Detecting Anomaly in Large-scale Network using Mobile Crowdsourcing

Yang Li, Jiachen Sun, Wenguang Huang, Xiaohua Tian,
Shanghai Jiao Tong University, Shanghai, China
Email: {sky660099, sjc0806, blueskygundam, xtian}@sjtu.edu.cn

***Abstract*—In this paper, we propose a tree modeling-based data mining method to detect anomalies from crowdsourced network data. We design an algorithm to extract potential network anomalies from decision trees. Moreover, we propose a criteria to evaluate the severity of anomaly in terms of three factors: standard deviation, weight sum and impurity decrease. To enhance generalization performance, we randomly generate sample subspace of the original dataset as the input for each subtree and compact detected anomalies from all subtrees. We carry out experiments based on the crowdsourced network measurement dataset containing five million samples, which contains round trip time (RTT) from more than 5,000 users. Experiments show that the proposed method can effectively detect high-latency network anomalies. Moreover, the random forest-based approach can achieve an improvement of approximately 25% of generalization performance compared to the single decision tree approach.**

***Index Terms*—Network Anomaly Detection, Crowdsourcing, Decision Tree, Random Forest**

I. Introduction

Today, OTT business is growing rapidly due to the extensive coverage of the mobile Internet. In OTT, users can access a variety of services provided by Internet companies, such as video streaming and text transmission, through traditional networks. Unlike traditional communication services, OTT only utilizes the carrier's underlying network support, while business services are provided by Internet companies. For example, based on the network broadband service provided by the telecom operator, WeChat can provide video chat service to users.

In order for the OTT service to be available globally, the coverage of the underlying network channel provided by the operator must be large enough to cover as many users as possible. Therefore, an integrated network of multiple ISPs is used as the backbone network of the OTT service. Unfortunately, the instability of the backbone network, such as frequent network failures, has had a serious negative impact on the quality of service (QoS), causing huge losses to Internet companies.

The fundamental reason behind the decline in OTT service quality is the lack of effective operation and maintenance methods to cope with the network failures under the large scale network architecture. In the near future, network scale and network complexity will both increase dramatically, such as the upcoming Internet of Things (IOT) era. However, traditional network operation and maintenance methods are costly in supporting sustainable operations. Therefore, how to theoretically reconstruct the operation and maintenance mode of the OTT network scenario is an urgent problem to be solved. One step further, the most important thing of the operation and maintenance system is how to effectively deal with the network anomaly of the OTT large-scale network, so that the entire network can be operated in a targeted manner.

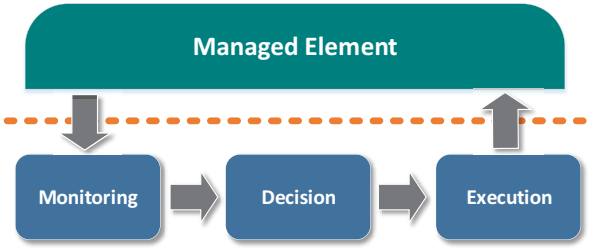


Fig. 1. The control loop of network operation and maintenance

Figure 1 illustrates the control loop for network operation and maintenance. The entire process is divided into three phases: monitoring, decision making, and execution. In the OTT scenario, the information service quality of the network will directly determine the user experience, thus determining the core competitiveness of the Internet company. Therefore, in the design of large-scale network operation and maintenance systems, it is necessary to introduce a new management architecture based on intelligent decision-making. This first puts forward new requirements for network measurement methods in the monitoring stage, namely how to obtain network measurement information in large-scale networks efficiently at a low cost.

Currently there are two main types of traditional network measurement methods.: (1) Active measurement. The packet probe is sent to the target network location to collect measurements of related network performance metrics, such as RTT and packet loss rate. The disadvantage is that only the network performance between the probe and the target being measured can be obtained, and the accuracy and effectiveness of the measurement depends on how to select a suitable observation path. In addition, the cost of deployment of active measurements is high and it is difficult to deploy on a large scale.; (2) Passive measurement. It captures traffic information with the help of the specific devices deployed on network links. The disadvantage is that it is difficult to measure the end-

to-end performance experienced by users. Apart from that, it requires customized hardware support so that the deployment cost is also high.

