Prophet: Fast Accurate Model-based Throughput Prediction for Reactive Flow in DC Networks

Kai Gao*§, Jingxuan Zhang†‡, Y. Richard Yang†‡[1], Jun Bi*§[1]

* Institute of Network Science and Cyberspace, Tsinghua University
† Department of Computer Science, Yale University
‡ Department of Computer Science, Tongji University
§ Tsinghua National Laboratory for Information Science and Technology

***Abstract*—As modern network applications (*e.g.*, large data analytics) become more distributed and can conduct application-layer traffic adaptation, they demand better network visibility to better orchestrate their data flows. As a result, the ability to predict the available bandwidth for a set of flows has become a fundamental requirement of today's networking systems. While there are previous studies addressing the case of non-reactive flows, the prediction for *reactive flows*, *e.g.*, flows managed by TCP congestion control algorithms, still remains an open problem. In this paper, we identify three challenges in providing throughput prediction for reactive flows: *throughput dynamics*, *heterogeneous reactive control mechanisms*, and *source-constrained flows*. Based on a previous theoretical model, we introduce a novel learning-based prediction system with a key component named *fast factor learning* (FFL) model. We adopt novel techniques to overcome practical concerns such as scalability, convergence and unknown system parameters. A system, Prophet, is proposed leveraging the emerging technologies of *Software Defined Networking* (SDN) to realize the model. Evaluations demonstrate that our solution achieves significant accuracy in a wide range of settings.**

I. Introduction

The last decade has witnessed the rapid development of infrastructures for distributed applications such as public Cloud platforms [1]–[3]. For many of these distributed applications, such as Content Delivery Networks (CDN) [4] and large-scale data analytics systems [5], [6], knowing the network performance in advance helps them to make scheduling decisions for better performance. Hence, predicting network performance has become a fundamental functionality for today's high performance distributed applications.

While many existing studies [7]–[11] work on predicting network performance, they tend to focus on reserved resources and not consider the case of *best-effort, reactive* flows (*e.g.*, flows that are managed by TCP congestion control). Best-effort, reactive flows can have multiple benefits, including full resource utilization with fairness guarantees, and hence contribute a large portion of total traffic in many networks [12]. As a result, predicting network performance involving such flows is an important problem but remains open.

Despite the importance, the problem is difficult due to several challenges.

- First, *the prediction system must handle the dynamics of reactive flows*. A fundamental difference between predicting the throughput of flows with bandwidth reservations and reactive flows is that reactive flows are *adaptive*. In a system with reactive flows, throughput of existing flows will change when flows arrive and die. Thus, the available resources cannot be computed by simply observing current state.
- Second, *the prediction system must handle heterogeneous reactive mechanisms*. In particular, the widely used reactive flow control mechanism, TCP, is host-based congestion control, which allows different hosts to use different implementations. For example, Linux kernel has a kernel option `tcp_congestion_control` and a socket option `TCP_CONGESTION` to set congestion algorithms system-wide and for each flow [13]. Such heterogeneity complicates the throughput prediction of TCP flows [14], [15].
- Third, *the prediction system must handle source constraints*. Throughput of a reactive flow is constrained by not only network resources but also application-level factors such as flow preferences or data availability. Thus, bandwidth estimation methods operating at the transport layer, as proposed in many previous TCP designs [16]–[18], does not suffice the need.

To handle both dynamics and heterogeneous reactive mechanisms, we take advantage of a previously proposed unifying theoretical model for heterogeneous reactive mechanisms [19]–[21].

Although the model provides a solid starting point, it leaves two key issues unaddressed. First, the theoretical model has a key parameter which is unknown in advance. We call this parameter a *scaling factor* as discussed in Section III-A and refer to the issue of computing the scaling factor as the scaling-factor computation challenge. Second, the theoretical model does not include source constraints which are important in real settings but introduce substantial complexity, as the constraints of background flows are not even known.

In this paper, we solve the scaling-factor computation challenge by deriving a novel method called *Fast Factor Learning* (FFL). While a naive approach which requires a unique variable for each individual flow can introduce scalability and convergence issues in practice, FFL uses 1) a novel technique based on equivalence classes, to substantially reduce the num-

[1]Prof. Yang (yry@cs.yale.edu) and Prof. Bi (junbi@tsinghua.edu.cn) are the corresponding authors of this paper.

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

ber of variables, improving scalability, and 2) a novel method based on gradient descent using sampling guidance, to achieve accelerated convergence. To address the source constraint challenge, we use learning to infer source constraints and extend our design to compute accurate throughput prediction.

Our **main contributions** in this paper are three-fold:

- We formally define the problem of *predicting throughput for reactive flows with source constraints*. A simple solution is first proposed for the most basic setting and we introduce two novel approaches to extend the solution to more general settings. In particular, we have adopted a novel learning component based on historical traffic samples to provide *accurate (relative errors less than 10%), fast (responses in less than 0.05s)* throughput prediction.
- We propose Prophet, a novel system to provide the service of throughput predictions for reactive flows, based on our theoretical findings.
- We implement a prototype and evaluate it extensively. It is demonstrated that Prophet can provide accurate estimation results even in multiple scenarios.

The rest of the paper is organized as follows. In Section II we formally define the problem of throughput prediction for reactive flows. Theoretical analysis is conducted and we design novel approaches to solve the prediction problem in Section III. In Section IV, we propose Prophet, a system to provide the flow query service based on the theoretical results and advanced monitoring. The evaluation results are presented in Section V. Finally, we compare with related work in Section VI and summarize our paper in Section VII.

II. Problem Statement

A. Motivation

Unlike predicting throughput for non-reactive flows which can be achieved by checking the available reservation, predicting throughput for reactive flows is much more difficult. Consider the example in Fig. 1, there are two flows in the original network, denoted as *background flows* f_1 and f_2. The two flows are *fully utilized*, meaning they do not have any source constraints and consume all available bandwidth.

Consider the prediction for a new flow q_1 arriving at the network. If the flow is not source-constrained, it can get 1/3 of the total bandwidth, *i.e.* 200 Mbps as in Fig. 1(b). In the general case, the new flow can be source-constrained, and the preceding result is only an upper bound. Assume the source limit is 50 Mbps, q_1 will only consume 50 Mbps as in Fig. 1(c).

The ability to predict the throughput is more important for a set of flows. Specifically, consider two simultaneous flows q_1 and q_2. If the two new flows are not source-constrained, they will both achieve 150 Mbps. However, if one flow, say q_1, is source-constrained, the result becomes more complex. As shown in Fig. 1(d), assume that q_1 is source-limited to 60 Mbps, one may compute that q_2 will achieve 180 Mbps (=(600-60)/3). If q_1 is limited to 180 Mbps, on the other hand, both q_1 and q_2 will receive 150 Mbps. In general settings with complex networks and complex flow source constraints, the prediction service must be able to respect source constraints.

