Learning Transfers via Transfer Learning

Md Arifuzzaman and Engin Arslan
Computer Science and Engineering,
University of Nevada, Reno
arif@nevada.unr.edu, earslan@unr.edu

***Abstract*—Detecting performance anomalies is key to efficiently utilize network resources and improve the quality of service. Researchers proposed various approaches to identify the presence of the types of anomalies including heuristic (e.g., change point detection) and Machine Learning (ML) models. These models typically collect socket statistics to decide whether or not a given traffic is anomalous. Although these models have high accuracy in the networks that they are trained for, they perform poorly when transferred to a different network mainly because the models are heavily dependent on bandwidth and RTT related metrics such as the number of packets delivered and minimum round trip time. In this paper, we propose a novel parameter transformation method to eliminate network dependence of the ML models thereby increasing the performance of the transfer learning for anomaly diagnosis models significantly. We finally validate the findings through experimental evaluations conducted on simulated and production networks using multiple congestion control algorithms.**

***Index Terms*—Transfer Learning, Feature Transformation, Network Anomaly Detection, Random Forest.**

I. Introduction

Performance of data transfers in production networks are adversely affected due to being exposed to various types of network anomalies such as packet loss and jitter. Detecting such anomalies is key to take necessary actions in a timely manner thereby increase resource efficiency and meet stringent performance requirements of distributed workflows. Operating systems provide several performance metrics for active network flows (e.g., number of transmitted packets, lost packets, average round trip time), which can be used to identify the presence as well as the type of anomalies.

However, analyzing performance metrics in real-time to detect and diagnose performance anomalies is beyond the capabilities of human operators due to high volume and velocity of input data. Thus, researchers developed automated solutions to process real-time performance statistics to pinpoint the root causes of performance anomalies in a timely manner. Specifically, ML/AI solutions are highly effective as they can extract complex relationships between a large number of features to make high-precision predictions compared to traditional statistical models. However, they often suffer from poor performance when transferred to new domains, making it extremely challenging to adopt them in production networks where labeled data collection is not possible.

In this paper, we develop models for network anomaly diagnosis problem that can achieve high accuracy when transferred to new networks without requiring new data collection. To achieve this goal, we first identify key features that can be used to infer the types of anomalies. We then innovate a feature transformation method to convert domain dependent metrics (e.g., such as number of retransmitted packets and average round trip time) to domain independent forms to enable the transfer of anomaly diagnosis problems across the networks. Compared to standard normalization (i.e., column-based normalization) which requires test data features to be in the same range as in training data features to work as expected, we normalize the selected features using the other features of the same data entry. For instance, we divide the number of retransmitted packet count to total transmitted packet count reported in the same entry such that it will indicate retransmission rate, which is not dependent of network bandwidth. Experimental results show that parameter transformation significantly improves the performance of transfer learning for anomaly diagnosis ML models. Specifically, while the accuracy of ML models when transferred to new networks are below 60% using standard normalization, the proposed row-based transformation improves the performance to up to 90%.

We further explore the impact of the congestion control algorithm on the performance of the proposed models. Our preliminary analysis on TCP Cubic, HTCP, and BBR reveals that the proposed parameter transformation also results in high accuracy when applied between different congestion control algorithms. In summary, we make the following contributions in this paper:

- We run extensive data collection in multiple emulated and productions networks to develop supervised machine learning models that can identify the root causes of performance anomalies of network transfers by processing socket statistics. We find that these models are over 90% accurate when they are tested in the network that they are trained for, but only attain less than 60% accuracy in different networks.
- We apply feature engineering to only use a subset of features in the model training and introduce novel parameter transformation to eliminate domain dependence for selected features. The results indicate that parameter transformation is an effective approach to minimize the domain dependence for the ML models, therefore the performance of transfer learning improves over 75% in all networks including production ones. We also investigate the reasons behind why the performance of transfer learning is unable to match with domain performance of

the ML models.
- We evaluate the performance of proposed feature transformation for widely used congestion control algorithms TCP Cubic, HTCP, BBR. We find transfer transfer learning performs fairly well (above 85% accuracy) across the congestion control algorithms, thus making it even simpler to transfer to new networks even if congestion control algorithms is different.

II. Related Work

Throughput fluctuations are the norm in research networks where several components of end-to-end data transfers are shared resources (i.e., storage, DTN, and network). As a result, system administrators tend to blame resource contention as the most likely reason for poor transfer throughput. However, non-congestion-related anomalies, such as faulty equipment, transient routing changes, and unanticipated impacts of system configuration changes, occur frequently but go undetected due to lack of comprehensive monitoring and anomaly detection solutions. Many universities and research centers use PerfSonar [1] to identify common networking problems such as decrease in throughput and increase in round trip time. Since it uses periodic active measurements to measure the network performance, it is unable to detect anomalies that take place between measurement intervals.

Most existing work proposed solutions to detect the presence of network anomalies without providing any information about underlying reasons. For example, Zhang et al. proposed Principal Component Analysis (PCA) for PerfSonar logs to identify the feature set that can be used to differentiate anomalous and normal traffic [2]. Lakhina et al. used Principal Component Analysis (PCA) on link utilization data (i.e., SNMP) to detect volume-related anomalies [3]. They proposed a threshold-based detection algorithm to identify significant variance in *residual* traffic and distinguish the flows involved in the anomalous event. Similarly, Rao et al. applied PCA on TCP performance metrics (captured by Tstat [4]) to extract principal components for normal traffic in high-speed networks [5]. Mapping normal and anomalous traffic into a new space using the first two principal components revealed a clear distinction between these two groups. They showed that the features found by normal traffic analysis do not reflect the behavior of abnormal traffic through which one can determine whether a given traffic pattern is anomalous. Ana et al. used Random Forest Regressor to related flow sizes to TCP performance metrics as captured by Tstat [6]. Once the flow size of transfers is estimated, the it is used determine if the flow size of a transfer is unexpectedly small or large, which is finally used to infer anomalies. Dao et al. implemented a change point detection algorithm using time-series throughput data [7]. The proposed solution first splits observed transfer rates into two as base time and congestion delay. Base time defines the maximum rate of a transfer in a given network determined by the network connection, the storage systems, and the CPU of the machines involved in the transfers. Congestion delay, on the other hand, is the increase in base time due to network congestion. The base time of any time range is calculated using minimum quantile regression, which is then used to estimate transfer completion time. Then, this value is subtracted from actual transfer time to find deviation in completion time, which would indicate congestion. Finally, an anomaly detection signal is triggered if several transfers in a time range exhibit significant deviation from estimated completion time.

