Automatic Network Traffic Anomaly Detection and Analysis using Supervised Machine Learning Techniques

by

Astha Syal

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master

of

Computing and Information Systems

Program

YOUNGSTOWN STATE UNIVERSITY

December, 2019

Automatic Network Traffic Anomaly Detection and Analysis using Supervised Machine Learning Techniques

Astha Syal

Signature:

Astha Syal, Student Date

Approvals:

Alina Lazar, Thesis Advisor Date

Dr. Coskun Bayrak, Committee Member Date

Dr. Feng Yu, Committee Member Date

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

ABSTRACT

Today, internet has become an important tool for the entire public. It is the source of information, education, entertainment, and convenience. To maintain the efficiency and performance of the large computer networks supporting the internet, it is important to monitor and analyze the overall network traffic. During evening hours, when most people access internet at the same time for social media browsing, accessing their data or watching Netflix, with the increase in utilization, the network traffic can become congested and therefore the speed decreases. This research aims to identify network variables that cause these disturbances, thus impacting the overall speed of the network and leading it to a state of "congestive collapse". Machine learning models can be built using data passively collected in the network's logs and can be used in real-time to predict the traffic in the next time frame so network administrators could tune the network variables that are causing these disturbances. The models proposed here are able to quickly detect large intervals of low performing network transfers, which requires attention from network engineers.

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Alina Lazar. Under her guidance and support I was able to successfully perform and complete my thesis. Working with her, I have received exposure beyond the textbooks and have experienced an immense amount of growth in my learning.

I would like to thank my committee members Dr. Coskun Bayrak and Dr. Feng Yu for giving their time and attention in assessing the research work.

I want to thank Alex Sim and John Wu from the Scientific Data Management Research Group at Lawrence Berkeley National Laboratory and Jinoh Kim from the Department of Computer Science and Information Systems at Texas A&M University-Commerce for providing the datasets and guidance for this project.

A special thanks to my parents, family and friends for immense support and guidance throughout my program. I would also thank the Department of Computer Science and Information Systems and the College of Graduate Studies for the financial assistance during my graduate studies.

Table of Contents

List of Figures

List of Tables

1 Introduction

Network engineers have to spend significant amount of time and effort in order to optimize and tune the computer networks performance as a result of the increase of computer networks in size and complexity. Therefore, network traffic monitoring and analysis have become very important and are in general used to explain the current state of the network and to identify the most important variables. Even though network engineers have developed certain methodologies and tools for traffic monitoring and problem detection, to diagnose the network performance data collection is the first step.

For high performance scientific applications that generate lots of data, high network transfers are a requirement for moving the data from one site to another. Science DMZ includes several dedicated data transfer nodes and data movement tools. Large scientific facilities use science DMZ to achieve high performance. Healthy operations within the diverse and complex varieties of computer networks can be maintained using network traffic analysis [17]. To predict future network traffic volume and unexpected events in real-time, online traffic monitoring information which is collected overtime can be used.

For successful scientific experiments and computations that require large complex data files to be transferred across long distances, reliable file transfers are essential. Protocols such as TCP and UDP are used by network to support file transfers. Degradation of network performance under packet losses and duplication can adversely impact scientific applications. Therefore, to find and repair network problems that cause such degradation, network statistics are being monitored by scientists and engineers.

Application of machine learning techniques and models have been observed in traffic classification and prediction, intrusion detection, network adaptation and configuration extrapolation. These methods, when used with data from passive network measurements, are useful to explain the status of network traffic and identify bottlenecks. However, analyzing the network traffic data using machine learning and statistical methods [5, 25] is challenging for several reasons.

The automatically collected network data is usually high volume of high dimensional heavy stream data. Problems need to be identified in real-time so that it can provide alerts in case of unexpected events.

Another important challenge is that there are very few labeled datasets to build and evaluate supervised machine learning methods [1]. For network data, even though it is relatively easy to collect, there is no automatic way to label it based on the problem which needs to be solved. In order to construct the high quality ground truth datasets manually, especially in the case of analyzing real network traffic traces, the process comes with its own constraints such as time consumption, domain knowledge and privacy issues.

Machine learning models have high generalization characteristics, therefore, the models built on existing datasets are expected to be adaptive to the high-variance future network traffic values. Since the nature of network system is quite dynamic, it becomes unacceptable to have a model that could retrain for every time interval manifesting significant network changes.

Well-performing machine learning models are usually built on an assumption that the training data follows the same distribution as the target distribution. In the dynamically changing network environment, this assumption is not always true or practical. In a previous study Winstein and Balakrishnan [26] showed that the machine learning models built using training data collected from a specific network environment can, to some degree, perform well in other network environments. The methods and results presented in this thesis provide a first step towards identifying important features to detect any network problems.

Even though a good performance can be achieved using state of the art machine learning algorithms when trained on networking data, the accountability and interpretability properties of these models make its implementation quite challenging as most of the resulting models are still black boxes, not humanly readable or easily interpretable. As a result, expert network engineers fail to understand the behavior of the model and so are not able to integrate their knowledge in real systems to provide new ways for network adaptation and configuration extrapolation.

Network engineers examine the average network throughout in a certain time window to monitor the performance of the network. Many high throughput transfers signify that the network is running without problems, while a set of stable low throughput transfers is usually a sign of network congestion known as "congestive collapse". Network congestion occurs when a network link is transferring more data packets than it can handle and as a result the network as a whole exhibits reduced quality of service. This network state is usually due to interference among the server's shared physical resources involved in these transfers, such as network links, disk storage systems, and CPUs. Typical effects include packet loss, duplication, and retransmission. A typical consequence of congestion is imminent decrease in network throughput. The identifying factors [18] that contribute to the decrease of network throughput are very important in determining resource allocations to use in scheduling requests.

The goal of this thesis is to demonstrate that the states of congestive collapse can be identified by using supervised machine learning techniques to passive measurements of network flow datasets. Labels can be assigned based on the average throughput to categorize network traffic flows grouped by time intervals which can help with the analysis of large network datasets. After initial binary labeling of these flows to a small predefined number of clusters, a classification model of the traffic can be generated using the features of each time window. New incoming flows can then be classified on the fly and assigned to one of the two classes (low versus high throughput time window)

Specific contributions to the thesis are as follows:

· Label the time intervals as 'slow' or 'normal' by using statistical analysis methods to extract throughput threshold values.

· Provide a 2-dimensional representation of the datasets to understand the structure of the data using unsupervised dimensional reduction and visualization methods based on UMAP.

· Classification experiments performed on real network data collected from eight DTNs using `Tstat`.

· Checking the results of this supervised learning approach, especially the precision and recall values, using throughput plots for evaluation and comparison.

· Using feature extraction to rank the most important features contributing towards the results. Also, doing a comparative study of the classification results of those features to the results containing all the features.

This thesis is organized as follows. Section 2 compares our approach with previous work, while section 3 is an overview of the proposed methodology and describes its main steps. Section 4 provides an overview of the datasets, while sections 5 thoroughly discusses the experiments performed on eight large real traffic datasets. Finally, section 6 draws conclusions and presents future developments for this work.

2 Related Work

Network telemetry consists of a set of measurements describing the current state of network. Since the state of network keeps changing, it becomes challenging to build a real time and fine-grained network telemetry system. The traditional bottom-up approach that the operators use to extract useful information from the network is described by Yu [27]. This study talks about the drawbacks of the current approach. The main drawbacks is the lack of integrated data to provide network-wide view since the operators have too much of information to process. In addition the network monitoring tools are only able to capture aggregated or sampled information. The paper proposes top-down approach using declarative measurement abstractions in which the operators can specify the measurement queries in a declarative way, independent of the underlying measurement system. It also introduces a run-time system that translates high-level queries to low-level configurations, dynamically allocates resources to queries and manages hosts and routing changes.

One important task of traffic application classification is to identify the applications and protocols associated with the traffic flow [4] since it identifies the malicious or inappropriate application in a network which needs to be handled. The paper presents a methodology to classify unknown application traffic which has not yet been thoroughly researched on. The model identifies unknown applications, without restrictions on traffic type, by analyzing the incoming data which likely doesn't belong to the model. It utilizes K-nearest neighbor machine learning approaches to the application data which requires an initial set of explicitly labelled data and uses Kolmogorov-Smirnov statistic as the distance function. The model has achieved an overall accuracy of 92.67% with 91.98% on known traffic and 92.81% on unknown traffic.