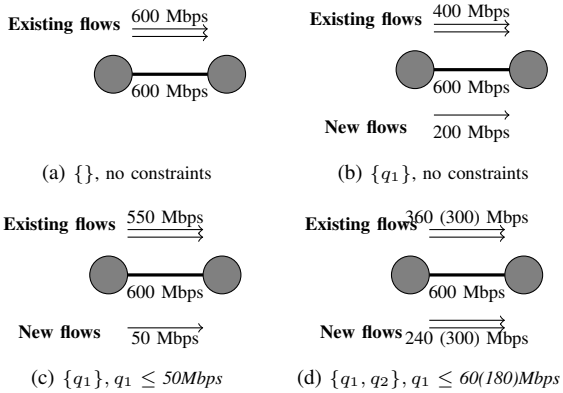


Fig. 1: Throughput for Different Flows and Source Constraints.

B. Problem Definition

Network: We consider a data center network with N full duplex edge connections where network congestion only happens at these edge connections [22]. The network core can be seen as a non-blocking switch and each connection can be seen as two separate links. The links are numbered from 1 to $2N$ and the set of links is denoted by L. The k-th link is represented as l_k, and we use c_k to represent its capacity. The capacity vector $\boldsymbol{c}$ is defined as a column vector that $\boldsymbol{c} = \langle c_1, \ldots, c_{(2N)} \rangle^T$.

Background flows: There are K running reactive flows in the network. We number these "background" flows from 1 to K where the i-th flow is denoted by f_i whose source is denoted by s_i and destination by d_i. F is a set consisting of all the background flows. The throughput of the i-th flow is denoted as x_i and the throughput vector $\boldsymbol{x} = \langle x_1, \ldots, x_K \rangle^T$. The i-th flow f_i may have a source rate limit of τ_i, where $\boldsymbol{\tau} = \langle \tau_1, \ldots, \tau_K \rangle$. **We also abuse the symbols F and f_i in the general case when there is no need to distinguish between background flows and queried flows**.

Flow query: We consider a flow query Q which consists of M reactive flows numbered from 1 to M. We refer to the i-th flow as q_i, and denote its throughput by y_i. The throughput vector $\boldsymbol{y}$ is a column vector defined as $\boldsymbol{y} = \langle y_1, \ldots, y_M \rangle^T$. The i-th flow q_i may have a source rate limit of π_i, where $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_M \rangle$.

Routing: We consider the case that routing is already known. The routing matrices A and B are binary matrices of size $2N \times K$ and $2N \times M$ respectively, which represents how flow f_i and q_i traverses the network. In particular, $a_{ki} = 1$ if and only if f_i traverses l_k and $b_{ki} = 1$ if and only if q_i traverses l_k.

Based on the network system model, we define the problem of predicting throughput for reactive flows.

Problem 1 (Throughput Prediction for Reactive Flows)**.** Given a network with reactive flows and a flow query Q with source constraints $\boldsymbol{\pi}$, return the predicted throughput $\hat{\boldsymbol{y}}$.

One can prove that when there are non-reactive flows in the network, *i.e.* a set of flows S where any flow $f \in S$ has a fixed

TABLE I: Symbols

Scope	*Symbol*	*Meaning*
Network	N	# of access ports/links
	L	Set of links where $\|L\| = 2N$
	l_k	The k-th link
	c_k	Link capacity of l_k
Background Flows	K	# of running flows in a network
	F	Set of running flows where $\|F\| = K$
	f_i	The i-th running flow
	s_i/d_i	Source/destination host of f_i
	x_i	Throughput of flow f_i
	τ_i	Source constraint for f_i
Flow Query	M	# of flows in a flow query
	Q	Set of flows in a query where $\|Q\| = M$
	q_i	The i-th flow
	y_i	Throughput of flow q_i
	π_i	Source constraint for q_i
Routing	a_{ki}	$a_{ki} = 1$ indicates f_i traverses l_k
	b_{ki}	$b_{ki} = 1$ indicates q_i traverses l_k
Monitoring & Learning†	ρ_i	Utility scaling factor for f_i or q_i
	p_i	Equivalent class index for f_i or q_i
	$\bar{\rho}_j$	$\rho_i = \bar{\rho}_{p_i}$
	α_i	Utility parameter for f_i or q_i
	x_i^*	Equilibrium throughput for f_i or q_i
	$s\hat{y}m$	Estimated result of *sym* $(x_i, y_i, \dots)$
	$s\tilde{y}m$	*sym* $(x_i, y_i, \dots)$ for the sampled flows

† Unless explicitly stated in the context of flow queries, $\rho_i, p_i, \dots$ and x_i^* are associated with f_i.

bandwidth guarantee, throughput estimation for the reactive flows can be converted to the problem of *throughput prediction for reactive flows* by subtracting the reserved bandwidth from c_k. Thus, we only consider reactive flows in this paper.

III. Design Foundation

In this section, we start with the most basic setting (*i.e.*, homogeneous reactive mechanism and no source constraints) to form a fundamental understanding of how to predict throughput for reactive flows. We then increase the complexities by *considering heterogeneous TCP implementations* in Section III-B and *handling source-constrained flows* in Section III-C, as demonstrated in Fig. 2.

A. Starting Point

Our throughput prediction solution is based on the *Network Utility Maximization* (NUM) [20] which is the theoretical foundation for many TCP designs. Let $U_i(x_i)$ denote the utility when the throughput of f_i is x_i, the allocation for each flow is a solution to the following optimization problem:

$$\textit{System} \qquad \max \sum_{f_i \in F} U_i(x_i) \tag{1}$$

$$\textit{Subject to} \qquad A\boldsymbol{x} \leq \boldsymbol{c}.$$

The problem of *throughput prediction for reactive flows* can then be solved by finding the optimal solution of the following optimization problem:

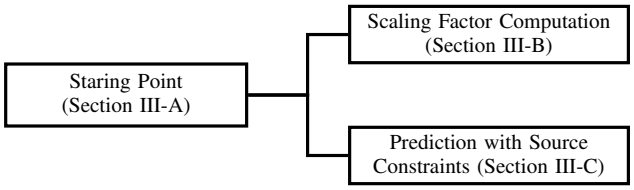


Fig. 2: Design Road Map.

$$\textit{System} \qquad \hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} \bar{U}_i(y_i) \right) \tag{2}$$

$$\textit{Subject to} \qquad \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \boldsymbol{c}.$$

To solve this optimization problem, it is essential to know the utility function. In particular, our method is based on the unifying model introduced by Srikant [19] where utility function of a TCP flow is modeled as:

$$U(x) = \rho \cdot \frac{x^{1-\alpha}}{1-\alpha}. \tag{3}$$

The parameter α is determined by the TCP congestion control algorithm. For example, the values of α for TCP Vegas and Reno [19] are 1 and 2 respectively. Existing works [23]–[25] have proposed sophisticated solutions on how to identify the TCP implementations, which means that we can assume that the α of each flow is already known. Thus, given x, the value of a utility function is proportional to the unknown parameter ρ, which we refer to as the *scaling factor*.