In the area of anomaly diagnosis, several work proposed developing supervised machine learning models to detect and diagnose performance anomalies that scientific workflows are exposed to such packet loss, duplication, and reordering [8]–[10]. For example, Tu et al. [10] proposed hyperparameter optimization for Gradient Boosting (XGBoost) ML model to detect and diagnose packet loss, duplication and reordering anomalies using *Tstat* transfer logs. The results show that hyperparameter optimization can be executed quickly and lead to up to 28% gain in F-score compared to "off-the-shelf" performance. We thus use Gaussian process-based Bayesian optimization (via python `scikit-optimize` library) to discover the optimal the hyperparameters settings that maximizes the performance of the models. In a previous work, we applied Deep Neural Network to process performance metrics (e.g., TCP statistics, file system counter, etc.) and diagnose the root causes of performance anomalies for file transfers, such as I/O interference, packet loss, packet corruption, and overloaded end hosts [11]. Despite yielding high accuracy for the networks that they are trained for, none of these solutions address the domain dependence problem as the proposed supervised learning models are heavily dependent on network settings such as bandwidth and RTT. In a previous work, we conducted preliminary analysis on the impact of parameter transformation on the performance of transfer learning for anomaly diagnosis models [12]. This work significantly extends the previous work by (1) we share insights into why and how parameter transformation improves the performance of transfer learning, (2) present experimental results for real-world networks, (3) assess the impact of congestion control algorithms on the performance of the models both for same network and different network evaluations, (4) we present future directions to enhance the performance of anomaly diagnosis models such as reducing the scale of training data collection

III. Network Anomaly Diagnosis

Network anomalies can lead to severe performance degradation such as decreased throughput and increased network delay. For example, packet loss anomaly with as small as 0.1% (i.e., 1 in every 1000 data packets are lost) rate can lead to $5-7\times$ decrease in throughput. Therefore, it is crucial to detect and mitigate them in a timely manner to sustain high network performance and improve the quality of experience for users. Equally important to anomaly detection is the root cause analysis since troubleshooting efforts are highly dependent on the types of anomalies. As an example, packet loss can be caused by increased network congestion, thus one can apply traffic engineering to evenly distribute the load over alternate

paths. On the other hand, high packet corruption indicates faulty cable or network devices that needs to be replaced to mitigate the issue. Yet, most previous work in this area only focus the detection of performance anomalies without providing any information about underlying reason(s), leaving daunting root cause analysis task to network operators.

To address this issue and expedite the troubleshooting process, we develop models that cannot only detect the presence of anomalies but also can predict the underlying reasons. Specifically, we focus on five network anomaly conditions as *jitter, packet duplication, packet reorder, packet corruption*, and *packet loss*. *Jitter* (aka delay jitter) defines the variation in end-to-end delay, which can be caused by various reasons including network congestion, route changes, and faulty hardware [13]. We create three jitter anomalies as 30%, 50%, and 60% with respect to base RTT. For example, if base network delay (i.e., RTT) is 100ms, then 30%, 50%, and 60% jitter rates will cause 30ms, 50ms, and 60ms variation in observed RTT, respectively. *Packet duplication* (duplicate henceforth) anomaly causes some packets to be transmitted destination multiple times. It can happen when a sender does not receive an acknowledgement message for some packets within a specific time duration, so it resends them assuming that the original packets were not delivered even if they were indeed delivered. It can also happen because of defective network devices duplicating some data packets and sending them along with the original ones. It can be defined as the percentage of packets duplicated, hence we simulate 20%, 30%, 40% duplication rates.

Packet reorder (reorder henceforth) is among common network anomalies in which data packets are delivered out of order. It can happen when networks experience routing instabilities which causes packets of the same flow to take different routes and be exposed to different network delays. It can also occur when multipath routing is used to split flows to multiple routes to take advantage of available bandwidth in alternative routes. Similar to the duplicate anomaly, it is defined as the percentage of packets that are reordered, thus we simulate three levels as 20%, 30%, and 40%. *Packet corruption* (corruption henceforth) causes data packets to be exposed to multiple bit errors in a way that it cannot be recovered through available error correction codes such as cyclic redundancy check (CRC). It happens due to faulty cables and network devices. As it is a relatively rare anomaly type, we simulate small percentages as 0.1%, 0.5%, 1%. Finally, *packet loss* (loss henceforth) causes data packets to be dropped due to non-congestion related events. While packet loss can also happen due to network congestion, its ratio is typically very small in congestion cases as most congestion control algorithms reduce sending rate drastically to alleviate the congestion. Thus, high packet losses typically take place due to defective network equipment [14]. To reproduce these anomalies, we use Linux Network Emulator (`netem`) [15] which allows the emulation of network configurations and common performance anomalies.

