Optimal Probing with Statistical Guarantees for Network Monitoring at Scale

Branislav Kveton[b], Muhammad Jehangir Amjad[a], Christophe Diot[a], Dimitris Konomis[c], Augustin Soule[a], Xiaolong Yang[a]

[a]*Google Inc.,*
[b]*Amazon,*
[c]*MIT,*

Abstract

Monitoring large-scale cloud networks is a complex task because their scale is prohibitively large, monitoring budgets are limited, network topologies are not regular and the estimates produced are a function of traffic patterns. In this work, we take a statistical approach to estimating a network metric, such as the latency of a set of paths, with guarantees on the estimation error. We aim to do so in an *intelligent* and *scalable* manner, without observing all existing traffic, and minimizing the estimation error at a fixed probing budget per unit of time. Our algorithms produce a distribution of probes/samples across network paths which can be used in conjunction with existing probers (or samplers). These algorithms are based on A- and E-optimal experimental designs in statistics, which guarantee a bounded estimation error for any monitoring budget. Unfortunately, these designs are too computationally intensive to be used in production at scale. We propose a scalable and near-optimal approximate implementations based on the Frank-Wolfe algorithm. We validate our approaches with two metrics (latency and loss) in simulations on real network topologies, and also using a production probing system in a real cloud network. We show major gains in reducing the probing budget compared to both production and academic baselines, while maintaining low errors in estimates, even with very low probing budgets.

1. Introduction

Monitoring is key to many management and traffic engineering tasks which enable high service availability. The research community has enthusiastically on-boarded the challenge of designing scalable cloud network monitoring systems (see, e.g. [20, 18, 32, 19, 34, 22, 16]), an effort which generally relies on the ability to (i) collect exhaustive data in a timely manner, (ii) optimally store the data or a summary of it, and (iii) perform efficient computations using the data, in order to accurately estimate network metrics. A large majority of these research papers focuses on data center networks (to take advantage of their architectural properties), using sketches and optimal data structures to increase scalability [20]. Unfortunately, monitoring data center networks separately is not sufficient to track cloud

Email addresses: `bkveton@amazon.com` (Branislav Kveton), `muhammadamjad@google.com` (Muhammad Jehangir Amjad), `christophediot@google.com` (Christophe Diot), `dkonomis@mit.edu` (Dimitris Konomis), `augustins@google.com` (Augustin Soule), `xiaolongyang@google.com` (Xiaolong Yang)

customers' experience as customer traffic is generally not local to a single data center.

At the global cloud infrastructure scale, monitoring every link and endpoint in the network by measuring all metrics on a real-time basis is impossible to achieve in production. First, the geographical and infrastructural scale of these networks is prohibitively large and usually growing at a rapid pace. Secondly, routers and hosts in the network often have limited memory and CPU resources. Within hosts, the goal is to preserve as many resources for revenue-generating usage as possible. As a result, not only is *it impossible to measure and monitor the network at all points and at all times, there is no acceptable amount of CPU or memory that can be reserved for these tasks. Practically, this translates to a "using as little as possible" philosophy for monitoring tasks, generally enforced by router and host-level maximum probing (or sampling) budgets*.

We claim that for network monitoring, *better does not necessarily mean more*. We take an approach that consists of judiciously collecting the best possible measurements for any monitoring budget. We formulate an optimization problem whose constraints model global and local budget limitations, and whose solution is an optimal distribution of probes over paths of interest in a network topology. Using this optimal distribution, one then generates probes and collects measurements that are subsequently used for the estimation of network metrics with statistical guarantees.

We use networks *paths* as the primary structure of interest. A path can be viewed as a sequence of links (such as hops between routers) joining two end points (i.e. hosts), or more abstractly as a feature vector with path characteristics.

Problem. Given global and local probing budgets and the network topology (or other network "features") as inputs, we want to determine the *optimal probing distribution* across all network paths to accurately estimate a network metric of interest across *every path*,

This work is only concerned with the process of optimally distributing a total probing budget among the network's paths (per unit time) *before* the probes are scheduled. We assume that the probing mechanism is a black-box, and we do not deal with how probes are spaced (in time) on each path, such as Poisson or Uniform.

While for simplicity reasons, this paper deals with probing, our approach applies directly to sampling. However sampling only applies to links with existing traffic, making it necessary to know the traffic matrix in order to optimize the sampling rate).

This work finds its roots in network tomography [3] (the ability to infer network metrics - mainly loss and latency - from probes sent in the network) and optimal monitor placement [4] (i.e. where to monitor and at what rate, in order to perform accurate measurement of metrics with limited impact on network resources) in the early twenty-first century. The closest related prior work is Chua et al. [6] where the authors present a method to select (approximately) the best subset of network paths to measure given scale constraints. We use the algorithm proposed in [6] as a baseline for comparison to our methodology. Relevant literature is reviewed in more detail in Section 7.

Proposed Solution. We formulate an optimization problem where we minimize the average or maximum estimation error for a metric of interest, such as latency across all paths, subject to measurement constraints, such as probing budgets at each node in the network. The output is an optimal distribution of probes across the paths in the network, per time unit.

We show that our framework is fairly generic. In Section 3.1, we introduce linear models and discuss their properties that allow us to estimate network latency with guarantees. We use a well-known property of linear regression, known as the optimal design (Sections 3.2 and 3.3), where the optimal probing distribution over network's paths can be determined without any prior observations, simply as a function of the features of the paths. As it is not possible to model all network metrics using linear models, we introduce generalized linear models (GLMs) in Section 4.5. This model also allows us to apply the same principles of optimal design. In Section 8 and Appendix Appendix A, we also discuss a general reduction for any non-linear model, which would take us beyond just latency and loss that are the metrics studied in this paper.

While optimal designs provide solutions to the optimization problem with statistical guarantees, they are known to be computationally challenging and they are not usable in production environments. In Section 4, we propose an approximate solution using the Frank-Wolfe algorithm [11], which is near-optimal and enables a scalable implementation in very large networks.

We evaluate our approach using both simulations (with real network topologies) and probing in a real production cloud network. We compare our probing strategy to two baselines: (i) `QR` algorithm proposed in Chua et al. [6], which selects the "best" subset of paths using SVD and QR decomposition; and (ii) a `Prod` baseline which distributes probes across all paths evenly (uniformly). We choose `QR` because it is the closest related prior work which selects only a few important paths and `Prod` because it is a common choice in practice, as it can be implemented easily in resource-constrained production networks. We show that our probing strategy produces more accurate metric estimates (using latency and loss) than both the baselines for similar probing budgets (Sections 5 and 6). Most importantly, even with very low budgets, the estimation error remains low, which makes our approach usable for many telemetry tasks.

Contributions Summary. The main contributions of this paper are (1) the design of probing distributions that minimize the error of a performance metric for a given budget; (2) a near-optimal implementation that allows deployment in large-scale networks, and (3) a thorough evaluation through simulation and probing in a large real network, which shows good performance even for small probing budgets. A key strength of our approach is that it produces an optimal probing distribution without having observed any prior traffic on the network.

2. Setting

The problem of statistical inference can be viewed as follows. The goal is to estimate an unknown function $f_* : \mathcal{X} \to R$ that maps d-dimensional *features* $x \in \mathcal{X}$ to real values $y \in R$, where $\mathcal{X} \subseteq R^d$ is a set of all d-dimensional feature vectors. In the networking domain, one example of f_* is the *latency of a path* in a network with d edges. In this case, x is a vector of edge indicators in a path, $y = f_*(x)$ is the expected latency of that path, and $\mathcal{X} \subseteq \{0, 1\}^d$ is the set of all paths.

We estimate f_* from a dataset

$$\mathcal{D} = \{(x_i, y_i) : x_i \in \mathcal{X}, y_i \in R, i \in [n]\}$$

of n training examples, where x_i is the feature vector of the i-th example and y_i is a noisy realization of $f_*(x_i)$. Our goal is to learn a best possible approximation to f_* with guarantees. We focus on two kinds of guarantees. The first is the *worst case*, where we want to approximate f_* well for all $x \in \mathcal{X}$. Then a natural metric to optimize is the maximum squared error of function f,

$$\mathcal{L}_{\max}(f) = \max_{x \in \mathcal{X}} (f(x) - f_*(x))^2 \,. \tag{1}$$

In our latency prediction example, $\mathcal{X}$ is the set of all paths and $\mathcal{L}_{\max}(f)$ is the maximum squared error of estimated path latencies using f.

We also consider the *average case*. Specifically, let $\mathcal{P}$ be a distribution over feature vectors in $\mathcal{X}$. Then we may want to optimize the average error over $x \sim \mathcal{P}$,

$$\mathcal{L}_{\text{avg}}(f) = E_{x \sim \mathcal{P}} \left[(f(x) - f_*(x))^2\right] \,. \tag{2}$$

In our latency prediction example, $\mathcal{P}$ is a distribution over all paths $\mathcal{X}$ and $\mathcal{L}_{\text{avg}}(f)$ is the mean squared error of estimated path latencies using f, weighted by $\mathcal{P}$.

General approach. Now we are ready to state our goal. For a fixed error metric, such as $\mathcal{L}_{\max}$, and a budget of n measurements, we want to collect a dataset $\mathcal{D}$ such that we can learn a good approximation $\hat{f}$ to f_* from $\mathcal{D}$, as measured by $\mathcal{L}_{\max}(\hat{f})$. Note that this optimization problem is inherently difficult, since $\hat{f}$ is learned from the measurements $(y_i)_{i=1}^n$ in $\mathcal{D}$, which are unknown before the dataset $\mathcal{D}$ is collected. Therefore, in the rest of this work, we focus on a simpler optimization problem of determining the relative frequency (distribution) of feature vectors x in $\mathcal{D}$ that would lead to a good approximation $\hat{f}$ with guarantees, after f_* is measured at those feature vectors. This problem is known as the *optimal experimental design* [24, 1, 26].

Our approach is summarized in Algorithm 1. In line 2, we determine the *probing distribution* $\alpha = (\alpha_x)_{x \in \mathcal{X}}$ of feature vectors, where α_x is the *probing probability* of feature vector x. This is the main optimization problem studied in this work. In lines 3-7, we collect a dataset of n measurements, where each x appears $\alpha_x n$ times in expectation. Finally, in line 8, we learn an approximation $\hat{f}$ to f_* from dataset $\mathcal{D}$.

3. Optimal Designs

To optimize errors (1) and (2), we focus on a class of functions $f : \mathcal{X} \to R$ where $|f(x) - f_*(x)|$ is bounded by an expression that only depends on features $(x_i)_{i=1}^n$, and not on measurements $(y_i)_{i=1}^n$. One such class are linear models (Section 3.1). In Sections 3.2 and 3.3, we derive probing distributions α that optimize the errors (1) and (2) in linear models. This is the optimization problem in line 2 of Algorithm 1.

