Swift and Accurate End-to-End Throughput Measurements for High Speed Networks

Md Arifuzzaman and Engin Arslan
University of Nevada, Reno
marifuzzaman@unr.edu, earslan@unr.edu

***Abstract*—Active probing is extensively used in high-speed research networks for performance troubleshooting and transfer optimization. For end-to-end (i.e., disk-to-disk) throughput measurements, current active probing practice involves transferring a set of files and measuring throughput upon the completion of the transfer, which leads to long probing times. We present $FastProb$ that takes an adaptive approach to determine the duration of probing transfers based on the stability behavior of reported instantaneous throughput values. $FastProb$ employs a hybrid machine learning model which utilizes binary classifiers to determine the "predictability" of probing transfers and regression models to actually predict the transfer throughput upon convergence. Experimental results show that $FastProb$ lowers the duration of probing transfers by 48% while attaining up to 61% higher measurement accuracy. We further incorporate $FastProb$ into an online file transfer optimization algorithm to demonstrate that shortening the duration of probing transfers results in 35% higher overall throughput for data transfers in production high-speed networks.**

***Index Terms*—End-to-end network measurement, Network probing, Transfer modeling, Throughput optimization**

I. Introduction

High-speed research networks (HSNs) with up to 400 Gbps bandwidth have been built to accommodate the growing demands of distributed science applications. However, network and end system-related performance issues hinder the effective utilization of these networks, necessitating comprehensive monitoring solutions to identify and mitigate performance anomalies in a timely manner. Active probing plays an important role in achieving this goal as it is used for various purposes including *anomaly detection* [1], [2] and *transfer optimization* [3]. Thus, more than $2,000$ research and education institutions use PerfSonar to conduct periodic probes between participating sites to monitor network metrics (e.g., packet loss, delay, etc.) and *detect anomalies* proactively [1]. Active probing is also widely used for *transfer optimizations* to evaluate the performance of different transfer settings in real-time. For example, congestion control algorithms run sample transfers to evaluate the performance of different TCP sending rates in terms of goodput and packet loss such that the optimal sending rate can be discovered in real-time [4], [5].

Although most network metrics (e.g., delay, jitter, and flow path) can be measured quickly with minimal impact, throughput measurements can adversely affect production traffic by causing congestion. In particular, disk-to-disk throughput measurements in HSNs require concurrent file transfers to probe "true" end-to-end performance[1], which can have significant impact on production traffic if executed for an extended period of time. Determining the optimal duration for probing transfers in HSNs is a challenging task as small values may lead to incorrect measurements while large values will increase the impact on production flows.

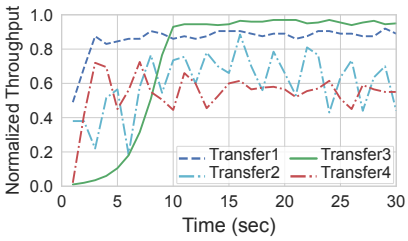


Fig. 1. Transfers converge at different pace in different networks, thus an adaptive approach is necessary to conduct probing transfers.

Previous approaches to conduct probing transfers in HSNs can be categorized into two groups as static and adaptive. In the static method, a fixed data size [3] (e.g., 10GB dataset) or duration [7] (e.g., 10 seconds) is used to execute probing transfers. The static approaches require fine-tuning of data size or time duration for each network as suboptimal values can cause inaccurate measurements or long probing times. To illustrate this, please refer to Figure 1 which presents the normalized throughput of end-to-end transfers from multiple HSNs. It is clear that while transfer throughput stabilizes in less than 5 seconds in one network (i.e, blue line), it does not converge within 30 seconds in another network (i.e, Transfer 2). The slow convergence of transfer throughput can be attributed to many reasons including but not limited to slow connection setup (especially when using advanced security protocols such as GSI [8]), high bandwidth delay product, and network congestion. One-time optimization of the data size or probing duration may not be sufficient as available network bandwidth may change over time due to the dynamic nature of resource interference. Adaptive approaches, on the other hand, process instantaneous throughput values that are populated periodically (e.g., once a second) using statistical methods such as time-series analysis [9] to terminate probing transfers as soon as the instantaneous throughput values converge. Despite performing better than static approaches, experimental analysis reveals that existing adaptive approaches fall short

[1]High performance computing clusters rely on parallel file systems (e.g., Lustre and GPFS) which require I/O parallelism to attain more than 10 Gbps read/write I/O throughput [6]

978-3-903176-47-8 ©2022 IFIP

to capture the intricate relationship between instantaneous throughput values and overall transfer throughput, and result in high measurement errors.

In this paper, we first propose machine learning (ML) regression models to process instantaneous throughput values and estimate the throughput of end-to-end probing transfers accurately. We find that although the ML regression models outperform existing adaptive approaches, they require probing duration to be determined by end users, thus failing to offer an automated solution. To overcome this challenge, we introduce $FastProb$ which pairs the regression models with classifiers to choose optimal probing duration in real-time. The results as gathered in several production HSNs show that $FastProb$ reduces probing duration by up to 48% while achieving up to 61% higher measurement accuracy in comparison to state-of-the-art end-to-end throughput probing techniques. In summary, the contributions of this paper are as follows:

- We gather and analyze $38K$ file transfer logs from four different HSNs and show that their throughput fluctuates significantly, making it hard to use any naive solutions to estimate transfer throughput by processing instantaneous throughput values (§ III).
- We propose a hybrid machine learning model, $FastProb$, that combines classification and regression models to achieve swift and precise estimation of throughput for end-to-end transfers (§ IV).
- We run extensive evaluations both in production and dedicated networks to assess the performance of $FastProb$ and compare it against the state-of-the-art (§ V).
- Finally, we incorporate $FastProb$ to a transfer optimization algorithm to demonstrate the impact of optimized probing transfers on the performance of transfer optimization algorithms in three production HSNs (§ VI).

