t+1}$ are independent of all the previous OD flows. Additionally, we assume $\mathbf{x}_{t+1}$ is a linear combination of the current flows perturbed by a Gaussian white noise ν_t,

$$\mathbf{x}_{t+1} = \mathbf{C}\mathbf{x}_t + \nu_t \tag{4}$$

In most of the cases the OD flows are independent, therefore we can think of $\mathbf{C}$ as an identity matrix. In this case, the model becomes a Wiener process which has the self-similarity property, one of the main characteristics of flows in network traffic, as shown in [3] and [10].

Using this notation, we design a state space model (SSM) that describes the evolution of the flows and the flow statistics. We define the state vector $\mathbf{s}_t$ as the composition of $\mathbf{x}_t$ and $\mathbf{y}_t$. The state vector is the latent variable of the SSM since it cannot be measured directly. The transition matrix $\mathbf{F}$ of the SSM could be constructed by combining the matrices $\mathbf{C}$ and $\mathbf{A}$. We can illustrate $\mathbf{s}_t$ and $\mathbf{F}$ with the block matrix representations,

$$F = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{A} & \mathbf{I} \end{bmatrix}, \quad \mathbf{s}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_t \end{bmatrix} \tag{5}$$

Here $\mathbf{0}$ is a $c \times r$ zero matrix and $\mathbf{I}$ is $r \times r$ identity matrix. The state vector $\mathbf{s}_t$ evolves according to the following linear system,

$$\mathbf{s}_{t+1} = \mathbf{F}\mathbf{s}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{6}$$

The transition noise $\mathbf{w}_t$ is a zero mean Gaussian process with covariance matrix $\mathbf{Q}$. The upper left $c \times c$ part of $\mathbf{Q}$ captures the flow correlations and fluctuations. The lower right $r \times r$ part indicates the reliability of flow statistics. For instance, if a switch is prone to malfunction, the flow statistics provided by this switch are not reliable and therefore the corresponding entries in $\mathbf{Q}$ are large.

Lastly, we construct proper measurement matrices $\mathbf{H}_t$ for the SSM model. We simply add c columns of zeros to the left hand side of aforementioned $\mathbf{M}_t$ matrices,

$$\mathbf{H}_t = \begin{bmatrix} \mathbf{0} & \mathbf{M}_t \end{bmatrix} \tag{7}$$

We cannot measure the values of OD flows, that is why the left part of $\mathbf{H}_t$ consists of zeros, and the OD flows must be inferred from the noisy measurements of flow statistics,

$$\mathbf{z}_t = \mathbf{H}_t \mathbf{s}_t + \mathbf{m}_t, \quad \mathbf{m}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \tag{8}$$

We assume the measurements are obtained with an error $\mathbf{m}_t$ possibly caused by the latency or the traffic created by the controller. Nevertheless, the accuracy of the measurements are usually quite high in practical SDN. Thus, we expect that the values in the covariance matrix $\mathbf{R}$ are small.

4.2 Kalman Filtering

We now introduce our Kalman filtering approach that infers the hidden state $\mathbf{s}_t$, the OD flows and the flow statistics, from the available information $\mathbf{z}_{1:t}$. In a state space model the posterior distribution $p(\mathbf{s}_t | \mathbf{z}_{1:t})$ can be expressed by two recursive equations which are called prediction and update. The predicted prior distribution can be obtained by using the Markov property,

$$p(\mathbf{s}_t | \mathbf{z}_{1:t-1}) = \int p(\mathbf{s}_t | \mathbf{s}_{t-1}) p(\mathbf{s}_{t-1} | \mathbf{z}_{1:t-1}) \tag{9}$$

And the prior distribution can be updated via Bayes rule when $\mathbf{z}_t$ becomes available,

$$p(\mathbf{s}_t | \mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t | \mathbf{s}_t) p(\mathbf{s}_t | \mathbf{z}_{1:t-1})}{p(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \tag{10}$$

Since our model is a linear Gaussian SSM, the prediction (9) and update (10) equations linearly transform a Gaussian distribution to another one. Therefore we are able to compute the mean and the covariance matrices of $p(\mathbf{s}_t | \mathbf{z}_{1:t-1})$ and $p(\mathbf{s}_t | \mathbf{z}_{1:t})$ optimally by using the Kalman techniques. Let $\mathbf{s}_{t|t-1}$ and $\mathbf{P}_{t|t-1}$ denote the mean and the covariance matrix of the predicted prior distribution $p(\mathbf{s}_t | \mathbf{z}_{1:t-1})$ and similarly let $\mathbf{s}_{t|t}$ and $\mathbf{P}_{t|t}$ denote the mean and the covariance matrix of the updated distribution $p(\mathbf{s}_t | \mathbf{z}_{1:t})$ after the measurement. The Kalman filter calculates these variables in an online manner, i.e. it only requires the previous state estimate $\mathbf{s}_{t-1|t-1}$ and its covariance matrix $\mathbf{P}_{t-1|t-1}$. The Kalman filter [6] can be summarized as follows,