We discuss local budget constraints in Section 3.4. One of our goals is to ensure generality and feasibility in a large and global modern network. We consider these and a generalization to non-linear models in Section 4.

Algorithm 1 Estimating $\hat{f}$ from probes.

```
1: Inputs: Budget of $n$ measurements
2: Determine probing distribution $\alpha = (\alpha_x)_{x \in \mathcal{X}}$
3: $\mathcal{D} \leftarrow \emptyset$
4: for $i = 1, \dots, n$ do
5:     Set $x_i \leftarrow x$ with probability $\alpha_x$
6:     Probe $x_i$ and observe $y_i$
7:     $\mathcal{D} \leftarrow \mathcal{D} + (x_i, y_i)$
8: Learn $\hat{f}$ from the collected dataset $\mathcal{D}$
9: Outputs: Learned function $\hat{f}$
```

3.1. Linear Models

In linear models, the unknown value $f_*(x)$ is linear in x and given by $f_*(x) = x^\top \theta_*$, where $x \in R^d$ is a known feature vector and $\theta_* \in R^d$ is an unknown parameter vector. The unknown θ_* is estimated from noisy observations

$$y = f_*(x) + \epsilon = x^\top \theta_* + \epsilon \,, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \,,$$

which are obtained by adding Gaussian noise to the true value $f_*(x)$. We assume that all examples in dataset $\mathcal{D}$ are generated this way. Specifically, $y_i = x_i^\top \theta_* + \epsilon_i$ for all $i \in [n]$, where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ are drawn independently.

A popular way of estimating θ_* is by minimizing the sum of squares $\hat{\theta} = \arg\min_\theta \sum_{i=1}^n (y_i - x_i^\top \theta)^2$, the so-called *least squares regression*. When $\sum_{i=1}^n x_i x_i^\top$ is invertible, this problem has a unique closed-form solution

$$\hat{\theta} = G^{-1} \sum_{i=1}^n y_i x_i \,, \quad G = \sum_{i=1}^n x_i x_i^\top \,.$$

The above matrix G is known as the *sample covariance matrix* [14]. It is known that $\hat{\theta}$ follows a multivariate normal distribution with mean $E[\hat{\theta}] = \theta_*$ and covariance $\mathrm{Cov}(\hat{\theta}) = \sigma^2 G^{-1}$, for any fixed $(x_i)_{i=1}^n$.

Given a new example with feature vector x, the true value $f_*(x) = x^\top \theta_*$ is estimated by $\hat{f}(x) = x^\top \hat{\theta}$. Since $\hat{\theta} \sim \mathcal{N}(\theta_*, \sigma^2 G^{-1})$, it can be shown [14] that $\hat{f}(x)$ is also normally distributed,

$$\hat{f}(x) \sim \mathcal{N}(f_*(x), \sigma^2 x^\top G^{-1} x) \,. \tag{3}$$

From tail inequalities for Gaussian random variables [10], it follows that

$$(\hat{f}(x) - f_*(x))^2 \leq 2\sigma^2 log(1/\delta) x^\top G^{-1} x \tag{4}$$

holds with probability at least $1 - \delta$. Therefore, the problem of designing a good approximation $\hat{f}$ to f_*, with a high-probability bound on $(\hat{f}(x) - f_*(x))^2$, reduces to designing a good sample covariance matrix G, with an upper bound on $x^\top G^{-1} x$. Note that $x^\top G^{-1} x$ does not depend on the measurements y_i in $\mathcal{D}$; it only depends on the features x_i. Because of that, we can derive desirable probing distributions α that depend only on $\mathcal{X}$.

Although linear models are simple, they are useful for modeling a canonical networking problem of predicting latency. In particular, since the latency of a path is the sum of latencies on edges of that path, the latency is a linear function. That is, let $\theta_* \in R^d$ be a vector of mean latencies of d edges, where $\theta_*(e)$ is the mean latency of edge e. Let $x \in \{0, 1\}^d$ be a vector of edge indicators in a path. Then $f_*(x) = x^\top \theta_*$ is the mean latency of path x.

3.2. E-Optimal Design

This section discusses our first approach to optimizing G. The description of this and later methods requires some notation and definitions from linear algebra, which we introduce next. For any symmetric matrix $M \in R^{d\times d}$, we denote by $\lambda_i(M)$ the i-th largest eigenvalue of M. We let $\lambda_{\max}(M) = \lambda_1(M)$ and $\lambda_{\min}(M) = \lambda_d(M)$. The trace of M is the sum of its eigenvalues, $\operatorname{tr}(M) = \sum_{i=1}^{d} \lambda_i(M)$. We say that M is *positive semi-definite (PSD)*, and we denote it by $M \succeq 0$, if $x^\top M x \geq 0$ for all $x \in R^d$.

Now we are ready to introduce our first approach. Minimization of the maximum error in (1) reduces to optimizing G as follows. Suppose, without loss of generality, that $\mathcal{X}$ in (1) is a unit sphere, $\mathcal{X} = \{x : \|x\|_2 = 1\}$. Then

$$\mathcal{L}_{\max}(\hat{f}) \propto \max_{x\in\mathcal{X}} x^\top G^{-1} x = \lambda_{\max}(G^{-1}) = \lambda_{\min}^{-1}(G)\,.$$

The first step is from the definition of $\mathcal{L}_{\max}$ in (1) and (4), where we omit the constant factor of $2\sigma^2 \log(1/\delta)$. The first equality is the definition of the maximum eigenvalue, and the second equality is a result in linear algebra, that the maximum eigenvalue of a PSD matrix M is the reciprocal of the minimum eigenvalue of M^{-1} [28].

Therefore, minimization of (1) amounts to maximizing the minimum eigenvalue of the sample covariance matrix G. This is known as the *E-optimal design* and can be formulated as a *semi-definite program (SDP)* [31]

$$\begin{aligned} \max \quad & \tau \\ \text{s.t.} \quad & \sum_{x\in\mathcal{X}} \alpha_x x x^\top \succeq \tau I_d\,, \\ & \sum_{x\in\mathcal{X}} \alpha_x \leq 1\,, \quad \forall x \in \mathcal{X} : \alpha_x \geq 0\,. \end{aligned} \tag{5}$$

This SDP has two types of variables. The variable τ is the minimum eigenvalue of the sample covariance matrix and it is maximized. The variable α_x denotes the probing probability of feature vector x (line 2 of Algorithm 1).

3.3. A-Optimal Design

Minimization of the average error in (2) reduces to optimizing the sample covariance matrix G as follows. Let $\mathcal{X}$ be the unit sphere in Section 3.2 and $\mathcal{P}$ be a uniform distribution over it. Let $U \Lambda U^\top = G^{-1}$ be the eigendecomposition of G^{-1}, which is PSD by definition. Then

$$\begin{aligned}\mathcal{L}_{\text{avg}}(\hat{f}) &\propto E_{x\sim\mathcal{P}}\left[x^\top G^{-1}x\right] = E_{x\sim\mathcal{P}}\left[x^\top U\Lambda U^\top x\right] \\ &= E_{x\sim\mathcal{P}}\left[x^\top \Lambda x\right] \propto \sum_{i=1}^{d}\lambda_i(G^{-1}) = \operatorname{tr}\left(G^{-1}\right)\,.\end{aligned}$$

The first step is from the definition of $\mathcal{L}_{\text{avg}}$ in (2) and (4), where we omit the constant factor of $2\sigma^2\log(1/\delta)$. The second equality holds since U only rotates x and $\mathcal{X}$ is a sphere. The next step follows from $\mathcal{X}$ being a sphere.

Therefore, minimization of (2) amounts to minimizing the sum of the eigenvalues of G^{-1}, which is equal to its trace. This is known as the *A-optimal design* and can be formulated as the following SDP [31]

$$\begin{aligned}\min\quad & \sum_{i=1}^{d}\tau_i \qquad\qquad (6)\\ \text{s.t.}\quad & \forall i\in[d]:\begin{bmatrix}\sum_{x\in\mathcal{X}}\alpha_x xx^\top & e_i\\ e_i^\top & \tau_i\end{bmatrix}\succeq 0\,,\\ & \sum_{x\in\mathcal{X}}\alpha_x\leq 1\,,\quad \forall x\in\mathcal{X}:\alpha_x\geq 0\,,\end{aligned}$$

where e_i is the i-th element of the standard d-dimensional Euclidean basis. This SDP has two types of variables. The variable τ_i represents the i-th eigenvalue of the inverse sample covariance matrix. The sum of these variables $\sum_{i=1}^d \tau_i$ is $\operatorname{tr}\left(G^{-1}\right)$ and it is minimized. The variable α_x is the probing probability of feature vector x (line 2 of Algorithm 1).

3.4. Local Budget Constraints

Both optimal designs (5) and (6) output a probing distribution $\alpha = (\alpha_x)_{x\in\mathcal{X}}$ subject to a single global constraint $\sum_{x\in\mathcal{X}}\alpha_x = 1$. In practice, local constraints are common. For instance, we may want to enforce that most paths do not start in a single source node, simply because such a probing strategy cannot be implemented due to resource availability constraints. Such constraints can be enforced as follows. Let $\operatorname{src}(x)$ be the source node of path x and $S = \{\operatorname{src}(x) : x\in\mathcal{X}\}$ be the set of all sources. Then

$$\forall s\in S:\sum_{x\in\mathcal{X}}\mathbb{1}\{\operatorname{src}(x)=s\}\,\alpha_x\leq b$$

is a set of linear constraints in α that limit the probing probability from any source s to at most b. Because the constraints are linear in α, they can be easily incorporated into (5) and (6) without changing the hardness of these problems. We experiment with local budget constraints in Section 5.5.

4. Practical Implementation

The experimental designs in Section 3 are theoretically sound and well understood. Unfortunately, it is impractical to solve them exactly as SDPs. We report our initial experience with this approach, as well as approximating it, in Section 4.1. To address this issue, we propose a general practical solution that combines the strengths of linear programming and gradient descent in Section 4.2. We instantiate this solution in E- and A-optimal designs in Sections 4.3 and 4.4, respectively; and evaluate it comprehensively in Sections 5 and 6. Finally, we also show how to extend optimal designs to generalized linear models in Section 4.5. We discuss more general non-linear models in Appendix Appendix A.

4.1. Initial Experience

Exact solutions to the SDPs in (5) and (6) are computationally costly [2]. The issue is that they are generally solved using interior-point methods [7], which are slow in practice and scaling them up is a problem-specific art. As a result, the design of practical and general algorithms for large-scale SDPs remains an open research problem.