II. Related Work

Active probing is widely used to measure network characteristics (such as delay [10], bandwidth [11], [12], loss rate [13], and topology [14]), and detect and localize performance issues [15]. However, most previous work uses probing to understand and optimize internet/web traffic, which has different characteristics than traffic in high-speed research networks. For example, while web traffic consists of many short-lived low-speed flows (1-100 Mbps), research network traffic is dominated by large flows whose throughput is the order of gigabits-per-second with up to 100Gbps [16]. Research and internet traffic also differ in terms of root causes of performance problems. While network issues (e.g., routing instabilities and poor performance of congestion control algorithm) are main reasons for performance problems for internet traffic, end-system issues (e.g., I/O limitations or interference) constitute the majority of performance problems in research networks. Therefore, existing network bandwidth measurement techniques that conduct memory-to-memory transfers (e.g., iPerf [17], Pathload [18], FastBTS [19], and Pathchirp [20]) are not well suited for high-speed research networks.

Previous studies proposed fixed-size [1], [3], fixed-duration [7], and time-series [9] models to conduct end-to-end probes in HSNs. Fixed-size approaches transfer a dataset consisting of one or more files and wait for its completion to calculate the throughput. Despite its simplicity, it can cause long transfer times when the dataset is not configured carefully. Yildirim et al. proposed regression analysis to determine the optimal dataset size based on network and dataset characteristics [3]. The proposed model, however, does not consider background traffic, which can change drastically, significantly increasing the probing duration due to decreased throughput. Fixed-duration approaches also have similar limitations as there is no single probing duration that would work optimally in all networks.

An alternative approach to optimize the duration of probing transfers involves collecting and processing instantaneous throughput reports. Probing transfers can be scheduled with large datasets and terminated as soon as it is sufficient to make an estimation using measured instantaneous throughput reports. Sapkota et al. applied time-series analysis and machine learning model to process instantaneous throughput values and predict the throughput of probing transfers as soon as possible with high accuracy [9], [21]. Our experimental results however show that time-series models are vulnerable to throughput fluctuations, causing more than 30% error rate in predictions. The proposed deep neural network-based machine learning model, despite improving the accuracy over the time-series models, requires long probing duration.

III. Problem Definition

Probing transfers are used to measure the maximum achievable file transfer throughput in HSNs. Although it is possible to execute probing transfers long enough (e.g., 60 seconds) to accurately measure the achievable throughput, shortening the probing duration is desirable for many reasons including but not limited to (i) reduced impact on production traffic when probing transfers use dummy data to check the health of the network, (ii) fast convergence time for online transfer optimization algorithms that rely on probing transfers to evaluate the performance of different transfer settings such as buffer size and the number of parallel connections [3]. Yet, file transfers exhibit distinct convergence time and stability patterns in HSNs due to various static (e.g., file size, bandwidth, and delay) and dynamic (e.g., file system and network congestion) factors, making it hard to choose a probing duration that would work in all networks. Even more challenging is the fact that different transfers in the same network can have completely different throughput patterns due to differences in dataset and background traffic. Hence, an adaptive approach is necessary to dynamically determine how long to execute a probing transfer to accurately measure achievable throughput.

Most transfer applications (e.g., sftp, GridFTP, and rsync) report instantaneous transfer throughput periodically for ongoing transfers, which can be used to estimate the throughput

TABLE I
FILE SYSTEM AND NETWORK SPECIFICATIONS OF TEST NETWORKS. A TOTAL OF $38K$ FILE TRANSFERS ARE CONDUCTED IN FOUR NETWORKS.

Network	Storage	Bandwidth	RTT	# of transfers
HPCLab	RAID-0 SSD	40G	0.2ms	10,136
ESnet	RAID-0 SSD	100G	88ms	6,831
XSEDE-1 (Stampede2-Expanse)	Lustre	40G	38ms	10,927
XSEDE-2 (OSG-Bridges2)	Lustre	10G	12ms	10,171
Total				**38,065**

of a transfer[2] quickly. Assume that throughput of a transfer is reported once in every i seconds, then n throughput reports will be available at $t = n \times i$ as follows $\{t_i, t_{2i}, t_{3i}, \dots t_{ni}\}$, where t_i is throughput of the transfer at $t = i$ second. If this transfer is executed long enough, we can calculate its throughput as the average of all instantaneous throughput reports, $t_{avg} = \sum_{j=1}^{n} \frac{t_{i \times j}}{n}$. The goal of this work is then to process instantaneous throughput reports as they become available to predict the throughput of a transfer, t_{avg}, as early as possible to terminate probing transfers quickly.