DTNs at a scientific computing facility were statistically analyzed before by Liu et. al [17]. It contained TCP logs collected by `Tstat` [20] together with other logs . Transfers performed during the year 2017 were examined at three different levels: individual file transfers, user transfer requests and TCP flows. They identified the areas that needed improvements in transfer performance and resource utilization along with some insights on transfer, file and flow characteristics using these logs. Therefore, the study shows that we can obtain some useful insights by combining analysis of logs from different layers of the data movement stack. Also, some of the findings on flow, file and transfer characteristics are applicable to other large facilities.

There is still no good way for the network engineers to immediately differentiate between normal and anomalous network transfers as network data at the flow level collected with `Tstat` is unlabeled. Using perfSONAR, Rao et al. [23] collected network data in a controlled environment which was used to detect and identify network transfer anomalies such as packet loss, duplication, and retransmission sequencing that affect file transfer performance especially related to congestion. In order to highlight abnormal behavior and determine important features, an approach that used the combination of dimensionality reduction and statistical analysis was used. Simple experiments were performed using unsupervised feature extraction to show that to extract certain characteristics from known network transfer datasets, the proposed method is proven to efficient. The real network data extracted from `Tstat` logs was used for these experiments and specific file details such as file size, workflow stage and link details were ignored so that the core network properties can be extracted as general normal features. The patterns extracted using dimensionality reduction based on Principle Component Analysis (PCA) and clustering can help build feature filters to select data for any future machine learning methods. This method can be utilized by researchers and network engineers to build relationships among sensitive parameters such as congestion and availability with transfer file type. The end goal was to study the impact of packet loss and congestion on end-to-end performance

Another solution to this problem was proposed by Rezaei [24]. Provided enough amount of data for bandwidth and duration tasks, traffic class prediction tasks can be trained with only a small number of samples which would eliminate the need of large amount of labelled data. The model uses three time-series features: packet length, inter-arrival time and direction of the first k-packets for 1-Dimensional Convolutional neural network (CNN). Through experiments with two public datasets: QUIC and ISCX, the paper illustrates that the multi-task learning approach outperforms both single and transfer learning approaches.

Another approach by Dao [8] only looks at the throughput characteristic of the network transfers, instead of analyzing `Tstat` features. This approach employs a change point detection method first divides the network flows into time windows based on their time stamp and then applies a non-parametric model for each window which describes the congestion. Network transfers that take significantly longer than typical expected time were appointed as ”slow” or ”abnormal transfers”. When many ”slow” network transfers occur in the same time window, it is worthwhile to alert the network engineers and prompt them to investigate the abnormal behavior of the system.

Kim [13, 15, 12, 14] proposed another unsupervised approach, based on clustering which aims to keep track of the network state based on the aggregation of multidimensional variables. The state of the network with regard to the monitored variables was represented by the clustered result which can also be compared with the observed patterns from previous time windows that enables intuitive analysis. The type of data being analyzed to construct clustered patterns affects the definition of the network state. The proposed method was proven through two popular use cases, one that estimated the traffic breakdown and the other that identified the anomalous states. The applicability of their method was shown by applying it to `Tstat` data collected from ESnet.

TCP anomalies such as packet loss contributing to the changes in the network throughout have already been studied before [18]. Therefore, correctly identify all these the anomalies is very essential. Previous research [14, 3] have shown statistical correlation between multiple variables collected in the `Tstat` logs and the network traffic throughput. Recently, to predict network traffic volume from some flow statistics, such as flow counts per time interval, Hidden Markov Model and Recurrent Neural Networks have been proposed [7] . To compute network throughput is assumed to more challenging than computing these flow statistics.

In regard to accurately identifying TCP anomalies occurring during file transfers based on passive measurements of TCP traffic collected using `Tstat`, another unsupervised/supervised technique have been proposed in [16]. Real large datasets collected from several DTNs were used to validate this method. Through the preliminary results, the correlation between the percentage of TCP anomalies and the average throughput in any given time window can be identified.

To extract information from raw network flows data and to meet current network data analysis requirements including scalability, auto configuration capability, human readability of results, as well as evaluation of the model quality over time, the big data framework SeLINA, proposed by Apiletti et al. [3] was designed. At first the data was clustered using an automated tuning algorithm based on DBSCAN. Then, the clustering labels were used as input to rank the features using a decision tree algorithm .

For the quality measurement and analysis of the degree of change that takes place in the network for continued evaluation and comparison, SeLINA uses the average Silhouette index. When a drastic change is observed in this index, the main DBSCAN clustering model is automatically rebuild to incorporate the new incoming traffic data. The system's ability to identify over-time changes in the network is highlighted by the experiments conducted. The self-tuning property for the main algorithms, a step that normally requires sophisticated, expert level fine-tuning has been one the major contributions of this paper.

Flowzilla [11] is a methodology designed to detect data transfer anomalies over the network in order to prevent network congestion and overutilization of network devices. The training data from `Tstat`, which is adaptive to the changing network load, is used to build the Model. The Feature Extraction Filter, which uses Random Forest Regression (RFR) to extract a subset of features from `Tstat` database generates the training data. Adaptive Threshold Mechanism is then utilized to detect the anomalies. It includes a threshold calculator which is designed to detect the threshold values based on previous data and the Detector, that calculates the anomalous flows based on difference in model's predicted flow size and threshold value. This framework has achieved an overall accuracy of 92.5% .

We have built our model based on the Flowzilla approach [11] but with some added differences and improvisations. The approach we followed classified the network transfers based on their throughput rather than their size. Instead of performing classification on the individual transfers, we did it on time intervals. Finally, most significant, our approach utilized `Tstat` instead of the limited set of features used by Flowzilla.

3 Methods

Traffic flows collected in the `Tstat` logs have no feature or variable to designate them as anomalies. However, for this paper we consider labeling the network transfers using the average throughput per time window and an adaptive threshold set as the first quartile of the dataset. Then, supervised machine learning algorithms are used to predict which flows are slow and which are normal. Specifically, a supervised approach based on the gradient boosting approach-LightGBM was found suitable to build models that automatically classifies the traffic flow time windows into two separate groups with similar characteristics in terms of their throughput. All the steps of this approach are highlighted in Fig. 1.

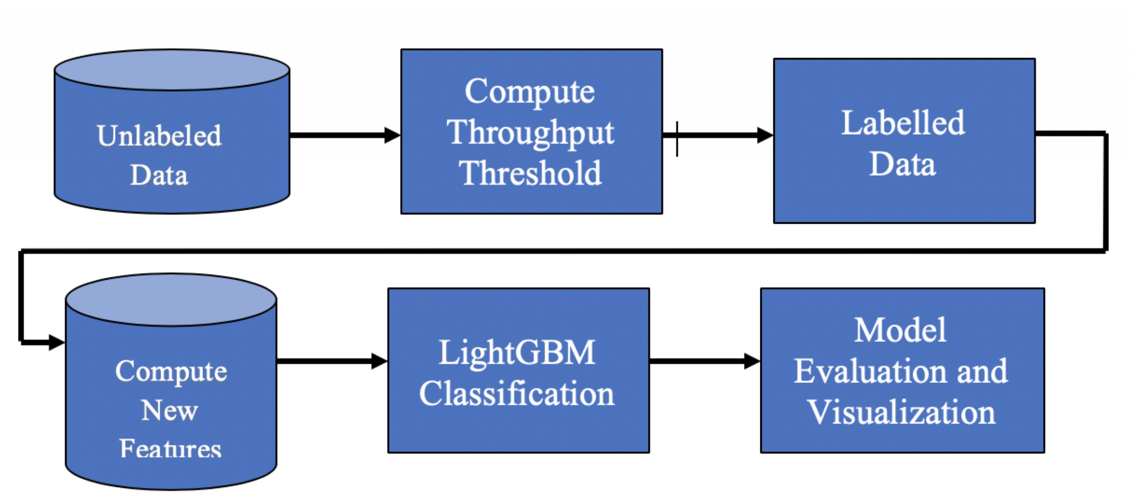

Figure 1: Proposed machine learning guided methodology for identifying and categorizing low performing network flows.