Initially, we tried to solve the E-optimal SDP in (5) exactly. In this paragraph, we summarize our findings on synthetic problems in Section 5, which we call topologies A-C. We defer detailed descriptions of these problems to Section 5. For now, the only relevant information is the scale of the SDPs. In topologies A / B / C, the number of features d (edges) is 251 / 197 / 440; and the number of variables (paths) is $|\mathcal{X}| = 500$ which is a small subset of those that we use in the experiments in Section 5. To solve these SDPs, we used CVXPY [8], a publicly available library for convex programming, on a 32 core workstation with 128 GB RAM. The run times for topologies A / B / C were $4\,400$, $1\,800$ and $31\,700$ seconds, respectively. The A-optimal SDP in (6) crashed due to running out of memory for all topologies. These are clearly not practical solutions.

Our first attempt at solving (5) approximately was based on the fact that enforcing $A \in R^{d \times d}$ to be PSD equates to satisfying infinitely many constraints $x^\top A x > 0$, for any $x \in R^d$, which were linear in our parameterization of A. We used this fact to design an approximate cutting plane algorithm, which generated x and solved a sequence of LPs. This approach reduced the run time by two orders of magnitude when the number of added constraints was small. However, we also observed a significant drop in the quality of the solutions. The quality of the solutions increased with more constraints, but so did the computational cost. In general, this approach was hard to scale because the added LP constraints were not sparse, which is necessary for efficient LP solutions. For more details, refer to Appendix Appendix B.

The above attempt can be viewed as solving (5) and (6) exactly on an outer polytope of the feasible sets. Due to the drop the solution quality, we considered an opposite approach next. Roughly speaking, we do not change the feasible set and solve our SDPs approximately using the Frank-Wolfe algorithm [11]. This resulted in near optimal solutions and orders-of-magnitude reduction in run time, as shown in Section 5.3. We summarize the run time differences of all experiments in this paper together in Appendix Appendix C.

4.2. Frank-Wolfe Algorithm

The Frank-Wolfe algorithm [11] is a popular algorithm for constrained optimization problems of the form

$$\min_{\alpha \in \mathcal{A}} f(\alpha)\,, \tag{7}$$

where f is the optimized function, $\mathcal{A} \in R^d$ is a convex feasible region defined by linear constraints, and $\alpha \in \mathcal{A}$ is the optimized parameter vector. The algorithm solves (7) iteratively as a sequence of linear programs (LPs)

$$g(\alpha) = \arg\min_{\tilde{\alpha} \in \mathcal{A}} \tilde{\alpha}^\top \nabla f(\alpha) \,, \tag{8}$$

where $\nabla f(\alpha)$ is the gradient of f at α.

The algorithm works as follows. The input to iteration i is a feasible solution $\alpha^{(i)} \in \mathcal{A}$. Based on this solution, the objective in (7) is approximated by a linear function at $\alpha^{(i)}$ and the corresponding problem is solved as $\alpha' = g(\alpha^{(i)})$. Then the Frank-Wolfe algorithm solves a line search problem

$$\alpha^{(i+1)} = \arg\min_{c \in [0,1]} f(c\alpha^{(i)} + (1 - c)\alpha') \,,$$

where $\alpha^{(i+1)}$ is the next feasible solution. Roughy speaking, this procedure can be viewed as gradient descent on f where $\alpha^{(i+1)}$ is guaranteed to be feasible. This is a major advantage over gradient descent [2], which may require a projection to $\mathcal{A}$. The Frank-Wolfe algorithm converges when f and $\mathcal{A}$ are convex.

The Frank-Wolfe algorithm strikes an elegant balance in our setting. On one hand, the LP in (8) can represent the linear constraints in (5) and (6). On the other hand, the hard problem of eigenvalue optimization is solved by gradient descent. In what follows, we apply this approach to A- and E-optimal designs.

4.3. Frank-Wolfe E-Optimal Design

The E-optimal design in Section 3.2 maximizes the minimum eigenvalue of the sample covariance matrix,

$$\max_{\alpha \in \Delta} \lambda_{\min}(G_\alpha) \,, \tag{9}$$

where $G_\alpha = \sum_{x \in \mathcal{X}} \alpha_x x x^\top$ is a sample covariance matrix parameterized by probing distribution $\alpha \in \Delta$ and Δ is the set of all distributions over $\mathcal{X}$. Then from the definitions of $\lambda_{\min}$ and G_α, we have that

$$\begin{aligned} \lambda_{\min}(G_\alpha) &= \min_{v \in R^d : \|v\|_2 = 1} v^\top \left(\sum_{x \in \mathcal{X}} \alpha_x x x^\top \right) v \\ &= \min_{v \in R^d : \|v\|_2 = 1} \sum_{x \in \mathcal{X}} \|v^\top x\|_2^2 \, \alpha_x \,. \end{aligned}$$

Since $\sum_{x \in \mathcal{X}} \|v^\top x\|_2^2 \, \alpha_x$ is linear in α for any v, we have that $\lambda_{\min}(G_\alpha)$ is concave in α. Thus $-\lambda_{\min}(G_\alpha)$ is convex in α and we can solve (9) by the Frank-Wolfe algorithm (Section 4.2).

To instantiate the algorithm, we let

$$\mathcal{A} = \Delta \,, \quad f(\alpha) = -\lambda_{\min}(G_\alpha) \,.$$

Since $\lambda_{\min}(G_\alpha)$ is linear in α,

$$\nabla f(\alpha) = -\nabla \lambda_{\min}(G_\alpha) = -\left(\|v_{\min}^\top x\|_2^2\right)_{x \in \mathcal{X}} \,,$$

where $v_{\min}$ is the eigenvector associated with $\lambda_{\min}(G_\alpha)$.

The time to compute the gradient $\nabla f(\alpha)$ is linear in $|\mathcal{X}|$, the number of partial derivatives. Since this is the most computationally-demanding part of our implementation of the Frank-Wolfe algorithm, its run time is nearly linear in $|\mathcal{X}|$ and the number of iterations. Thus, the approach scales to large problems.

4.4. Frank-Wolfe A-Optimal Design

The A-optimal design in Section 3.3 minimizes the trace of the inverse sample covariance matrix,

$$\min_{\alpha \in \Delta} \operatorname{tr}\left(G_\alpha^{-1}\right) , \tag{10}$$

where G_α and Δ are defined as in Section 4.3. Since (10) is convex in α [2], we can solve it by the Frank-Wolfe algorithm (Section 4.2).

To instantiate the algorithm, we let

$$\mathcal{A} = \Delta\,, \quad f(\alpha) = \operatorname{tr}\left(G_\alpha^{-1}\right) .$$

For any invertible matrix $M \in R^{d \times d}$, we have that [25]

$$\partial \operatorname{tr}\left(M^{-1}\right) = \operatorname{tr}\left(\partial M^{-1}\right) = -\operatorname{tr}\left(M^{-1}(\partial M)M^{-1}\right) .$$

Now we apply this identity to $M = G_\alpha$ and note that the partial derivative of G_α with respect to α_x is $\partial G_\alpha / \partial \alpha_x = x x^\top$. This yields

$$\nabla f(\alpha) = -\left(\operatorname{tr}\left(G_\alpha^{-1} x x^\top G_\alpha^{-1}\right)\right)_{x \in \mathcal{X}} .$$

4.5. Generalized Linear Models

Generalized linear models (GLMs) [21] extend linear models to non-linear functions, in a way that inherits their attractive statistical and computational properties. Specifically, in the GLM, the measurement $y \in R$ at feature vector $x \in \mathcal{X}$ has an exponential-family distribution with mean $f_*(x) = \mu(x^\top \theta_*)$, where μ is the *mean function* and $\theta_* \in R^d$ are model parameters [21]. For any mean function μ, it is well known that the matrix of second derivatives of the GLM loss at solution θ_*, also called the Hessian, is

$$H = \sum_{i=1}^{n} \dot{\mu}(x_i^\top \theta_*) x_i x_i^\top , \tag{11}$$

where $\dot{\mu}$ is the derivative of the mean function μ. This Hessian plays the same role as the sample covariance matrix G in (4) [5]. Hence, up to the factor of $\dot{\mu}(x_i^\top \theta_*)$, the optimal designs in Sections 3.2 and 3.3 optimize the maximum and average errors (Section 2) in GLMs.

We use Poisson regression, an instance of GLMs, in Section 5.4 to estimate packet losses. In this problem, $f_*(x) = \exp[x^\top \theta_*]$ is the probability that the packet is not dropped on path $x \in \mathcal{X}$ and $\theta_* \in R^d$ is a vector of log-probabilities that packets are not dropped on individual path edges.

5. Synthetic Experiments

In this section, we evaluate the quality of the estimates produced by the Frank-Wolfe probing distributions. All experiments are on simulated data. We first introduce (Section 5.1) the nomenclature required to understand the setup, experiments, results, and evaluations. Next we describe the common experimental setup in detail (Section 5.2) followed by the specifics of latency experiments and results (Section 5.3), and the packet loss experiments and results (Section 5.4). Finally, we study the impact of local probe budgets on our methods (Section 5.5).

5.1. General Setup

Network Graph. A network topology is converted to a network graph comprised of nodes and edges. The nodes in this graph are points of presence (POPs), which are a collection of routing and networking equipment. The edges are the physical links between the POPs. All sources and destinations in a POP are coalesced into a single POP node. This is a simplification of the network topologies that we want to study and far smaller than major cloud infrastructures. However, it is representative of the connectivity encountered in these networks, which is sufficient for demonstrating the strengths of our approaches. We denote the number of edges in the network by d and perform our experiments using the following 3 network topologies:

- Topology A: 101 nodes, 251 edges, $5\,050$ paths.
- Topology B: 82 nodes, 196 edges, $3\,321$ paths.
- Topology C: 192 nodes, 439 edges, $15\,428$ paths.

Note that these three topologies have very similar normalized edge (average for A/B/C: 3.71/3.6/4.37) and node degrees (average for A/B/C: 4.97/4.78/4.57), making it difficult to interpret the impact of these graph properties on the performance of the methods studied in this paper. The normalized average edge degree is defined as the average edge degree (average number of paths an edge appears on) divided by the ratio of total number of paths over total number of edges. The average node degree is defined as the average number of edges of the node.

Metrics of Interest. We experiment with two metrics: **latency** and **packet loss**. For latency, we estimate the time of a packet to traverse a path in a network. For packet loss, we estimate the probability of a packet being dropped at any point on a path. The shades areas in our plots represent standard error of the estimates.

Budget Constraints. We assume a global budget of n probes per unit time. Since the quality of metric estimates improves with more probes (Sections 5.3 and 5.4), all compared methods probe exactly n times. We experiment with local budget constraints in Section 5.5.