B. Scalable, Fast Scaling Factor Computation

In this section, we describe how to reduce the scale of the estimation problem and propose the theoretical foundation of estimation the scaling factors $\boldsymbol{\rho} = \langle \rho_1, \dots, \rho_K \rangle$ where ρ_i is the scaling factor for flow f_i.

For a given $\boldsymbol{\rho}$, the optimization problem (1) has a unique optimal solution since the utility functions are concave. Thus, we consider $\boldsymbol{x}$ as a function of $\boldsymbol{\rho}$, as in Lemma 1.

Lemma 1 (Optimality of Equilibrium Bandwidth)**.** Consider a network system where each running TCP flow f_i has its scaling factor ρ_i. If the equilibrium bandwidth of all running flows $\boldsymbol{x}^*$ fits the condition $A\boldsymbol{x}^* = \boldsymbol{c}$, it should be the maximal point of the following *Lagrange function*:

$$\mathcal{L}(A, \boldsymbol{c}, \boldsymbol{\rho}; \boldsymbol{x}, \boldsymbol{\lambda}) = \sum_i \rho_i \frac{x_i^{1-\alpha_i}}{1-\alpha_i} - \boldsymbol{\lambda}^{\mathrm{T}}(A\boldsymbol{x} - \boldsymbol{c}).$$

Proof. This lemma can be trivially proved by substituting $U_i(x_i)$ with $\rho_i \frac{x_i^{1-\alpha_i}}{1-\alpha_i}$ in the *Lagrangian* of the optimization problem (1). □

The best estimation of $\boldsymbol{\rho}$ maximizes the likelihood between the actual bandwidth allocation ($\boldsymbol{x}$) and the one ($\hat{\boldsymbol{x}}$) estimated

from $\boldsymbol{\rho}$. In particular, we use the *least square* to measure the error of $\hat{\boldsymbol{x}}(\boldsymbol{\rho})$, *i.e.*,

$$E(\boldsymbol{\rho}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|^2. \tag{4}$$

In this paper, we make a mild assumption that a host uses the same TCP implementation throughout our prediction. This is a rational assumption since hosts typically do not change the TCP algorithms very frequently.

1) Improving scalability with equivalent classes: A first issue to be resolved is to *reduce the number of variables to be estimated*. In an arbitrary network, each flow potentially has a unique link price (dual variables in the NUM problem) so that its scaling factor ρ is also unique. For a network with K flows, a brute-force approach has to solve a K-variable estimation problem, which can be large in real world cases.

A key observation is that *many flows have similar scaling factors*. In a classic Clos topology [26], any connection can only have 1 out of 3 different hop count values, and respectively 3 different propagation delays. Since scaling factors usually depend only on TCP implementation and round trip time, scaling factors of flows with the same source can be approximately classified into three different groups.

For a network where there are P different scaling factors, the scaling factor of each flow belongs to one of the P equivalent classes. Let p_i denote the equivalent class index of flow f_i, which is uniquely determined by the source host and the routing matrix. Let $\bar{\rho}_i$ denote the scaling factor value of the i-th equivalent class, we have $\rho_i = \bar{\rho}_{p_i}$. *The problem of estimating $\boldsymbol{\rho}$ is now reduced to estimating the scaling factors of all equivalent classes $\bar{\boldsymbol{\rho}}$.*

By grouping flows with similar scaling factors, we substantially reduce the number of variables from $N(N-1)$ to $3N$. In a Clos topology where $k = 4$, our reduced model only has 48 variables while the brute-force method has 240.

2) Achieving fast convergence with gradient descent method: The second issue is *convergence*. We derive the gradient of $\bar{\boldsymbol{\rho}}$ to speed up the computation.

Proposition 1. Given a routing matrix A, a capacity vector $\boldsymbol{c}$ and the equilibrium flow rates $\boldsymbol{x}$, there is no neighbourhood $U(\bar{\boldsymbol{\rho}}_{\mathbf{0}}, \delta)$ of the scaling factor $\bar{\boldsymbol{\rho}}$ which makes the following formula always true:

$$\forall \bar{\boldsymbol{\rho}} \in U(\bar{\boldsymbol{\rho}}_{\mathbf{0}}, \delta), \quad \det(A\Lambda^{-1}A^{\mathrm{T}}) \equiv 0,$$

where $\Lambda = \mathrm{diag}(\rho_{p_i}\alpha_i x_i^{-\alpha_i - 1})$.

Proof. Since $\det(A\Lambda^{-1}A^{\mathrm{T}})$ is a polynomial of $\bar{\boldsymbol{\rho}}$, if it is always zero in the neighbourhood $U(\bar{\boldsymbol{\rho}}_{\mathbf{0}})$, it must be zero in $\Re^P$. But we can always find a diagonal matrix Λ which makes $\det(A\Lambda^{-1}A^{\mathrm{T}})$ non-zero. So the assumption is not valid. □

Now we show that the gradient of the error estimation function $E(\bar{\boldsymbol{\rho}})$ can be computed very efficiently.

Substituting $\boldsymbol{\rho}$ with $\bar{\boldsymbol{\rho}}$ in Lemma 1, we have

$$\mathcal{L}(A, \boldsymbol{c}, \bar{\boldsymbol{\rho}}; \boldsymbol{x}, \boldsymbol{\lambda}) = \sum_i \bar{\rho}_{p_i} \frac{x_i^{1-\alpha_i}}{1-\alpha_i} - \boldsymbol{\lambda}^{\mathrm{T}}(A\boldsymbol{x} - \boldsymbol{c}),$$

where $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ subject to the following equations:

$$\nabla_{\boldsymbol{x}}\mathcal{L} = 0 \quad \Rightarrow \qquad \forall j, \quad \sum_k a_{kj}\lambda_k = \bar{\rho}_{p_j} x_j^{-\alpha_j}, \tag{5}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = 0 \quad \Rightarrow \qquad \forall k, \quad \sum_j a_{kj} x_j = c_k. \tag{6}$$