3.1 LightGBM

Gradient Tree Boosting is a technique which has been proven to give state-of-the-art results on many stantard classification benchmarks. The proposed method employs LightGBM algorithm to identify time intervals with low averages of the throughput values over the entire dataset. LightGBM (Light Gradient Boosting Machine) model uses a histogram based learning algorithms[22]. On supplying the training dataset to the model, it uses Gradient-based One Side Sampling which performs random sampling on instance with small gradients and keeps all the instances with large gradients, focusing mainly on under-trained instances without causing much impact on data distribution. In order to reduce the number of features, it uses Exclusive Feature Bundling (EFB) which sorts the features in descending order based on their degrees in the graph constructed with weighted edges. Each feature in the ordered lists are then checked and assigned to an existing bundle or a new bundle is created. This algorithm can avoid unnecessary computation of zero feature values. This approach provides higher efficiency, has faster training speed, uses less memory and provides better accuracy as compared to other classification models. We also compared the results of this model to other models such as Linear SVM, Random Forest and XGBoost.

3.2 XGBoost

Another Gradient Boosting Algorithm is XGBoost which uses parallel tree boosting. It introduces weighted quantile sketch for tree learning and sparsity-aware algorithm for sparse data[6]. Since network data is quite sparse and unpredictable in nature, this algorithm has proven to be quite efficient with the network datasets.

It enables parallel and distributed computing which has resulted in faster learning process. XGBoost uses a new regularization technique to overcome the limitations of overfitting which makes it faster and more robust during model tuning and differentiates it from other gradient boosting.

The main difference between XGBoost and LightGBM can be illustrated through the way their decision trees are formed. XGBoost uses Level-wise tree growth which checks all the previous leaves for each new leaf[2] while LightGBM uses Leaf-wise tree growth, mentioned in above section as histogram implementation, which offers several advantages over XGBoost in terms of accuracy, training speed and large scale data handling. This difference could also be reflected through the Table 4 in our Experiments and Results section.

3.3 Linear SVM

SVMs are supervised learning models that usually provide high performance for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two classes, an SVM training algorithm builds a linear hyperplane with the specific property of having the largest margin. In other words, the individual data points are mapped so that the points of the two categories are divided by the optimal hyperplane that has the largest gap. The optimization step of linear SVMs can be solved efficiently using the coordinate descent algorithm, thereby reducing the convergence iteration to linear time, making this algorithm run very fast compared to other classification methods.

3.4 Random Forest

Random Forest is the most flexible and easy to use classification algorithm. On providing a training data to the model, it randomly selects the subset for it and creates a set of decision trees for them. Each tree predicts the class prediction and the class with maximum votes becomes the predicted class. It handles missing values and doesn't suffer overfitting, which makes the model robust and accurate.

3.5 Feature Importance - SHAP

Most of the machine learning models we use today are black boxes, therefore, interpreting the model output holds great importance in evaluating the performance of the model. Shap (SHapely additive exPlanations) values are capable of explaining the output of any machine learning models using a high speed algorithm especially design for tree ensemble methods. The model supports implementations for XGBoost, LightGBM, CatBoost, and scikit-learn tree models making it a good fit for our experiments. We supplied the training sets through SHAP which assigns each feature an importance value for the prediction. Through this, we identify top ten features which creates most impact on the output of our model. We then run the same set of experiments for these selected features and compare the results with the classification evaluation results containing all the features. This helps us to identify which features are involved towards predicting a specific class outcome and how much impact they have on the results.

3.6 Dimensionality Reduction

Uniform Manifold Approximation and Projection (UMAP) [19] is a new dimension reduction technique that can be used for visualizations similar to other manifold data embedding techniques, and also for general non-linear dimension reduction. It is based on manifold theory and fuzzy topological data analysis. The algorithm builds a weighted k-neighbor graph to efficiently approximate the k-nearest-neighbor computation and calculates spectral embeddings that are later optimized using the stochastic gradient descent algorithm. The algorithm is founded on assumptions that the data is uniformly distributed on a Riemannian manifold, an assumption that does not always hold for real data.

To understand the underlying structure of this particular dataset, a two-dimensional representation in an embedding space for the node 5 dataset is presented in Figure 2. Similar to principal component analysis (PCA) representations, the values of the x-axis and y-axis of the UMAP scatterplot are nothing more than a representation in the embedding two-dimensional space. Part (a) of this figure shows all the individual flow with their calculated throughput. Red represents flows with low throughput (throughput lower than the first quartile), blue means normal (throughput between the first quartile and the third quartile), while green shows the high performing flows (throughput larger than the third quartile). Figure 2 (b) shows the same network transfers, colored using the same encoding; however this time the flows are assigned with the majority class of the one-hour time windows in which they belong.

Given the time series property of the network transfers datasets, it does not make sense to split them randomly into training and testing or perform standard cross-validation shuffling, but instead a ”time series cross-validator” is more appropriate. The procedure used to split time series is described next. One moment in time can to be chosen as the delimiter between the training and testing sets. Also, to perform cross-validation at each split, test indices must be higher than the indices used for testing before, and the entire training set needs to have timestamps from before the test set. Unlike standard cross-validation methods, successive training sets are supersets of those that come before them, but they can also be limited to a certain size. Cross-validation is the preferred validation method for larger datasets because it better estimates the generalization ability of the model, which is very important for the problem we are trying to solve.

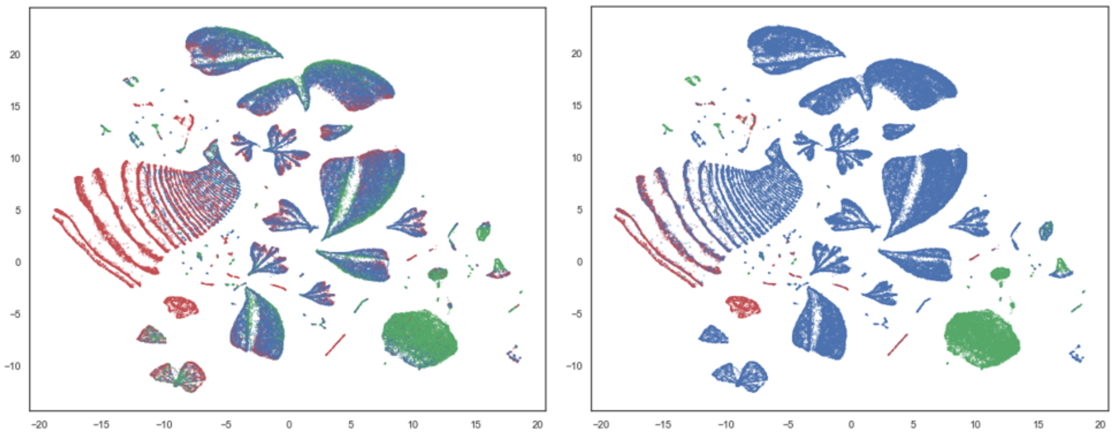

Figure 2: UMAP 2-dimensional visual representation of the network traffic flows collected from node 5 and colored based on their throughput values. Red means low, blue means normal and green means high. (a) individual flows (b) majority labels assigned based on one hour time intervals.

3.7 Performance Evaluation

Quantitative classification evaluation, which evaluates the goodness of classification results, can be done using the traditional measures such as accuracy, precision, recall, and F1 score. Precision is the ratio between the correctly classified positive instance and number of all positive instances. It gives an idea of the amount of elements from the positive class that were misclassified. Recall is the ratio between the correctly classified positive instance and the number of all instances classified as positive. The F1 score is an average between precision and recall. Precision, recall, and the F1 score are very important, especially when the training and/or testing datasets are highly imbalanced.

By showing the correctly classified time intervals versus the incorrectly classified on the time series average throughput plots, a qualitative classification evaluation is possible.

4 Datasets

Today, network traffic statistics at the flow level can be collected using passive monitoring tools such as `Tstat` [10]. A passive probe located on the access link that connects each Data Transfer Node (DTN) located at the National Energy Research Scientific Computing Center (NERSC) to the ESnet (Department of Energy's dedicated science network) inspects all packets flowing on the link and extracts the information to be summarized. The `Tstat` software rebuilds each TCP and UDP network flow by matching incoming and outgoing segments.