- **Prediction Phase:** The state vector and the covariance matrix are predicted by using the transition model,

$$\mathbf{s}_{t|t-1} = \mathbf{F}\mathbf{s}_{t-1|t-1} \tag{11}$$

$$\mathbf{P}_{t|t-1} = \mathbf{F}\mathbf{P}_{t-1|t-1}\mathbf{F}^T + \mathbf{Q} \tag{12}$$

- **Update Phase:** The state vector and the covariance matrix is updated by combining the prediction and the information obtained through the measurement

$$\mathbf{G}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T(\mathbf{R} + \mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T)^{-1} \tag{13}$$

$$\mathbf{s}_{t|t} = \mathbf{s}_{t|t-1} + \mathbf{G}_t(\mathbf{z}_t - \mathbf{H}_t\mathbf{s}_{t|t-1}) \tag{14}$$

$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{G}_t\mathbf{H}_t)\mathbf{P}_{t|t-1} \tag{15}$$

The basic idea behind the Kalman filter is to estimate the state as a weighted average of the predicted state and the measured state. The weights (the Kalman gain **G**) are calculated using the covariance matrices such that the more certain values have larger weights, and therefore the new state obtained by the weighted average is closer to the more trusted values. For details, please refer to [6].

4.3 Switch Selection

The Kalman filter does not require a fixed measurement matrix $\mathbf{H}_t$. Therefore, another problem is how to choose the matrix $\mathbf{H}_{t+1}$ to be used in the next measurement. Here we want to construct a $\mathbf{H}_{t+1}$ so that after measurement we obtain the most certain state estimate. Obviously measuring all the elements in the state vector, i.e. using $(c + r) \times (c + r)$ identity matrix as $\mathbf{H}_{t+1}$ gives the most certain estimate. However, querying all the switches is a costly operation, therefore we are constrained to select a small number (a subset) of switches at a time, say k switches.

Assume we have a set $\mathbb{H}$ of measurement matrices for all the combinations of the switches. Calculating the entropy for each matrix in $\mathbb{H}$ and choosing the one that minimizes the entropy is not a feasible solution due to the exponentially growing cardinality $|\mathbb{H}| = \binom{m}{k}$. Instead, we use the following greedy algorithm, namely min-entropy switch selection, to find a suboptimal solution,

Algorithm 1 Min-Entropy Switch Selection

```
$\mathbf{H}^* \leftarrow [\ ]$ # Empty Matrix
for $i = 1, 2, \ldots, k$ do
    $min\_h \leftarrow h(p(\mathbf{s}_{t+1}|\mathbf{H}^*))$
    for $j = 1, 2, \ldots, m$ do
        $\tilde{\mathbf{H}} \leftarrow \begin{bmatrix} \mathbf{H}^* \\ \mathbf{H}^{(j)} \end{bmatrix}$
        if $h(p(\mathbf{s}_{t+1}|\tilde{\mathbf{H}})) < min\_h$ then
            $\hat{\mathbf{H}} \leftarrow \tilde{\mathbf{H}}$
            $min\_h \leftarrow h(p(\mathbf{s}_{t+1}|\tilde{\mathbf{H}}))$
        end if
    end for
    $\mathbf{H}^* \leftarrow \hat{\mathbf{H}}$
end for
```

Here $\mathbf{H}^{(j)}$ is the matrix used to measure the flow statistics in the flow tables of switch j, and h is the following function that calculates the entropy of $p(\mathbf{s}_{t+1}|\tilde{\mathbf{H}})$,

$$h(p(\mathbf{s}_{t+1}|\tilde{\mathbf{H}})) = \frac{1}{2}ln(|2\pi e\mathbf{P}_{t+1|t+1}|) \tag{16}$$

where $|\cdot|$ denotes the determinant operation. We do not need the value of the measurement to calculate $\mathbf{P}_{t+1|t+1}$, therefore we can use the Kalman equations as if the state vector is measured by $\hat{\mathbf{H}}$ to obtain $\mathbf{P}_{t+1|t+1}$ as seen in Equations (13)-(15).