A. Data Collection

We first use Emulab [16] to create several network settings with various bandwidth and delay values to measure the performance of anomaly diagnosis models in different network settings. We build a simple network topology in which a sender and a receiver is connected via switch. We configure nine networks using a combination of three bandwidth (100Mbps, 1000Mbps, 5000Mbps) and delay (10ms, 30ms, 100ms) values. We refer to these networks as $<Bandwidth(Mbps)>M<RTT(ms)>ms$ format. For example, 5000M10ms refers to the setup where network bandwidth is 5000 Mbps and Round Trip Time (RTT) is set to 10 ms. We run 200 `iPerf3` [17] transfers (each 20 seconds long) for each anomaly type as well as normal condition. The different rates of the same anomaly groups (e.g., 0.1%, 0.5%, and 1% packet loss anomalies) are tagged with the same label. The default congestion control algorithms is set to TCP Cubic, but Section IV-A presents results for different congestion control algorithms.

As we aim to predict the types of performance anomalies beyond their presence, simply using transfer throughput is not sufficient as multiple anomalies lead to throughput degradation in a similar degree. Thus, we benefit from existing tools that report several performance metrics regarding to the status of transfers such as the number of bytes transmitted and the number of retransmitted packets. For example, Linux utility *Netstat* [18] captures a multitude of performance metrics that can be used to debug performance issues. However, it reports combined results for all active flows, thus it is not always suitable to debug the performance issues of individual flows. Since we are using dedicated instances for Emulab experiments and only create one transfer at a time, the values reported by *Netstat* can be used to analyze the performance of that flow. Another Linux utility *ss* [19] provides flow-level socket statistics for active connections [19]. Finally, *Tstat* [4] uses *tcpdump* to inspect active network connections and reports a large number of performance metrics upon the completion of flows. Since these tools report a different set of metrics (some of which may overlap), we captured performance metrics using all three of them to compare their effectiveness in capturing the anomalies.

B. Feature Selection

Tstat, *Netstat*, and *ss* report up to 150, 64, and 29 metrics, respectively. Since most of them either not consistently reported or have fixed values, we first apply feature selection before training ML models as follows: We randomly select 20% of gathered transfer logs to train a Random Forest classifier. In the training phase, Random Forest model calculates a significance score for each input parameter to quantify their impact on the classification of anomalies. We choose features whose combined importance score is more than 95% of sum of all features' importance scores. We repeat this process ten times using a different subset of the transfer logs to identify a set of features that are consistently selected in each of the ten repetitions. At the end, we select 14 features for *Tstat*, 7

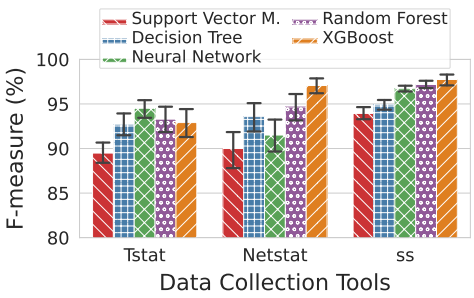


Fig. 1. Performance comparison of ML models for network anomaly diagnosis problem. The results are average of nine networks created in Emulab.

features for *Netstat*, and 7 features for *ss* data to train ML models.

C. Model Training

We train several supervised ML classifiers using the data collected in Emulab. Out of many we evaluated, we only report the performance of Random Forest (RF), Decision Tree (DT), XGBoost (XGB), Neural Network (NN), Support Vector Machine (SVM) models as others (e.g, Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and Long Short Term Memory (LSTM)) either fail to obtain competitive results or require extensive manual work to find the optimal model architecture. As an example, CNN is able to perform slightly better $(2-3\%)$ compared to Random Forest model, but finding the right neural architecture (e.g., the number of hidden layers, the number of dropout layers, etc.) is a time-consuming process, thus we focus on ML models whose training cost is reasonable. We leverage scikit-optimize library to optimize the hyperparameters of the selected models. Specifically, it finds the number of trees and tree depth for RF, tree depth for DT, the number of layers, neuron count per layer, activation function, solver method, and learning rate for NN. For XGB, it tunes learning rate, number of estimators, max depth, minimum child weight, gamma, regularization parameter (alpha), subsampling, and colsample_bytree ratio. We also apply standard normalization on the dataset after splitting data into training and test groups using $80\% - 20\%$ split ratio. Please note that test dataset normalization uses the same scaling metrics (i.e., average and standard deviation) that are calculated during the normalization of training dataset to avoid data leakage.

We first train a separate model for each of nine networks with different bandwidth and RTT settings. The trained models are then evaluated using the test dataset of the same network. To evaluate the performance of ML models, we adopt 5-fold cross validated F-score (also referred as F-measure) value which is an harmonic mean of precision (i.e., the number of true positive results divided by the number of all positive results) and recall (i.e., number of true positive results divided by the number of all samples that should have been identified as positive) values [20].

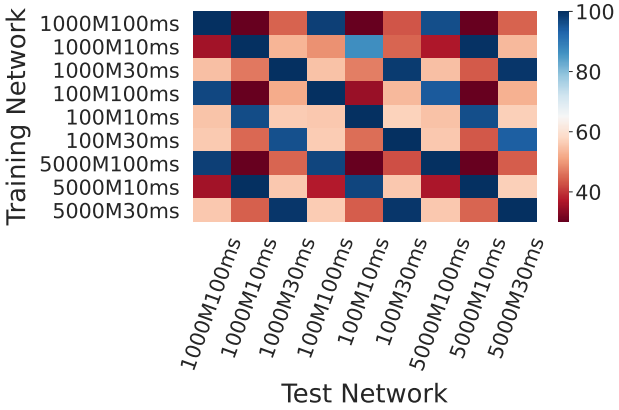


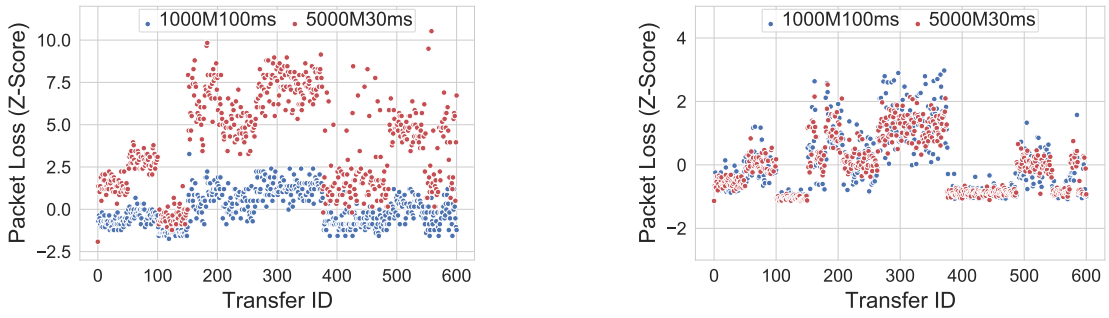
Fig. 2. Performance of XGBoost models for transfer learning.