Table 1: Datasets Statisctics

Node	# of Flows	60min	30min	5min
1	2,447,602	4,232	8,450	47,310
2	1,119,470	2,639	4,372	13,618
3	5,131,592	4,029	7,845	36,089
4	455,244	3,286	5,716	21,006
5	135,531	421	659	2,140
6	166,116	598	1,060	4,359
7	157,247	412	659	2,439
8	169,233	626	1,096	4,539

`Tstat` offers output statistics at packet and flow level. The flow-level analysis provides a summary of the connection properties that is logged [21] for further analysis. It can be used to collect many different statistics for TCP, UDP, and RTP/RTCP traffic. For TCP connections, congestion window size, out-of-sequence segments, duplicated segments, number of bytes and segments retransmitted, and RTT are some of the statistics that it collects. `Tstat` distinguishes between completed and not completed flows, and between clients (hosts that actively open a connection) and servers (hosts that passively listen for connection requests). `Tstat` also records UDP messages. However, since UDP communication contributes a very low percent of the total bytes moved from/to the major computer center, we did not included UDP communications in this study.

Among the measurements collected by `Tstat`, some of the metrics are believed to be correlated to both system configuration and possible performance issues. For example, the measure of Round Trip Time (RTT) is usually related to both the distance from the server, but also possible to reveal congestion on the path. Similarly, both reordering and duplicate probabilities increase during periods of congestion. The duration and amount of carried data are used to compute the actual throughput and could also distinguish between the type of service the flow carries, e.g., short-lived signaling flows carrying little data rather than long lived data flows carrying a large amount of data. We included all these `Tstat` measurements in our experiments.

At the large scientific facility 90K of TCP flows are collected per node daily and an approximate total of 10GB of compressed data logs are collected yearly on ten DTNs. The `Tstat` data used for this study was collected and provided by the NERSC computing facility at LBNL. The `Tstat` data contains source and destination IP addresses, and so is not publicly available for privacy reasons. To simplify our analysis, for this study we eliminated all flows that carry less than 10MB of data both ways. Table 1 shows the number of network transfer in each dataset and also the number of time intervals. The datasets for nodes 1 to 4 contain six months worth of transfers collected between 01/01/2017 - 06/28/2017, while the datasets for nodes 5 to 8 are smaller and have approximately one month worth of data collected between 06/01/2017 - 06/28/2017.

For each dataset all the features with constant values are eliminated. We also eliminate features involved in the calculation of the throughput because it is used to assign the output labels. Table 2 shows the summary statistics for the calculated throughput for first four nodes and 3 shows the summary statistics for the last four nodes which are the smaller nodes. The first quartile values in this table are used as threshold values for our experiments. Network transfers that take place closer in time are highly correlated compared with transfers that are far apart. To account for this important characteristic of the datasets we add two additional features which where assigned based on the previous and two time intervals preceding the current time window.

All the features are then normalized using the MinMax Scaling procedure. The datasets are divided into time intervals based on three time frequencies: 5, 30 and 60 minutes. Averages for all the features including throughput are calculated and saved for further input into the classification algorithm.

Table 2: Summary Statistics (Node 1 - Node 4)

	node1	node2	node3	node4
count	2,447,602	1,119,470	5,131,592	455,244
mean	129.46	98.836	881.61	174.46
std	317.78	115.29	978.5	375.75
min	0.00003	0.00009	0.00005	0.00007
5%	13.941	20.175	43.448	7.66
25%	31.636	56.643	174.85	42.672
50%	57.901	82.353	377.81	87.981
75%	111.15	103.64	1521.1	162.7
max	9853.994	9883.041	9889.773	9725.279

Table 3: Summary Statistics (Node 5 - Node 8)

	node5	node6	node7	node8
count	135,531	166,116	157,247	169,233
mean	105.23	262.35	91.439	268.05
std	69.401	335.53	81.206	631.1
min	0.0008	0.0007	0.005779	0.008
5%	22.825	22.329	21.242	26.714
25%	57.376	64.642	47.877	60.551
50%	92.339	107.12	77.219	93.842
75%	135.83	198.24	117.02	149.5
max	3003.8	5920	3048	3271.2

5 Experiments and Results

We ran a set of experiments for Node 1 through Node 8 using the following classification methods: Linear SVM, Random Forest, XGBoost and LightGBM. Through the average results in Table 4, we can observe that accuracy is higher than 85% for all the methods and recall does not fall below 63% in any of the result cases. The complete evaluation results for all nodes can be observed in Table 5, Table 8, Table 9 and Table 10. Since we observed the best results from LightGBM for our model, we ran all our experiments using LightGBM methodology. However, LightGBM is one of the slowest methods.

Table 4: Average evaluation of Node 1 through Node 8 for all classification methods

Method	Runtime(secs)	Accuracy	Precision	Recall	F1-Score	%CI
Linear SVM	0.197	0.9024	0.8522	0.7097	0.7690	29.05
Random Forest	0.260	0.8577	0.8899	0.6395	0.7113	28.60
XGBoost	0.831	0.8692	0.8799	0.6768	0.7389	28.18
LightGBM	2.902	0.8962	0.8998	0.7497	0.7999	29.00

5.1 Classification Evaluation

To detect time intervals of slow network transfer we adopt a supervised classification method based on LightGBM. Let $X = \{(x_1, y_1), ...(x_N, y_N)\}$ be the labeled anomaly detection dataset with N total instances, where x_i represents the input feature vector that can be defined in a d dimensional space as $x_i = \{x_i^1, x_i^2, ...x_i^d\}$. This set of feature values (client/server IP Address, client/server protocol, RTT values, maximum segment size) is extracted from the raw `Tstat` data. The corresponding binary class label $y_i \in \{y_1, y_2, ...y_N\}$ for each input vector x_i represent normal speed time intervals of abnormally slow intervals. This class label is assigned based on the adaptive threshold defined using the first quartile of the average throughput of the training dataset. Therefore, the training dataset will always contain 25% of slow time intervals transfers. In the end, the classification models are designed to predict whether the average throughput for the network transfers flows in a given time window that is below the throughput threshold for that node.

In the detection or testing phase, based on a classifier trained with set X, every instance in a test set is assigned to the class of either normal or slow type of transfer. It is important to note that because of the dynamic nature of the network, the network traffic data would change with time; thus, the adaptive threshold needs to be periodically updated and the classification model for anomaly detection must be learned with new training data, for the purpose of keeping high accuracy for online detection.

We build several models, a different one for each combination of node dataset and time window. The experiments are done for 5, 30 and 60 minutes time intervals. For the first set of experiments, we used the entire datasets for all the eight nodes and divided each of them based on the time stamp in training and testing sets. The last week is used for testing and the rest to build the model. Results of these experiments are presented in Table 5 in terms of accuracy, precision, recall and F1 score and percent of time windows under the threshold.

Table 5: Classification Evaluation: Light GBM

	min	Runtime	Acc.	Prec.	Recall	F1	%CI
node1	5Min	23.000s	0.9567	0.9407	0.8981	0.9189	27.34
	30Min	8.700s	0.9144	0.8423	0.7276	0.7808	20.94
	1H	6.600s	0.8979	0.8247	0.6107	0.7018	19.66
node2	5Min	8.100s	0.9638	0.9526	0.9059	0.9286	25.97
	30Min	5.400s	0.9376	0.9198	0.8427	0.8796	27.03
	1H	5.100s	0.9174	0.8803	0.8013	0.8389	26.85
node3	5Min	20.000s	0.8954	0.7560	0.9018	0.8225	26.86
	30Min	7.700s	0.9389	0.8248	0.8794	0.8512	19.87
	1H	6.600s	0.9354	0.8516	0.8684	0.8599	22.82
node4	5Min	9.300s	0.9699	0.9333	0.8512	0.8804	14.37
	30Min	5.900s	0.9544	0.9618	0.7550	0.8459	16.58
	1H	4.600s	0.9564	0.9359	0.7604	0.8391	14.95
node5	5Min	2.400s	0.9062	0.9825	0.8528	0.9130	57.77
	30Min	1.100s	0.4857	0.7000	0.1111	0.1918	57.79
	1H	0.510s	0.6911	0.8537	0.6481	0.6139	54.47
node6	5Min	4.000s	0.9349	0.9524	0.6557	0.7767	17.26
	30Min	1.700s	0.9115	0.8776	0.6719	0.7611	20.98
	1H	0.930s	0.9588	0.9118	0.8857	0.8986	20.58
node7	5Min	3.300s	0.9587	0.9967	0.9061	0.9492	42.63
	30Min	1.000s	0.8824	0.9740	0.7732	0.8621	47.54
	1H	1.200s	0.8211	1.0000	0.6207	0.7660	47.15
node8	5Min	4.400s	0.9524	0.9567	0.7921	0.8625	18.86
	30Min	2.200s	0.9197	0.8750	0.7000	0.7778	20.06
	1H	1.000s	0.9277	0.8750	0.7000	0.7778	18.07

The results presented in Table 5 shows overall accuracy and precision more than 82% for all the nodes except for Node 5, 30mins time window, which also shows a lower recall, less than 60% as compared to the recall for all other nodes. We also observe best results for Node 7 and Node 2 as represented in 8 and Figure 4. On observing the results from these two figures, it clearly states that our model correctly identifies contiguous intervals of low network performance. The higher number of false positives as reflected by the recall rate is especially during times when time intervals with high throughput alternate with time intervals of low throughput.