The Algorithm 1 constitutes a query list of m switches in m steps. In each step, it calculates the amount of decrease in the entropy to be gained for each switch when the switch is added to the query list, then it adds the best switch. We note that the algorithm tries to form a list of switches having mostly independent flow statistics. In other words, it is unlikely that the algorithm adds a switch to the list whose flow statistics can be calculated by the flow statistics of the other switches already in the list.

We can also determine the number of switches to be queried with an adaptive strategy. In this case, we start with an empty query list, and we continue to add switches to the list until the entropy decreases below a given threshold. We can make the Algorithm 1 perform this adaptive strategy by simply changing the outer `for` statement with a `while` statement whose condition is $min\_h < \tau$ where τ is the threshold.

5 DISCUSSION AND EXTENSIONS FOR KEY USE-CASES

In this section, first we discuss how the Kalman parameters can be learned by using Markov Chain Monte Carlo (MCMC) methods. Then we demonstrate how the marginal likelihood computed by the Kalman filter can be used to detect anomalies. Lastly we show how to build a robust Kalman filter in dynamic networks where topology changes over time.

5.1 Parameter Estimation

In our framework, we need to know the initial state $\mathbf{s}_{0|0}$ and its covariance $\mathbf{P}_{0|0}$ as well as the covariance matrices of the transition and the measurement models, $\mathbf{Q}$ and $\mathbf{R}$, to apply the Kalman filter. Let θ denote these unknown parameters. The posterior distribution of θ given all the measurements can be written in terms of the likelihood and the prior distributions using Bayes rule,

$$p(\theta|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_{1:t}|\theta)p(\theta)}{p(\mathbf{z}_{1:t})} \tag{17}$$

The computation of the denominator is known to be hard. In fact, it usually does not have an analytical solution. Nevertheless, the Metropolis-Hastings method (MH) [16] allows us to draw samples from $p(\theta|\mathbf{z}_{1:t})$ using unnormalized $p(\mathbf{z}_{1:t}|\theta)p(\theta)$. MH draws candidate samples from a proposal distribution $q(\cdot|\theta^{(i-1)})$ and then accepts the samples with a probability α. After a burn-in period,

Algorithm 2 Metropolis-Hastings for Parameter Estimation

```
$\theta^{(0)} \sim q(\cdot)$
for $i = 1, 2, \ldots$ do
    $\theta^{c} \sim q(\cdot|\theta^{(i-1)})$
    $\alpha \leftarrow min\left(1, \frac{p(\mathbf{z}_{1:t}|\theta^{c})p(\theta^{c})}{p(\mathbf{z}_{1:t}|\theta^{(i-1)})p(\theta^{(i-1)})}\frac{q(\theta^{(i-1)}|\theta^{c})}{q(\theta^{c}|\theta^{(i-1)})}\right)$
    $u \sim \mathcal{U}(0,1)$
    $\theta^{(i)} \leftarrow \begin{cases} \theta^{c}, & \text{if } u \le \alpha \\ \theta^{(i-1)}, & \text{otherwise} \end{cases}$
end for
```

MH produces samples from the target distribution $p(\theta|\mathbf{z}_{1:t})$. In Algorithm 2, we run the Kalman filter for each sample θ to calculate the marginal likelihood $p(\mathbf{z}_{1:t}|\theta)$. In Kalman filtering, $p(\mathbf{z}_{1:t}|\theta)$ is a product of Gaussian densities and can be computed recursively as follows,

$$p(\mathbf{z}_{1:t}|\theta) = \prod_{i=1}^{t} p(\mathbf{z}_i|\mathbf{z}_{1:i-1}, \theta) \tag{18}$$

$$= \prod_{i=1}^{t} \mathcal{N}(z_i; \mathbf{H}_i \mathbf{s}_i, \mathrm{S}_i) \tag{19}$$

where $\mathrm{S}_i := \mathbf{R} + \mathbf{H}_t \mathbf{P}_{t|t-1} \mathbf{H}_t^T$. At each time step, the marginal likelihood is updated by the density of the new measurement [16].

5.2 Anomaly Detection

The marginal likelihood $p(\mathbf{z}_{1:t}|\theta)$ provided by the Kalman filter can also be used to detect abrupt flow related changes and inconsistencies in a network. After obtaining the new measurement $\mathbf{z}_t$, we calculate the probability of measuring $\mathbf{z}_t$ given previous measurements, $p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \mathcal{N}(z_i; \mathbf{H}_i \mathbf{s}_i, \mathrm{S}_i)$, and check if the probability is below a given threshold. If the probability is too small, it means that the measured flow statistics are not normal and cannot be explained by the current state and the model. Furthermore, by looking at the state covariance $\mathbf{P}_{t|t}$ we are able to find the responsible switch. For instance, we measure a packet count of a certain rule in a flow table and observe that the packet count is reset to zero, although the state estimate says that there is an active flow matching with the rule. This may indicate the corresponding switch is malicious or malfunctioning. That capability enables utilization of our scheme for monitoring various network requirements such as security and availability.