While the traditional network measurement methods can not meet the requirements of OTT, some work adopts the crowdsourcing strategy of collecting network measurement data from a large number of users. For example, MopEye [1] can provide large-scale network performance data based on end-to-end crowdsourcing, and then use traditional statistical methods to perform simple mathematical analysis. CniCloud [2] collect cellular network data by crowdsourcing from various devices and provides some simple analysis tools. [3]–[5] extract the relation between network performance and other features through mathematical analysis. In addition, the DYSWIS system [6] chooses collecting experts' suggestions to build a rule system for anomaly detection and troubleshooting. Espinet deployes measurement probes on each link of the network, thus crowdsourcing traffic information to rule out network failures [7]. Bischof obtaines the relevant characteristics of ISP by crowdsourcing information from the network-intensive application running on the terminal system [8].

However, for such large-scale crowdsourced datasets containing a large amount of noise, traditional mathematical methods lack robustness and cannot effectively utilize the correlation of various feature dimensions of the data. In general, we are facing the following challenges:

1) The dataset contains multiple anomalies which are caused by various types of factors with different dimensions.
2) Depending on the degree of performance degradation and the extent of the impact, each anomaly's severity is different.
3) Crowdsourced data is noisy and inaccurate due to the influence of devices and environment while collected discontinuously in time.

In order to cope with the challenges brought by the crowdsourced network measurement data, we propose a data mining method to detect high-latency network anomalies. We utilize decision trees model to extract all possible anomalies from the dataset. Concretely, we obtain anomaly information by analyzing the tree structure which is generated from crowdsourced data. Moreover, we propose a criteria to evaluate each anomaly's severity thus we can select out important anomalies which have a larger influence on the network. Finally, with the help of random forest, we optimize our analysis model to be more robust and general.

To the best of our knowledge, we are the first to investigate anomaly detection based on crowdsourcing and decision trees. The main contributions of this paper are as follows:

1) We propose a data mining method to extract the cause of network anomaly based on decision trees' structure. While existing works mainly use decision trees for predictions, decision trees‘ interpretability can help ue further exploiting the hidden information of network anomaly. Results show that we can efficiently extract all potential cause of network anomaly from crowdsourcing dataset.
2) We propose "confidence", a criteria to evaluate the anomaly severity objectively according to the degree of performance degradation and the scale of impact. We choose three factors to quantify the anomaly severity: standard deviation, weight sum and impurity decrease. Experiments in Section 5 also proves the accuracy of the criteria.
3) We propose a robust forest-based data mining algorithm by integrating our single tree-based approach with random forest. We utilize bootstrapping sampling to generate random sample subspace of the original dataset as the input for each subtree. Then we compact all their mining outputs to get final results. By applying the idea of random forest, the variance of the model can be reduced and the generalizaion can be enhanced.

The remainder of this paper is organized as follows: Section 2 introduces the preliminaries of our work and the basic idea of our modeling approach. In Section 3, we present the detailed design of the anomaly mining, including data exploration and tree modeling. Section 4 unveils the design of evaluation criteria. Experiments are given in Section 5. We conclude our work and give a glimpse of the future work in Section 6.

II. Overview

A. Dataset

MopEye [1] is an Android-based application that automatically collects RTT information, which is accurate to the granularity of the user and the application being used.

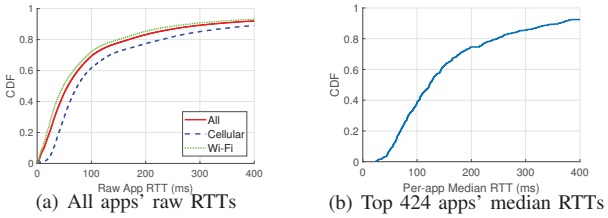
Therefore, compared to other network measurement tools such as Mobiperf [9], MopEye measurement does not generate network overhead, and the measured RTT can accurately reflect the network latency experienced by each application. In addition, MopEye can run on Android smartphones without root privileges, which greatly reduces the barriers to crowdsourcing. MopEye uploads the RTT and other information such as the user's location and signal strength to the server.