To gain insights into the throughput of file transfers in HSNs, we conducted $38K$ file transfers in four HSNs (as given in Table I) between September and October 2021 using GridFTP and logged instantaneous throughput values in one second intervals. HPCLab network consists of two data transfer nodes that are located in the same local area network and connected by a $40G$ switch. The nodes are equipped with direct-attached SSD drives that are configured into a RAID-0 array. In the ESnet network, two data transfer nodes (both located at Berkeley, CA) are connected via $100G$ wide-area network loop that spans between Berkeley, CA and Chicago, IL to create a wide-area network connectivity. Both ESnet and HPCLab networks are isolated, so transfers are not affected by background traffic. XSEDE-1 and XSEDE-2 networks, on the other hand, are shared production environments that connect supercomputing centers Stampede2, Expanse, Bridges2, and Open Science Grid (OSG) [22]. All XSEDE sites use Lustre as a parallel file system. Since file size affects transfer throughput and convergence behavior [23], we transferred different datasets with various file sizes (ranges between 512 KB and 1 GB) and counts (ranges between 30 and $180,000$). We also tuned a few application-layer transfer configurations, such as the number of concurrent file transfers and parallel network connections, to capture their impact on transfer convergence behavior. Consequently, the dataset contains transfer logs representing a wide range of network conditions, workload characteristics, and transfer settings. All transfers are executed at least 60 seconds using GridFTP which reports transfer throughput at most one second intervals. Thus, each transfer log consists of throughput reports as $\{t_1, t_2, t_3, \dots t_n\}$ where t_i is the throughput of i^{th} second and n is greater than 60. We also calculated average throughput, t_{avg}, by taking average of all instantaneous throughput values, $t_{avg} = \frac{t_1 + t_2 \dots + t_n}{n}$ and

[2]Throughput is defined as average throughput when a transfer is executed long enough such as 30 seconds or more.

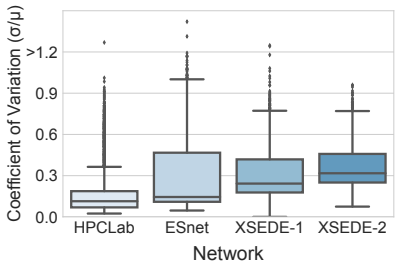


Fig. 2. End-to-End data transfers exhibit significant throughput fluctuations in all networks.

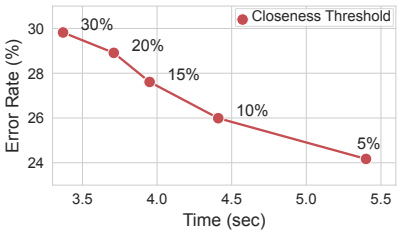
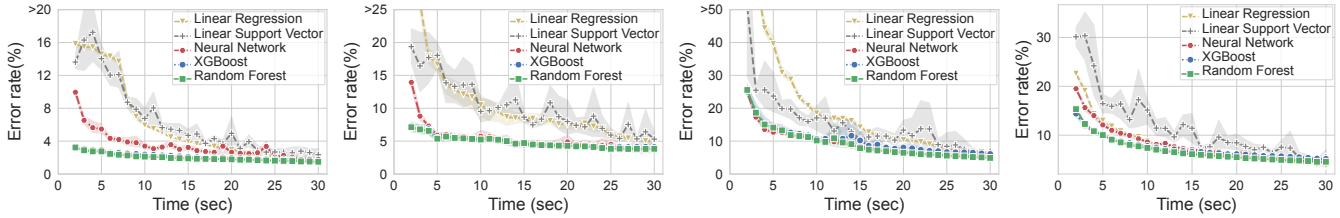


Fig. 3. A simple approach to detecting throughput convergence based on the closeness of consecutive instantaneous throughput values results in more than 24% for XSEDE-1 transfers even with a 5% distance threshold.

appended it to instantaneous throughput logs.

Figure 2 shows the Coefficient of Variance (CV) for transfers, which is calculated by dividing the standard deviation in instantaneous throughput values by mean (t_{avg}). CoV value is calculated for each transfer independently, so large CoV values indicate considerable fluctuations in instantaneous throughput. As HPCLab is an isolated testbed, the throughput of transfers fluctuates the least among others. ESnet transfers, on the other hand, fluctuate despite running in an isolated environment. This can be attributed to high bandwidth-delay characteristics (i.e., $100G$ bandwidth and 89ms delay) of ESnet network which results in slower convergence speed. Transfers in XSEDE networks also exhibit high fluctuations mainly due to the shared nature of network and I/O resources. These results indicate that simple solutions to detect throughput convergence by processing instantaneous throughput reports would fail to perform well. To validate this claim, we implemented *closeness-based* throughput convergence detection method that processes instantaneous throughput values to determine if throughput has stabilized. For example, suppose instantaneous throughput values of a transfer for the first six seconds are reported as $\{100, 800, 1200, 1600, 1250, 1400\}$. Then, the convergence detection algorithm with a 20% closeness threshold will mark the transfer throughput as *"converged"* at the sixth second since it is the first time that consecutive instantaneous throughput values fall within the 20% range of each other. Once the convergence decision is made, one can take the average of the last two instantaneous throughput values (throughput of the fifth and sixth second in the above example) as the throughput of the transfer, t_{pred}.

We evaluated the closeness based throughout convergence detection using transfer logs gathered in XSEDE-1 network. The error rate is calculated as the percentage of the difference between the predicted throughput, t_{pred}, and the average throughput of transfers, t_{avg}. Please note that t_{avg} is the average of all instantaneous throughput reports captured dur-



(a) HPCLab (b) ESnet (c) XSEDE-1 (d) XSEDE-2

Fig. 4. Performance comparison of regression models in different networks. Although Random Forest performs, Neural Network, and XGBoost models can achieve good performance, optimal probing duration is not the same in all networks, necessitating an automated solution.

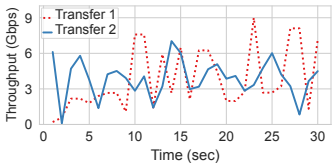


Fig. 5. Time series and classification based convergence detection methods [9], [21] result in high error rate when transfer throughput stalls shortly (49% prediction error for *Transfer-1*) and long probing times under high throughput fluctuation scenarios ($15s$ probing duration for *Transfer-2*).

ing data collection whereas t_{pred} is the average last two instantaneous throughput report when the closeness criteria is met. Figure 3 presents the average prediction time and error rate for XSEDE-1 transfers when the closeness threshold is ranged between 5% and 30%. Clearly, the error rate is too high even when setting the closeness threshold to 5%. This is because throughput stalls for $1-2$ seconds before starting to increase again for a non-negligible portion of transfers due to delayed connections and transient I/O and network resource interference (please see Transfer 3 and 4 in Figure 1).