5.3 Estimation in Dynamic Networks

The proposed scheme is highly flexible for addressing dynamic networks with changing structure, flows and management mechanisms since the Kalman filter allows the transition matrix $\mathbf{F}$ to be changed over time. This property enables us to build a scalable and robust Kalman filter. In our method, whenever the state of the network is changed or some rules in the flow tables are updated, we update the transition matrix $\mathbf{F}$. For example, suppose a new switch is connected to the network, and in response, the controller updates the rules in the flow tables to change the routing. In this case, first we wait until the rules are updated, then we construct a new $\mathbf{F}$. During this transitional period, we may skip a couple of measurements in our scheme, which is also allowed by the Kalman filter.

We note that in the networks where the number of hosts is large but the number of active flows is small, $\mathbf{F}$ and $\mathbf{P}_{t|t}$ become extremely large sparse matrices since they grow at the order of $O(n^4)$. However, this problem can be easily mitigated by using sparse matrix computations.

6 EXPERIMENTS

In this section, we explain our experiment design and environment. We provide our experimental results and evaluate our method by comparing it to alternative techniques.

6.1 Experiment Design

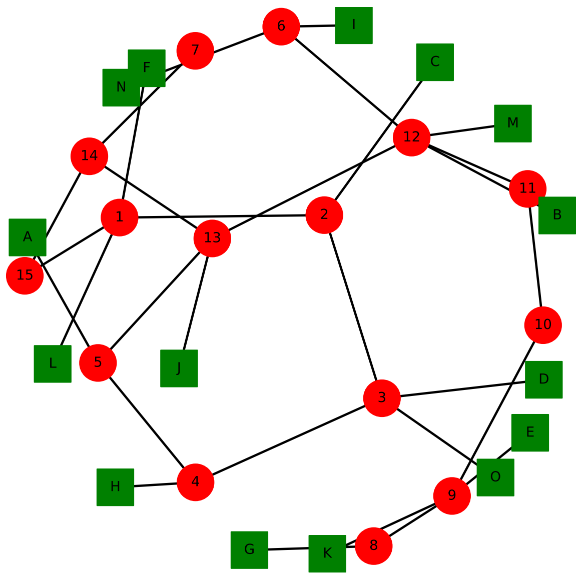

Figure 1: The random topology used in the simulations. The red circles represent switches and the green squares represent hosts.

In our experiments, we utilized *Mininet* [9] to simulate a software-defined network consisting of switches supporting *OpenFlow v1.3* specification [11]. We created a random topology with 15 switches and 15 hosts using Watts-Strogatz graph model [19]. The experimental topology is shown in Figure 1, where red circles and green squares represent switches and hosts, respectively. We generated flows between each pair of hosts using the tool *iperf* [4].

We built our SDN application on top of Python-based *Ryu* [15] controller. Our application first detects the network topology and adds the routing rules (*in-port,destination-mac*) to the flow tables according to the shortest path algorithm. Subsequently, it starts to periodically query the switches and construct the traffic matrix at fixed time intervals (in our experiments $\Delta t = 10$ seconds). After gathering the packet counts, the controller estimates the TM by using the Kalman filtering scheme explained in Section 4.2. Then it decides the subset of switches to be queried in the next cycle. The overall mechanism is depicted in Figure 2.

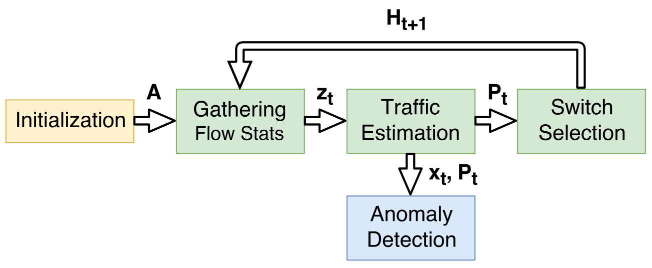

Figure 2: Overview of the inference mechanism.