The solution of the bandwidth allocation problem for a given $\bar{\boldsymbol{\rho}}$ is a function of $\bar{\boldsymbol{\rho}}$. We denote it as $\hat{\boldsymbol{x}}$ and use the following symbols for simplicity:

$$\delta x_{ji} = \frac{\partial \hat{x_j}}{\partial \bar{\rho}_i}, \; \delta\lambda_{ki} = \frac{\partial \hat{\lambda_k}}{\partial \bar{\rho}_i}.$$

Now consider the partial derivatives for (5) and (6) at $\boldsymbol{\rho}$, we have

$$\begin{cases} \bar{\rho}_{p_j}\alpha_j x_j^{-\alpha_j - 1}\delta x_{ji} + \sum_k a_{kj}\delta\lambda_{ki} = 0 & \forall i, \forall j \text{ s.t. } p_j \neq i, \\ \bar{\rho}_i\alpha_j x_j^{-\alpha_j - 1}\delta x_{ji} + \sum_k a_{kj}\delta\lambda_{ki} = x_j^{-\alpha_j} & \forall i, \forall j \text{ s.t. } p_j = i, \\ \sum_j a_{kj}\delta x_{ji} = c_k & \forall i, \forall k. \end{cases}$$

It is equivalent to the following linear equation:

$$\begin{pmatrix} \Lambda & A^{\mathrm{T}} \\ A & O \end{pmatrix} \begin{pmatrix} \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \\ \mathbf{J}_{\boldsymbol{\lambda}}(\bar{\boldsymbol{\rho}}) \end{pmatrix} = \begin{pmatrix} \Gamma \\ \boldsymbol{c}J_{1,P} \end{pmatrix}, \tag{7}$$

where

$$\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) = (\delta x_{ji}), \mathbf{J}_{\boldsymbol{\lambda}}(\bar{\boldsymbol{\rho}}) = (\delta\lambda_{ki}),$$

$$\Gamma_{j,i} = \begin{cases} x_j^{-\alpha_j} & \text{if } p_j = i, \\ 0 & \text{if } p_j \neq i. \end{cases}$$

If the coefficient matrix of (7) is reversible, $\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})$ can be computed very efficiently. From Proposition 1, we know there is no continous $\bar{\boldsymbol{\rho}}$ making the determinant of this coefficient matrix $\det(O - A\Lambda^{-1}A^T)$ always zero, which means we can always generate the next $\bar{\boldsymbol{\rho}}$ iteratively or make a small disturbance to get a reversible coefficient matrix.

Now we can use the gradient descent method to minimize the error. To eliminate the drawback of using a fixed step with Cartesian coordinates, we choose spherical coordinates and the gradient is calculated as follows.

Let $\hat{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})$ denote the estimated value of $\boldsymbol{x}$ by $\bar{\boldsymbol{\rho}}$ and $(r, \boldsymbol{\phi})$ be the spherical coordinate transform of $\bar{\boldsymbol{\rho}}$, $\hat{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) = \hat{\boldsymbol{x}}(\boldsymbol{\phi})$. Let $\boldsymbol{e}$ be the unit vector of $\bar{\boldsymbol{\rho}}$, *i.e.* $\boldsymbol{e} = \frac{\bar{\boldsymbol{\rho}}}{\|\bar{\boldsymbol{\rho}}\|}$. $\boldsymbol{e}$ is also a function of $\boldsymbol{\phi}$. Let $\mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})$ be the Jacobian matrix of $\boldsymbol{e}(\boldsymbol{\phi})$. We can get:

$$\nabla_{\boldsymbol{\phi}} E = 2(\hat{\boldsymbol{x}} - \boldsymbol{x})^{\mathrm{T}}\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})\mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})^{\mathrm{T}}. \tag{8}$$

C. Prediction with Source-Constrained Background Flows

We now consider the more general problem of throughput prediction for source-constrained flows. Two major challenges are raised: 1) how to compute the scaling factors with source constraints, and 2) how to obtain the source constraints for background flows.

For the first challenge where $\boldsymbol{\tau}$ is already given, we extend the optimization problem in (2) to include source constraints in

our prediction framework. The throughput prediction problem is equivalent to solving the following optimization problem:

$$\textit{System} \quad \max \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} U_i(y_i) \right) \tag{9}$$

$$\textit{Subject to} \quad \begin{pmatrix} A & B \\ I & O \\ O & I \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{\tau} \\ \boldsymbol{\pi} \end{pmatrix}.$$

Thus, the throughput estimation in the FFL model is tranformed to a new format using the augmented routing matrix and constraint vector.

For the second challenge where $\boldsymbol{\tau}$ is also unknown, the problem is further complicated. Instead of taking the naive approach by augmenting the original FFL model with $\boldsymbol{\tau}$ as variables, we use a much simpler method. Our solution is based on the following observation.

A fundamental difference between source-constrained flows and non-reactive flows, even though they may appear similar because flows of both types may have a fixed transmission rate, is that *source-constrained flows only transmit at a fixed rate when its throughput is less than the equilibrium throughput*.

Proposition 2 (Hard Constraint)**.** Let $\langle \boldsymbol{x'}, \boldsymbol{y'} \rangle^T$ be the optimal solution of (2) and $\langle \boldsymbol{x''}, \boldsymbol{y''} \rangle^T$ be the optimal solution of (9). We have the following conclusion:

$$x''_i = \tau_i, \qquad \forall i, x'_i > \tau_i$$
$$y''_j = \pi_j, \qquad \forall j, y'_j > \pi_j.$$

Proof. Consider the Lagrangian of optimization problems (2) and (9) (denoted as $\mathcal{L}_{(2)}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}; \boldsymbol{\lambda})$ and $\mathcal{L}_{(9)}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}; \boldsymbol{\lambda}, \boldsymbol{\mu})$ respectively). Without loss of generality, we consider a specific i such that $x'_i > \tau_i$. First, consider the derivatives of x_i, we have

$$\nabla_{x_i} \mathcal{L}_{(2)} = \frac{\partial U_i}{\partial x_i}(\cdot) - \sum_k a_{ki} \lambda_k \tag{10}$$

$$\nabla_{x_i} \mathcal{L}_{(9)} = \frac{\partial U_i}{\partial x_i}(\cdot) - \sum_k a_{ki} \lambda_k - \mu_i. \tag{11}$$

Since $\boldsymbol{x''}$ also satisfies the constraints of (2), consider the gradient of x_i at $\boldsymbol{x''}$ and the corresponding $\boldsymbol{\lambda''}$. Since $x'_i > \tau_i \geq x''_i$, $\nabla_{x_i} \mathcal{L}_{(2)}(\boldsymbol{x''}, \boldsymbol{\lambda''}) > 0$ but $\nabla_{x_i} \mathcal{L}_{(9)}(\boldsymbol{x''}, \boldsymbol{\lambda''}) = 0$. Thus, $\mu_i > 0$ and according to the *Karush-Kuhn-Tucker* conditions, $x''_i = \tau_i$.