Baseline I (`QR`). The algorithm presented in [6] is a natural baseline for comparison because [6] tries to solve a similar problem, choosing the best subset of paths to probe. This algorithm works as follows. First, we compute the singular value decomposition (SVD) of the path-edge matrix M, $M = U\Sigma V^{\top}$, where U and V are orthonormal matrices.

Next we choose the first $k = \mathrm{rank}(M)$ columns of U, denoted by U_k, and apply the pivoting QR decomposition to it. The first k chosen rows of U_k represent important rows of M, paths that explain most information contained in M. The budget is then spread evenly across these k chosen paths.

Baseline II (`Prod`). `Prod` spreads the budget of n measurements evenly across all paths. It is a natural point of comparison because it also produces a probing distribution without prior knowledge of the traffic on the network. The shortcoming of `Prod` is that it over-probes links that appear in many paths and under-probes those that appear in a few. For additive metrics, such as latency, each link in the network can be viewed as a model parameter. Therefore, the accuracy of estimated model parameters using `Prod` probing can vary a lot. Our methods are designed to precisely address this issue, and compute the probing distribution to *optimize* the accuracy of learned model parameters.

We remind the reader that the time spacing of probes on a path is outside the scope of this paper.

Evaluation Criteria. We use the *maximum error* in (1) and *average error* in (2) in all experiments in this section. The maximum error is the highest error over a set of paths. The average error is measured on the same paths and reflects the average performance. For the maximum error in (1), the set of paths is $\mathcal{X}$. Therefore, this error corresponds to the most mispredicted path latency/loss probability. For the average error in (2), $\mathcal{P}$ is a distribution over paths $\mathcal{X}$. We design $\mathcal{P}$ such that we can evaluate the quality of learned models uniformly over all of their parameters. In particular, $\mathcal{P}$ is defined by the following generative process. First, we choose an edge, uniformly at random. Then we choose any path with that edge, uniformly at random. This choice guarantees that any edge appears in a sampled path from $\mathcal{P}$ with probability at least $1/d$. That is, for any $i \in [d]$, $P_{x \sim \mathcal{P}}\left(x(i) = 1\right) \geq 1/d$, where $x(i)$ is the i-th entry of vector x.

Trade-offs. Our approach trades off measurement accuracy and probing budget, as it is not always feasible in large scale networks to measure with the maximum accuracy. We do that with theoretical guarantees. Therefore, when compared to the baselines, we look for *achieving a fixed level of accuracy with lower budget or achieving a higher accuracy with a fixed budget*. Naturally, the higher the probing budget, the higher the accuracy, for both our approach and the baselines.

5.2. Evaluation Methodology

The set of paths $\mathcal{X}$ contains all the shortest paths in the network. For each metric of interest, we have a vector of ground truth model parameters $\theta_* \in R^d$ and we conduct an experiment that compares four probing distributions, essentially calculating one vector of estimated model parameters $\hat{\theta} \in R^d$ for each. The first probing distribution from the E-optimal design (Section 3.2), solved using the Frank-Wolfe algorithm (Section 4.3). We call it `E-optimal`. The second probing distribution is from the A-optimal design (Section 3.3), also solved using the Frank-Wolfe algorithm (Section 4.4). We call it `A-optimal`. In both methods, the number of Frank-Wolfe iterations is 300, which is sufficient to obtain near-optimal solutions in all of our experiments. The last two methods are the `QR` and `Prod` baselines.

All methods operate under a total budget of n measurements, with n varying from $3,000$ to $30,000$. The choice of n in our experiments may seem low. In practice, probing budgets vary and can be significantly higher. Our chosen

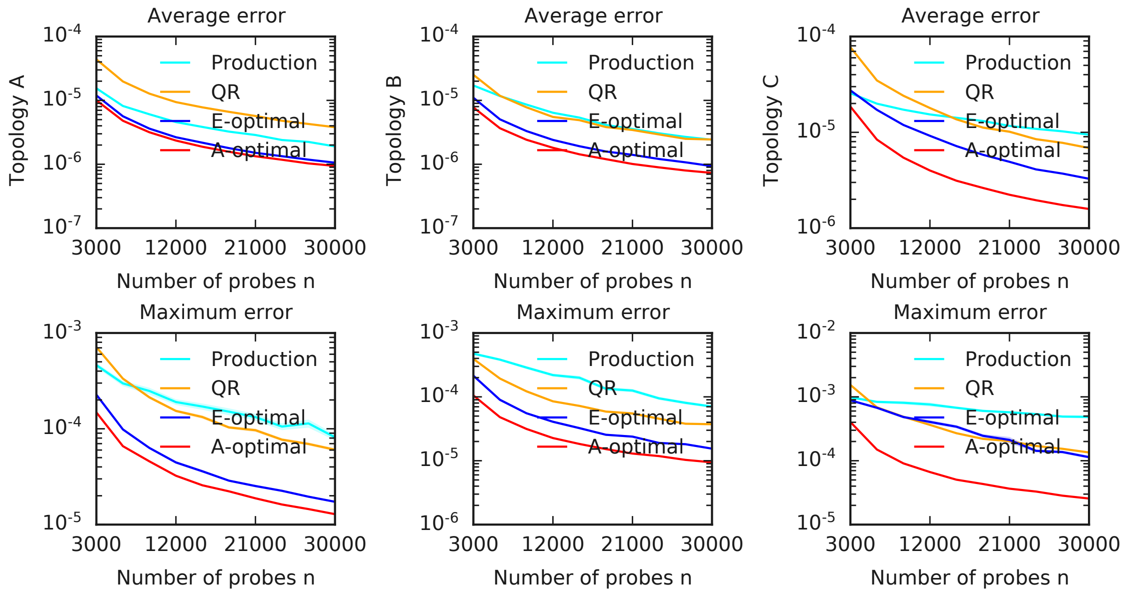

Figure 1: Latency estimation errors in three network topologies. The average errors are in the first column and the maximum errors are in the second.

budgets allow us to study the estimation error in situations where the budget is more scarce, which is typically when it is most important to know the measurement error.

For each probing distribution and budget n, we report the average of 300 independent runs of the following end-to-end experiment:

1. Compute a probing distribution $\alpha = (\alpha_x)_{x \in \mathcal{X}}$ (line 2 in Algorithm 1). For `E-optimal` and `A-optimal`, we solve (9) and (10), respectively; for `QR`, we set $\alpha_x = 1/k$ if path x is among the chosen paths by `QR` and zero otherwise; and for `Prod`, we set $\alpha_x = 1/|\mathcal{X}|$ for each path $x \in \mathcal{X}$.

2. Collect dataset $\mathcal{D}$ by probing paths $\mathcal{X}$ according to α (lines 4-7 in Algorithm 1). The noisy observations y_i for feature vectors x_i are generated as described in Sections 5.3 and 5.4.

3. Estimate model parameters $\hat{\theta} \in R^d$ from $\mathcal{D}$ (line 8 in Algorithm 1), using least-squares regression (Section 3.1) for latency and Poisson regression (Section 4.5) for packet loss.

4. Use $\hat{\theta}$ and θ_* to compute the maximum (1) and average (2) errors.

5.3. Latency Experiments

Generating Noisy Latency Observations. Let $\ell_*(e)$ be the mean latency of edge e in seconds. We calculate $\ell(e)$ as the physical distance between the nodes of edge e divided by the speed of light in fiber, which is approximately one

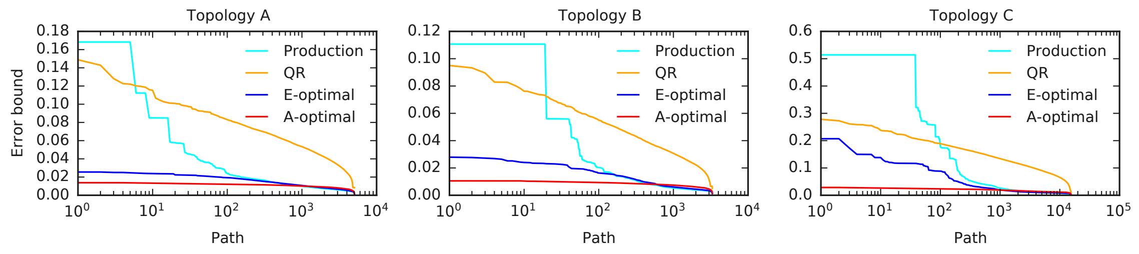

Figure 2: Predicted errors in (4), in the descending order of the per-path errors for each method. The x-axis is on the logarithmic scale to highlight highest predicted errors, which contribute the most to the actual errors.

third of the speed of light in vacuum. The edge latency varies from 0.001s to 0.025s (topology A), 0.036s (topology B), 0.011s (topology C).

We formulate our estimation problem as follows. The mean latency of edge e is $\theta_*(e) = \ell(e)$ and $\theta_* \in R^d$ is the vector of mean latencies of all edges. The mean latency of path x is $f_*(x) = x^\top \theta_*$, where $x \in \{0,1\}^d$ is a vector of edge indicators. The *noisy latency observation* of path x is $y = x^\top \theta_* + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise. The noise ε is a *measurement noise* and we set $\sigma = 0.01$, based on prior experience with latency variance.

Results. All latency estimation errors are reported in Figure 1. We observe four major trends. First, for all methods, the errors decrease as budget n increases. This is expected, since the quality of latency estimates improves with the number of probes. Second, the rate of the error decrease, from $n = 3\,000$ to $n = 30\,000$, is about 10 fold. This follows from how the squared error relates to the sample covariance matrix G in (4). In particular, for any probing distribution, 10 times more probes yield 10 times higher eigenvalues of G and thus, 10 times lower eigenvalues of G^{-1}; which then provide a 10 times lower error bound in (4). Third, the reported errors are acceptable, even for small probing budgets. For instance, in all network topologies, the average squared error of `A-optimal` with budget $n = 30\,000$ is around 10^{-6}. This means that the absolute path latency is mispredicted by 10^{-3} on average, which is at least an order of magnitude less than the range of the latencies in our networks.

A fourth observed trend is that `A-optimal` consistently dominates `E-optimal` and `E-optimal` consistently dominates the baselines (`QR` and `Prod`). That is, `E-optimal` has a lower error than `QR` and `Prod` for all budgets and attains any fixed error with a lower budget than either of them. We observe this trend for both the average and maximum errors, and over all network topologies. `A-optimal` dominates `E-optimal` in a similar fashion. Perhaps surprisingly, `E-optimal` performs worse than `A-optimal` in minimizing the maximum error. We hypothesize that this is because the path with the maximum potential error, the maximum eigenvector of G^{-1}, is not among paths $\mathcal{X}$. Last, given all topologies have very similar node and edge degree, it is not possible to study the impact of these network topological properties on the methods presented in this paper.