We adopted a simple transition and measurement model in the experiments. In our model, the initial packet rates of OD flows are independent and normally distributed with mean μ_0 and variance σ_0. They fluctuate independently and normally with variance σ_v^2. The packet count vector $\mathbf{y}_t$ is a zero vector, because we reset all the count values to zero in the flow tables at the initialization step. The measurement errors of the packet counts are independent white noises with variance σ_ϵ^2. Using these simplifications, we constructed the parameter set of the Kalman filter as $\theta = \{\mu_0, \sigma_0, \sigma_v, \sigma_\epsilon\}$.

We generated flows for two hours and recorded all the packet counts in the flow tables. In order to test the stability of our method, we deliberately kept all the virtual switch processors busy for 10 seconds in the second hour, which made some switches malfunction and unable to properly count the packets. As a result, we obtained packet counts inconsistent with the linear system in (1).

We used the first hour of the trace to learn the Kalman parameters. We ran the Metropolis-Hastings (Algorithm 2) to sample from $p(\theta|\mathbf{z}_{1:t})$ in (17). At each iteration, we ran the Kalman filter with the complete measurement matrix ($r \times r$ identity matrix) to calculate the likelihood in (18), and we used a uniform prior of θ. We generated 32K samples and discarded first 2K as burn-in samples. After that, we constructed the approximation $\tilde{p}(\theta|\mathbf{z}_{1:t})$ and we chose the sample θ^* which produces the maximum likelihood. We do not here include the plots of the marginals of $\tilde{p}(\theta|\mathbf{z}_{1:t})$ due to the lack of space.

We used the second hour of the trace to evaluate our inference framework. We simulated our application as if it ran on a network where the packet counts in the flow tables were the same as the ones in the trace. We used three strategies for switch selection: *Round-Robin*, *Random* and *Min-Entropy*. In *Round-Robin*, we generate a random permutation of the switches and select the subset of switches in circular order, in *Random*, we choose the subset uniformly at random, and in *Min-Entropy* we use the method described in Section 4.3. We ran the simulation for different sizes of the query list, $k = 1, \ldots, 10$. We recorded the TM vectors $\mathbf{x}_t$ in (14) for each time step, strategy and size of the query list. Lastly, we repeated the experiment $N = 100$ times to reduce the random variation in *Round-Robin* and *Random* strategies.

6.2 Experimental Results

We construct a baseline TM by using all the packet counts for each time step. In our case, the routing matrix $\mathbf{A}$ in (1) is a full rank matrix with more rows than columns. The linear system (1) is overdetermined and in general has no solution. Thus, we want to find the approximate TM which best fits the packet counts in the trace.

$$\mathbf{b}_t = \underset{\mathbf{x}_t}{\operatorname{argmin}} \|(\mathbf{y}_{t+1} - \mathbf{y}_t) - \mathbf{A}\mathbf{x}_t\|_2 \tag{20}$$

The least squares problem in (20) could be solved using $\mathbf{A}^\dagger$, the Moore-Penrose pseudoinverse of $\mathbf{A}$, which can be algebraically formulated as follows,

$$\begin{aligned} \mathbf{b}_t &= \mathbf{A}^\dagger(\mathbf{y}_{t+1} - \mathbf{y}_t) & (21)\\ &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{y}_{t+1} - \mathbf{y}_t) & (22)\end{aligned}$$

In Figure 3, we depict the packet rate estimates of a specific OD flow along with the baseline rate. The estimates are calculated

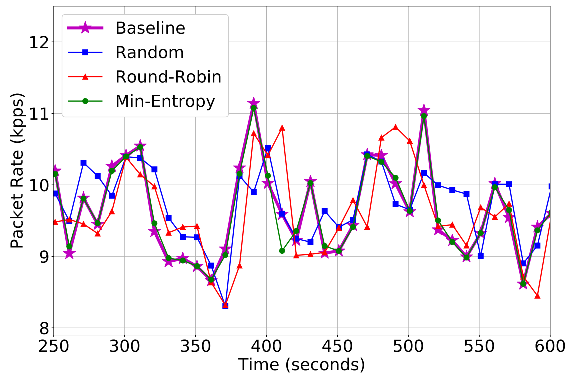

Figure 3: Estimated and baseline packet rates of a specific OD flow. (for $k = 5 \Rightarrow 33.3\%$)

by three switch selection strategies querying only 5 switches at a time. For better visualization, we only show a short time interval from $t = 250s$ to $t = 600$. We see that all the strategies are able to track the baseline flow. However, the *Min Entropy* estimate is much closer to the baseline than the other estimates for most of the time. Moreover, the deviations of *Random* and *Round Robin* from the baseline become quite large at some time points.