D. Model Evaluation

Figure 1 presents an average F-score for the nine network settings we created in Emulab. We observe that SVM attains the lowest performance for all three datasets whereas NN, DT, RF, and XGB models all attain high $(92-97\%)$. Among them, XGB performs the best $(96-97\%$ F-score) when trained with *Netstat* and *ss* datasets. In terms of the type of the dataset, all models achieve higher and more stable performance using the *ss* dataset. Specifically, while the performance of XGB classifiers ranges between 87% and 98% using *Tstat* dataset, its performance stays within 95% and 99% using *ss* dataset. This can be attributed to the fact while we can query *ss* multiple times to capture the performance statistics of a specific time duration (e.g., $t = 5s$), *Tstat* reports metrics only once at the end of transfers. This in turn leads to increased noise in the data as reported values are affected performance fluctuations that happen during startup and tear-down periods. Similarly, the models have nonegligible fluctuations when trained with *Netstat* data which can be explained due to lack of per-flow reports and absence of RTT related metrics. Therefore, we use *ss* data for the rest of the analysis.

As it is time consuming to gather training data for a variety of network settings, we explore the performance of the ML models in diagnosing errors when they are tested in different network settings. The ability to transfer anomaly detection and diagnosis models to new networks is a key to increase the adoption of ML solutions since it may not be possible to gather training data in all networks, in particular in shared production ones. Therefore, we evaluate the performance of XGB classifiers that are trained with a data collected in one network to diagnose errors in another network. As we trained a separate model for each of nine settings created in Emulab, we test each of these models in the other eight networks to evaluate their performance. To give an example, the model trained for 100M10ms network is evaluated in all other eight network settings (i.e., 100M30ms, 100M100ms, 1000M10ms, etc.) to measure if it's still able to diagnose the root causes of the performance anomalies.

The results as presented in Figure 2 indicate that despite yielding over 97.8% accuracy when the XGB models are



(a) Before Parameter Transformation

(b) After Parameter Transformation

Fig. 3. Distribution of retransmission values (*retrans*) in training (1000M100ms) and test (5000M30ms) networks for packet loss anomaly. The proposed transformation method improves the similarity of data distribution between training and test dataset despite having different bandwidth and RTT settings.

TABLE I
SS METRICS USED TO TRAIN PREDICTION MODELS

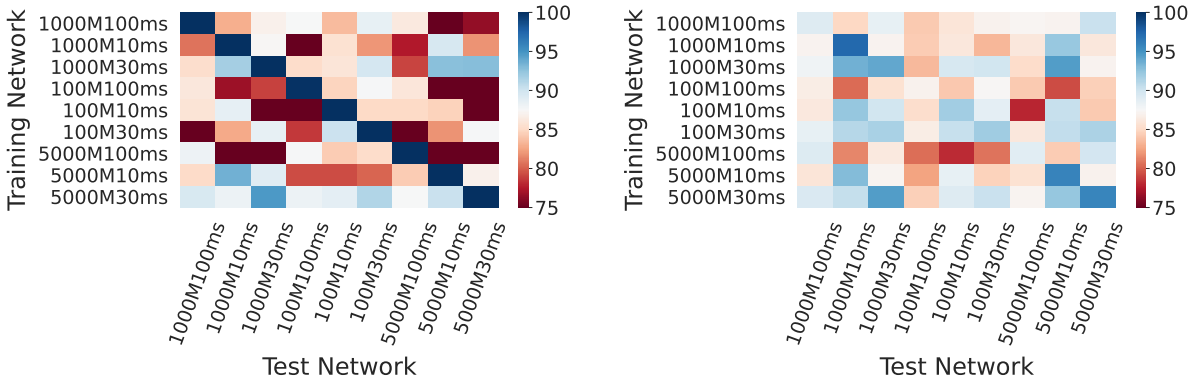
Feature Name	Base Metric	Importance Score
minrtt	rtt_avg	0.21
rtt_std	rtt_avg	0.07
ssthresh	cwnd	0.12
dsack_dups	segs_out	0.16
retrans	segs_out	0.19
notsent	segs_out	0.08
reord_seen	segs_out	0.16

tested in networks that they are trained for, their performance degrades severely when they are transferred to new networks as the average F-score of all nine models is 58% with some returning less than 30% F-score. This is mainly due to the difference in the range of parameters in different network settings. For example, maximum RTT value of 20ms in a network with average RTT of 10ms is a sign of jitter anomaly. On the other had, the same maximum RTT value of 20ms will be classified as normal when average RTT of the network is 20ms. Standard normalization does not resolve the issue as it does not change the distribution of the test dataset. It is also not feasible to normalize training and test datasets independently since that requires test dataset to contain all anomaly types and be available altogether, an assumption that cannot be guaranteed. Except RF classifier which yields only slightly higher $(2-3\%)$ performance than XGB, we observe similar or worse transfer learning performance when using other ML models (e.g., Neural Network) or dataset types (i.e., *Tstat* and *Netstat*). It is therefore evident that transfer learning does not work for network anomaly diagnosis when a target network has different bandwidth and RTT values compared to training network.