Our next step is to better understand why `A-optimal` performs so well and why `QR` performs so poorly. To do so, we use the error bound in (4). For a given method and path, the error bound is the maximum high-probability error for latency prediction of that path by that method. A lower predicted error should correlate with a higher accuracy of latency estimates. Therefore, if one method is more accurate than another, we would expect most of its predicted errors to be lower. To visualize this, we show the predicted errors of all paths for each method in Figure 2, in the descending order of the per-path errors. We observe that the `A-optimal` predicted errors are lower overall than those of `E-optimal`, which are lower overall than those of `Prod`. The predicted errors of `QR` are the highest, which indicates poor performance. This is consistent with the actual errors in Figure 1. We conclude that the optimization of (4), which is done in both `E-optimal` and `A-optimal`, leads to better performance.

Finally, we comment on the average run times of computing the Frank-Wolfe `E-optimal` designs: 66s (topology A), 57s (topology B), and 53s (topology C). This is at least two orders of magnitude lower than if we computed them exactly (Section 4.1): 4 400s (topology A), 1 800s (topology B), and 31 700s (topology C). The Frank-Wolfe `A-optimal` designs are computed in 140s (topology A), 121s (topology B), and 219s (topology C). We summarize the run time differences from all experiments in this paper in Appendix Appendix C.

5.4. Packet Loss Experiments

Generating Noisy Loss Observations. We denote by $\theta_*(e)$ the log-probability that the packet is not dropped on edge e and by $\theta_* \in R^d$ all edge log-probabilities. So $\exp[\theta_*(e)]$ is the probability that the packet is not dropped on edge e. Using the mean latency of edge e in Section 5.3, $\ell(e)$, we define $\theta_*(e)$ as

$$\theta_*(e) = -\frac{\ell(e)}{10 \max_{e \in [d]} \ell(e)} \,.$$

This means that longer edges are more likely to lose packets. This is simply a modeling choice to generate edges with variable losses. By definition, $\theta_*(e) \in [-0.1, 0]$ and thus $\exp[\theta_*(e)] \in [0.9, 1]$. So the probability of dropping a packet on any edge is at most 0.1.

As in Section 5.3, x is a vector of edge indicators in a path. Thus $f_*(x) = \exp[x^\top \theta_*]$ is the probability that the packet is not dropped on path x, under the assumption that the events of dropping packets on path edges are independent. The *noisy packet-loss observation* of path x is $y \sim \mathrm{Ber}(f_*(x))$, an indicator that the packet is not dropped with mean $f_*(x)$. Note that estimation of θ_* can be solved by Poisson regression, an instance of GLMs (Section 4.5). The rest of the experimental setup is the same as in Section 5.3.

Results. Packet-loss estimation errors are reported in Figure 3. These results are consistent with those described earlier for latency (see Figure 1). In particular, we observe that the errors decrease as budget n increases and that the rate of this decrease is inversely proportional to the budget. That is, from budget $n = 3\,000$ to $n = 30\,000$, the decrease is also about 10 fold. Such a good result is surprising, since `E-optimal` and `A-optimal` minimize the average and maximum errors in GLMs only loosely, as discussed in Section 4.5. We also observe that the reported errors are acceptable. For instance, in all network topologies, the average squared error of `A-optimal` with budget $n = 30\,000$ is below 5×10^{-4}. This means that the absolute path loss is mispredicted by 2×10^{-2} on average, which is nearly an order of magnitude lower than the maximum packet loss on an edge, which is 0.1 by our modeling assumptions.

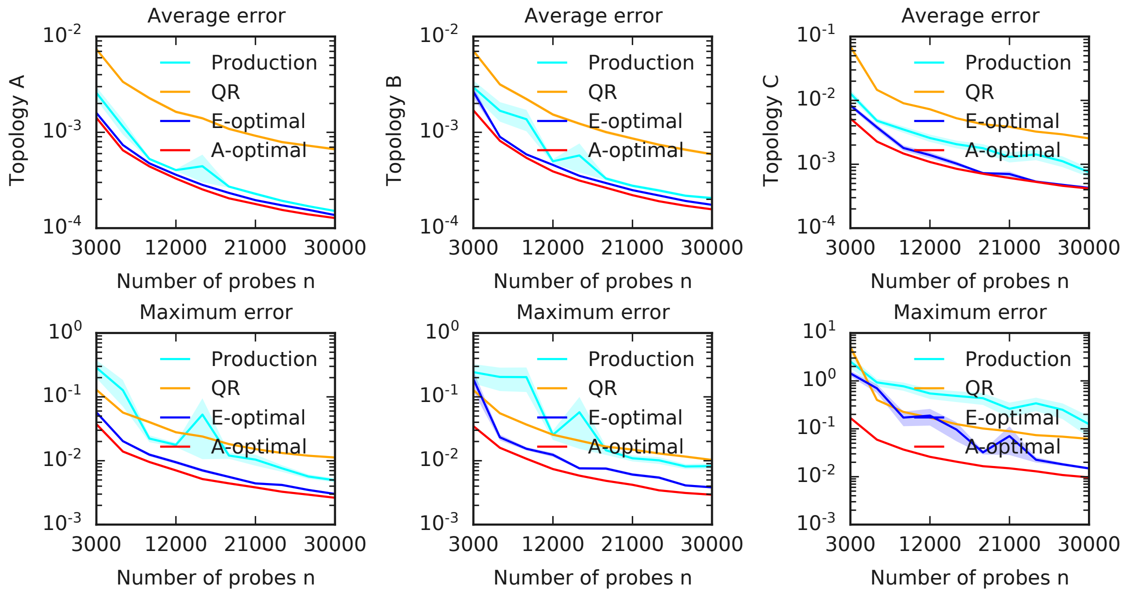

Figure 3: Packet-loss estimation errors in three network topologies. The average errors are in the first column and the maximum errors are in the second.

We also observe that `E-optimal` and `A-optimal` consistently dominate both `QR` and `Prod`. In particular, they have lower errors than these baselines for all budgets and attain any fixed error with a lower budget than either of them. We observe this trend for both the average and maximum errors, and over all network topologies. The differences between `E-optimal` and `A-optimal` are less pronounced than in Figure 1, although clearly `A-optimal` never performs much worse than `E-optimal`.

5.5. Local Budget Experiments

The goal of this section is to evaluate local budget constraints described in Section 3.4. We consider the most common constraint in network monitoring, that none of the nodes in the network is a source or destination of "too many" probes. This constraint can be incorporated in our optimal designs as follows. Let $\mathcal{V}$ be the set of all nodes in the network and $\mathrm{src}(x)$ be the source of path x. Then, for each $v \in \mathcal{V}$, the constraint is

$$\sum_{x \in \mathcal{X}} \mathbb{1}\{\mathrm{src}(x) = v\} \, \alpha_x \leq \frac{\sum_{x \in \mathcal{X}} \mathbb{1}\{\mathrm{src}(x) = v\}}{|\mathcal{X}|} + b \,,$$

where $b \geq 0$ is an *excess local budget* over uniform probing distribution and $\sum_{x \in \mathcal{X}} \mathbb{1}\{\mathrm{src}(x) = v\} \,/\, |\mathcal{X}|$ is the fraction of paths starting in node v. The destination constraints are defined analogously.

We experiment on topology A with three local budgets $b \in \{0.1, 0.01, 0.001\}$. The latency estimation results are reported in Figure 4. We expect that lower local budgets would result in worse optimal designs, because both `E-optimal` and `A-optimal` are more constrained in how they can probe. We observe this with the maximum error at

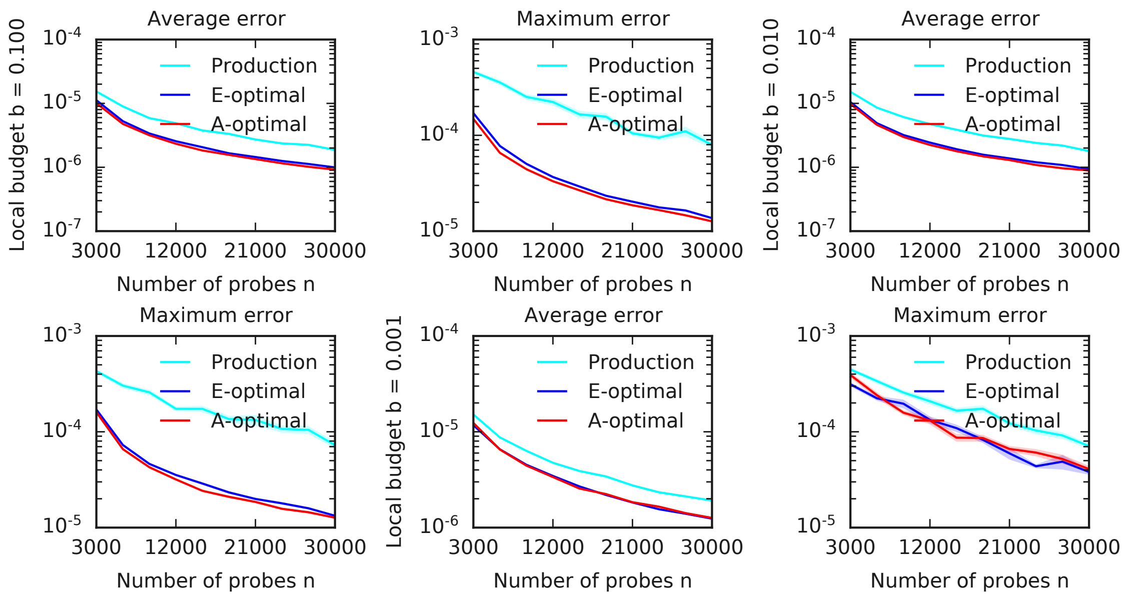

Figure 4: Latency estimation errors in topology A for various local budgets. The average errors are in the first column and the maximum errors are in the second.

$n = 30\,000$, which increases from 10^{-5} ($b = 0.1$) to 3×10^{-5} ($b = 0.001$). We also observe slight changes in the average error, which increases from slightly below 10^{-6} ($b = 0.1$) to slightly above 10^{-6} ($b = 0.001$).

In Figure 5, we show the predicted errors for latency and various local budgets. This plot is similar to Figure 2. When the local budget is low ($b = 0.001$), the probing distributions in `E-optimal` and `A-optimal` cannot differ much from `Prod`, and some paths have high predicted errors of 0.1. This leads to lower accuracy. On the other hand, when the local budget is high ($b = 0.1$), the probes can be distributed more intelligently and this results in the highest predicted errors of mere 0.02, similarly to no local budget constraints (topology A in Figure 2).

