Towards Generalizable Network Anomaly Detection Models

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

***Abstract*—Finding the root causes of network performance anomalies is critical to satisfy the quality of service requirements. In this paper, we introduce machine learning (ML) models to process TCP socket statistics to pinpoint underlying reasons of performance issues such as packet loss and jitter. More importantly, we introduce a novel feature engineering method to transform network-dependent metrics (e.g., total packet count and round trip time) in training datasets into network-independent forms to be able to transfer the models to new network settings without requiring to retrain them. Experimental results in various network settings show that the proposed feature engineering approach improves the performance of the models in previously unseen network settings from around 60% to nearly 90%. We believe ability to transfer ML models across networks will pave the way for wide adoption of ML solutions in production networks where collecting labeled data is not possible.**

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

I. Introduction

As the number of interconnected devices soars, networks are becoming increasingly complex. Maintaining such complex networks to ensure high quality of service has become a challenging task. Data-driven, automated solutions such as machine learning (ML) models can significantly simplify network management by shedding light into issues that existing solutions fall short to detect and diagnose. Unlike traditional statistical models, ML can extract complex relationships between a large number of features to make high-precision predictions. However, gathering input data to train highly-accurate models is a major impediment in the wide adoption of ML in production systems. While one can use isolated network testbeds to collect data and train ML models, these models cannot be directly used in different network settings as many input parameters are likely to be highly dependent on network settings such as bandwidth and delay. While simulated or emulated testbeds can be utilized to reproduce target network settings, any change in the target testbed, such as latency increase or bandwidth upgrade, will make pre-trained models obsolete.

In this paper, we attempt to derive ML models for network anomaly diagnosis problem that can be transferred to new networks. To do so, we first collect flow-level performance metrics at the end hosts and apply feature transformation to those that are network-dependent such as number of packets sent and average round trip time. Unlike standard data normalization (i.e., column-based normalization) which transforms input parameters into $[0-1]$ range by processing entire dataset, the proposed transformation normalizes each entry using other features of the same entry, i.e., row-based normalization. For example, packet loss is reported as the number of lost packets which is bandwidth-dependent as same packet loss ratio will correspond to higher packet loss count in networks with higher bandwidth. Thus, instead of using its absolute value, we divide it to total transmitted packets such that it will fall into the same range regardless of network bandwidth, thereby eliminating network dependency.

Experimental results show that parameter transformation significantly improves the performance of transfer learning for anomaly detection models. Specifically, when parameter transformation is not applied, anomaly diagnosis models that are trained using datasets from a specific network setting yield less than 60% accuracy when tested in different network settings even if standard data normalization is applied. With the help of novel parameter transformation, the accuracy reaches to more than 90% which is only slightly $(5 - 10\%)$ lower compared to the performance of ML models when they are tested in networks that they are trained for.

II. Related Work

Machine learning (ML) has been extensively used to detect network performance anomalies [1]–[5]. Most of the existing work aimed at detecting the presence of network anomalies without providing any information about underlying reasons. For example, Hou et al. performed online change-point detection analysis on time-series dataset to detect unexpected increase in round trip time (RTT) [5]. The authors of [4] projected perfsonar measurements data into new subspace by applying Principal Component Analysis (PCA) which clustered normal, correlated, and uncorrelated anomaly samples together to discern anomalous behavior from normal one. The authors of [3] applied both Boosted Decision Trees and Neural Network to detect anomalous packet loss behavior. 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 such as I/O interference, packet loss, packet corruption, and overloaded end hosts. [1]. Giannakou et al. used Random Forest Regression to process *tstat* logs and predict packet retransmission rate [2].

ML is also widely utilized to develop anomaly-based intrusion detection methods. The network packets are captured at the Network Interface Card level to be examined and filtered before being delivered to feature extraction model to compute flow attributes. The attributes then are assembled into feature vectors that provide input samples for the classifier's training, testing, and validation phases. The authors of [6] proposed an anomaly detection technique based on correlation information in traffic data samples. The idea is to build a separate covariance matrix for benign and malicious traffic using training data which then can be used in the production to classify flows. In almost all of previous work, authors build anomaly detection models that are network-specific, creating an impractical scenario for their deployment to new networks. In this paper, we are making an attempt to develop transferable anomaly diagnosis models to not only detect the presence of anomalies but also predict their root causes.

III. Data Collection

We use Emulab [7] to create networks with different bandwidth and delay settings and gather training data for ML models. We build a simple network topology in which a sender and a receiver is connected via switch. Nine network settings are created using combinations of three bandwidth (100Mbps, 1000Mbps, 5000Mbps) and delay (10ms, 30ms, 100ms) values. Throughout the paper, we refer to these network settings as $b{<}Bandwidth(Mbps){>}d{<}RTT(ms){>}$ format. For example, b5000d10 refers to the setup where network bandwidth is 5000Mbps and Round Trip Time (RTT) is set to 10ms.