Previous studies proposed time-series analysis and classification methods to process instantaneous throughput values and detect throughput convergence [9], [21]. Autoregressive (*AR*) model uses linear regression to processes instantaneous throughput values and predict the transfer throughput for the next time interval. It terminates the transfer when the prediction falls within a certain range (default is 5%) of actual throughput observation. Deep Neural Network (*DNN*) classifier similarly processes instantaneous throughput values to detect the convergence of throughput and uses the average of the last four instantaneous throughput values to predict the throughput. Figure 5 presents an example where both AR and DNN models fail to make accurate predictions. Since throughput appears to have converged in the first few seconds of *Transfer-1*, both models terminate the transfer at $t = 5$ and underestimates the actual average by 49%. On the other hand, they fail to detect a convergence for *Transfer-2* until the maximum time limit (i.e., 15 second) thus cause long probing duration. We therefore apply machine learning (ML) regression models as they are better at capturing otherwise intricate dynamics between input parameters (instantaneous throughput values) and output (average throughput) to lower prediction times and increase the prediction accuracy.

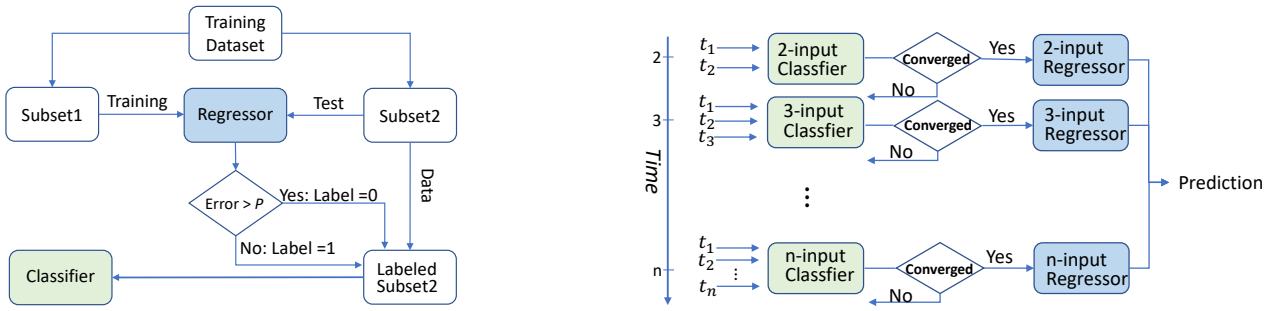
IV. Modeling Throughput of File Transfers

Regression models use instantaneous throughput values $t_1, t_2, ..., t_n$ are used to predict the average throughput of a transfer, t_{pred}, as

$$t_{pred} = \alpha_0 + \alpha_1 t_1 + \alpha_2 t_2 + ... + \alpha_n t_n \quad (1)$$

where α is the coefficient vector estimated in the model fitting phase. Instead of using absolute throughput values as inputs to regression models, we feed normalized values to minimize the bias. As a normalization method, we adopted standard normalization method (aka feature scaling), which separately scales each data column. As linear regression-based models suffer from multicollinearity, we also applied principal component analysis (PCA) to transform features (i.e., instantaneous throughput values) so that they become linearly uncorrelated. Note that PCA transformation is not applied to the input of machine learning models that do not assume feature independence, such as Random Forest and Extreme Gradient Boosting (aka XGBoost) Regression models. Finally, we use all components of PCA as it is used to overcome the multicollinearity problem not to reduce the number of features.

Next, we use the gathered $38K$ transfer logs to train Linear Regression (LR), Support Vector Regression (SVR), Neural Network (NN), XGBoost (XGB), and Random Forest (RF) Regression models to predict average transfer throughput, t_{avg}, using instantaneous throughput values. Since we envision using the prediction models in real-time to determine the stopping condition of probing transfers, the number of available instantaneous throughput reports will start from zero and increase one by one as time passes. For instance, if we measure the throughout of probing transfers at one-second intervals, we will have one instantaneous throughput value at $t = 1s$, two instantaneous throughout values at $t = 2s$, and so on. Hence, n in Equation 1 depends on how long a probing transfer is executed. Since the optimal probing duration is different for each network, we first trained a separate model for each possible probing duration (i.e., one second, two seconds, three seconds, etc.) as the number of input features is dependent on probing duration. For example, evaluating the Random Forest Regressor at $t = 3s$ and $t = 4s$ requires two models; one takes 3 inputs and the other takes 4 inputs. Although it is possible for the 3-input model to process $t = 4s$ data by taking its last 3 reported instantaneous throughput values, we instead chose to derive a separate model to take advantage of all available reported values for improved performance. As



(a) Training (b) Inference

Fig. 6. Illustration of training (a) and inference (b) phases of $FastProb$. It trains classifier-regression model pairs. The classifiers are used to determine if reported instantaneous throughput values are sufficient to accurately predict the average throughput of a transfer using corresponding regression models.

probing transfers are expected to execute for a short period, we limit the maximum probing duration to 30 seconds and train 30 separate models for each ML model type.