IV. Transfer Learning

As original models do not perform well when transferred to new networks due to bandwidth and RTT difference between the training and test networks, we investigate ways to either eliminate or transform the domain-dependent features. For example, the number of transmitted bytes is proportional to network bandwidth, thus using it when training a model adversely affects the performance of the model when it is transferred. One possible option is to only use the metrics that are not dependent to network bandwidth and RTT such as maximum segment size and MTU value. However, this leads to significant performance decrease as most important features are dependent to bandwidth and RTT. Thus, we explore the possibility of transforming the features to network independent forms. For instance, we can divide the number of packets *retransmitted* to the number of packets *transmitted* to transform it to retransmission rate, which is not directly related to the absolute value of bandwidth. Similarly, average RTT can be transformed to RTT-independent form by dividing it to maximum RTT value of the transfer. By extending this idea, we convert all 7 features of *ss* dataset into network independent forms as shown in Table I. In a nutshell, we divide RTT-related metrics to average RTT value (i.e., rtt_avg), packet count metrics to total transmitted packets (i.e., $segs\_out$). Note that the base metrics are not among the 7 metrics selected after the feature selection due to having low impact on classifying the anomalies. As a result, we did not eliminate any of the metrics, but rather transformed them to bandwidth and RTT independent forms.

Figure 3 demonstrates the impact of the proposed feature transformation. It shows the normalized value of *retrans* (the number of retransmitted packets) parameter when packet loss anomaly is injected with 0.1%, 0.5%, and 1% ratios. We first normalize the values of packet loss for network A, then use the same normalization metrics (i.e., average and standard deviation) to normalize the network B to simulate a scenario where a model is trained using training data from network A and tested with data from network B. Note that the same packet loss rates are introduced in both networks, thus similar outcomes are expected in terms of transmission rates. However, when regular standardization is applied, *trans* values do not overlaps as expected due to difference in absolute values. On the other hand, when it is transformed using the proposed transformation technique, the values from both network fall



(a) XGBoost (b) Random Forest

Fig. 4. Performance of Random Forest and XGBoost models for transfer learning experiments in Emulab after applying feature transformation.

into the same range, letting ML models to distinguish the types of anomalies in new network settings.

Figure 4(a) presents the F-score of the XGBoost models after applying the proposed feature transformation. While its performance of transfer learning was on average 58% (with as low as 20% in some cases) when the input parameters are used in the raw format, the proposed feature transformation has improved the performance to 84% with the lowest performance of 63.85%. As XGB is known for its overfitting issues [21], we also evaluate the performance of the RF models in Figure 4(b). Clearly, the RF models obtain higher and more stable performance then XGB models with average and lowest F-scores of 88% and 75.63%, respectively. We therefore present the results of RF models in the rest of the paper as they outperform the XGB models in transfer learning experiments significantly in exchange of only slightly $(2-3\%)$ lower performance for same network experiments.

It is important to note that transforming the features before training the RF classifiers causes a slight performance degradation when the models are evaluated in the same network that they are trained for. The performance of the RF models (i.e., F-score) are on average 97% before applying the parameter transformation (as shown in *ss* column Figure 1), whereas it decreases to 94% after the parameter transformation. We believe that this is a reasonable trade-off compared to considerable gain achieved for transfer learning performance.

Despite the significant improvement, transfer learning performance still falls behind the same network performance by around $10-15\%$. This can be attributed to the difference in distribution of some features. Figure 5 shows the average value of `dsack_dups` (duplicate selective acknowledgement) parameter for *duplicate* and *Reorder* anomaly classes as observed in each of nine network settings. Reported `dsack_dups` value is higher in the case of *duplicate* anomaly compared to *reorder* anomaly in each network. However, its value for *reorder* anomaly overlaps with its value in *duplicate* anomaly in different networks, which confuses the models when they are transferred to new networks. As an example, its average value is 0.28 for reorder anomaly (after the parameter transformation) in 1000M30ms which is significantly smaller than its value for *duplicate* anomaly in the same network. However, its value is higher than compared to *duplicate* anomaly in 1000M10ms network for which the value of `dsack_dups` is around 0.21. Consequently, even if the transformed value of `dsack_dups` is always higher for *duplicate* anomaly compared to *reorder* anomaly in each network, this does not hold true across the networks, causing some reorder anomaly cases to be categorized as duplicate anomaly when models are transferred. Since transfer learning requires features to follow a similar distribution in training and test datasets to work well [22], [23], such differences in some features across the networks adversely affect the performance of the models when transferred to new settings.

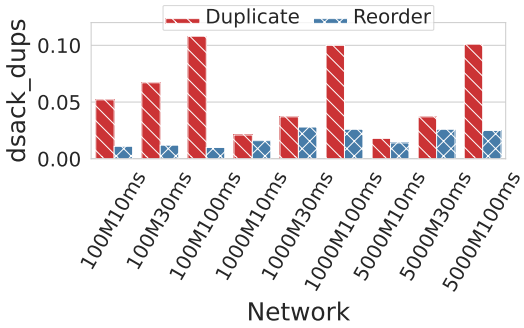


Fig. 5. Comparison of transformed *dsack_dups* feature for different anomaly types Emulab. Although its value is always smaller for *reorder* anomaly compared to *duplicated* anomaly in each network, this does not hold true across the network, thus causes performance issues for transfer learning.