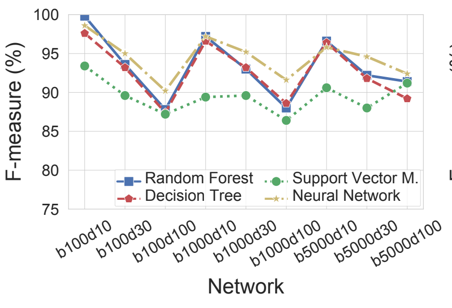
To reproduce network anomalies, we use Linux Traffic Control (`tc`) [8] utility that modifies kernel packet scheduler to create custom configurations. `tc` can create five types of network anomalies as jitter, packet duplication, packet reorder, packet corruption, and packet loss. We run 200 TCP transfers that are 30 second long for each anomaly type as well as normal condition using `iPerf3` [9]. A total of 97,200 transfers (9 network settings $\times$ 6 traffic conditions $\times$ 3 levels of anomaly $\times$ 200 repetitions) are conducted to collect a training dataset for ML models. Please note that we inject one anomaly type at any given time and leave the assessment of predictability of multi-class anomalies as a future work. As a result, the training dataset consists of six classes as normal, loss, duplicate, reorder, corrupt, and jitter. Thus, 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 congestion control algorithms is set to TCP Cubic.

There are several ways to capture TCP statistics for active and completed transfers. For example, Linux utility *netstat* captures a large number of TCP statistics that can be used to debug performance issues [10]. However, it reports one set of metrics for entire system, thus it is hard to confidently relate to the changes in performance metrics to individual flows. Another Linux utility *ss* provides per-flow TCP statistics for active TCP connections [11]. Finally, *tstat* is a network monitoring software that can be utilized to capture flow-level TCP statistics. Unlike *ss* which relies on Linux kernel TCP to capture performance metrics, *tstat* uses `tcpdump` to inspect active connections and reports a large number of metrics once the connections are terminated [12]. Since all of these tools report a different set of metrics (some of which may over lap), we captured TCP statistics using all three of them to compare their effectiveness in training accurate models that can help to pinpoint underlying reasons for TCP performance issues. We applied Random Forest-based feature engineering to find importance score of each metric and selected the subset of metrics whose sum of importance scores contribute to 95% of total importance score of all metrics to reduce feature space.

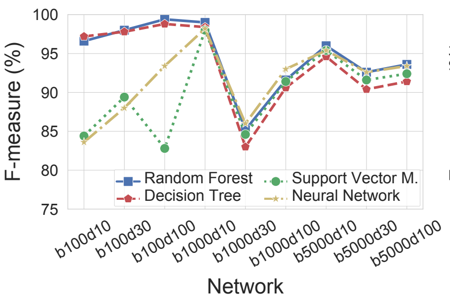
IV. Model Training

We applied several ML models to process training data and derive classification models. Out of all, we only report Random Forest (RF), Decision Tree (DT), Neural Network (NN), Support Vector Machine (SVM) model results. We leverage scikit-optimize library to optimize the hyperparameters of each model. 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, and finally kernel and gamma values for SVM. We also apply standard normalization on the dataset after splitting training and test dataset. 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.

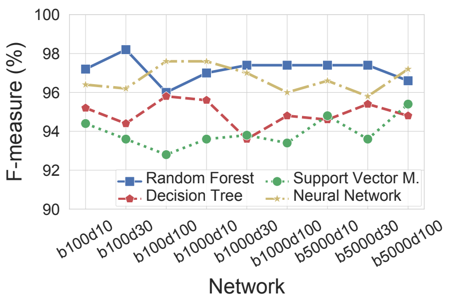
We first derive a separate model for each network setting after splitting the dataset as training and test with $80\% - 20\%$ split ratio. In other words, we train a separate model for each of nine networks created in Emulab with different bandwidth and RTT settings. The trained models are then evaluated based on their performance in the test dataset from same network. To evaluate the performance of ML models, we adopt F-score which combines 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) rates [13]. Figure 1 presents 5-fold cross validation results for different network bandwidth and RTT settings. We observe that SVM underperforms in almost all settings as its F-scores is $5 - 10\%$ lower than the F-score of the other models. On the other hand, NN, RF, and DT models perform similarly for all three datasets as they are all able to achieve over 90% F-score, on average. In particular, RF performs slightly better with 93%, 94%, and 97% average F-scores for *tstat*, *netstat*, and *ss* dataset, respectively. Hence, we only present RF model results in the rest of the analysis. It is also important to note that while the ML models experience performance fluctuations when *tstat* and *netstat* datasets are used for training (e.g., b100d100 in *tstat* and b1000d30 in *netstat*), their performance is more stable for *ss* dataset. This can be attributed to the fact that even if some performance metrics are reported by all three tools, there are few metrics that are only reported by one or two of them.