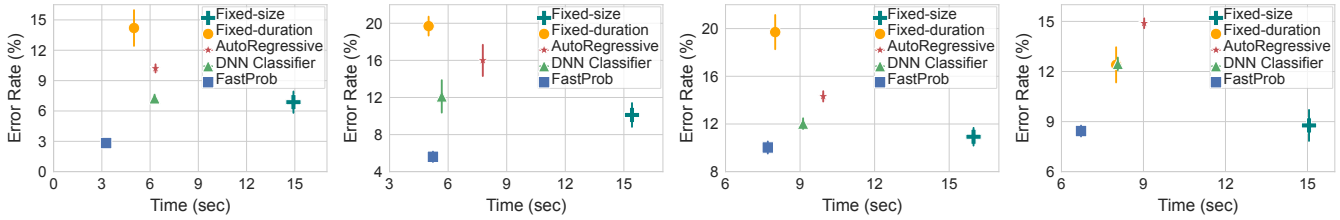
To train an n-input model, we take the first n instantaneous throughput values of each transfer log and feed them to the model along with the actual average throughput of the transfer, t_{avg} as a label. For example, for a transfer log with following instantaneous and average throughput values $< 100, 800, 900, 1100, \cdots >, < 1320 >$, we pass $< 100, 800 >$ as an input to 2-second regression models with a label $< 1320 >$, $< 100, 800, 900 >$ as an input to 3-second regression models with a label $< 1320 >$, and so on. To separate transfer logs as training and test, we use timestamp-based partitioning, which sorts all transfer logs based on their start time and places first 80% into the training and last 20% into the test category. This is intended to capture the *data shift* problem which can adversely affect the performance of ML models when system conditions change over time. We conducted 5 cross-validations, for which we first split the transfer logs into 6 groups using time-based partitioning. Then, we train the models using the transfer logs in the first group and test against the transfers in the second group; retrain models using the logs in the first two groups and test against the transfers in the third data group, and so on. We use Mean Absolute Percentage Error (MAPE) to calculate the error rate of models.

We use Gaussian process-based Bayesian optimization (using scikit-optimize library) to discover the optimal hyperparameters for the models, such as kernel and regularization values for SVR, number of trees and maximum depth for RF and XGB. We used AutoKeras [24] to tune the hyperparameters of NN models, which perform Neural Architecture Search (NAS) to find out the best performing architecture for the given dataset. Since we derive 30 models for each network, a total 120 architecture search is required, which incurs a significant training cost. Thus, we performed the architecture search for randomly selected subset of 20 DNN models and realized that all searches returned similar architectures that consist of $5 - 8$ Dense, ReLU, and Dropout layers. Hence we adopted a 7 layer architecture for all DNN models that are composed of Normalizer, Dense, ReLU, Dense, Dropout, ReLU, Dense layers respectively.

The results, as presented in Figure 4 show that the error rate of most ML models decreases as the number of inputs (i.e., probing duration) increases. The error rates of SVR and LR models are significantly higher compared to other models. In particular, they cause over 30% error rate in XSEDE-1 and XSEDE-2 networks due to failing to capture an accurate relationship between instantaneous and average throughput when instantaneous throughput fluctuates significantly. NN, XGBoost, and RF models all have competitive results as they attain less than 6% error with 6 inputs (i.e., 6-second probing) for HPCLab and ESnet networks. Although their error rates increase for XSEDE transfers due to higher throughput fluctuations, they can keep the error rate less than 12% for 6-second probing and less than 10% for 10-second probing intervals.

Although ML regression models can achieve less than 10% prediction error rate in all networks, the minimum duration to achieve it is not the same for all networks. For instance, one can use 3-input RF regression model to achieve less than 10% error rate in HPCLab and ESnet, but require 10-input RF regression model to achieve the same error rate in XSEDE-1. Even more challenging is the fact that different transfers in shared, production networks (i.e., XSEDE-1 and XSEDE-2) can converge at different times as some transfers fluctuate more than others in the same network due to difference in background traffic, dataset characteristics, and transfer settings. Therefore, it is important to determine the duration of each probing transfer in real-time based on its behavior, then use the corresponding regression model to make the prediction for average throughput.

Adaptive Regression with $FastProb$: To achieve this goal, we introduce $FastProb$ that pairs the regression models with a classifier which to determine whether or not a given regression model would be able to make high accuracy prediction for a probing transfer using its available instantaneous throughput values. As an example, if we want to find out whether or not we can use a 3-input regression model to predict the throughput of a probing transfer at $t = 3s$, we can train a binary classifier that will process populated instantaneous throughput values. $< t_1, t_2, t_3 >$, to determine if the 3-input regression model will be able to make accurate prediction for this transfer. If the classifier returns "yes", then we can use the 3-input regression model to predict the average throughput and terminate the probing transfer. Otherwise, we



(a) HPCLab (b) ESnet (c) XSEDE-1 (d) XSEDE-2

Fig. 7. Performance comparison of different probing algorithms. $FastProb$ offers more than 50% improvement in error rate and transfer duration compared to the state-of-the-art solutions.

let the transfer continue for another second and use the 4-input classifier to determine if the 4-input regression model would be able to predict the transfer throughput accurately using $< t_1, t_2, t_3, t_4 >$. Hence, each regression model is paired with a classification model to estimate if the regression model is likely to return an accurate prediction for a given transfer as shown in Figure 6(b). We kept the maximum runtime for probing transfers to 15 seconds in $FastProb$ as we noticed that the performance of the RF regressors does not improve significantly after 15 seconds (i.e., 15-input regressor) in Figure 4.