A. Impact of Congestion Control Algorithm

Although TCP Cubic is the most commonly deployed congestion control algorithm (CCA), researchers have been working to develop alternative solutions to overcome the limitations of Cubic such as slow convergence in long fat networks and poor performance in the presence of random packet losses. As an example, Google introduced BBR in 2016 to for improved resilience to random packet losses and

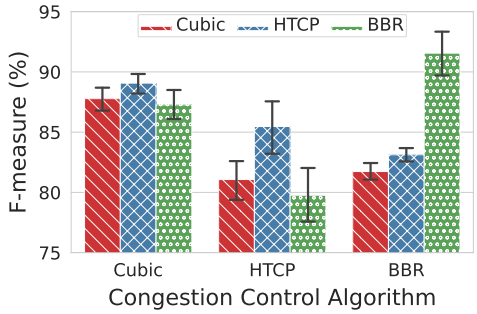


Fig. 6. Transfer learning performance across congestion control algorithms.

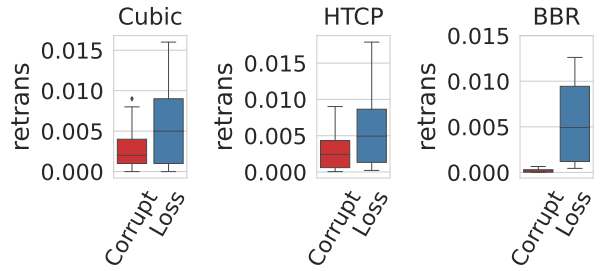


Fig. 7. Retransmission rate (*retrans*) is an important metric to distinguish *Corrupt* and *Loss* anomalies. BBR returns separate range of for it under the two anomaly type, thus the anomaly diagnosis models attain higher performance when trained with BBR datasets.

increased overall resource utilization [24]. Likewise, many high-speed networks adopt HTCP [25] to enhance transfer performance in long fat networks. We thus assess the impact of CCA on the proposed parameter transformation method. Figure 6 presents an average F-score for transfer learning experiments (i.e., each of nine RF model are tested using the data from other eight network settings) when CCA is set to Cubic, BBR, and HTCP. The reported values are average F-scores for the nine RF models; i.e., average of all values in Fig 4 except the values on diagonal line. In addition, the figure reports the performance of the models when they are evaluated against different CCA. That is, the left most blue line (i.e., HTCP) reports the F-score for a model that is trained in one network setting using Cubic as a CCA and evaluated in other eight networks when using HTCP as a CCA. Thus, it reports the performance of the models when both network settings (i.e., bandwidth and RTT) and CCA are different than the ones in training data.

It is clear that the RF classifiers yield the best when using BBR as a CCA. Specifically, BBR trained models achieve around 92% F-score for transfer learning when the test datasets are also captured using BBR. In contrast, Cubic and HTCP models yield 87% and 85% respectively when they are tested against the datasets that are collected in different network settings using the same CCA. We realize that the performance difference is due to variation in the range of some features when BBR is used instead of Cubic or HTCP. Figure 7 shows transformed and normalized value of retranmission rates (*retrans*) for *corrupt* and *loss* anomalies in Emulab (1000M30ms) for Cubic, HTCP and BBR. Since *retrans* is

TABLE II
SPECIFICATIONS OF REAL-WORLD NETWORKS.

ID	Source	Destination	Bandwidth	RTT
WAN-1	DTN1	DTN2	40G	10 ms (emulated)
WAN-2	TACC	UC	1G	32 ms
WAN-3	TACC	SDSC	1G	58 ms

TABLE III
TRANSFER LEARNING PERFORMANCE FOR CUBIC DATASET

Training Dataset	Test Dataset	F-measure (%)
Emulab	WAN-1	81.9 ± 2.8
WAN-1	Emulab	81.1 ± 3.7
WAN-2	WAN-3	81.2 ± 1.6
WAN-3	WAN-2	79.8 ± 1.8

one of the most important features used to separate these two anomaly types, its distribution plays a key role in finding the right class. Clearly, it can easily be used to separate *corrupt* and *loss* anomalies across the networks when using BBR as there is clear difference in the range of values. On the other hand, its value overlaps for *corrupt* and *loss* anomalies when using Cubic and HTCP, causing models to misclassify some *loss* anomalies as *corrupt* and vice versa. We aim to conduct a deeper analysis on why BBR acts differently in the case of packet corruption in a future work.

B. Performance Evaluation in Real-World Testbeds

In this section, we evaluate the performance of the proposed parameter transformation for transfer learning experiments in three real-world networks. Table II lists the locations of source/destination end points, bandwidth and RTT for these networks. Two data transfer nodes (DTN1 and DTN2) located in the same local area network used to collect data in an isolated environment. We injected 10ms delay to emulate wide area network behavior. We use Texas Advanced Computing Center (TACC, located in Austin, Texas), University of Chicago (UC, located in Chicago, Illinois), and San Diego Supercomputer Center (SDSC, located at San Diego, California) to conduct transfers under different bandwidth and RTT conditions. Similar to Emulab results, we observe high F-scores for the models when they are trained and tested in the same network.

Table III present the performance of transfer learning across different testbeds. Since both Emulab and WAN-1 is configured with simulated RTT, we compare them against each other, Compared to $85-90\%$ transfer learning F-score we obtain when testing models in different networks in Emulab (Figure 6), the performance degrades by around $5-10\%$ when the models are transferred to WAN-1. We notice that this is due to `netem`'s inconsistent behavior in different networks. As an example, injecting 1% packet corruption results in very close to 1% packet loss ratio in all Emulab transfers whereas it causes the packet loss rate to fluctuate between 0.5% and 0.8% for WAN-1 transfers. We leave a deeper analysis on `netem`'s performance difference in different networks as a future work.