The collection of MopEye crowdsourcing network measurement datasets lasted from May 2016 to January 2017, resulting in more than five million RTT measurements.So far, MopEye has attracted 4,014 user installations from 126 countries. The dataset includes 5,252,758 RTT measurements from 6,266 Android apps from 2,325 smartphones. A cumulative distribution function diagram of the application's raw RTT values and RTT median values is shown in Figure 2 [1]. From the left image, we can see that about 70% of the original RTT values are distributed below 100ms. The figure on the right shows the distribution of the median RTT of 424 applications with more than 1000 measurements.

B. System Architecture

The system architecture is composed by two stages, as shown in Figure 3:

Rule Mining: We first focus on cleaning and processing the raw data to make it more explainable and better fitted in



(a) All apps' raw RTTs (b) Top 424 apps' median RTTs

Fig. 2. CDF plots of apps' raw RTTs and median RTTs

TABLE I
NEW FEATURES AFTER FEATURE ENGINEERING

Features	Dimensions	Type
AppName	4000+	Categorical
Carrier	2000+	Categorical
Signal	5	Ordinal
Speed	5	Ordinal
SynHour	24	Categorical
Longitude	36	Ordinal
Latitude	18	Ordinal
DestLongitude	36	Ordinal
DestLatitude	18	Ordinal
NetType	2	Categorical

the modeling stage. After that, we build decision trees model with CART algorithm [10] on the basis of processed dataset. According to the information provided by tree structure, we can extract rules in a comprehensive way, where rule indicates the potential cause leading to network anomalies.

Rule Evaluation: After tree modeling, we obtain all candidate rules. We design "confidence", a criteria to evaluate each rule's severity according to its degree of performance degradation and the range of impact. The criteria is composed by three factors: standard deviation, weight sum and impurity decrease. Moreover, to achieve a better generalization performance, we utilize random forest to generate a bunch of subtrees. We apply the criteria to the forest and compact their outputs to get the final results.

III. RULE MINING

In this section we focus on the mining of rules from the dataset. To achieve a better performance, we first preprocess the raw dataset from two aspects: feature engineering, instance clustering. Then we describe the detail of the decision tree modeling and the procedure of rules mining.

A. Data Exploration

1) Feature Engineering: The presence of outliers and missing values in the dataset can cause model bias and degrade

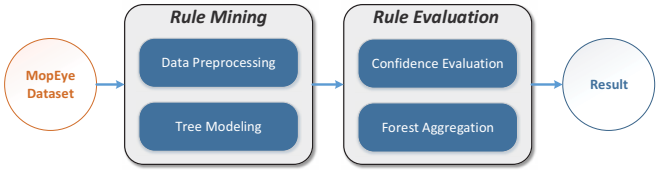


Fig. 3. System Architecture

TABLE II
NOTATION LIST

Notation	Implication
x	The node or the rule
$N(x)$	Number of samples in x
S_x	Sample subspace of x
K	Number of classes
P_x^k	Class proportion of label k in x
y_i	Label of i-th sample
w_i	Weight of i-th sample
$W(x)$	Weight sum of x

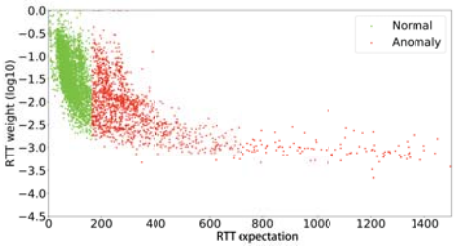


Fig. 4. The weight of instances decreases as the expectation increases. (The green dots represent normal instances while the red dots represent anomaly instances).

the performance of the model. On the one hand, a reasonable value interval is set for each feature to remove outliers. For example, the signal strength needs to be distributed between -100 and 0. On the other hand, for simplicity and efficiency, the missing values are complemented by the average of the features. All features are converted into standard numerical forms and discretized into an ordinal vector like $\{1, 2, 3 \cdots n\}$. The features after engineering are displayed in Table I.

2) Instance clustering: From the dataset, we can be observed that the RTT value fluctuates within a large range under the same measurement scenario. In order to better characterize the RTT performance, we cluster all samples with same features into "instances". If the RTT expectation and standard deviation of the instance sample subspace are both less than the threshold, the instance is marked as normal, otherwise anomaly. In the experiment, we set the thresholds for expectation and standard deviation to 160 and 100, respectively, where 160 is the RTT expectation of all samples. In addition, we use the reciprocal of the standard deviation as the weight to measure the quality of the instance, with an upper bound as 1.