We conclude that our approach can easily incorporate local budget constraints. Perhaps surprisingly, even when the local budget is $b = 0.01$ (which means that no more than 1% of the probed paths can start or end at any given node) the corresponding `E-optimal` and `A-optimal` solutions offer similar performance as with global budget only.

6. Real-World Experiments

In Section 5, the ground truth and its observations were generated from *well-behaved* functions with desirable properties. For instance, the mean latency of a path in Section 5.3 is a linear function of its edge latencies, and the latency of that path is observed with Gaussian noise. Real latencies and packet losses are not always *well-behaved*. We conduct a set of larger-scale *real-world experiments*, with actual observed latencies and packet losses. We also use this section to show that our Frank-Wolfe solutions can be used in practice.

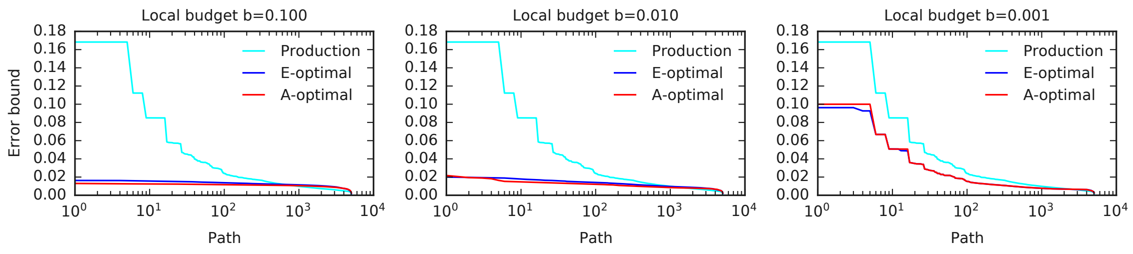

Figure 5: Predicted errors in (4), in the descending order of the per-path errors for each method. The x-axis is on the logarithmic scale to highlight highest predicted errors, which contribute the most to the actual errors. The plots are for topology A and various local budgets.

6.1. *Network Topology*

All experiments in this section are conducted on a single network topology, which is a random subset of a production network that belongs to a global cloud services provider. We cannot reveal the percentage of the network that our topology represents. However, our experimental network is sufficient in size and realistic in complexity: 306 nodes, 762 edges, and 39 316 paths. This network is larger than any of the networks in Section 5 and brings additional evidence that our approach can be applied at scale in a cloud network.

6.2. *Experimental Setup*

Our setup mimics a real-world scenario where the probing distribution is recomputed once per unit time, such as one or five minutes. Then the probes are sent and they generate *observations* for each path. Finally, these observations are used to estimate the latency or packet loss of each path in the network, whether or not that path is probed. As in the synthetic experiments, we compare our optimal designs to the `QR` and `Prod` baselines.

6.3. *Ground Truth*

The main challenge in evaluating a real world setting is that the ground truth, the actual latency or packet loss of a path, is never perfectly known. However, in order to compare different methods, we need some notion of it. In this work, we determine the ground truth as follows. First, we fix a short period of (unit) time. We experiment with one and five minute periods. We refer to the 1-minute experiment as *topology M1* and to the 5-minute experiment as *topology M5*. Next, over that fixed time period, we send 50 to 100 probes along *all paths* in the subset of the network chosen. In total, this translates to over 4 million probes. Then we determine the mean latency and packet loss of a path as the empirical average of its observations generated by the probes. For a given path x, these correspond to the ground truth latency and packet loss $f_*(x)$ in Sections 5.3 and 5.4, respectively.

Our choice of short time periods is not arbitrary: it allows us to determine the ground truth accurately. Over longer time periods, the metrics of interest, especially packet loss, would smooth out and not accurately represent *near real-time* and highly variable values.

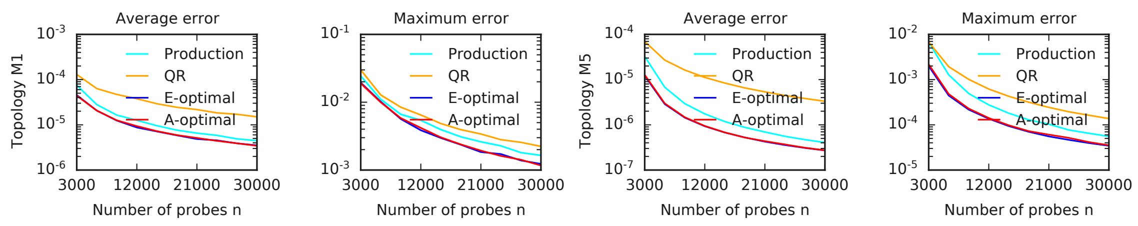

Figure 6: Latency estimation errors in large-scale networks M1 and M5.

6.4. *Emulating Observations*

As described earlier, we send 50 to 100 probes along all paths in the network over a short period of time and collect observations. Then we emulate the behavior of our methods on these observations. As an example, suppose that we have 100 observations of paths x_1, x_2, and x_3; and the emulated `A-optimal` decides on a distribution of probes such that 6, 9, and 0 probes (per unit time) are assigned to each of these paths. Then the observations are obtained by sampling 6 random observations from the set of 100 collected observations for path x_1 and 9 from those for path x_2.

6.5. *Results*

Latency and packet loss estimation results are reported in Figures 6 and 7. These results are consistent with the trends observed in the synthetic experiments (Sections 5.3 and 5.4). Specifically, the errors decrease as budget n increases, and `E-optimal` and `A-optimal` consistently dominate both the baselines (`QR` and `Prod`). For latency, in all network topologies, the average squared error of A-optimal with budget $n = 30\,000$ is around 4×10^{-6}. This means that the absolute path latency is mispredicted by 2×10^{-3} on average. For packet loss, the average squared error of `A-optimal` with budget $n = 30\,000$ is below 2×10^{-4}, which means that the absolute path loss is mispredicted by 10^{-2} on average. Both absolute errors are one order of magnitude lower than the estimated quantities, which means that our predictions are reasonably accurate.

The error differences among `E-optimal`, `A-optimal`, `QR`, and `Prod` are less pronounced than in Section 5. One reason is that the ground truth is not idealized. As an example, the path latency is not necessarily linear in its features, which contributes to an additional error that could only be eliminated by a better estimation model than linear regression.

6.6. *Run Times*

The costs of computing `E-optimal` and `A-optimal` are 346 and 961 seconds, respectively. This is six times higher than in the synthetic experiments, for a topology that is six times larger. This is expected, as the computation cost of our Frank-Wolfe solutions is linear to the number of paths[1]. While this may seem too long at first, we consider it to

[1]The run time differences of all experiments in this paper are summarized in Appendix Appendix C.

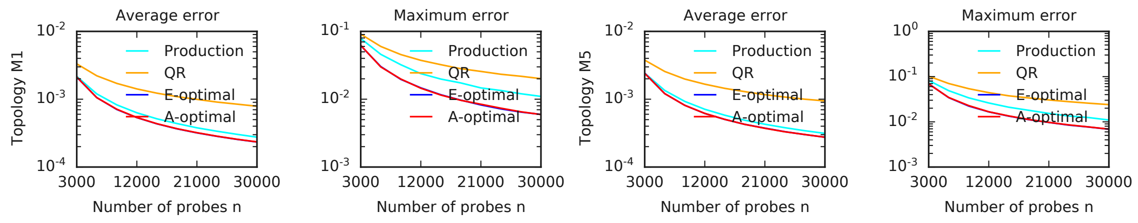

Figure 7: Packet-loss estimation errors in large-scale networks M1 and M5.

be practical. In particular, since our network routes are usually stable over 10 minute intervals, we do not expect to recompute the probing distributions more frequently than that.

7. Related Work

Monitoring systems for large scale networks is a rich area. Our focus here is on systems that can scale to the size of the main cloud infrastructures (not only to one Data Center) and that can provide multiple network performance metrics with a known accuracy at all time, whatever the resources available to the telemetry systems are. We are going to address large scale telemetry systems first, and then discuss our approach and methodology.

Most recently published systems attempt to monitor the network infrastructure at full coverage (e.g. [20, 18, 32, 19, 34, 22, 16**?**]). These systems are generally designed for data centers and they make the assumptions that they operate with a fixed amount of resource (memory, CPU) in network nodes and/or end hosts. In practice, we claim that due to hardware diversity and operational practices (in most cloud networks, telemetry operates under a maximum budget in CPU time or probing/sampling frequency), these assumptions cannot be verified and full coverage cannot be achieved, jeopardizing the quality of the estimated metrics. This explains why we adopt a statistical approach in which network performance metrics can be estimated with a known accuracy, whatever the available resources and monitoring budget are. Compared to the approaches cited above, we do not make any assumption on minimum CPU or memory in the nodes in the network. We can operate with any probing/sampling budget, which could lead to low (but known) accuracy of our estimations. In this landscape, NetBouncer [29] is adopting a statistical approach for failure inference (it does not support any other metric). It relies on specific hardware to be deployed in switches (to support IP in IP tunneling) and is limited to Data Center monitoring. The approach we propose in this work uses a different optimization technique and can estimate multiple performance metrics with a know accuracy.

Our work also pertains to the area of network tomography [3, 12, 17, 13, 23]. Tomography has focused historically on the theoretical (e.g. [9, 33]) and practical feasibility [**?**] of loss and latency measurements. Very often though, tomography papers deal with loss estimation and/or do not focus on optimal probing but on high quality inference. There are exceptions discussed below ; none of the papers we are aware of are dealing with the general framework approach as we define it, building a scalable implementation and evaluating it experimentally. In [30], the optimal sampling problem is discussed using an EM algorithm. Our statistical framework is more general as (i) it allows the estimation of network performance metrics with measurable accuracy and (ii) it only uses the network topology without requiring any traffic observations. In [15], the authors use A- and D-optimal designs to compute probing distributions to estimate loss and delay variation. The problem statement and the methodology share similarities with our work. However, the two papers have different objectives. More importantly, [15] does not design a scalable implementation, which is the key to make these techniques applicable in practice, and as a consequence performance evaluation is limited to numerical simulations.

Our work is also related to optimal monitor placement [4]. The objectives of [4] are quite similar to ours, as the authors try to select where and how frequently to sample traffic in network nodes in order to minimize monitoring resource consumption, and adapt to the traffic dynamics (that it takes as an input to the optimization algorithm). The method is quite different though (gradient projection for constrained optimization), and while we optimize the probing rate to estimate network performance metrics on all network links, they optimize the sampling rate in network nodes to estimate the performance of a specific set of Origin-Destination (OD) pairs. The method in [4] takes the network topology and the routing matrix (i.e. which OD pair take with path) as input; it returns a set of monitors and their sampling rates that are optimal with respect to the set of OD pairs to measure. Measurement accuracy is not estimated.