Similarly we can prove $\forall j, y'_j > \pi_j$, $y''_j = \pi_j$. □

Our identification method on $\boldsymbol{\tau}$ is based on Proposition 2 but uses it in a reversed way. If the estimated equilibrium rate $\hat{x}_i$ is "significantly larger"[1] than the actual throughput x_i, we consider this flow being source-constrained with a rate limit of x_i and use it in future computations. In a series of samples, the source constraint of a given flow f_i cannot be "significantly smaller" than the average actual throughput, $\bar{x}_i$.

[1]We allow a 10% relative error, *i.e.*, if $x > (1 + 10\%)y$, we say x is significantly larger than y.

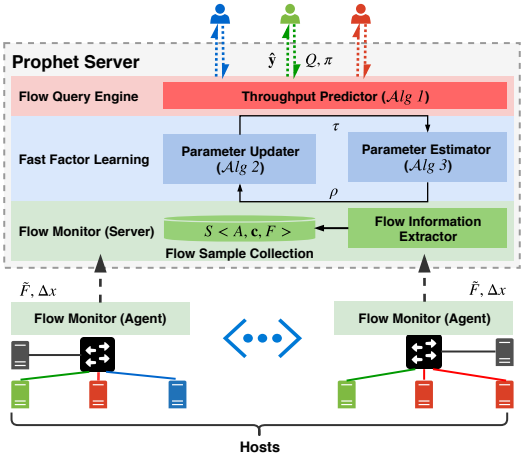


Fig. 3: The Prophet System Overview

IV. The Prophet System

From the observations and analysis in Section III, we have derived what kind of information is necessary to solve the problem of throughput prediction for reactive flows. In this section, we introduce the details of the Prophet system based on the results.

A. System Overview

As demonstrated in the figure, Prophet consists of the following components:

- **Flow query engine**: A flow query engine receives flow queries from users. It uses the estimated parameters learned by the FFL component and calculates the estimated throughput prediction for flow queries based on (9).
- **Fast factor learning**: The *fast factor learning* (FFL) component uses flow samples collected by flow monitors to train a TCP utility function estimation model and learn the parameters for each flow continuously. The component iteratively updates the parameters once the information of new flows is reported. Using the estimated parameters, the FFL component can construct utility functions.
- **Flow monitor**: A flow monitor monitors the traffic for a set of given hosts, extracts the header information which is further processed by TCP classifiers. It is also responsible for monitoring real-time traffic at the access switch.

B. Throughput Prediction for Reactive Flow Queries

For a flow query Q, consider the throughput prediction problem in (9). Background flows F and the routing matrix A are provided by the network controller, the routing matrix of queried flows B is computed from Q, source constraints of queried flows $\boldsymbol{\pi}$ are (optionally) provided along with Q, and finally source constraints for background flows $\boldsymbol{\tau}$, parameters of utility functions $\boldsymbol{\alpha}$ and $\bar{\rho}$ are provided by the FFL component.

Thus, we can conduct throughput prediction for Q by solving the problem of (9), as in Algorithm 1. The algorithm constructs utility functions for background flows (Line 2-3)

Alg. 1: Throughput Prediction for Reactive Flows

Input: $F, A, \boldsymbol{c}, \boldsymbol{\tau}, \boldsymbol{\rho}; Q, \boldsymbol{\pi}$
Output: $\hat{\boldsymbol{y}}$ - estimated equilibrium bandwidth

1 **Function** PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \boldsymbol{\rho}; Q, \boldsymbol{\pi}$)
2 $\quad \boldsymbol{\alpha}, \boldsymbol{p} \leftarrow$ IDENTIFYTCP(F)
3 $\quad$ **foreach** $f_i \in F$ **do**
4 $\quad\quad U_i(x) \leftarrow \bar{\rho}_{p_i} \frac{x^{1-\alpha_i}}{1-\alpha_i}$
5 $\quad \boldsymbol{\alpha'}, \boldsymbol{p'} \leftarrow$ IDENTIFYTCP(Q)
6 $\quad$ **foreach** $q_i \in Q$ **do**
7 $\quad\quad \bar{U}_i(x) \leftarrow \bar{\rho}_{p'_i} \frac{x^{1-\alpha'_i}}{1-\alpha'_i}$
8 $\quad B \leftarrow$ GETROUTINGMATRIX(Q)
9 $\quad U(\boldsymbol{x}, \boldsymbol{y}) \leftarrow \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} \bar{U}_i(y_i)\right)$
10 $\quad A' \leftarrow \begin{pmatrix} A & B \\ I & O \\ O & I \end{pmatrix}, \boldsymbol{c'} \leftarrow \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{\tau} \\ \boldsymbol{\pi} \end{pmatrix}$
11 $\quad \hat{\boldsymbol{y}} \leftarrow \arg\max U(\boldsymbol{x}, \boldsymbol{y})$ subject to $A' \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \boldsymbol{c'}$
12 $\quad$ **return** $\boldsymbol{y}$

Alg. 2: Parameter Updates based on Flow Samples

1 **Function** *ModelLearningDaemon*
2 $\boldsymbol{\alpha} \leftarrow \emptyset$, $\boldsymbol{\tau} \leftarrow \emptyset$, $\bar{\boldsymbol{\rho}} \leftarrow \mathbf{1}$, $S \leftarrow \emptyset$
3 **for** $k = 1, \ldots, +\infty$ **do**
4 $\quad$ **for** $i = 1, \ldots, |\tilde{F}^{(k)}|$ **do**
5 $\quad\quad$ **if** $\tilde{f}_i \notin F$ **then**
6 $\quad\quad\quad \tau_i \leftarrow +\infty$,
7 $\quad \boldsymbol{\alpha}, \boldsymbol{p} \leftarrow$ IDENTIFYTCP($\tilde{F}^{(k)}$), $\boldsymbol{c} \leftarrow \boldsymbol{c_0} - \boldsymbol{\Delta x}^{(k)}$
8 $\quad F \leftarrow \tilde{F}^{(k)}$, $A \leftarrow$ GETROUTINGMATRIX(F)
9 $\quad S \leftarrow \langle A, \boldsymbol{c}, F \rangle$
10 $\quad \bar{\boldsymbol{\rho}} \leftarrow$ ESTIMATEPARAMETERS($\boldsymbol{\tau}, \bar{\boldsymbol{\rho}}, S$)
11 $\quad \hat{\boldsymbol{x}} \leftarrow$ PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \bar{\boldsymbol{\rho}}; \emptyset, \emptyset$)
12 $\quad \boldsymbol{\tau} \leftarrow$ UPDATE($\hat{\boldsymbol{x}}, \tilde{\boldsymbol{x}}$)