In fact, the RTT expectation of an instance has a negative correlation with its weight, as shown in Figure 4. The y-axis represents the instance weight and is represented by the logarithm of 10. It can be seen that the weight sum of the anomaly instance is less than the weight sum of the normal instance, leading to the problem of class imbalance and the bias of the model. Therefore, we stretch the two classes' weight sum to be equal by multiplying a fixed factor to normal instances' weight.

B. Tree Modeling

In Section 2, we introduce the basic idea to extract rules from decision trees. Here we introduce the details of the decision tree modeling.

1) Mining Procedures: We utilize a toy example to illustrate how we extract rules from decision trees. Table III shows the mini dataset with five instances, where NetType and Speed are features and Performance is the label. NetType has two cases 0 and 1 and Speed has three cases from 0 to 2, while Performance is classified with normal and anomaly. Then we build decision tree-based on the dataset as Figure 5 shows. To deduce rules from the tree, We traverse all nodes except for the root node and obtain information from their ancestor nodes. For sample subspace of node 1, they all satisfy "NetType=0", which is the split of their parent node. Therefore, we obtain our first potential rule as {"NetType=0"}. For node 2, we observe its Gini 0.5 is larger than parent node's 0.32. Because of the increase of impurity, this node is not informative enough to extract rules and we just skip it. For node 3, it has two ancestor nodes, which raises the feature requirements as "NetType≠0" and "Speed≤0.5", thus we can derive the potential rule as {"NetType=1","Speed=0"}. Similarly, the sample subspace of node 4 satisfies "NetType≠0" and "Speed¿0.5", which forms two potential rules {"NetType=1","Speed=1"} and {"NetType=1","Speed=2"}. However, the latter one leads to an empty sample subspace and we only keep {"NetType=0","Speed=1"} as valid rule. Now we have traversed all nodes and extracted 3 rules from the decision tree.

To give a brief analysis, we can find that there are total 11 kinds of potential rules (5 for one feature type and 6 for two feature type). How can decision trees finish the filtering of rules? The first reason is that decision trees are generated from existing samples. Feature combinations leading to empty sample subspace will not be expressed in decision trees. For example, there will not exist a tree node that requires Speed to be 2 and NetType to be 1 because there are no instances satisfy this requirement. The second reason is that rules which is not informative will be eliminated implicitly due to the greedy property of decision trees. For example, eliminated rule {"Speed=1"} contains two samples with one being anomaly and another being normal, this is obviously not informative while valid rule {"NetType=0"} leads to three samples all being anomaly.

2) Criteria Analysis: We measure impurity of tree nodes or rules by Gini. Gini means the possibility of two randomly selected samples belonging to different classes. Here We carry a simple analysis on Gini to find out its relation to the impurity of nodes. We present a notation list displayed in Table II. Assume samples are classified with labels from 1 to K. We can calculate the proportion of each class on node x as:

$$P_x^k = \frac{\sum_{i \in S_x} w_i \times \mathrm{I}(y_i = k)}{W(x)}. \tag{1}$$

The Gini of node x is calculated as:

$$G(x) = \sum_{k=1}^{K} P_x^k (1 - P_x^k) = 1 - \sum_{k=1}^{K} \left(P_x^k\right)^2. \tag{2}$$

Then we can obtain the variance of class proportion in x:

$$\begin{aligned} D(P_x) &= 1/K \sum_{j=1}^{K} \left(P_x^j - \bar{P}_x\right)^2 = 1/K \sum_{j=1}^{K} \left(P_x^j - \frac{1}{K}\right)^2 \\ &= 1/K \sum_{j=1}^{K} \left(\left(P_x^j\right)^2 - \frac{2}{K} P_x^j + \frac{1}{K^2}\right) \\ &= 1/K (\sum_{j=1}^{K} \left(P_x^j\right)^2 - \frac{1}{K}). \end{aligned} \tag{3}$$

According to equation 2 and equation 3, we get the relation between variance and Gini:

$$G(x) = 1 - (K \times D(P_x) + 1/K). \tag{4}$$

We can observe that when K is fixed, there is a linear relation between class proportion's variance and Gini. The bigger the variance is, the smaller the Gini is. To be noticed, big variance implicitly means the samples are more deviated to one class, thus leading to a relatively pure node. Therefore ,we can conclude that smaller Gini indicates smaller impurity.