(a) *tstat*



(b) *netstat*



(c) *ss*

Fig. 1. Performance comparison of ML models in identifying the type of network anomalies

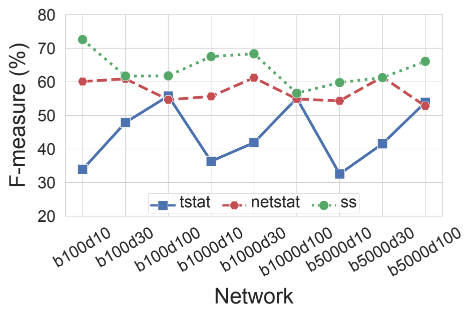


Fig. 2. Performance of Random Forest models for transfer learning.

TABLE I
SS METRICS USED TO TRAIN PREDICTION MODELS

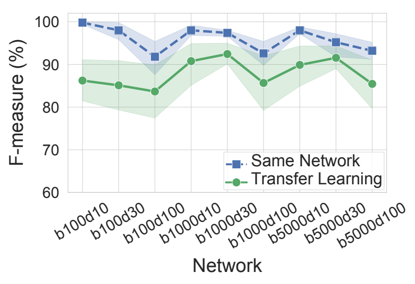
Feature Name	Base Metric	Importance Score
minrtt	rtt_avg	0.12
rtt_std	rtt_avg	0.13
unacked	cwnd	0.05
ssthresh	cwnd	0.08
dsack_dups	segs_out	0.17
retrans	segs_out	0.20
notsent	segs_out	0.08
reord_seen	segs_out	0.17

We next evaluate the performance of the ML models for transfer learning. In the context of this paper, we refer transfer learning as an ability to use an ML model that is trained in one network setting to detect anomalies in another network setting. For instance, it evaluates the performance of a model that is trained in a network with 500Mbps bandwidth and 10ms delay to detect anomalies in a network with 100Mbps bandwidth and 10$o$ms delay. Figure 2 illustrates the performance of transfer learning for all three datasets using an RF model. It is clear that despite yielding over 95% accuracy when training and test datasets come from the same network, performance of the model degrades severely when they are transferred to different network setting as F-score fluctuates between $30-70\%$. This is mainly due to the difference in the range of parameters in different network settings. For example, maximum RTT value of 30ms in a network with average RTT of 10ms denotes to the presence of jitter anomaly. On the other had, the same maximum RTT value of 30ms would be deemed normal when RTT of a network is around 30ms. Data normalization does not help to overcome this issue either because normalization will distort data range when training and test datasets have different bandwidth and delay settings. It is also not feasible assumption to normalize training and test datasets independently since it is unrealistic to assume that test dataset will contain all anomaly types and be available altogether. As a result, it is evident that transfer learning does not work for network anomaly detection when target network has different bandwidth and delay settings than the training network and raw values of performance metrics are used.

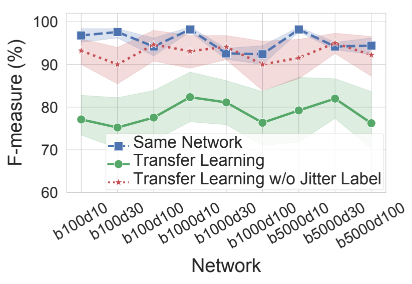
Parameter Transformation: To tackle this issue, we focus on parameter transformation to convert network-dependent metrics into network-independent forms. As an example, if we divide packet retransmission count in *ss* dataset by total packets sent, then it will return packet loss rate, which is independent of network bandwidth as it is guaranteed to be between 0 and 1 in all networks. Similarly, average RTT can be transformed to network agnostic form by dividing it to maximum RTT of the transfer. By extending this idea, we transformed each feature by dividing it to a related feature in the same transfer report such that bandwidth and RTT dependence can be removed. Table I lists the selected metrics for *ss* logs along with base metrics that are used to transform metrics. In a nutshell, we divide RTT related metrics to average or minimum RTT values and packet count metrics to total transmitted packets. Although not presented, we came up with similar transformation for *tstat* and *netstat* metrics as well.