Alg. 3: TCP Parameter Estimation

Input: $\boldsymbol{\tau}$, $\bar{\boldsymbol{\rho}}$, S
Output: $\hat{\bar{\boldsymbol{\rho}}}$ - updated estimated parameters

1 **Function** ESTIMATEPARAMETERS($\boldsymbol{\tau}, \bar{\boldsymbol{\rho}}, S$)
2 $\quad \boldsymbol{\phi} \leftarrow$ CARTESIANTOSPHERICAL($\bar{\boldsymbol{\rho}}$)
3 $\quad \varepsilon \leftarrow \varepsilon_0$
4 $\quad$ **while** $\varepsilon > tolerance$ **do**
5 $\quad\quad E \leftarrow 0, \nabla_{\boldsymbol{\phi}} E \leftarrow \mathbf{0}$
6 $\quad\quad$ **foreach** $< A, \boldsymbol{c}, F > \in S$ **do**
7 $\quad\quad\quad \bar{\boldsymbol{\rho}} \leftarrow$ SPHERICALTOCARTESIAN($\boldsymbol{\phi}$)
8 $\quad\quad\quad \hat{\boldsymbol{x}} \leftarrow$ PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \bar{\boldsymbol{\rho}}; \emptyset, \emptyset$)
9 $\quad\quad\quad E \leftarrow E + \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|^2$
10 $\quad\quad\quad \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \leftarrow$ SOLVE($Equation\ 7$)
11 $\quad\quad\quad \nabla_{\boldsymbol{\phi}} E \leftarrow \nabla_{\boldsymbol{\phi}} E + 2(\hat{\boldsymbol{x}} - \boldsymbol{x})^{\mathrm{T}} \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})$
12 $\quad\quad \boldsymbol{\phi} \leftarrow$ SGD($\boldsymbol{\phi}, E, \nabla_{\boldsymbol{\phi}} E$)
13 $\quad\quad \varepsilon \leftarrow \frac{E}{\sum_{\boldsymbol{x}} \dim \boldsymbol{x}}$
14 $\quad \hat{\bar{\boldsymbol{\rho}}} \leftarrow$ SPHERICALTOCARTESIAN($\boldsymbol{\phi}$)
15 $\quad$ **return** $\hat{\bar{\boldsymbol{\rho}}}$

and queried flows (Line 4-6) respectively. It then builds the routing matrix for the queried flows in Line 7 and constructs the optimization problem in (9). The estimated throughput is calculated by solving the optimization problem.

C. Computing Scaling Factors and Source Constraints

In this section, we describe how the information required by throughput prediction are learned from flow samples.

Let $\tilde{F}$ denote the sampled flows and $\boldsymbol{\Delta x}$ as the aggregated throughput of source-constrained small flows that are considered as non-reactive flows as in Proposition 2. Given the sampled flows $\tilde{F}$ and $\boldsymbol{\Delta x}$, we need to update five parameters that are essential to predict throughput: active background flow set F, its routing matrix A, source constraints $\boldsymbol{\tau}$, link capacities $\boldsymbol{c}$, and utility function parameters $\bar{\boldsymbol{\rho}}$.

We number the flow samples received from 1 incrementally. Let $\tilde{F}^{(k)}$ and $\boldsymbol{\Delta x}^{(k)}$ denote the k-th samples. Algorithm 2 describes how we process the samples.

For the i-th flow $\tilde{f}_i \in \tilde{F}^{(k)}$, $\tilde{\alpha}_i$ is determined by the TCP algorithm used by its source $\tilde{s}_i$. We use mechanisms from previous studies [23]–[25] to identify TCP implementations and set the α for $\tilde{f}_i$. The equivalence class index p_i is identified as a combination of $\tilde{s}_i$ (TCP implementation) and the distance between $\tilde{s}_i$ and $\tilde{d}_i$ (link price). These steps are represented using a function called IDENTIFYTCP.

Source constraint $\tilde{\tau}_i$ is set to $+\infty$ at the beginning (Line 6) and is continuously updated throughout the life cycle of f_i in our prediction model. We use $\tau_i^{(k)}$ to denote the values after k samples. Further processing on $\boldsymbol{\tau}^{(k)}$ is discussed later.

The link capacities are calculated by subtracting the total throughput of source-constrained flows $\boldsymbol{\Delta x}$ from the physical link capacity $\boldsymbol{c_0}$ (Line 7). The active background flow set $F^{(k)}$ is set to $\tilde{F}^{(k)}$ directly (Line 8). We get the routing matrix A from the network controller (Line 8).

The values of $\tau_i^{(k)}$ is updated using the methods introduced in Section III-C. The steps are denoted as function UPDATE (Line 11-12). In particular, we use the sampled throughput $\tilde{x}_i$ as an estimation of actual x_i.

The input of Algorithm 3 is a set of flow samples. Each flow sample is formulated as a 3-tuple $< A, \boldsymbol{c}, F >$ where A and $\boldsymbol{c}$ are as defined in Table I and F is a set of hash code. Each hash code in F indexes a record of flow f_i, which contains the TCP identification result α_i and p_i, the measured equilibrium bandwidth x_i and the estimated source constraint τ_i.

Algorithm 3 first transforms initial scaling factor $\bar{\boldsymbol{\rho}}$ to spherical coordinates $\boldsymbol{\phi}$. It then updates $\boldsymbol{\phi}$ by the sample set S iteratively and estimates the error ε until ε is less than a given threshold. In each loop, the algorithm passes the current $\bar{\boldsymbol{\rho}}$ to Algorithm 1 to get a bandwidth estimation for each sample in S. Then it calculates the value of E and $\nabla_{\boldsymbol{\phi}} E$ which are defined in (4) and (8). With this information, the algorithm applies *Stochastic Gradient Descent* to update $\boldsymbol{\phi}$. After the result is converged, the algorithm transforms $\boldsymbol{\phi}$ back to the Cartesian coordinates $\hat{\bar{\boldsymbol{\rho}}}$, which minimizes the error of bandwidth estimation E, and return it.

D. Extracting Flow Information

In this section, we describe how flow samples are collected at flow monitors.

From the analysis in Section III, Prophet requires the actual flow throughput to be collected. However, in a real world scenario, some adjustment must be taken into consideration.

We use sampling to gather the information on the access switches. Each flow is mapped to the TCP 5-tuple. The throughput of a flow is estimated as the total bytes in a sampling period divided by the sampling time. We construct $\tilde{F}$ by selecting the flows with throughput larger than 5% of the total bandwidth as *sampled flows* and the rest are considered *source-constrained* with a small rate limit, whose total throughput of the source-constrained flows Δx_k on link l_k is measured.