3) Node Spliting Strategy: According to CART [10], each tree node will be split into two child nodes to decrease impurity. To extract all potential rules, we set no depth limit for the generation of decision trees. The tree node will continue spliting until reaching completely pure, i.e., Gini equals to 0. However, if we prune the tree to avoid overfitting, we have to suffer the loss of potential rules as the shrinkage of tree structure. Therefore, We propose another method based on random forests in Section 4 to increase the generalization performance.

There are categorical and ordinal features as we can see in Table II. For different kinds of feature, the strategy to find the best spliting point also varies. Define all possible splits as set R, the performance of one certain split θ can be measured as:

$$G(S_x, \theta) = \frac{W(S_l)}{W(x)} G(S_l) + \frac{W(S_r)}{W(x)} G(S_r), \tag{5}$$

where S_l and S_r represent two child nodes generated by split θ. Then we need to traverse all features to find split θ^* that minimizes equation 5:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left(G(S_x, \theta)\right). \tag{6}$$

If feature v is ordinal with m unique values, we first sort them in ascending order: $\{v_1, v_2, \cdots, v_m\}$. We can observe that there are $m-1$ possible splits from v_1 to v_{m-1} and we traverse all to find the best one for this feature. The sample subspace of S_l and S_r can be represented as below, where $1 \leq i \leq m-1$:

TABLE III
MINI DATASET

NetType	Speed	Performance
0	0	anomaly
0	2	anomaly
0	1	anomaly
1	1	normal
1	0	anomaly

TABLE IV
EVALUATION OF RULES

Rule	Label	Confidence
NetType=0	anomaly	0.48
NetType=1, Speed=0	anomaly	0.16
NetType=1, Speed=1	normal	0.16

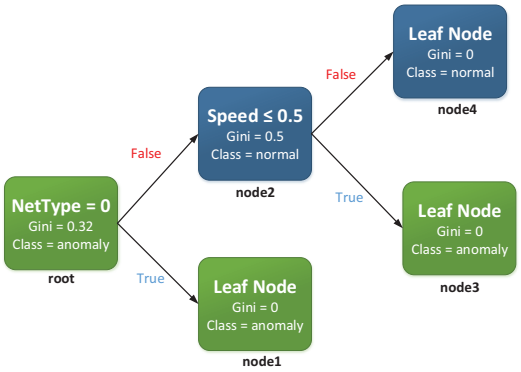


Fig. 5. Decision tree generated from the mini dataset.

$$S_l(v_i) = \{s \in S_x \mid v(s) \leq v_i\},$$
$$S_r(v_i) = \{s \in S_x \mid v(s) > v_i\}. \tag{7}$$

If feature v is categorical with values $\{v_1, v_2, \cdots, v_m\}$, there are total $2^{(m-1)} - 1$ possible splits, which is a huge number of cases to check. According to CART [10], the search space can be reduced to $m - 1$, as equation 8 shows, where $1 \leq i \leq m - 1$:

$$S_l(v_i) = \{s \in S_x \mid v(s) = v_i\},$$
$$S_r(v_i) = \{s \in S_x \mid v(s) \neq v_i\}. \tag{8}$$

After obtaining the best splits of all features, we compare the performance of them to determine the final spliting point of this node.

IV. RULE EVALUATION

After obtaining all rules from decision trees, we now focus on evaluating rules to obtain anomalies' severity. We propose "confidence" as the evaluation criteria which is composed by three factors: standard deviation, weight sum and impurity decrease. Moreover, we enhance the generalizaion performance of our method by utilizing random forests. We introduce the details of forest generation and present the holistic algorithm of the forest-based approach.