To train the prediction models, we first split the training set into two categories as Subset1 (70%) and Subset2 (30%), then train a regression model using the Subset1. The regression model is evaluated for transfers in Subset2 and error rates are calculated for each transfer. Next, the transfers with less than a certain error rate, P, are marked with the label 1 and others with a label 0. Finally, the labeled transfers of Subset2 are used to train a binary classifier. In the above example, we first use Subset 1 to train a 3-input regression and then test it on Subset2 to label them based on the performance of the regression model. Finally, a 3-input binary classifier is trained to decide whether or not the 3-input regression model can be used to make accurate predictions for probing transfer at their third second.

We implemented various combinations of XGBoost (XGB), Neural Network (NN), and Random Forest (RF) classifier-regressor pairs and evaluated them in terms of prediction time and estimation accuracy. For instance, we combined an NN classifier with an XGB regressor to test the performance of using NN as the binary classifier and XGB as the regression model. Although we omitted the full results due to space limitations, the best performance is achieved when RF classification models are paired with RF binary classifiers. Hence, $FastProb$ is composed of 14 RF Classifier-RF Regressor pairs. We again tuned the hyperparameters of the classifiers (as described in Section IV) to maximize the performance. As $FastProb$ uses confidence threshold (P) to determine the label of transfer in Subset2 as illustrated in Figure 6(a), we compared the model performance using different P values between $1 - 20\%$ for XSEDE-1 transfers. While higher P values (e.g., 20%) result in significantly high error rates in exchange for lower probing times, lower P values (e.g., 1%) lead to long probing times in exchange of higher accuracy. Thus. we used $P = 5\%$ as it strikes a good balance between prediction accuracy and probing duration.

TABLE II
COMPARISON OF $FastProb$ AGAINST RF REGRESSION MODEL.

Network	RF Regressor		$FastProb$	
	Error (%) - 3 Sec	Error (%) - 10 Sec	Error (%)	Time (sec)
HPCLab	2.8	2.1	2.8	3.2
ESnet	6.6	5.1	5.6	5.2
XSEDE-1	18.9	10.1	9.9	7.7
XSEDE-2	12.1	7.3	8.4	6.7

V. EXPERIMENTAL EVALUATIONS

We first compare the performance of $FastProb$ against the Random Forest regression model (as presented in Figure 4) in Table II. Since the regression models require probing duration to be specified by the user, we picked two fixed values as 3 and 10 seconds and used 3- and 10-input RF regression models as we observe that the models can achieve less than 10% error rate in $3 - 10$ seconds for different networks. We observe that while 10−input can keep the error rate less than 10%, it is an unnecessarily long probing duration for some networks such as HPCLab and ESnet. 3− input regression model, on the other hand, causes up to 18% average error rate and more than 50% for 8.4% of transfers. $FastProb$ can strike a balance between probing time and error rate as it can attain very similar error rates compared to the 10− input regressor model while requiring less than 7.7 seconds in all networks and less than 5.2 seconds in ESnet and HPCLab. This is mainly due to its ability to distinguish stable/predictable transfers from others such that they can be terminated quickly. While one can possibly choose a different probing duration for each network after analyzing the performance of the regression models (e.g, 3 seconds for ESnet and 10 seconds for XSEDE-1), $FastProb$ eliminates this step and automatically detects the optimal duration for each network with the help of its binary classifiers.

Comparison to state-of-the-art: We next compare $FastProb$ against state-of-the-art static (i.e., fixed-size, fixed-duration), and adaptive (i.e., Autoregressive [9] and DNN [21]) probing methods. *Fixed-size* approach transfers a fixed dataset and calculates the transfer throughput based on transfer duration. Yildirim et al. developed regression models to estimate optimal size for probing transfers and found that using $10 - 23\%$ of the original dataset size results in the best trade-off between accuracy and duration [3]. Therefore, we used $2 - 60$ GiB data depending on network settings (larger data in high delay, bandwidth networks) to match with the data size used in [3]. *Fixed-duration* method runs probing

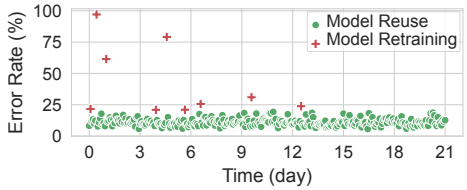


Fig. 8. Incremental training can be used to mitigate the need for exhaustive data collection as well as to adopt changing network conditions. (+) indicates retraining of the model due to degrading model performance.

transfers for a predetermined amount of time and calculates throughput based on the amount of data transferred [7]. Although earlier work set the probing duration to as much as 120 seconds [7], we kept it at 5 seconds for HPCLab and ESnet networks and 8 seconds for XSEDE networks to present its results in a similar time scale as other approaches. *Autoregressive* is a time-series model that uses third-degree linear regression to process instantaneous throughput values and predict the throughput for the next time interval. It then lets the transfer run for one more interval and compares its prediction against the observed throughput. If the prediction is close enough to the actual throughput (default is 5%), it then assumes that the model has captured the throughput behavior of the transfer thus terminates the transfer immediately and uses the model to predict the throughput upon convergence. Deep Neural Network (*DNN) Classifier* trains a model that can determine when to stop a probing transfer based on the convergence pattern of instantaneous throughput values. Once the convergence is detected, it terminates the transfer and uses an average of the last four throughput values to predict the average throughput.