We evaluate the overall performance of our inference framework by using the relative root-mean-square error (RRMSE) metric,

$$E = \frac{\sqrt{\sum_{j=1}^{N}\sum_{i=1}^{c}\sum_{t=t_0}^{T}(x_t^{i,j} - b_t^i)^2}}{\sqrt{N\sum_{i=1}^{c}\sum_{t=t_0}^{T}(b_t^i)^2}} \tag{23}$$

Here the superscript j denote the trial number and N is the total number of trials, and we discard first $t_0 = 250$ seconds as warm-up period. We plot the relative errors of the switch selection strategies

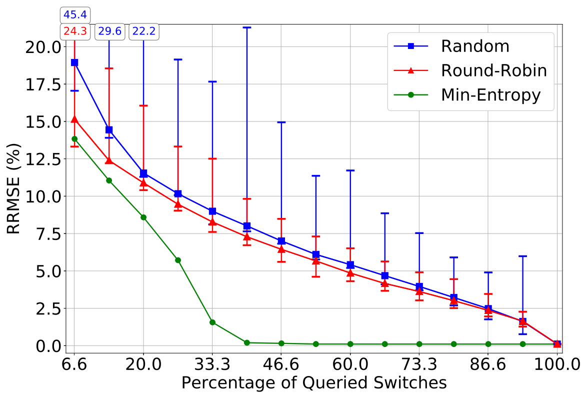

Figure 4: RRMSE vs percentage of queried Switches. The error bars correspond to the worst and the best trials. (for $m = 15$)

for different number of queried switches in Figure 4. We also plot the error bars indicating the relative errors of the best and the worst trials. We observe as the number of queried switches increases, the error of *Min-Entropy* drops faster than the other strategies, and it reaches zero at point $k = 6$. This is because *Min-Entropy* chooses the switches that give the most information about the OD flows when combined, and for $k \geq 6$ it is able to constitute a query list that solves the linear system (1). On the other hand, the other methods choose a switch without considering whether the information it brings could be inferred from the existing query list. We see *Random* and *Round-Robin* exhibit roughly similar error profiles. However, the largest errors of *Round-Robin* are much smaller. Moreover, we point out *Min-Entropy* is deterministic, i.e. given the same model it always produces the same query list, thereby does not have error bars in that figure. Figure 4 also displays the trade-off between the accuracy and the measurement overhead. The measurement overhead could be decreased in exchange for loss of accuracy, though it is apparent that querying a large number of switches, i.e. $k \geq 6$, using *Min-Entropy* leads to overkill.

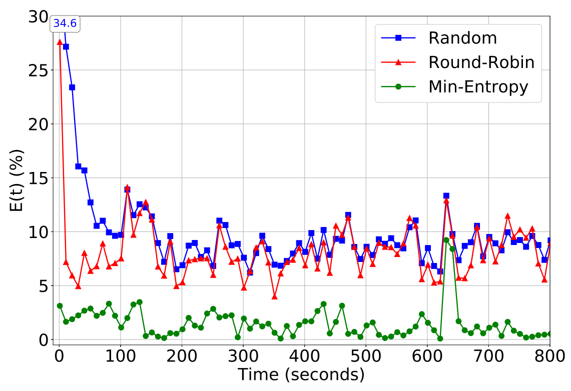

Figure 5: RRMSE of the switch selection strategies at each time step. (for $k = 5 \Rightarrow 33.3\%$)

Next, we consider spatial and temporal errors for a fixed size of the query list $k = 5$. The temporal error is the RMSE of the overall estimation errors of OD flows at a particular time.

$$E(t) = \frac{\sqrt{\sum_{j=1}^{N} \sum_{i=1}^{c} (x_t^{i,j} - b_t^i)^2}}{\sqrt{N \sum_{i=1}^{c} (b_t^i)^2}} \tag{24}$$

Figure 5 shows that the initial temporal errors of the strategies are significantly high. The reason for that is the Kalman filter is initialized by a prior with a large variance (high entropy), and it requires some time to converge to the OD flows, which we call warm-up period t_0. We observe that *Min-Entropy* converges immediately after the first measurement since its ultimate aim is to maximize the certainty of the estimate. It also outperforms the other strategies for the entire simulation. We see a peak in the figure around $t = 630s$ because some malfunctioning switches significantly corrupt the packet counts at that time as explained in Section 6.1. However, the Kalman filter successfully handles the outliers and corrects itself in a very short time. We exclude the corrupted part and the warm-up period in all other assessments since the large errors arising in these exceptional cases degrades the overall performance evaluation of the strategies.