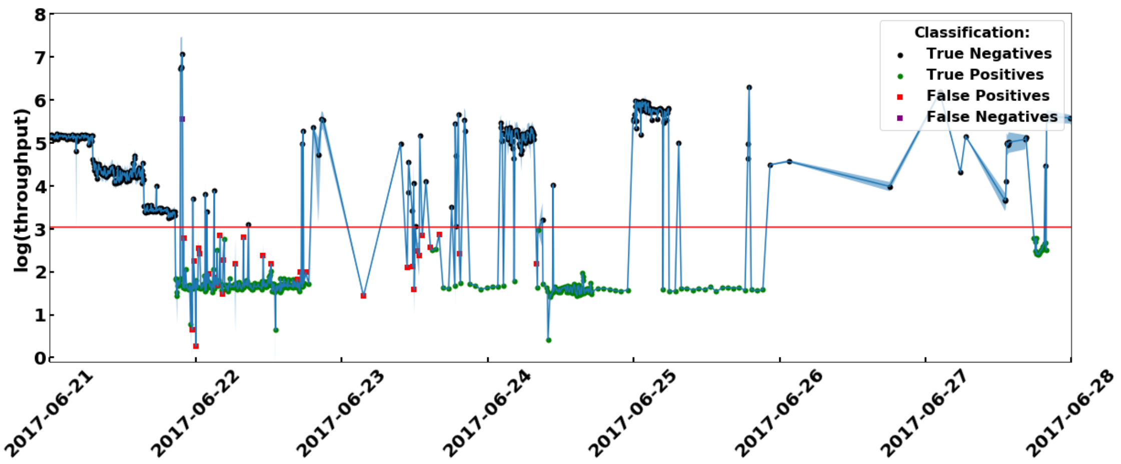

Figure 3: Average throughput for node 7, 5min time windows. The red line is throughput threshold. Recall is 90%

We also take a look at the results for Node 5 (Figure 5) which has the lowest accuracy and recall. We also observe that Node 5 carries least number of total transfers among all nodes which makes it harder to classify and has a very high ratio of transfers with high throughput. Also, there are six false negatives, all of them are very close to the average throughput threshold.

For datasets containing larger number of flows, flows collected over a period

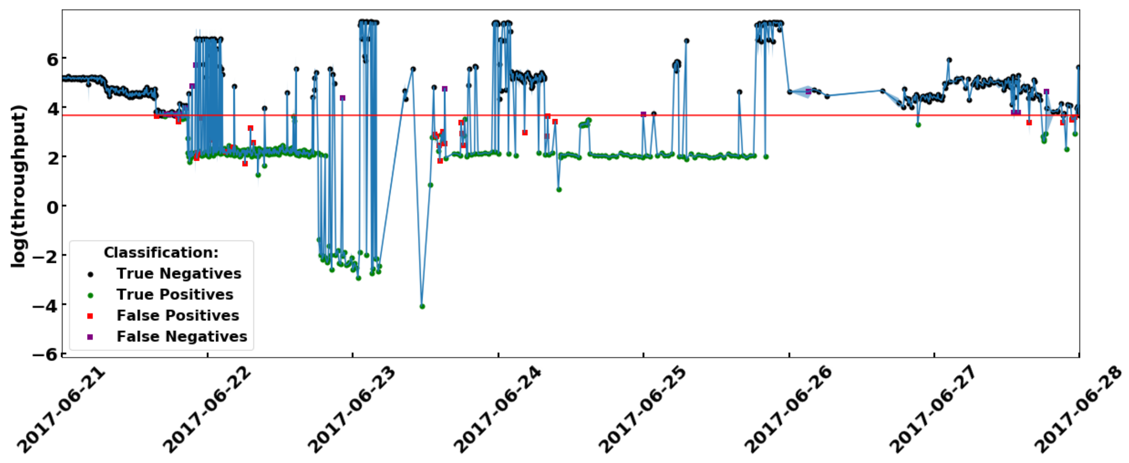

Figure 4: Average throughput for node 2, 5min time windows. The red line is throughput threshold. Recall is 90%

of 6 months, we ran the second set of experiments for Node 1 through Node 4. For these datasets we run cross validation using a Time Series Split with k=6 sets. The results are presented in Table 6. We obtain an overall accuracy greater than 84%, the lowest precision is recorded at 79% and the recall does not fall below 81%. We also observe that the best results are obtained for Node 3 at the time interval of 30 minutes (Figure 6), and the worst results are obtained for Node 2 at 1 hour time interval (Figure 7). Even in case of lowest recall, we can observe that we can still identify the network state with continuous low throughput values, indicating that the model is able to identify the state of low throughput very accurately.

As shown in Table 4 the number of time intervals in our experiments vary from 412 to 47,310. It is well known [9] that LightGBM is capable of training datasets with over 500,000 instances in seconds. It takes less than 23 seconds to train the dataset for node 1 with 5 minutes time intervals which is the largest dataset used in our experiments. Data preprocessing and training can be done offline and so it doesn't

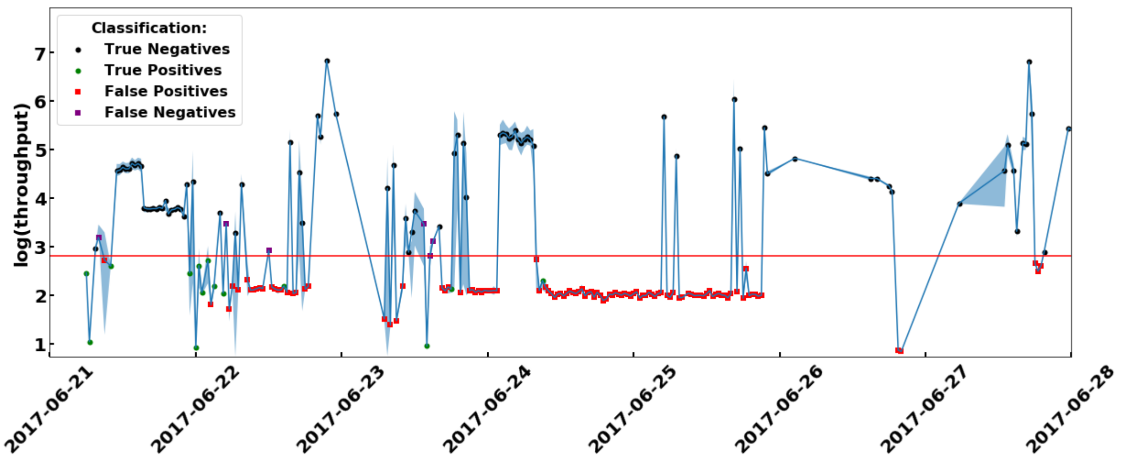

Figure 5: Average throughput for node 5, 30min time windows. The red line is throughput threshold. Recall is 11%

slow down the detection. Testing works fast, making this method suitable for quick detection.

For cross-validation using a Time Series Split with $K = 6$ sets in Table 11, Table 12 and Table 13, we observe accuracy greater than 85% and the recall doesn't fall below 75%. The results from all our classification evaluation models doesn't hold much variations in the output. Therefore, we can conclude that our model works very efficiently in classifying the low throughput network transfers in a given time window.

5.2 Feature Selection and Evaluation

On obtaining overall best results from Node 7, we ran SHAP over the Node 7 flows (Figure 8) to analyse which features contributes towards the prediction of a particular class. We only looked over the top ten features holding most impact over the prediction.