TABLE IV
A SUBSET OF SS METRICS THAT CAN BE USED TO TRAIN ML MODELS

Feature Name	Base Metric	Importance Score
minrtt	rtt_avg	0.30
dsack_dups	segs_out	0.19
retrans	segs_out	0.34
reord_seen	segs_out	0.17

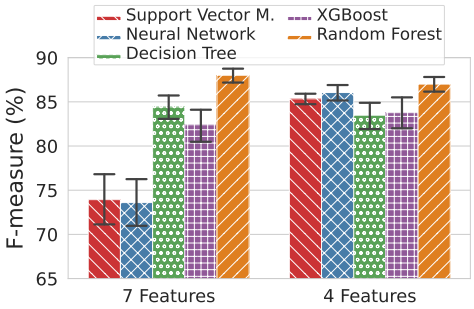


Fig. 8. Transfer learning performance comparison between 7 features and 4 features models

Similarly, the performance of transfer learning is around 80% when tested between WAN-2 and WAN-3 networks. It is also important to note that WAN-2 and WAN-3 are shared, production networks, thus potential network interference can always lead to mislabeled data during data collection phase.

V. DISCUSSION AND FUTURE DIRECTIONS

To the best of our knowledge, this paper makes a first attempt to develop transferable machine learning models for network anomaly diagnosis. The results indicate that the proposed novel feature engineering can improve the performance of transfer learning from around $50-60\%$ to over 80% F-score across all networks. In this section, we investigate ways to further improve the performance as well as model explainability.

Model Explainability: As mentioned in Section III-B, we find that 7 out of 29 features reported by *ss* tool are sufficient to capture 95% of the variation in anomaly type (or lack of it). While RF models yield the best performance, it is not the best option when it comes to the interpretability. Decision

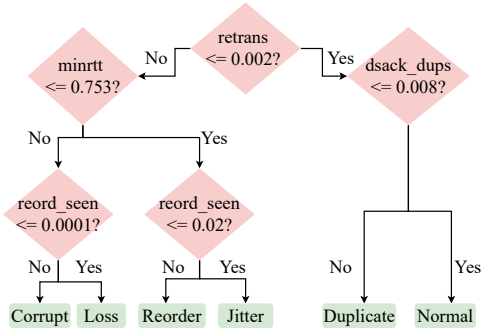


Fig. 9. A visualization of sample decision tree model created using 4 features of *ss* dataset.

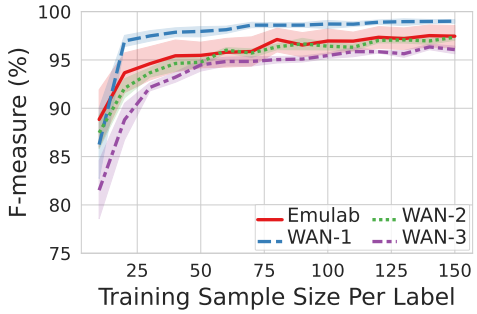


Fig. 10. Performance comparison of ML models with respect to training sample size

Tree, on the other hand, can yield competitive results (95% vs 97% for same network experiments as shown in Figure 1) while improving the model explainability significantly. As an example, the RF models typically consist of 100 trees whose depth ranges between 5 and 31, making it extremely difficult to inspect. In comparison, DT models have depth of $4-19$ when trained with the same dataset, which are, despite being simpler than RF models, still hard to debug. To further enhance the interpretability, we assess the impact of reduced feature set size. Specifically, we train all the models using only 4 of 7 original features based on importance scores as listed in Table I. The selected features and their updates scores as well as base metrics used to transform them are given in Table IV. Figure 8 presents the transfer learning performance of the ML models for Emulab dataset (Cubic as a CCA) when they are trained with fewer features. Surprisingly, the performance of SVM and NN improved significantly (around 10%) when some of the features are discarded, which can be the result of inconsistent distribution of values for discarded features. The features mentioned in Table IV have very consistent and expected behavioral patterns across the networks. For example, the value of *reord_seen* parameter is high in all networks when the *reorder* anomaly is injected. Note that tree based models are more resilient against including somewhat inconsistent parameters as all major branches in these model are formed using the most important features, thus the discarded features are used to classify a small number of samples. Yet, the performance of DT and RF models, however, suffer slightly with $1-2\%$ loss in F-score. In return, the interpretability of the DT model improves significantly. We find however that the performance of the RF and DT models do not decreased when the feature set size is reduced to 4, thus using simpler models does not necessarily require performance sacrifice. Figure 9 illustrates a sample Decision Tree model for one of the Emulab networks when 4 features are used for training. Such simple models make it easier to gain insights into the decision-making logic of the model. For example, we can observe that *duplicate* anomaly can be distinguished by looking at the values of *retrans* and *dsack_dups* metrics.

Impact of Congestion Control Algorithm: We notice that transfer learning experiments mostly fails to distinguish packet *loss* and *corrupt* anomalies when using Cubic as a CCA (Table III). This is mainly due to the fact that both anomalies

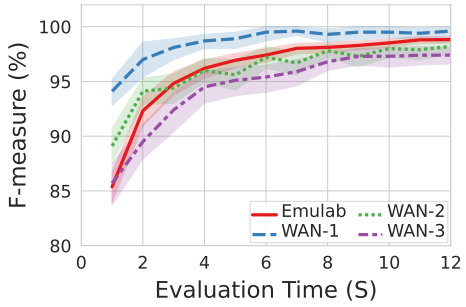


Fig. 11. Performance of Random Forest models in different evaluation time for transfer learning.

trigger a similar action (i.e., retransmitting the loss/corrupted packets) for Cubic, making it hard to find a good feature that can separate them. We validated this claim by evaluating the performance of transfer learning experiments after merging *loss* and *corrupt* anomalies into a single category, which led to the performance of transfer learning to increase over 95% in all networks. Moreover, as shown in Figure 7, BBR returns more distinct values for *retrans* feature for *loss* and *corrupt* anomalies, thus leading to higher transfer learning performance. Specifically, while the performance of transfer learning across the networks is around 80% when using Cubic and HTCP, it reaches to 85% for Emulab-WAN1 and 94% for WAN2-WAN3 experiments. Thus, BBR offers a better opportunity fit for detecting the six types of anomalies we targeted in the work.