Figure 7 demonstrates the performance comparison of the models. Probing duration takes more than 14 seconds when using the *fixed-size* approach in all networks. In return, it achieves a lower error rate compared to *Autoregressive* and *DNN Classifier* models. In contrast, *fixed-duration* yields shorter execution times while causing relatively higher error rates in most networks. The *Autoregressive* model keeps its execution time less than 10 seconds for all networks but returns a $1.5 - 3.5$ times higher error rate than $FastProb$, which can be attributed to its termination condition. It stops the probing transfers when the prediction made by the model is close to actual observation in the next interval. However, this comparison is susceptible to immature terminations when the predictions fall within a close range of observed throughput not because of throughput convergence but merely due to throughput fluctuations. *DNN Classifier* yields a better error rate and probing duration than the *Autoregressive* and static approaches (i.e., fixed-size and fixed-time) in most networks as it can adapt its execution time based on instantaneous reports. On the other hand, $FastProb$ outperforms *DNN Classifier* in terms of error rate and probing duration. The improvement ratio ranges between $17 - 61\%$ for error rate and $12 - 48\%$ for the probing period. The highest performance gain occurs in HPCLab, where it yields a 61% lower error rate and 48% lower probing time compared to the DNN classifier.

Incremental training: Gathering rich and diverse training datasets may not be possible in every network. Thus, an ability to train an initial model with limited data and update later as more data becomes available is critical to enable the adoption of supervised learning models. Even in networks with sufficient training datasets, evolving nature of networks and end systems in terms of configurations (e.g., network bandwidth and file system settings) and usage behavior demands model retraining based on new observations. Therefore, we implemented incremental training for $FastProb$ as follows: We first train an initial model using transfer logs of XSEDE-1 network that are collected in the first day of data collection phase, which contains around 500 transfer logs. We then evaluated the model against transfer logs of following days in batches (30 transfer logs in each batch). If the error rate of a batch of transfers exceeds a certain threshold (by default 20%), we retrain $FastProb$ using all previous transfer logs; otherwise keep using the same model. We repeat this process around 300 times (nearly $9,000$ transfers in total with each batch containing 30 transfers) for the data collected in three weeks time-frame. We selected XSEDE-1 to demonstrate the efficacy of incremental training in the most challenging network as all probing solutions obtain their worst performance in XSEDE-1.

Figure 8 presents the error rate for each interval along with timestamps in which we retrained $FastProb$ due to increased error rate. It is clear that we retrain $FastProb$ more often in the first few days compared to the last ones as initial models fails to perform well due to limited training data. In total, the model is retrained only 9 times out of 300 evaluations. The average prediction error rate is 11.93% with an average duration of 7.65 seconds. Although the performance of $FastProb$ with incremental training algorithm is similar to the cross-validation results (Figure 7), its standard deviation is noticeably higher in cross-validated experiments. This can be attributed to having larger training and test data in cross-validated experiments. One drawback of incremental training when used with Random Forest models is that it requires complete retraining of the models using full historical data as decision tree-based models do not support incremental learning. We believe that this is not a significant limitation as it takes only around 2 minutes to train $FastProb$ with all XSEDE-1 data that contains $10,000$ transfers logs using a server with 16GB RAM and Intel i7-7700 CPU @ 3.60GHz processor. Moreover, the frequency of retraining decreases rapidly as the model performance improves over time. Alternatively, one can replace RF in $FastProb$ with models that support incremental training such as Neural Network. However, we observe that NN models take significantly longer to train/update compared to training RF models from scratch. Specifically, it took 8 seconds to train the RF model with the first 500 transfer logs whereas it took 138 seconds for NN. Similarly, retraining the RF model when its error rate increases takes around 30 seconds whereas updating the NN model with only new data takes more than 200 seconds. Thus, RF model does not only offer advantage over NN model in terms of model performance but also in terms of training cost.

VI. An Application Scenario: Tuning Transfer Settings for Bulk Data Transfers

To demonstrate the benefit of improved probing accuracy and duration, we integrated $FastProb$ into a simple real-time transfer tuning algorithm. The algorithm searches for optimal concurrency levels for file transfers to increase the transfer throughput. Concurrency refers to the simultaneous transfer of multiple files using different processes and network connections. It is widely used to overcome I/O and network limitations in HSNs as a single I/O process or network connection falls short to reach maximum possible performance [6], [25]. However, the optimal concurrency value is not the same for all networks as it depends on several factors including file system configuration, network bandwidth, dataset characteristics, and network and I/O interference. The dynamic and unpredictable nature of some of these factors such as network and I/O congestion demands an adaptive approach to evaluate the performance of different concurrency levels at the runtime.

We implemented a simple search algorithm that evaluates a range of concurrency values for their performance (i.e., achieved throughput) using probing transfers to find the value that yields maximum throughput such that it can be used to transfer the rest of dataset. We tested this algorithm in HPCLab, Stampede2-Expanse, and BlueWaters-Expanse networks using 106 GB, 60 GB, and 960 GiB datasets consisting of 1 GiB files. While achievable throughput is around $20-30$Gbps in HPCLab and Stampede2-Expanse networks, it is nearly 85Gbps in BlueWaters-Expanse network, thus we arranged dataset size proportional to average transfer throughput in each network to give the optimization algorithm enough time to find the optimal solution. We check the concurrency values between 1 and n where n is defined as 5, 12, and 20 for HPCLab, Stampede2-Expanse, and BlueWaters-Expanse networks, respectively. Although training data was available for HPCLab and Stampede2-Expanse networks to train custom $FastProb$ classifier-regressor pairs, no such data was available for BlueWater-Expanse network. Hence, we used the $FastProb$ model trained with Stampede2-Expanse dataset to optimize probing transfers in BlueWaters-Expanse as both networks have similar characteristics in terms of high bandwidth between end points and shared parallel file systems at the end hosts. We repeated each experiment ten times and present average and standard deviation results for transfer throughput and probing time in Table III.