We then ran the LightGBM classification on these features in the training set

Table 6: Cross Validation Evaluation: Light GBM

		Accuracy	Precision	Recall	F1 Score
node1	5Min	0.9348 ±0.0124	0.9223 ±0.0101	0.8880 ±0.0300	0.9004 ±0.0222
	30Min	0.9134 ±0.0127	0.8862 ±0.0231	0.8388 ±0.0429	0.8550 ±0.0375
	1H	0.8954 ±0.0175	0.8594 ±0.0319	0.8162 ±0.0416	0.8316 ±0.0380
node2	5Min	0.9473 ±0.0082	0.9289 ±0.0128	0.8851 ±0.0214	0.9044 ±0.0169
	30Min	0.8988 ±0.0260	0.8515 ±0.0332	0.8502 ±0.0268	0.8496 ±0.0302
	1H	0.8462 ±0.0397	0.7967 ±0.0384	0.8148 ±0.0292	0.7957 ±0.0415
node3	5Min	0.9393 ±0.0101	0.9158 ±0.0203	0.8705 ±0.0485	0.8756 ±0.0364
	30Min	0.9400 ±0.0104	0.9172 ±0.0106	0.9006 ±0.0240	0.9049 ±0.0137
	1H	0.9212 ±0.0136	0.8939 ±0.0219	0.8556 ±0.0513	0.8595 ±0.0461
node4	5Min	0.9346 ±0.0163	0.9251 ±0.0135	0.8820 ±0.0245	0.8994 ±0.0204
	30Min	0.9285 ±0.0077	0.9187 ±0.0130	0.8552 ±0.0301	0.8773 ±0.0243
	1H	0.9240 ±0.0097	0.9002 ±0.0199	0.8405 ±0.0353	0.8614 ±0.0305

rather than using all the features. On looking at the results from Table 7, we can observe that, even though we only ran the features holding most impact on the model output, we do not see much difference in the resulting accuracy and precision values. We can observe a significant change in recall values since the features contributing towards the low throughput class have more impact now on the dataset. With the results presented in the Table 7, we can conclude that the top features contributing towards the predicted class have a dominant effect on overall results. Therefore, even if we include all the features in the training data, the results are not much impacted. This makes our model very accurate and consistent in classifying the low throughput network transfers in a given time window.

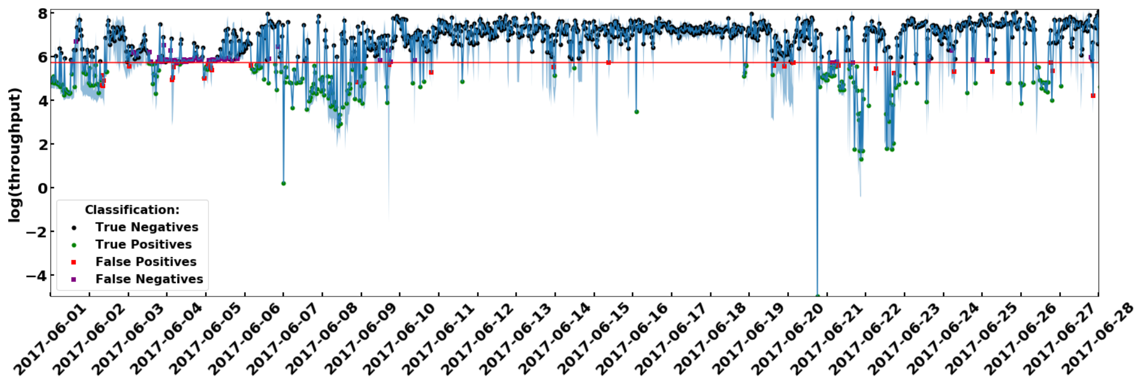

Figure 6: Average throughput for node 3, 30min time windows. The red line is throughput threshold. Recall is 90%

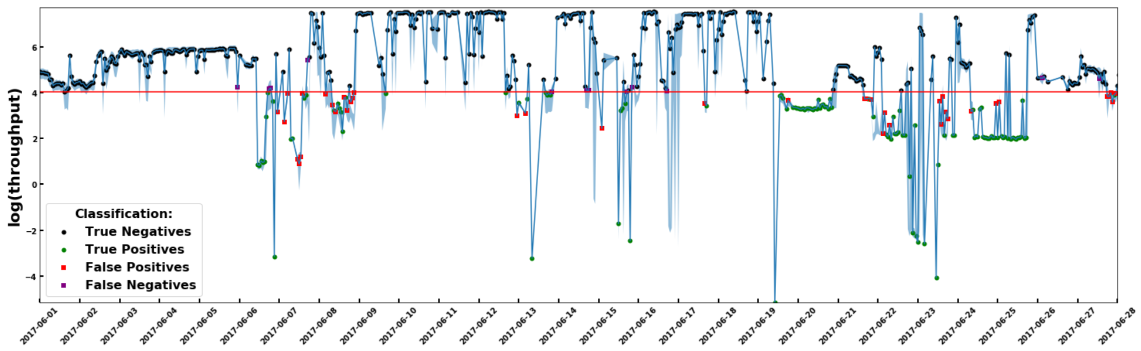

Figure 7: Average throughput node 2, 1H time windows. The red line is throughput threshold. Recall is 81%

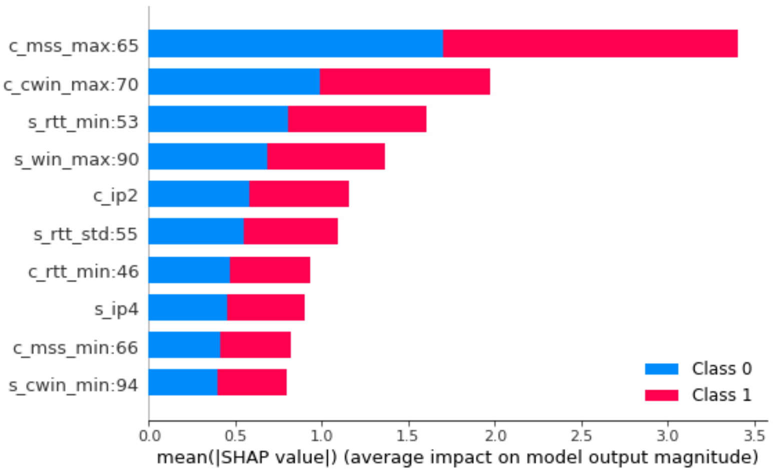

Figure 8: Feature selection using SHAP for Node 7

Table 7: Classification Evaluation Comparison for Node 7

	min	Runtime	Acc.	Prec.	Recall	F1	%CI
All Features	5Min	3.300s	0.9587	0.9967	0.9061	0.9492	42.63
	30Min	1.000s	0.8824	0.9740	0.7732	0.8621	47.54
	1H	1.200s	0.8211	1.0000	0.6207	0.7660	47.15
Selected Features	5Min	1.500s	0.9457	0.9932	0.8788	0.9325	42.63
	30Min	0.570s	0.8529	0.9855	0.7010	0.8193	47.54
	1H	0.330s	0.8455	0.9149	0.7414	0.8190	47.15

6 Conclusion

At large scientific facilities where petabytes are transferred daily, reliable network transfers are needed for successful operations. LightGBM classification is proposed to classify the traffic flows captured by `Tstat` so that the possible problems such as low throughput can be identified. Our system splits the `Tstat` log streams into chunks so as to make predictions in near real-time. The classification model does not need to be rebuilt from ground up rather it only needs to be updated in real-time. Our results show that this new method can be utilized to accurately detect abnormally low throughput time intervals.

This thesis presents a supervised data analytics system that effectively mines network traffic data. The proposed methodology is based on a two-phase approach 1) binary classification labels are assigned to the network transfers using adaptive throughput threshold ; and 2) new data labels are predicted using the classification model built in real-time to identify the network flows with low throughput.

LightGBM classification algorithm designed to handle large datasets is featured using this methodology. It can easily handle one year's worth of network traffic data. Network traffic data can be analyzed by exploiting the the features of this general purpose approach under different network conditions. This approach has been tested using datasets from eight out of ten data transfer nodes at the major computer center.