A. Evaluation criteria

We first compare the weight sum between anomaly instances and normal instances in the rule's sample subspace and then label the rule with the larger one's class. Equation 9 shows the three factors for calculating the confidence:

$$C(x) = \text{std}(x) \times W(x) \times I(x). \tag{9}$$

For a certain rule x, the standard deviation of class proportion is defined as:

$$\text{std}(x) = \sqrt{\frac{1}{K}\sum_k \left(P_x^k - 1/K\right)^2}, \tag{10}$$

where P_x^k stands for the proportion of instances labeled as class k. High standard deviation implies the instances are more biased to the rule's label, which deserves more confidence. Thus this metric will help to select out the biased and informative rules while eliminating those trivial rules. Moreover, a rule with more instances is obviously more reliable. Therefore, we calculate the sample subspace's weight sum as the second metric:

$$W(x) = \sum_{s \in S_x} w_s. \tag{11}$$

$$IQ(x) = \text{std}(x) \times W(x). \tag{12}$$

We define the multiplication of the above two metric as instance quality, as equation 12 shows. Higher instance quality deserves larger confidence; e.g., all instances are labeled with the same class and have a large weight sum. However, it is not persuasive enough just to take instance quality as the rule's confidence. Because of the greedy spliting of decision trees, the rule extracted from the shallow layer of the tree always contains much more instances than the rule extracted from the deep layer. For example, {"Verizon"} and {"Verizon", "LA"}, the former rule contains much more instances for it has no limit on the user location. This implicitly means shollow rule is easier to get a high instance quality for it has a bigger weight sum of sample subspace. For example, in binary case where classes are only normal and anomaly, assuming one shallow rule's weight sum is ten times larger than one deep rule's, even if the deep rule has the maximal standard deviation 0.5, the shallow rule's standard deviation only need to reach 0.05 to get higher instance quality. Obviously it is not that fair for deep rules. Therefore, a penalty term is needed to give proper punishment according to the depth of the rule. Here we choose impurity decrease as the penalty metric:

$$I(x) = G(r) - G(n_x), \tag{13}$$

where r stands for the root node and n_x stands for the node that the rule x lead to. As introduced in Section 2, deeper nodes in decision trees tend to have smaller Gini. As depth of the rule's node increases, the impurity decrease also increases. Thus we can conclude the impurity decrease implicitly represents the depth of rules. Therefore, the shallow

rule's confidence will be decreased due to smaller impurity decrease. Moreover, the degree of penalty is controlled by the impurity of rule's node. For example, if a shallow rule leads to a relatively pure node, the impurity decrease is also large thus the penalty will be weaken.

Here we explain the evaluation process with the help of the toy example shown in Section 3. First we label rules according to its sample subspace. From Table III, we can observe that {"NetType"=0} and {"NetType"=1, "Speed"=0} are anomaly and {"NetType"=1, "Speed"=1} is normal. For {"NetType=0"}, the standard deviation of it is 0.5 and the weight sum is 3. The impurity decrease is calculated as 0.32 - 0.0 = 0.32. Thus the confidence of {"NetType"=0} is $0.5 \times 3 \times 0.32 = 0.48$. Other rules' confidence is calculated in the same way as shown in Table IV and we rank them in descending confidence order.

We can see {"NetType=0"} has the highest confidence because it leads to three anomaly instances while {"NetType=1","Speed=0"} has a relatively lower confidence for having only one anomaly instances. To be noticed, we only consider binary class which are only anomaly and normal. In future work, we would like to extend binary case to multi-class case to enhance the granularity of anomaly detection, e.g. trivial anomaly and severe anomaly.

B. Enhancement of Generalization and Robustness

We first import randomness into the building of subtrees. For each subtree, we randomly produce a subspace of the original dataset as the input with the help of bootstrapping sampling [11]. Suppose original dataset D has N instances, we randomly select an instance with replacement and put it into the target dataset D^*. Repeating this operation for N times, D^* will get about 63 percent instances of D. For the spliting of each tree node, suppose we have M features, we randomly select $\sqrt{M}$ features to search for the best split.