Training Data Collection: We find that the models obtain near-optimal performance (over 95% F-score) when they are trained for each network exclusively. Hence, we explore the cost of gathering training data for each network. Our data collection involves running sample transfers for five different anomalies in three severity levels (e.g., 0.1%, 0.5%, 1% packet loss rates) in addition to normal transfer scenario. This requires conducting 16 transfers to gather one data sample for each category. Since we collected 200 samples for each category and ran each transfers for 20 seconds to ensure that they converge to a stable state before we capture the socket statistics, it took around 18 hours (16 transfer/category $\times$ 200 samples/category $\times$ 20 seconds/transfer) for each congestion control algorithm.

In Figure 10, we assess the impact of per class sample size on the performance of RF models. Although the original models used 160 samples to train a model (80% of all 200 samples we collected), we find that 30 samples per label is sufficient to achieve over 93% F-score and 50 samples per label can yield near-optimal performance in all networks. Thus, it is possible to train an RF model by gathering as low as 30 samples per class. We next investigate the impact of transfer duration on the model performance. In original data collection process, we ran each transfer for 20 seconds before capturing the socket statistics to guarantee that transfers are converged. However, Figure 11 shows that 6 seconds is sufficient for transfers to converge to a stable state, thus one can run each sample transfer as short as 6 seconds before capturing its statistics and terminating. Consequently, we claim that one can gather "sufficient" amount of training data for each network in 48 minutes (16 transfer/category $\times$ 30 samples/category $\times$ 6 seconds/transfer) to improve the performance of the ML models beyond the capabilities of transfer learning. To validate this claim, we re-evaluated the models using 30 samples from each category and capturing socket statistics for 6^th second. The resulting models achieved almost similar performance ($95-97\%$ F-score) in all networks compared to the original models.

VI. Conclusion

There is an increasing trend to leverage ML models to detect network anomalies as they can outperform the traditional approaches such as heuristic and statistical methods, however, existing solutions have two major limitations. First, they can only determine the presence of anomalies without providing any detail about underlying reason(s). Second, they cannot be transferred to different network settings as the models expect the test networks to have similar bandwidth and RTT values as in training networks. This paper proposes a novel feature transformation technique to eliminate the network dependence of anomaly diagnosis models such that they can be transferred to new networks. We believe that an ability to transfer ML models will pave the way for the wide adoption of ML models in production networks by removing the need for the training data collection. Experimental results gathered from both emulated and real-world networks indicate that the proposed models achieve over 95% accuracy when they are tested in network that they are trained for. Moreover, they yield more than 80% accuracy when transferred to new network settings. We further analyze the impact of the congestion control algorithms on the performance of transfer learning experiment and realize that BBR improves the accuracy of the prediction models due to resulting in more distinguishable outcomes for different anomaly types.

References

[1] "Perfsonar," https://www.perfsonar.net, 2021.

[2] Y. Zhang, S. Debroy, and P. Calyam, "Network-wide anomaly event detection and diagnosis with perfsonar," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 666–680, 2016.

[3] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[4] M. Mellia, A. Carpani, and R. L. Cigno, "Tstat: Tcp statistic and analysis tool," in *International Workshop on Quality of Service in Multiservice IP Networks*. Springer, 2003, pp. 145–157.

[5] N. S. Rao, M. Kiran, C. Wang, and A. Mandal, "Detecting outliers in network transfers with feature extraction," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), Tech. Rep., 2018.

[6] A. Giannakou, D. Gunter, and S. Peisert, "Flowzilla: A methodology for detecting data transfer anomalies in research networks," in *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, 2018, pp. 1–9.

[7] C. Dao, X. Liu, A. Sim, C. Tull, and K. Wu, "Modeling data transfers: Change point and anomaly detection," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1589–1594.

[8] M. Kiran, C. Wang, G. Papadimitriou, A. Mandal, and E. Deelman, "Detecting anomalous packets in network transfers: investigations using pca, autoencoder and isolation forest in tcp," *Machine Learning*, vol. 109, no. 5, 2020.

[9] G. Papadimitriou, C. Wang, K. Vahi, R. F. da Silva, A. Mandal, Z. Liu, R. Mayani, M. Rynge, M. Kiran, V. E. Lynch *et al.*, "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, pp. 387–400, 2021.

[10] H. Tu, G. Papadimitriou, M. Kiran, C. Wang, A. Mandal, E. Deelman, and T. Menzies, "Mining workflows for anomalous data transfers," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 1–12.

[11] S. Cooper, M. Bhuiyan, and E. Arslan, "Machine learning for data transfer anomaly detection," in *IEEE/ACM Supercomputing*, 2020.

[12] M. Arifuzzaman, S. Islam, and E. Arslan, "Towards generalizable network anomaly detection models," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 375–378.

[13] D. C. Verma, H. Zhang, and D. Ferrari, *Delay jitter control for real-time communication in a packet switching network*. International Computer Science Institute, 1991.

[14] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 362–375.

[15] "Network emulator," https://www.linux.org/docs/man8/tc-netem.html, 2021.

[16] "Emulab," https://www.emulab.net, 2021.

[17] "iPerf3," https://iperf.fr/, 2021.

[18] "Netstat," https://linux.die.net/man/8/netstat, 2021.

[19] ss-another utility to investigate sockets, "ss-another utility to investigate sockets," https://man7.org/linux/man-pages/man8/ss.8.html, 2021.

[20] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[21] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1937–1967, 2021.

[22] M. Long, J. Wang, G. Ding, S. J. Pan, and S. Y. Philip, "Adaptation regularization: A general framework for transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1076–1089, 2013.

[23] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu, "Transfer feature learning with joint distribution adaptation," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2200–2207.

[24] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[25] "H-tcp - congestion control for high delay-bandwidth product networks," https://www.hamilton.ie/net/htcp.htm, 2021.