Even though the proposed method has shown its ability to accurately identify large windows of low throughput, certain problems have been detected in case of isolated or alternating intervals. Therefore, to address this problem, the current system can be extended to (i)include more time related features, (ii) evaluate the pre-processing feature selection techniques that could eliminate the highly co-related features, (iii) enhance the design and integrate different analysis techniques which could be more appropriate for outlier detection

In future, we plan to find better ways to label the data or the 'slow' time intervals and also to investigate the generalization capabilities of the presented method; larger datasets will be used for training and testing. The model highly relies on selecting the right throughput threshold. In order to attain more efficient model performance, a series of experiments could be conducted by selecting other ways of computing adaptive throughput threshold and its impact over the model could be evaluated.

7 References

[1] Sebastian Abt and Harald Baier. 2014. Are we missing labels? A study of the availability of ground-truth in network security research. In *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, 40–55.

[2] Essam Al Daoud. [n. d.]. Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset. *pdfs.semanticscholar.org* ([n. d.]).

[3] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Danilo Giordano, Marco Mellia, and Luca Venturini. 2016. SeLINA: a self-learning insightful network analyzer. *IEEE Transactions on Network and Service Management* 13, 3 (2016), 696–710.

[4] Ryan Baker, Ren Quinn, Jeff Phillips, and Jacobus Van der Merwe. 2018. Toward Classifying Unknown Application Traffic. In *DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) Workshop*.

[5] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. 2018. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications* 9, 1 (June 2018), 16.

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining (KDD '16). ACM, New York, NY, USA, 785–794.

[7] Zhitang Chen, Jiayao Wen, and Yanhui Geng. 2016. Predicting future traffic using hidden markov models. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 1–6.

[8] Cecilia Dao, Xinyu Liu, Alex Sim, Craig Tull, and Kesheng Wu. 2018. Modeling data transfers: change point and anomaly detection. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1589–1594.

[9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* 9, Aug (2008), 1871–1874.

[10] Alessandro Finamore, Marco Mellia, Michela Meo, Maurizio M Munafo, Politecnico Di Torino, and Dario Rossi. 2011. Experiences of internet traffic monitoring with tstat. *IEEE Network* 25, 3 (2011), 8–14.

[11] Anna Giannakou, Daniel Gunter, and Sean Peisert. 2018. Flowzilla: A Methodology for Detecting Data Transfer Anomalies in Research Networks. In *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. IEEE, 1–9.

[12] Jinoh Kim and Alex Sim. 2017. A New Approach to Online, Multivariate Network Traffic Analysis. In *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 1–6.

[13] Jinoh Kim, Alex Sim, Sang C Suh, and Ikkyun Kim. 2017. An approach to online network monitoring using clustered patterns. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*. IEEE, 656–661.

[14] Jinoh Kim, Alex Sim, Brian Tierney, Sang Suh, and Ikkyun Kim. 2018. Multivariate network traffic analysis using clustered patterns. *Computing* (2018), 1–23.

[15] Jinoh Kim, Wucherl Yoo, Alex Sim, Sang C Suh, and Ikkyun Kim. 2017. A lightweight network anomaly detection technique. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*. IEEE, 896–900.

[16] Alina Lazar, Kesheng Wu, and Alex Sim. 2018. Predicting Network Traffic Using TCP Anomalies. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 5369–5371.

[17] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Yuanlai Liu. 2018. A comprehensive study of wide area data movement at a scientific computing facility. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1604–1611.

[18] Zhengyang Liu, Malathi Veeraraghavan, Jianhui Zhou, Jason Hick, and Yee-Ting Li. 2013. On causes of GridFTP transfer throughput variance. In *Proceedings of the Third International Workshop on Network-Aware Data Management*. ACM, 5.

[19] Leland McInnes and John Healy. 2018. Umap: Uniform manifold approxima-

tion and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[20] Marco Mellia, Michela Meo, Luca Muscariello, and Dario Rossi. 2008. Passive analysis of TCP anomalies. *Computer Networks* 52, 14 (2008), 2663–2676.

[21] Marco Mellia, Michela Meo, Luca Muscariello, and Dario Rossi. 2008. Passive analysis of TCP anomalies. *Computer Networks* 52, 14 (2008), 2663–2676.

[22] E A Minastireanu and G Mesnita. 2019. Light GBM Machine Learning Algorithm to Online Click Fraud Detection. *J. Inform. Assur. Cybersecur* (2019).

[23] Nageswara S Rao, Mariam Kiran, Cong Wang, and Anirban Mandal. 2018. *Detecting Outliers in Network Transfers with Feature Extraction.* Technical Report. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).

[24] Shahbaz Rezaei and Xin Liu. 2019. Multitask Learning for Network Traffic Classification. (June 2019). arXiv:cs.LG/1906.05248

[25] M Wang, Y Cui, X Wang, S Xiao, and J Jiang. 2018. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Netw.* 32, 2 (March 2018), 92–99.

[26] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13).* ACM, New York, NY, USA, 123–134.

[27] Minlan Yu. 2019. Network Telemetry: Towards a Top-down Approach. *SIGCOMM Comput. Commun. Rev.* 49, 1 (Feb. 2019), 11–17.

8 Appendix

Table 8: Classification Evaluation: Linear SVM

	min	Runtime	Acc.	Prec.	Recall	F1	%CI
node1	5Min	2.500s	0.9013	0.8632	0.7597	0.8081	27.34
	30Min	0.240s	0.8574	0.6878	0.5842	0.6318	20.94
	1H	0.130s	0.8664	0.6875	0.5878	0.6337	19.66
node2	5Min	0.360s	0.9312	0.9401	0.7853	0.8558	25.97
	30Min	0.100s	0.9055	0.9189	0.7133	0.8031	27.03
	1H	0.071s	0.8985	0.9008	0.6987	0.7870	26.85
node3	5Min	1.400s	0.9181	0.8765	0.8090	0.8414	26.86
	30Min	0.180s	0.9288	0.8261	0.8132	0.8196	19.87
	1H	0.110s	0.9024	0.8480	0.6974	0.7653	22.82
node4	5Min	1.400s	0.9396	0.9058	0.6468	0.7547	14.37
	30Min	0.160s	0.9121	0.8917	0.5350	0.6687	16.58
	1H	0.081s	0.9221	0.9107	0.5312	0.6711	14.95
node5	5Min	0.022s	0.9413	0.9758	0.9213	0.9478	57.77
	30Min	0.015s	0.8349	0.9891	0.7222	0.8349	57.79
	1H	0.008s	0.8862	0.8732	0.9254	0.8986	54.47
node6	5Min	0.063s	0.9311	0.8274	0.7596	0.7920	17.26
	30Min	0.017s	0.9279	0.8281	0.8281	0.8281	20.98
	1H	0.012s	0.9176	0.8621	0.7143	0.7813	20.58
node7	5Min	0.037s	0.9703	0.9782	0.9515	0.9647	42.63
	30Min	0.014s	0.9167	0.9545	0.8660	0.9081	47.54
	1H	0.008s	0.8780	0.9216	0.8103	0.8624	47.15
node8	5Min	0.100s	0.8824	0.8878	0.4307	0.5800	18.86
	30Min	0.020s	0.8562	0.7742	0.4000	0.5275	20.06
	1H	0.014s	0.8916	0.8000	0.5333	0.6400	18.07

Table 9: Classification Evaluation: Random Forest

	min	Runtime	Acc.	Prec.	Recall	F1	%CI
node1	5Min	8.600s	0.9507	0.9538	0.8616	0.9054	27.34
	30Min	1.500s	0.8941	0.8317	0.6201	0.7105	20.94
	1H	0.130s	0.8829	0.7978	0.5420	0.6455	19.66
node2	5Min	1.400s	0.9394	0.9536	0.8095	0.8735	25.97
	30Min	0.430s	0.8658	0.8789	0.5839	0.7017	27.03
	1H	0.280s	0.8830	0.8793	0.6538	0.7500	26.85
node3	5Min	4.600s	0.8811	0.7191	0.9145	0.8051	26.86
	30Min	0.910s	0.9149	0.7426	0.8755	0.8036	19.87
	1H	0.490s	0.9399	0.8684	0.8684	0.8684	22.82
node4	5Min	1.800s	0.9500	0.9737	0.6701	0.7939	14.37
	30Min	0.540s	0.9196	0.9187	0.5650	0.6997	16.58
	1H	0.330s	0.9361	0.9661	0.5938	0.7355	14.95
node5	5Min	0.120s	0.4370	0.7500	0.0381	0.0725	57.77
	30Min	0.072s	0.4495	0.7500	0.0714	0.1304	57.79
	1H	0.065s	0.5610	1.000	0.1940	0.3250	54.47
node6	5Min	0.200s	0.9538	0.9241	0.7978	0.8563	17.26
	30Min	0.082s	0.9541	0.9310	0.8438	0.8852	20.98
	1H	0.068s	0.9294	0.8710	0.7714	0.8182	20.58
node7	5Min	0.130s	0.9457	1.000	0.8727	0.9320	42.63
	30Min	0.070s	0.8137	1.000	0.6082	0.7564	47.54
	1H	0.065s	0.8211	1.000	0.6207	0.7660	47.15
node8	5Min	0.210s	0.9729	0.9482	0.9059	0.9266	18.86
	30Min	0.086s	0.9097	0.9231	0.6000	0.7273	20.06
	1H	0.075s	0.8795	0.7778	0.4667	0.5833	18.07