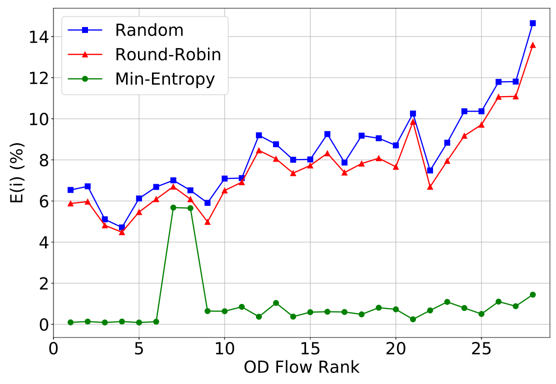

Figure 6: RRMSE of the switch selection strategies for largest OD flows. (for $k = 5 \Rightarrow 33.3\%$)

We also calculate the spatial errors, i.e. the errors of the estimated OD flows, in a similar way.

$$E(i) = \frac{\sqrt{\sum_{j=1}^{N} \sum_{t=t_0}^{T} (x_t^{i,j} - b_t^i)^2}}{\sqrt{N \sum_{t=t_0}^{T} (b_t^i)^2}} \tag{25}$$

We sort the OD flows from largest to smallest and we plot the errors of the first 28 OD flows for each strategy in Figure 6. These flows constitute 95% of the entire flow load in the network. We note the x-axis represents the rank of the OD flow instead of its id. As can be seen in Figure 6, *Min-Entropy* experiences the smallest errors, and the errors of the *Random* and *Round-Robin* are increasing as the flows are getting smaller, partly because the errors are calculated relatively. We also observe that *Min-Entropy* exhibits larger errors for certain OD flows (7th and 8th) because these flows are harder to estimate. The 7th and 8th flows traverse almost the same path, and therefore the Kalman filter confuse them with each other.

Lastly, we plot the change in the marginal likelihood in Equation (19) for $k = 7$ to show that the dramatic decrease in the probability $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$ is a strong indicator of an anomaly in the network. In Figure 7, we see that the likelihood of the corrupted measurement at $t = 630s$ is extremely small compared to the others. Furthermore, the likelihood of the new measurements becomes normal right after the switches start to work properly. This is because the new likelihood is calculated using all the previous measurements and the effect of the corrupted measurement is very small. For anomaly detection, we recommend querying a large number of switches, e.g. greater than 40%, because otherwise the Kalman filter might not detect the inconsistencies.

7 CONCLUSION AND FUTURE WORK

In this paper, we present a Kalman filtering based scheme to estimate an accurate and timely Traffic Matrix (TM) in SDN environment. The proposed method uses only partial information while

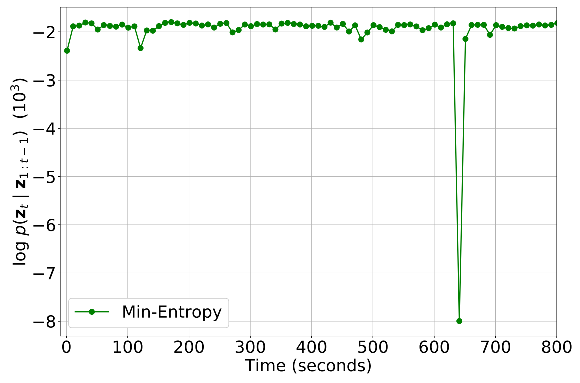

Figure 7: Log-probability of each measurement given all previous measurements. (for $k = 7 \Rightarrow 46.6\%$)

constructing the TM in order to avoid the communication and computational overhead. In addition, we propose a switch selection strategy which aims to increase the accuracy of the constructed TM by minimizing the information entropy of the estimate.

We use a network trace produced in *Mininet* environment to evaluate our inference framework and the switch selection strategy. We demonstrate that our framework estimates a TM almost as accurate as the one obtained by using all the flow statistics even when a few switches are queried at a time. We also compare our switch selection strategy with some conventional methods such as uniformly random selection and round robin selection, and show that our strategy outperforms these methods in terms of both spatial and temporal error characteristics.

In this study, we have assumed that the evolution of the network traffic is simply a linear Gaussian state-space model (LGSSM). In the future, we plan to design a more general framework capable of working with the nonlinear non-Gaussian models by utilizing Sequential Monte Carlo (SMC) methods instead of the Kalman filter. Moreover, integration of a network management mechanism that is based on our framework and instrumental in software-defined networks is another research direction.