V. Evaluation

We have developed a prototype of the Prophet system. In this section, we conduct extensive experiments and evaluate the accuracy of throughput prediction for reactive flows.

A. Settings

System configuration: We use Mininet 2.2.1 and Network Simulator ns-2.35 to simulate the network. The prototype is implemented in Python and evaluations are conducted on a 64-bit Ubuntu 14.04 LTS server with 4-core Intel(R) Xeon(R) E5-2609 v2 (@ 2.50GHz) CPU, 32G memory.

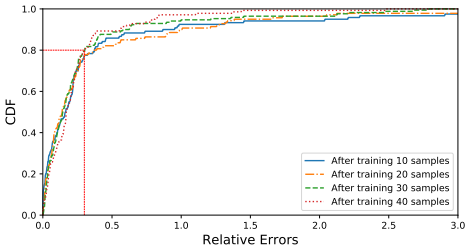
Network setting: The network used in our evaluations is a Clos topology with $k = 4$, *i.e.* 16 hosts. The link capacities between *core* and *aggregation*, and between *aggregation* and *edge* are set to 1 Gbps to ensure no congestion in the network core. The link capacity between an edge switch and a host is 1 Mbps. All links have the same propagation delay of $2us$.

Methodology: For each evaluation, we initiate some random background flows and capture their real time throughput. Then a random flow query is made for which we predict the throughput. The flows in the query are then added to the network. We capture their equilibrium throughput and compare the results with our estimations.

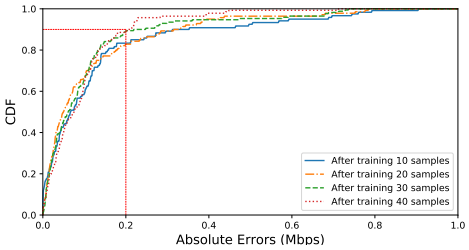
Metrics: We measure the following metrics: 1) **throughput error** between an estimation $\hat{\boldsymbol{x}}$ and the actual throughput $\boldsymbol{x}$, which represents the accuracy of a prediction and is calculated as $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|$ (absolute error) and $\frac{\|\hat{\boldsymbol{x}}-\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$ (relative error), 2) **scaling factor error**, which indicates the convergence speed of the training process, and 3) **execution time** of both the training and the prediction stages.

B. Throughput Prediction for A Single TCP Implementation

We evaluate the performance of Prophet in the setting with only one TCP implementation. In particular, we use TCP Vegas in a Clos topology with $K = 4$, with the number of flows in a query randomly picked in $[10, 20]$. We conduct 10 simulations on NS2. For each execution, an initial set of flows is launched first and 50 queries are made consecutively. For each query, we record the predicted throughput and add the flows to the network. After the network has reached the equilibrium, we record the throughput for the queried flows and compare with the prediction.



(a) CDF of Relative Errors



(b) CDF of Absolute Errors

Fig. 4: Errors for Throughput Prediction.

Fig. 4a demonstrates the cumulative distribution of relative errors of the queried reactive flows. As we can see, more than 80% of the flow queries have a relative error of less than 30% and more than 40% flows are less than 10%, which demonstrates *that our throughput prediction system can achieve high accuracy from 70% up to 90%.*

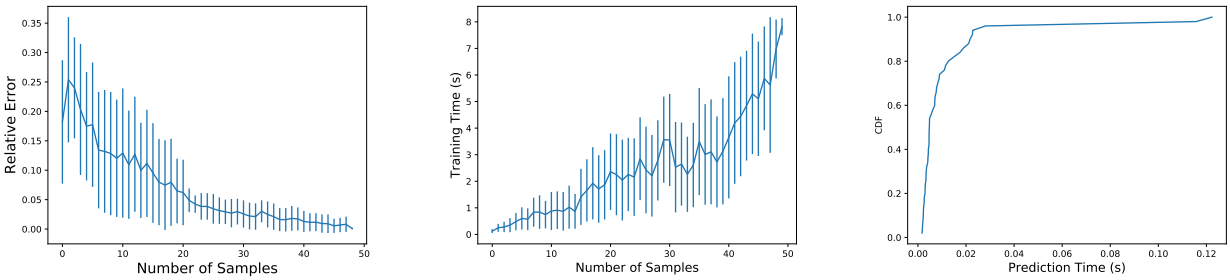
The curve in Fig. 5(a) demonstrates the relative errors between estimated scaling factors $\bar{\boldsymbol{\rho}}$ after training with k samples and the final scaling factors. As we can see, *the scaling factors converge very quickly* with a relative error of approximately 5% after around 20 samples.

The training time is presented in Fig. 5(b). Not surprisingly, the training time increases as the number of samples increase. However, the relatively large standard deviations and the jitters indicate that the execution time depends heavily on the actual flow distribution. From the previous analysis on the convergence of scaling factors, a reasonable cut-off value is 20, where the training time is mostly less than 4 seconds.

The prediction time is demonstrated in Fig. 5(c). Since the prediction is independent of sampling numbers, we are more interested in their distributions. As we can see from Fig. 5(c), the prediction for 10-20 flows is pretty fast. All queries are responded in less than 0.15s and about 90% of the queries only takes less than 0.05s. We conclude that *Prophet can efficiently predict throughput for reactive flows.*

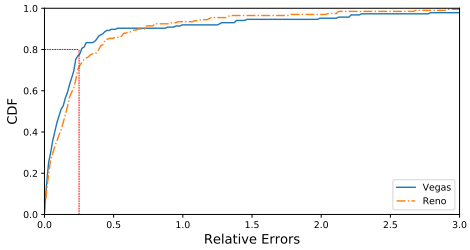
C. Throughput Prediction for Multiple TCP Implementations

We use the same methodologies as in the single TCP setting, with the exception that now multiple TCP implementations are

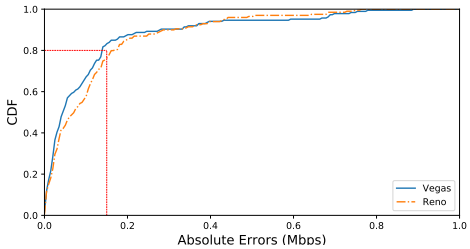


(a) Relative Errors of $\bar{\rho}$ (b) Training Time (c) Query Response Time

Fig. 5: Performance of a Clos Topology with $K = 4$, 50 Queries with $10 - 20$ Flows.



(a) CDF of Relative Errors



(b) CDF of Absolute Errors

Fig. 6: Errors for Multiple TCP Implementations.

used in the experiments. We randomly select TCP implementation algorithms for all the hosts in the network and make all flows from the same source use the same TCP implementation.