V. EVALUATION

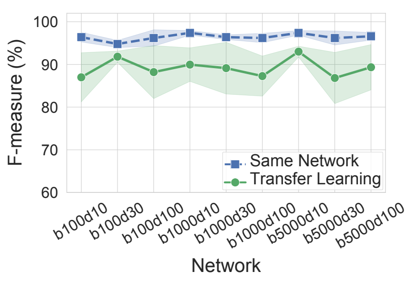
Figure 3 presents F-score of the RF models after applying feature transformation. *Same Network* label refers to the performance of the models when both training and test dataset come from the same network. *Transfer Learning* results present the average performance of the RF models when training dataset is from one network setting and test dataset is from other eight network settings. Compared to Figure 1 which reports around 98% F-score for the RF models when using raw *ss* performance metrics with standard normalization, the RF model trained with the transformed *ss* parameters still yields over 95% F-score when both training and test datasets have the same bandwidth and delay values (i.e, Same Network). On the other hand, while the performance of transfer learning



(a) *tstat*



(b) *netstat*



(c) *ss*

Fig. 3. Performance comparison of Random Forest model after applying feature transformation. The region around the lines indicate standard deviation for repeated experiments.

was between $30-70\%$ when input parameters are used in the raw format, feature transformation resulted in significant improvement with nearly 80% F-score for all three datasets. The average transfer learning performance is 86%, 79%, and 89% for *tstat*, *netstat*, and *ss* dataset, respectively.

We find that jitter anomaly is often misclassified when RF models are trained with *netstat* dataset. This is due to missing RTT-related metrics in *netstat* dataset. Based on importance scores as listed in Table I, it is clear that RTT metrics are important to make accurate classification decisions especially for jitter anomalies. Thus, the models trained with *netstat* logs perform worse in transfer learning. To validate this claim, we removed jitter anomaly samples from *netstat* logs re-evaluated the performance of transfer learning. Figure 3(b) shows that upon removing the jitter anomaly logs, the performance of transfer learning has improved significantly with 90% F-score.

In addition to missing RTT metrics, another major drawback of *netstat* is that it reports system-wide results. That is, it captures TCP statistics for all active TCP sockets and report cumulative values. Although it is useful to find out system-level issues such as buffer size limitations or kernel bugs, it cannot be used to debug performance problems of individual transfers. On the other hand, both *tstat* and *ss* report flow-level metrics, thus they are better fit to troubleshoot individual transfer issues. A key difference between *tstat* and *ss* is that *tstat* emits performance metrics after transfers are completed, which may be inconvenient since the presence of anomalies can be detected only after some transfers are fully exposed to them. *ss*, however, can provide real-time updates, thus is well-suited for timely detection of performance anomalies. As presented in Figure 3(c), *ss* dataset also helps the RF models to yield the highest F-score in transfer learning experiment.

VI. Conclusion

Machine learning based automated techniques are widely adopted to detect network anomalies quickly and accurately. However, the existing techniques in this area have two major limitations. First, they can only determine the presence of anomalies without providing any clue about underlying reasons. Second, they cannot be transferred to new networks as derived models rely on network-specific performance metrics such as bandwidth and round-trip time. This paper makes a first attempt to derive more efficient and transferable ML models to simplify performance troubleshooting as well as increase the adoption of ML models in production networks where collecting large scale training data may not feasible. Experimental results indicate that the proposed models achieve around 95% accuracy when finding the root cause of performance anomalies. Moreover, the models yield almost 90% accuracy when transferred to new network settings with the help of novel feature engineering method that transforms performance metrics into network-independent forms.

VII. Acknowledgments

This project is in part sponsored by the National Science Foundation (NSF) under award numbers 2007789 and 2019164.

References

[1] S. Cooper, M. Bhuiyan, and E. Arslan, "Machine learning for data transfer anomaly detection," in *IEEE/ACM Supercomputing*, 2020.
[2] A. Giannakou, D. Dwivedi, and S. Peisert, "A machine learning approach for packet loss prediction in science flows," *Future Generation Computer Systems*, vol. 102, pp. 190–197, 2020.
[3] 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.
[4] 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.
[5] B. Hou, C. Hou, T. Zhou, Z. Cai, and F. Liu, "Detection and characterization of network anomalies in large-scale rtt time series," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 793–806, 2021.
[6] M. Tavallaee, W. Lu, S. A. Iqbal, and A. A. Ghorbani, "A novel covariance matrix based approach for detecting network anomalies," in *6th Annual Communication Networks and Services Research Conference (cnsr 2008)*. IEEE, 2008, pp. 75–81.
[7] "Emulab," https://www.emulab.net/, 2021.
[8] "tc," https://man7.org/linux/man-pages/man8/tc.8.html, 2021.
[9] "iPerf3," https://iperf.fr/, 2021.
[10] "Netstat," https://linux.die.net/man/8/netstat, 2021.
[11] ss-another utility to investigate sockets, "ss-another utility to investigate sockets," https://man7.org/linux/man-pages/man8/ss.8.html, 2021.
[12] 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.
[13] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457309000259