ACKNOWLEDGMENTS

This work was supported in part by the Scientific and Technical Research Council of Turkey (TUBITAK) under grant number 117E165 and in part by the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610.

REFERENCES

[1] A. Atary and A. Bremler-Barr. 2016. Efficient Round-Trip Time monitoring in OpenFlow networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524501

[2] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. 1990. *Simple Network Management Protocol (SNMP)*. RFC 1157. United States.

[3] Mark E. Crovella and Azer Bestavros. 1997. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Trans. Netw.* 5, 6 (Dec. 1997), 835–846. https://doi.org/10.1109/90.650143

[4] Esnet. 2017. esnet/iperf. (Sep 2017). https://github.com/esnet/iperf Accessed: 2017-10-01.

[5] Yanlei Gong, Xiong Wang, Mehdi Malboubi, Sheng Wang, Shizhong Xu, and Chen-Nee Chuah. 2015. Towards Accurate Online Traffic Matrix Estimation in Software-Defined Networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15, Santa Clara, California, USA, June 17-18, 2015*, Jennifer Rexford and Amin Vahdat (Eds.). ACM, 26:1–26:7. https://doi.org/10.1145/2774993.2775068

[6] Mohinder S. Grewal and Angus P. Andrews. 2014. *Kalman Filtering: Theory and Practice with MATLAB* (4th ed.). Wiley-IEEE Press.

[7] Lavanya Jose, Minlan Yu, and Jennifer Rexford. 2011. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE'11)*. USENIX Association, Berkeley, CA, USA, 13–13.

[8] Diego Kreutz, Fernando M. V. Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-Defined Networking: A Comprehensive Survey. *CoRR* abs/1406.0440 (2014).

[9] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*. ACM, New York, NY, USA, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466

[10] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. 1994. On the Self-similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Trans. Netw.* 2, 1 (Feb. 1994), 1–15. https://doi.org/10.1109/90.282603

[11] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[12] L. Nie, D. Jiang, L. Guo, S. Yu, and H. Song. 2016. Traffic Matrix Prediction and Estimation Based on Deep Learning for Data Center Networks. In *2016 IEEE Globecom Workshops (GC Wkshps)*. 1–6. https://doi.org/10.1109/GLOCOMW.2016.7849067

[13] M. Polverini, A. Baiocchi, A. Cianfrani, A. Iacovazzi, and M. Listanti. 2016. The Power of SDN to Improve the Estimation of the ISP Traffic Matrix Through the Flow Spread Concept. *IEEE Journal on Selected Areas in Communications* 34, 6 (June 2016), 1904–1913. https://doi.org/10.1109/JSAC.2016.2559178

[14] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. 2002. Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment (IMW '02)*. ACM, New York, NY, USA, 91–92. https://doi.org/10.1145/637201.637213

[15] Ryu SDN Framework Community. 2014. Ryu SDN Framework. (2014). Retrieved 17 March, 2016, from http://osrg.github.io/ryu/.

[16] Simo Sarkka. 2013. *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA.

[17] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. 2010. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Proceedings of the 11th International Conference on Passive and Active Measurement (PAM'10)*. Springer-Verlag, Berlin, Heidelberg, 201–210.

[18] Paul Tune and Matthew Roughan. 2013. Internet Traffic Matrices: A Primer. (2013).

[19] D. J. Watts and S. H. Strogatz. 1998. Collective dynamics of'small-world'networks. *Nature* 393, 6684 (1998), 409–10.

[20] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. 2013. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In *Proceedings of the 14th International Conference on Passive and Active Measurement (PAM'13)*. Springer-Verlag, Berlin, Heidelberg, 31–41. https://doi.org/10.1007/978-3-642-36516-4_4

[21] Q. Zhang and T. Chu. 2016. Structure Regularized Traffic Monitoring for Traffic Matrix Estimation and Anomaly Detection by Link-Load Measurements. *IEEE Transactions on Instrumentation and Measurement* 65, 12 (Dec 2016), 2797–2807. https://doi.org/10.1109/TIM.2016.2599426

[22] Yin Zhang, Matthew Roughan, Nick Duffield, and Albert Greenberg. 2003. Fast Accurate Computation of Large-scale IP Traffic Matrices from Link Loads. *SIGMETRICS Perform. Eval. Rev.* 31, 1 (June 2003), 206–217. https://doi.org/10.1145/885651.781053

[23] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. 2009. Spatio-temporal Compressive Sensing and Internet Traffic Matrices. *SIGCOMM Comput. Commun. Rev.* 39, 4 (Aug. 2009), 267–278. https://doi.org/10.1145/1594977.1592600