A closely related work is that of Chua et al. [6], where the QR decomposition of the traffic matrix is used to extract a set of *important paths*. Roughly speaking, if the mean latency of these paths was known, we could estimate the latency of any other path in the network. This method is not designed for noisy measurements and we compare to it in Sections 5 and 6.

We also did not find any previous work discussing the trade-off between probing/sampling budget and performance metric measurement accuracy, and its application to monitoring very large cloud networks.

8. Conclusions

Starting from the claim that monitoring traffic exhaustively is both not necessary and not feasible, we propose a statistical approach to monitoring (using probes or sampling) large-scale network infrastructure, which has provable guarantees on the quality of estimates under the constraint of a limited probing budget. We show through simulation and experimentation on *real* network topologies (supporting global cloud services) and for two metrics (latency and loss) that our approach estimates the metrics with low error, even in the case of small probing budgets. We claim that our approach can be used in any production network to measure most network performance metrics in a scalable way (as long as the metric estimation problem can be expressed as a regression problem). Our results show that the computation time of optimal strategies is not an obstacle to operational deployment. Our approach does not make any assumption on switch/router hardware capabilities. It can operate in two modes: target budget or target measure accuracy. We also show that we outperform two baselines:(i) a best path subset selection algorithm proposed in [6] and (ii) a *simple* strategy that probes evenly across all paths. Our approach and results can be easily extended to traffic sampling as long as the traffic matrix is available. Although our initial results are promising, we still have a long path toward deploying our system in production networks.

First, existing monitoring systems support various tasks, from traffic engineering (managing the traffic based on policies) to security (detection of various attacks) and service availability (detection of unexpected events and failures). We showed that our framework is well suited for optimizing probing for network link loss and latency estimation. In future work, we will evaluate our framework on other telemetry tasks, such as estimating available bandwidth, flow size distribution, or top flows. We believe such extension is straightforward as long as the estimation problem can be formulated as regression (Sections 3.1 and 4.5). In addition, in Appendix Appendix A, we propose learning of a non-linear embedding that would transform potentially any estimation problem into regression, with appropriate features. Then we would be able to apply A- and E-optimal designs on the new learned features.

Second, the current work only deals with distributing a probing budget among the paths of interest in a network. In other words, it answers the question "where to probe and how much" before the actual probing is performed and is therefore independent of the mechanism according to which the number of probes are carried out per unit of time. Examples of such mechanisms (which we treat as black-box in this work) in the literature include *uniform sampling* and *Poisson sampling* [27], where inter-probe send times have the uniform or Poisson distribution, respectively. As a result, it is straightforward to combine our framework with any such probing mechanism in a 2-stage pipeline: (i) one first computes the number of probes that will be sent along each path, and (ii) the probing mechanism sends the probes and collects the measurements.

Third, the topologies we have studied represent subsets of the original cloud infrastructure topology. We cannot disclose details on the full size and corresponding computing time. However, we will rely on some "sharding" techniques to divide the full infrastructure topology (data center included as we probe from hosts) into geographical entities. This is not a problem though as it is the way Cloud services are structured.

Our current objective is to build an operational probing/sampling service that takes the following inputs: a topology, a path level (i.e. OD pairs) traffic matrix with packet count, a telemetry function (e.g. median latency, flow size distribution, link failures), and either a probing/sampling budget or a target statistical accuracy for the metric to compute. It will deliver an optimal sampling/probing strategy to perform in hosts.

We believe controlling the accuracy of the performance estimation is the only way to monitor at scale and at low time granularity in today's hyper-scale networks.

References

[1] Anthony Atkinson and Alexander Donev. 1992. *Optimum Experimental Designs*. Oxford University Press.

[2] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, USA.

[3] Tian Bu, Nick Duffield, Francesco Lo Presti, and Don Towsley. 2002. Network Tomography on General Topologies. *SIGMETRICS Perform. Eval. Rev.* 30, 1 (June 2002), 21–30. https://doi.org/10.1145/511399.511338

[4] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. 2006. Reformulating the Monitor Placement Problem: Optimal Network-Wide Sampling. In *2006 40th Annual Conference on Information Sciences and Systems*. 1725–1731. https://doi.org/10.1109/CISS.2006.286433

[5] Kani Chen, Inchi Hu, and Zhiliang Ying. 1999. Strong Consistency of Maximum Quasi-Likelihood Estimators in Generalized Linear Models with Fixed and Adaptive Designs. *The Annals of Statistics* 27, 4 (1999), 1155–1163.

[6] David B. Chua, Eric D. Kolaczyk, and Mark Crovella. 2005. A Statistical Framework for Efficient Monitoring of End-to-End Network Properties. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems*. ACM, 390–391.

[7] E. de Klerk. 2002. *Aspects of semidefinite programming: Interior point algorithms and selected applications*. Number 65 in Applied optimization, ISSN 1384-6485. Kluwer Academic Publishers, Netherlands. Pagination: xvi, 283.

[8] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research* (2016). http://stanford.edu/ boyd/papers/pdf/cvxpy$_paper.pdf Toappear.$

[9] N. G. Duffield, F. Lo Presti, V. Paxson, and D. Towsley. 2001. Inferring link loss using striped unicast probes. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, Vol. 2. 915–923 vol.2. https://doi.org/10.1109/INFOCOM.2001.916283

[10] Rick Durrett. 2010. *Probability: Theory and Examples* (4 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511779398

[11] Marguerite Frank and Philip Wolfe. 1956. An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly* 3, 1-2 (1956), 95–110.

[12] Denisa Ghita, Katerina Argyraki, and Patrick Thiran. 2013. Toward Accurate and Practical Network Tomography. *SIGOPS Oper. Syst. Rev.* 47, 1 (Jan. 2013), 22–26. https://doi.org/10.1145/2433140.2433146

[13] D. Ghita, H. Nguyen, M. Kurant, K. Argyraki, and P. Thiran. 2010. Netscope: Practical Network Loss Tomography. In *2010 Proceedings IEEE INFOCOM*. 1–9. https://doi.org/10.1109/INFCOM.2010.5461918

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA.

[15] Ting He, Chang Liu, Ananthram Swami, Don Towsley, Theodoros Salonidis, Andrei Iu. Bejan, and Paul Yu. 2015. Fisher Information-Based Experiment Design for Network Tomography. *SIGMETRICS Perform. Eval. Rev.* 43, 1 (June 2015), 389–402. https://doi.org/10.1145/2796314.2745862

[16] Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 576–590. https://doi.org/10.1145/3230543.3230559

[17] Yiyi Huang, Nick Feamster, and Renata Teixeira. 2008. Practical Issues with Using Network Tomography for Fault Diagnosis. *SIGCOMM Comput. Commun. Rev.* 38, 5 (Sept. 2008), 53–58. https://doi.org/10.1145/1452335.1452343

[18] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI '16)*. USENIX Association, USA, 311–324.

[19] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and General Sketch-Based Monitoring in Software Switches. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 334–350. https://doi.org/10.1145/3341302.3342076

[20] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 101–114. https://doi.org/10.1145/2934872.2934906

[21] P. McCullagh and J. A. Nelder. 1989. *Generalized Linear Models*. Chapman & Hall.

[22] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and Precise Triggers in Data Centers. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 129–143. https://doi.org/10.1145/2934872.2934879

[23] H. X. Nguyen, R. Teixeira, P. Thiran, and C. Diot. 2009. Minimizing Probing Cost for Detecting Interface Failures: Algorithms and Scalability Analysis. In *IEEE INFOCOM 2009*. 1386–1394. https://doi.org/10.1109/INFCOM.2009.5062054

[24] Andrej Pazman. 1986. *Foundations of Optimum Experimental Design*. Springer Netherlands.

[25] K. B. Petersen and M. S. Pedersen. 2012. The Matrix Cookbook. (nov 2012). http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html Version 20121115.

[26] Friedrich Pukelsheim. 1993. *Optimal Design of Experiments*. John Wiley & Sons.

[27] Matthew Roughan. 2006. A Comparison of Poisson and Uniform Sampling for Active Measurements. *IEEE Journal on Selected Areas in Communications* 24, 12 (2006), 2299–2312. https://doi.org/10.1109/JSAC.2006.884028

[28] G. Strang. 2016. *Introduction to Linear Algebra*. Wellesley-Cambridge Press. https://books.google.com/books?id=efbxjwEACAAJ

[29] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 599–614. https://www.usenix.org/conference/nsdi19/presentation/tan

[30] Y. Tsang, M. Coates, and R. Nowak. 2001. Passive network tomography using EM algorithms. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, Vol. 3. 1469–1472 vol.3. https://doi.org/10.1109/ICASSP.2001.941208

[31] Lieven Vandenberghe and Stephen Boyd. 1999. Applications of Semidefinite Programming. *Applied Numerical Mathematics* 29, 3 (1999), 283–299.

[32] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 561–575. https://doi.org/10.1145/3230543.3230544

[33] Yolanda Tsang, M. Coates, and R. D. Nowak. 2003. Network delay tomography. *IEEE Transactions on Signal Processing* 51, 8 (2003), 2125–2136. https://doi.org/10.1109/TSP.2003.814520

[34] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. 2015. Packet-Level Telemetry in Large Datacenter Networks. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 479–491. https://doi.org/10.1145/2829988.2787483

Appendix

Appendix A. Non-Linear Models

To extend our ideas to arbitrary non-linear models, we propose learning a non-linear mapping of features into a lower r-dimensional space, so that we get a linear regression problem in the new space. That is, we learn $g : R^d \to R^r$ such that $f_*(x) \approx g(x)^\top \theta_*$.

The function g is learned as follows. Let m be the number of tasks and $\mathcal{D}_j$ be the dataset corresponding to task $j \in [m]$. In networking, the task j can be viewed as an inference problem on day j, which is accompanied by its training set $\mathcal{D}_j$. We learn g by solving

$$\min_g \min_{\theta_1,\ldots,\theta_m} \sum_{j=1}^m \sum_{(x,y)\in\mathcal{D}_j} \left(g(x)^\top \theta_j - y\right)^2 , \tag{A.1}$$

where $\theta_j \in R^r$ are optimized model parameter for task j.

The key structure in above loss function is that function g is shared among the tasks. Therefore, its minimization leads to learning a common embedding g, essentially a compression of features x, that is useful for solving all tasks using linear regression. The above optimization problem can be solved using a multi-headed neural network, where g is the body of the network and head j outputs $g(x)^\top \theta_j$, the predicted value at feature vector x for task j.