Again we demonstrate the throughput prediction results in Fig. 6. As we can see, the prediction results are similar to the single TCP implementation case that for 80% of the flows, the relative errors are less than 30%. We can also see that for different TCP implementations, Reno and Vegas in this case, the prediction accuracy may vary slightly.

D. Summary

In our evaluations, we have demonstrated that Prophet achieves accurate throughput predictions for reactive flows for networks of either single or multiple TCP implementations. We also demonstrate that the training model can converge with a small number of samples (10-20), and provide fast responses (90% responses in less than 0.05s).

VI. Related Work

TCP throughput estimation. Throughput estimation, or bandwidth estimation, is a widely-studied topic and is the foundation of many TCP implementations. TCP Vegas [27] estimates the throughput as the *expected rate*, defined by *cwnd*/*propagation delay*. TCP Westwood [17] takes a series of throughput samples and uses different low-pass filters for throughput estimation. BBR [28] estimates the throughput by finding the optimal operating point where throughput stops increasing alongside RTT. These throughput estimation methods are usually deployed on end hosts and can only provide the estimation after a flow is instantiated. More importantly, they cannot predict throughput for *a set of flows*. Instead, Prophet predicts throughput both *before* flows are launched and for *a set of reactive flows*.

Network performance prediction. Centralized network management systems, such as SDN, have enabled the ability to provide network views to applications, which makes it possible to predict network performance [7]–[11], [29]. However, existing solutions provide either a deterministic allocation or simple and static predictions for reserved bandwidth allocation. Prophet is the first approach to predict the dynamic network performance for reactive flows to the best of our knowledge, especially with source constraint considerations.

The general bandwidth allocation framework for TCP has been introduced in prior studies (*e.g.*, [19]–[21]). While they have laid down the theoretical foundation of our work, Prophet takes one more step on designing an approximate but practical prediction system with the advanced monitoring capabilities.

Modeling coexisting TCP flows. Many researchers have studied the equilibrium of TCP flows when multiple different congestion avoidance algorithms coexist. Vojnovic *et al.* [30] have studied the fairness of TCP implementations based on *additive-increase and multiplicative-decrease*. Tang *et al.* [15] have proved that the equilibrium of mixed TCP flows still exists. They also give the sufficient conditions for global uniqueness of network equilibrium. Instead of developing an accurate model for coexisted TCP flows, Prophet estimates equilibrium rates for heterogeneous TCP flows based on model learning and real-time monitoring.

VII. Conclusion

In this paper, we systematically study the problem of predicting throughput for reactive flows. We propose a novel learning-based method to accurately predict the throughput for reactive flow queries, based on both theoretical models and advanced monitoring capabilities provided by SDN and NFV. A system, namely Prophet, is proposed to provide reactive flow query service for applications hosted in a data center network. Evaluations have demonstrated that our prediction method yields accurate throughput prediction for well-known TCP implementations with fast convergence and quick responses.

VIII. Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments. We have made some changes accordingly and include further discussions in an extended technical report [31] due to the limited space.

This research is sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

This work is also supported by the *National Science Foundation* (CC-IIE 1440745), *Google Research Award*, *National Key Research and Development Plan of China* (2017YFB0801701), and the *National Natural Science Foundation of China* (No. 61672385, No. 61472213 and No. 61502267).

References

[1] Amazon, "Amazon elastic compute cloud (Amazon EC2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[2] Google, "Google Cloud Computing, Hosting Services & APIs," 2017, https://cloud.google.com/.

[3] Microsoft, "Microsoft Azure Cloud Computing Platform & Services," 2017, https://azure.microsoft.com/en-us/.

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[6] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, M. Girone, C. Grandi, and others, "LHC Computing Grid," *Technical design report*, p. 8, 2005.

[7] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. O. Guedes, "Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks." in *WIOV*, 2011.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 242–253.

[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *NSDI*, vol. 11, 2011, pp. 22–22.

[10] R. Alimi, Y. Yang, and R. Penno, "RFC 7285, Application-layer traffic optimization (ALTO) protocol," 2014.

[11] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "NOVA: Towards on-demand equivalent network view abstraction for network optimization," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, Jun. 2017, pp. 1–10.

[12] CAIDA, "Analyzing UDP usage in Internet traffic," http://www.caida.org/research/traffic-analysis/tcpudpratio/index.xml.

[13] Linux, "Manual tcp (7)," 2017, http://man7.org/linux/man-pages/man7/tcp.7.html.

[14] . Budzisz, R. Stanojevic, A. Schlote, F. Baker, and R. Shorten, "On the Fair Coexistence of Loss- and Delay-Based TCP," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1811–1824, Dec. 2011.

[15] A. Tang, J. Wang, S. H. Low, and M. Chiang, "Equilibrium of Heterogeneous Congestion Control: Existence and Uniqueness," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 824–837, Aug. 2007.

[16] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Adaptive bandwidth share estimation in TCP Westwood," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, Nov. 2002, pp. 2604–2608 vol.3.

[17] S. Mascolo, C. Casetti, M. Gerla, S. Lee, and M. Sanadidi, "TCP Westwood: congestion control with faster recovery," *Univ. California, Los Angeles, Tech. Rep. CSD TR*, vol. 200017, 2000.

[18] A. Capone, L. Fratta, and F. Martignon, "Bandwidth estimation schemes for TCP over wireless networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 2, pp. 129–143, Apr. 2004.

[19] R. Srikant, *The Mathematics of Internet Congestion Control*. Springer Science & Business Media, 2012.

[20] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, pp. 237–252, 1998.

[21] S. H. Low, L. L. Peterson, and L. Wang, "Understanding TCP Vegas: A Duality Model," *J. ACM*, vol. 49, no. 2, pp. 207–235, 2002.

[22] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62.

[23] J. Oshio, S. Ata, and I. Oka, "Real-Time Identification of Different TCP Versions," in *Managing Next Generation Networks and Services: 10th Asia-Pacific Network Operations and Management Symposium, APNOMS 2007, Sapporo, Japan, October 10-12, 2007. Proceedings*, S. Ata and C. S. Hong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 215–224.

[24] ——, "Identification of Different TCP Versions Based on Cluster Analysis," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, Aug. 2009, pp. 1–6.

[25] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP Congestion Avoidance Algorithm Identification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1311–1324, Aug. 2014.

[26] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.

[27] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[28] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[29] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.

[30] M. Vojnovic, J.-Y. Le Boudec, and C. Boutremans, "Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2000, pp. 1303–1312.

[31] K. Gao, J. Zhang, Y. R. Yang, and J. Bi, "Prophet Technical Report," 2017, https://www.dropbox.com/s/tvgqx4v830c2nb0/prophet-tr.pdf?dl=0.