Table 10: Classification Evaluation : XGboost

	min	Runtime	Acc.	Prec.	Recall	F1	%CI
node1	5Min	14.000s	0.9514	0.9400	0.8785	0.9082	27.3469
	30Min	2.900s	0.9054	0.8312	0.6882	0.7529	20.94
	1H	1.500s	0.8934	0.8571	0.5496	0.6698	19.66
node2	5Min	3.200s	0.9537	0.9594	0.8578	0.9058	25.97
	30Min	1.200s	0.9367	0.9195	0.8392	0.8775	27.03
	1H	0.740s	0.9174	0.9091	0.7692	0.8333	26.85
node3	5Min	10.000s	0.8925	0.7527	0.8933	0.8170	26.86
	30Min	2.500s	0.9288	0.7895	0.8755	0.8303	19.87
	1H	1.300s	0.9354	0.8759	0.8355	0.8552	22.82
node4	5Min	5.300s	0.9550	0.9275	0.7451	0.8264	14.37
	30Min	1.600s	0.9461	0.9470	0.7150	0.8148	16.58
	1H	1.000s	0.9533	0.9231	0.7500	0.8276	14.95
node5	5Min	0.450s	0.4370	0.9167	0.0279	0.0542	57.77
	30Min	0.160s	0.4220	0.5000	0.0159	0.0308	57.79
	1H	0.120s	0.6829	0.9118	0.4627	0.6139	54.47
node6	5Min	1.000s	0.9321	0.9512	0.6393	0.7647	17.26
	30Min	0.270s	0.9180	0.8824	0.7031	0.7826	20.98
	1H	0.170s	0.8824	0.77781	0.6000	0.6774	20.58
node7	5Min	0.550	0.9470	0.9966	0.8788	0.9340	42.63
	30Min	0.180s	0.8676	1.0000	0.7216	0.8383	47.54
	1H	0.110s	0.8374	1.0000	0.6552	0.7917	47.15
node8	5Min	1.100s	0.9599	0.9543	0.8267	0.8859	18.86
	30Min	0.280s	0.9097	0.8367	0.6833	0.7523	20.06
	1H	0.180s	0.8976	0.7600	0.6333	0.6909	18.07

Table 11: Cross Validation Evaluation: Linear SVM

		Accuracy	Precision	Recall	F1 Score
node1	5Min	0.8920 ±0.0076	0.8587 ±0.0109	0.8132 ±0.0373	0.8272 ±0.0301
	30Min	0.8884 ±0.0127	0.8381 ±0.0339	0.7924 ±0.0519	0.8045 ±0.0501
	1H	0.8753 ±0.0118	0.8181 ±0.0427	0.7638 ±0.0504	0.7794 ±0.0502
node2	5Min	0.9157 ±0.0105	0.8666 ±0.0346	0.8237 ±0.0363	0.8369 ±0.0332
	30Min	0.8758 ±0.0195	0.8175 ±0.0359	0.8099 ±0.0270	0.8068 ±0.0327
	1H	0.8643 ±0.0132	0.7972 ±0.0246	0.8052 ±0.0234	0.7987 ±0.0229
node3	5Min	0.9128 ±0.0144	0.8702 ±0.0260	0.8244 ±0.0413	0.8375 ±0.0324
	30Min	0.9176 ±0.0150	0.8858 ±0.0146	0.8702 ±0.0225	0.8727 ±0.0148
	1H	0.8980 ±0.0139	0.8638 ±0.0099	0.8349 ±0.0444	0.8341 ±0.0348
node4	5Min	0.8785 ±0.0259	0.8474 ±0.0270	0.7887 ±0.0206	0.8106 ±0.0216
	30Min	0.8676 ±0.0245	0.8354 ±0.0235	0.7716 ±0.0244	0.7899 ±0.0231
	1H	0.8614 ±0.0257	0.8221 ±0.0282	0.7586 ±0.0286	0.7719 ±0.0274

Table 12: Cross Validation Evaluation: Random Forest

		Accuracy	Precision	Recall	F1 Score
node1	5Min	0.9313 ±0.0122	0.9262 ±0.9262	0.8747 ±0.0305	0.8931 ±0.0231
	30Min	0.9075 ±0.0124	0.8816 ±0.0229	0.8220 ±0.0441	0.8417 ±0.0385
	1H	0.9062 ±0.0127	0.8811 ±0.0286	0.8162 ±0.0444	0.8378 ±0.0404
node2	5Min	0.9349 ±0.0108	0.9368 ±0.0078	0.8289 ±0.0413	0.8631 ±0.0322
	30Min	0.8825 ±0.0234	0.8325 ±0.0328	0.8126 ±0.0288	0.8177 ±0.0296
	1H	0.8519 ±0.0232	0.7975 ±0.0339	0.7849 ±0.0216	0.7849 ±0.0284
node3	5Min	0.9298 ±0.01244	0.9000 ±0.0196	0.8541 ±0.0456	0.8625 ±0.0344
	30Min	0.9400 ±0.0103	0.9188 ±0.0089	0.8939 ±0.0265	0.9021 ±0.0165
	1H	0.9136 ±0.0126	0.8870 ±0.0221	0.8316 ±0.0469	0.8455 ±0.0431
node4	5Min	0.9125 ±0.0187	0.9153 ±0.0199	0.8273 ±0.0247	0.8589 ±0.0227
	30Min	0.9128 ±0.0056	0.9152 ±0.0048	0.8127 ±0.0256	0.8468 ±0.0193
	1H	0.9090 ±0.0098	0.9147 ±0.0158	0.8047 ±0.0291	0.8356 ±0.0219

Table 13: Cross Validation Evaluation: XGBoost

		Accuracy	Precision	Recall	F1 Score
node1	5Min	0.9282 ±0.0144	0.9150 ±0.0120	0.8808 ±0.0288	0.8930 ±0.0212
	30Min	0.9073 ±0.0159	0.8764 ±0.0239	0.8375 ±0.0434	0.8497 ±0.0379
	1H	0.8954 ±0.0179	0.8621 ±0.0307	0.8160 ±0.0413	0.8319 ±0.0373
node2	5Min	0.9404 ±0.0103	0.9252 ±0.0133	0.8643 ±0.0272	0.8887 ±0.0201
	30Min	0.9105 ±0.0166	0.8654 ±0.0229	0.8582 ±0.0213	0.8613 ±0.0218
	1H	0.8833 ±0.0153	0.8276 ±0.0248	0.8314 ±0.0214	0.8275 ±0.0227
node3	5Min	0.9333 ±0.0119	0.8998 ±0.0174	0.8605 ±0.0463	0.8674 ±0.0353
	30Min	0.9385 ±0.0101	0.9158 ±0.0111	0.8968 ±0.0276	0.9008 ±0.0159
	1H	0.9159 ±0.0152	0.8791 ±0.0278	0.8453 ±0.0526	0.8491 ±0.0493
node4	5Min	0.9222 ±0.0168	0.9170 ±0.0124	0.8555 ±0.0250	0.8786 ±0.0202
	30Min	0.9259 ±0.0073	0.9199 ±0.0135	0.8413 ±0.0333	0.8678 ±0.0285
	1H	0.9158 ±0.0121	0.8889 ±0.0220	0.8270 ±0.0372	0.8482 ±0.0338