Appendix B. Scaling Optimal Designs

Appendix B.1. Intuition

The SDPs 5 and 6 can be solved via an interior point algorithm [7]. Interior-point methods are however slow in practice and scaling the solution of an SDP is a problem-specific art. Enforcing PSD-ness of a square matrix $A \in R^{d\times d}$ amounts to enforcing infinitely many constraints $x^T Ax > 0$ (one per direction $x \in R^d$). An equivalent condition for a square matrix $A \in R^{d\times d}$ to be PSD is the positivity of all its eigenvalues: $\lambda_i(A) > 0, i \in [d]$.

Interestingly, the PSD constraints $x^T Ax > 0$ that show up in both E-optimal and A-optimal design optimization problems are linear in α and τ (i.e. of the form $c_\alpha^T \alpha + c_\tau^T \tau > 0$ for some $c_\alpha \in R^n$ and $c_\tau \in R^d$. We leverage this fact and come up with an approximate cutting plane algorithm for `E-optimal` and `A-optimal` design, where we solve a sequence of linear programs (LP)s as opposed to an SDP (see Algorithm 3). The main idea is that we no longer try to enforce $A \succeq 0$, or equivalently $x^T Ax > 0$ for all x, but rather only satisfy it for a finite subset of directions x. We choose 4 different types of directions x: (i) the canonical unit vectors $e_i \in R^d$, (ii) the $d(d-1)/2$ vectors $e_i \pm e_j$, (iii) a number of random *sparse* directions generated in a Monte Carlo fashion and (iv) the eigenvectors corresponding to the k (where k is a tunable parameter) most negative eigenvalues of A, in a greedy fashion.

Directions (i) and (ii) above are necessary A to be PSD, whereas directions (iv) are the ones along which the constraint $x^T Ax > 0$ is violated the most. Indeed, $x^T Ax = x^T \lambda x = \lambda |x\|_2^2 < 0$. Finally, directions (iii) aim to broaden the subspace where PSD-ness holds. They need to be sparse, since sparsity of constraints is a key factor for a linear program to scale.

Appendix B.2. A cutting plane algorithm for `E-optimal` *allocation*

We now formally describe the cutting plane algorithm for solving 5 approximately. We define the matrix $A \in R^{d\times d}$ as the $A = \sum_{x\in\mathcal{X}} \alpha_x x x^\top - \tau I_d$.

First, we add the simplex constraint $\sum_{i=1}^{n} \alpha_i \leq 1$ (constraint set R_1). Second, we add the d cuts in the directions of the unit vectors: $e_i^T A e_i > 0, i = 1, 2, \ldots, d$ (constraint set R_2). Third, we add the $d\cdot(d-1)/2$ cuts in the directions $e_i \pm e_j, i = 1, 2, \ldots, d, j = i+1, \ldots d$ (constraint set R_2). Fourth, we generate a total of M cuts in random directions sampled from a standard multivariate normal distribution, $r_j^\top A r_j > 0, r_j \sim \mathcal{N}(0, I), j = 1, 2, \ldots, M$ (constraint set R_4). We find that $M = 100$ works well in practice.

After an initial LP solve with the constraints $R = R_1 \cup R_2 \cup R_3 \cup R_4$, we iteratively solve a new LP while the matrix A is not positive definite or we run out of iterations, T. For each of these iterations, we augment the constraint set R greedily by adding cuts in the eigenvectors that correspond to the K most negative eigenvalues of A and perform a new LP solve. We used $T = 75$ iterations and $K = 5$ cuts in our implementation.

State of the art LP solvers can solve linear programs with millions of parameters and hundreds of thousands of constraints as long as the constraints are *sparse*. The eigenvectors however tend to be dense and this is the reason why the cutting plane algorithm cannot scale further the solution of `E-optimal` allocation.

Appendix B.3. A cutting plane algorithm for `A-optimal` *allocation*

We now formally describe the cutting plane algorithm for solving 6 approximately. The algorithm is very similar to the algorithm in the previous section, with the main difference that we know have a total of d matrices $A_i \in R^{d\times d}$ that we want to be approximately positive definite.

We define the matrix $A_i \in R^{d\times d}$ as the block matrix:

$$A_i = \begin{bmatrix} \sum_{x\in\mathcal{X}} \alpha_x x x^\top & e_i \\ e_i^\top & \tau_i \end{bmatrix}, i = 1, 2, \ldots, d.$$

Appendix B.4. Performance

For the network topology C for example, solving the `E-optimal` SDP takes 18hrs, whereas the `A-optimal` SDP cannot run due to memory limits. The cutting plane algorithm runs in approximately 1200 seconds for the `E-optimal` design and 2220 seconds for the `A-optimal` design, respectively, at the expense of only 1.8% loss in accuracy. This first successful attempt at scaling the `E-optimal` and `A-optimal` design problems led us to the conclusion that the PSD constraints need only approximately be satisfied. For `E-optimal` design, we observed that the cutting plane algorithm achieved the results above with just 60% of the eigenvalues of M being positive. For `A-optimal` design, the cutting plane algorithm reached the above results with an average of 61% of positive eigenvalues, where the average is over the d block matrices A_i.

Algorithm 2 Iterative Cutting Plane Linear Programming for E-Optimal Allocation.

```
1:  function CP-EOPT($X$, $T$, $M$, $K$)
2:      $R_1 \leftarrow \{\sum_{i=1}^{n} \alpha_i \leq 1\}$
3:      $A = \sum_{i=1}^{n} a_i x_i x_i^T - \tau \cdot I_d$
4:      $R_2 \leftarrow \{e_i^T A e_i > 0, i = 1, 2, \dots d\}$
5:      $R_3 \leftarrow \{z_{ij}^T A z_{ij} > 0, z_{ij} = e_i \pm e_j, i = 1, 2, \dots, m, j = i+1, i+2, \dots, d\}$
6:      $R_4 \leftarrow R_4 \cup \{r_j^\top A r_j > 0, r_j \sim \mathcal{N}(0, I), j = 1, 2, \dots, M\}$
7:      $R \leftarrow R_1 \cup R_2 \cup R_3$
8:      $O \leftarrow \max \tau$
9:      $t \leftarrow 0$
10:     $psd \leftarrow$ False
11:     while $(iter < I) \wedge (\neg psd)$ do
12:         $(\alpha, \tau) \leftarrow LP(O, R)$                                   ▷ LP Solve.
13:         $A = \sum_{i=1}^{n} a_i x_i x_i^T - \tau \cdot I_d$
14:         $(\Lambda, Z) \leftarrow$ eig($A$)
15:         if $(\Lambda[1] > 0)$ then
16:             $psd \leftarrow$ True
17:         else
18:             for $k = 1$ to $K$ do
19:                 if $\Lambda[k] < 0$ then                       ▷ $\Lambda$ sorted in ascending order.
20:                     $z \leftarrow Z[i]$
21:                     $C \leftarrow R \cup \{z^T M z > 0\}$
22:         $t \leftarrow t + 1$
23:     return $\alpha, \tau$
```

Algorithm 3 Cutting Plane Algorithm for A-Optimal Allocation.

```
1:  function CP-AOPT($X, T, M, K$)
2:      $R_1 \leftarrow \{\sum_{i=1}^{n} \alpha_i \leq 1\}$
3:      $R_2 \leftarrow \{\}$
4:      $R_3 \leftarrow \{\}$
5:      $R_4 \leftarrow \{\}$
6:      for $i = 1$ to $d$ do
7:          $A_i = \begin{bmatrix} \sum_{x \in \mathcal{X}} \alpha_x x x^\top & e_i \\ e_i^\top & \tau_i \end{bmatrix}$
8:          $R_2 \leftarrow R_2 \cup \{e_j^\top A_i e_j > 0, j = 1, 2, \dots d\}$
9:          $R_3 \leftarrow R_3 \cup \{z_{jk}^\top A_i z_{jk} > 0, z_{jk} = e_j \pm e_k, j = 1, 2, \dots, d, k = j+1, j+2, \dots, d\}$
10:         $R_4 \leftarrow R_4 \cup \{r_j^\top A_i r_j > 0, r_j \sim \mathcal{N}(0, I), j = 1, 2, \dots, M\}$
11:     $R \leftarrow R_1 \cup R_2 \cup R_3 \cup R_4$
12:     $O \leftarrow \min \sum_{i=1}^{d} \tau_i$
13:     $t \leftarrow 0$
14:     $psd \leftarrow$ False
15:     while $(t < T) \wedge (\neg psd)$ do
16:         $(\alpha, \tau) \leftarrow LP(O, R)$                                  ▷ LP Solve.
17:         for $i = 1$ to $d$ do
18:             $A_i = \begin{bmatrix} \sum_{x \in \mathcal{X}} \alpha_x x x^\top & e_i \\ e_i^\top & \tau_i \end{bmatrix}$
19:             $(\Lambda_i, Z_i)$ = eig$(A_i)$
20:             if $\Lambda_i[1] > 0$ then
21:                 $psd \leftarrow psd \wedge$ True
22:             else
23:                 for $k = 1$ to $K$ do
24:                     if $\Lambda_i[k] < 0$ then                    ▷ $\Lambda_i$ sorted in ascending order.
25:                         $z \leftarrow Z_i[k]$
26:                         $R = R \cup \{z^T A_i z > 0\}$
27:         $t = t + 1$
28:     return $\alpha, \tau$
```

Appendix C. Runtime Profile across Experiments

There are three types of experiments described in this work: Initial Experiments in Section 4.1, Synthetic Experiments in Section 5, and Real-world Experiments in Section 6.

Initial Experiments and Synthetic Experiments use the same topologies A / B / C but different algorithms (exact vs approximate SDP algorithms). In the Initial Experiments, the E-optimal is computed using an interior point algorithm in $4\,400$ / $1\,800$ / $31\,700$ seconds; in the Synthetic Experiments, the E-optimal is computed using the Frank-Wolfe algorithm in 66 / 57 / 53 seconds–this is at least two orders of magnitude lower. A-optimal in the Initial Experiment does not reach completion due to running out of memory for all topologies. Therefore, we don't compare the A-optimal run times between Initial Experiments and Synthetic Experiments. The detailed difference between these two algorithms and the trade-off between the increased loss in accuracy and decreased run times can be found in Appendix Appendix B.

The Synthetic Experiments and the Real-world Experiments use the same Frank-Wolfe algorithm but different typologies. In the Synthetic Experiment, for topologies A / B / C, the E-optimal allocation is computed in 66 , 57 , 53 seconds, the A-optimal is computed in 140, 121, and 219 seconds; In the Real-World Experiments, the E-optimal allocation is computed in 346 seconds and the A-optimal is computed in 961 seconds. While this six times higher than the average result of Synthetic Experiments, the complexity is the same because the number of paths is also six times larger.