Overall, $FastProb$ can keep probing time below 5 seconds in all three networks and achieves $2-5$x shorter probing times compared to the other solutions. Since HPCLab servers are located in the same local area network, instantaneous throughput reports are more stable compared to others networks. As a result, Autoregressive and DNN Classifier models can keep the probing time less than 5 second and yield only $10-15\%$ less throughput than $FastProb$. On the other hand, transfer throughput exhibits more fluctuations in wide area networks, causing longer probing transfers when using AutoRegressive and DNN Classifier models similar to *Transfer-2* in Figure 5. Consequently, $FastProb$ can lower search time for the online optimizations significantly and lead to 25% to 35% higher overall throughput in Stampede2-Expanse and BlueWaters-Expanse, respectively.

TABLE III
Performance of real-time transfer tuning algorithm using different probing techniques.

Model	HPCLab	Stampede2-Expanse	BlueWaters-Expanse
Average Transfer Throughput (Gbps)			
Fixed-duration	15.1	3.2	31.7
DNN Classifier	17.1	3.2	33.9
AutoRegressive	17.2	3.4	37.0
$FastProb$	19.6	4.3	49.4
Average Probing Time (sec)			
Fixed-duration	10.01	11.57	11.56
DNN Classifier	4.67	10.95	9.71
AutoRegressive	4.13	7.86	10.95
$FastProb$	2.11	4.52	2.62

VII. Conclusion

This paper introduces $FastProb$ to predict the throughput of file transfers upon convergence by processing instantaneous throughput values. $FastProb$ leverages Random Forest-based classification models to determine if the throughput of a transfer can be predicted based on available instantaneous throughput values. If the classification models return a positive response, then it uses a Random Forest-based regression model to process instantaneous throughput values to estimate convergence throughput. The results from a wide range of network, dataset, and configuration settings show that $FastProb$ outperforms the state-of-the-art solutions by nearly 50% both in terms of probing time and measurement accuracy. We integrated $FastProb$ into a real-time transfer optimization algorithm to demonstrate the benefit of optimizing probing transfers. The results from three different networks show that $FastProb$ can shorten the probing duration by $2-5x$, thereby improving the transfer throughput by up to 35% for optimization algorithms.

Acknowledgement

The work in this study was supported in part by the NSF grants 1850353 and 2007789.

References

[1] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. M. Swany, S. Trocha, and J. Zurawski, "Perfsonar: A service oriented architecture for multi-domain network monitoring," in *International conference on service-oriented computing*. Springer, 2005, pp. 241–254.

[2] J. Zhang, R. Gardner, and I. Vukotic, "Anomaly detection in wide area network meshes using two machine learning algorithms," *Future Generation Computer Systems*, vol. 93, pp. 418–426, 2019.

[3] E. Yildirim, J. Kim, and T. Kosar, "Modeling throughput sampling size for a cloud-hosted data scheduling and optimization service," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1795–1807, 2013.

[4] 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.

[5] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "{PCC}: Re-architecting congestion control for consistent high performance," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 395–408.

[6] Y. Liu, Z. Liu, R. Kettimuthu, N. S. Rao, Z. Chen, and I. Foster, "Data transfer between scientific facilities–bottleneck analysis, insights, and optimizations," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 122–131.

[7] D. Yun, C. Q. Wu, N. S. Rao, and R. Kettimuthu, "Advising big data transfer over dedicated connections based on profiling optimization," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2280–2293, 2019.

[8] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for grid services," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*. IEEE, 2003, pp. 48–57.

[9] H. Sapkota, B. A. Pehlivan, and E. Arslan, "Time series analysis for efficient sample transfers," in *Proceedings of the ACM Workshop on Systems and Network Telemetry and Analytics*, 2019, pp. 11–18.

[10] R. Durairajan, S. K. Mani, P. Barford, R. Nowak, and J. Sommers, "Timeweaver: Opportunistic one way delay measurement via ntp," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1. IEEE, 2018, pp. 185–193.

[11] A. K. Paul, A. Tachibana, and T. Hasegawa, "An enhanced available bandwidth estimation technique for an end-to-end network path," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 768–781, 2016.

[12] R. Fontugne, A. Shah, and K. Cho, "Persistent last-mile congestion: Not so uncommon," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 420–427.

[13] F. Baccelli, S. Machiraju, D. Veitch, and J. C. Bolot, "On optimal probing for delay and loss measurement," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 291–302.

[14] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 4, pp. 56–69, 2007.

[15] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy, "Inferring persistent interdomain congestion," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 1–15.

[16] "MyESnet," 2020, "https://my.es.net/".

[17] "iPerf," 2022, https://iperf.fr/.

[18] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer, 2002.

[19] X. Yang, X. Wang, Z. Li, Y. Liu, F. Qian, L. Gong, R. Miao, and T. Xu, "Fast and light bandwidth testing for internet users," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 1011–1026. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/yang-xinlei

[20] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in *Passive and active measurement workshop*, 2003.

[21] H. Sapkota, M. Arifuzzaman, and E. Arslan, "Sample transfer optimization with adaptive deep neural network," in *2019 IEEE/ACM Innovating the Network for Data-Intensive Science)*. IEEE, 2019, pp. 69–76.

[22] "Extreme Science and Engineering Discovery Environment," 2022, https://www.xsede.org/ecosystem/resources.

[23] R. Kettimuthu, Z. Liu, D. Wheeler, I. Foster, K. Heitmann, and F. Cappello, "Transferring a petabyte in a day," *Future Generation Computer Systems*, 2018.

[24] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1946–1956.

[25] M. Arifuzzaman and E. Arslan, "Online optimization of file transfers in high-speed networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.