To achieve a balance between performance and model complexity, we determine the number of subtrees needed to build by utilizing two metrics: (1) Out of Bag (OOB) score. (2) Average prediction accuracy. The basic idea of OOB is each subtree could be tested by the instances not involved in the training of this subtree, which can be represented as $D - D^*$. Breiman proves that OOB score can get the same accuracy as the test set does [12]. Thus OOB can be used to measure the generalization performance of random forest. Moreover, we ues 10-fold cross validation to measure the prediction performance of the random forest. The original dataset is divided into 10 sets, and each time we use one set as the test set and the rest as the training set to build random forest. Then we average the prediction accuracy of the ten trees to get the final prediction accuracy. These two metrics vary according to different number of subtrees chosen and we carry a tradeoff between them to select a proper scale for the forest. We will introduce the corresponding experiments in Section 5.

To get an intution about the generalization performance of the random forest and decision trees, we implement a small experiment to compare the prediction accuracy of them. Figure 6 shows the prediction performance of decision trees and random forest on LTE USA dataset and Wi-Fi USA dataset respectively. The x-axis represents the minimal number of instances (MNI) required to be at the leaf node, which is displayed as the logrithm of 2. With this parameter being larger, the maximal depth of the tree becomes lower, which implicitly restricts the complexity of the model. In this way we can compare the two methods more comprehensively. We still utilize 10-fold cross validation to measure the average prediction accuracy (APA). We can see random forest achieves a better performance than decision trees when MNI is relatively small, which means, the models is relatively complex. However, when MNI becomes larger, random forest performs a little worse than decision trees. This is because random forest cannot fit the training data well with shallow subtrees with randomness imported. In our work, we generate trees without depth limit, thus random forests would naturally have a better generalization performance than decision trees. To summarize the forest-based method, details of the holistic algorithm is shown in Algorithm 1.

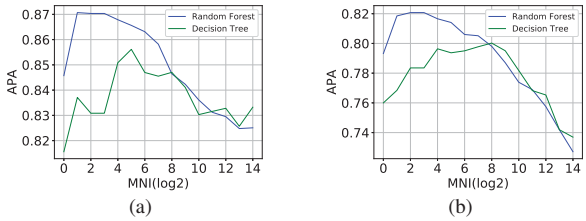


Fig. 6. Comparison of prediction performance between random forests and decision trees. (a) Average prediction accuracy on LTE USA dataset. (b) Average prediction accuracy on Wi-Fi USA dataset.

V. Experiments

A. Prerequisites

The basic settings of our experiments are composed by three parts: datasets, models and experimental setup.

Datasets: To facilitate experimental analysis, we split two sub-datasets from the MopEye dataset for experiments: LTE USA datasets and Wi-Fi USA datasets. The former cntains all USA's LTE records with 391,607 samples collected from 19 operators and 604 applications. The latter contains all USA's Wi-Fi records with 1,144,654 samples collected from 617 operators and 1,438 applications.

Models: There are three data mining approaches we test in the following experiment: apriori rule mining, tree-based approach and forest-based approach. Specifically, we choose a classical data mining approach, apriori association rule mining algorithm, to compare with our own approaches. Apriori algorithm is a data mining approach to extract the most relative rules from the raw data with two parameters set: support and confidence [13]. By trial and error, we empirically set support as 0.01 and confidence as 0.5 to get the best performance of apriori algorithm.

Algorithm 1 Holistic Algorithm

Input: $\{t_1, \cdots, t_N\}$
Output: $FinalAnalysis$
  $FinalAnalysis$ = ϕ
  for each tree t_i in $\{t_1, \cdots, t_N\}$ **do**
    $Rule$ = ϕ
    for each node n_j in t_i in depth-first order **do**
      p_j = path generated by backtracking n_j
      $p_j.info$ = feature ranges of p_j
      $n_j.rules$ = rules extracted from $p_j.info$
      for each rule r_m in $n_j.rules$ **do**
        $confidence(r_m) = \text{std}(r_m) \times W(r_m) \times I(r_m)$
        if r_m not in $Rule$ **then**
          $Rule$.**append**(r_m)
        else
          $Rule(r_m).confidence$ = $confidence(r_m)$
        end if
      end for
      $FinalAnalysis$.**append**(Rule)
    end for
    $FinalAnalysis$ = $FinalAnalysis$.**aggregation**()
  end for

Experimental Setup: We build up decision trees and random forest with the help of machine learning tool Scikit-Learn 0.19, which is based on Python. However, the decision trees library of Scikit-Learn only support the spliting strategy of oridinal features. We modify the cython source code and recomplie the decision trees library of Scikit-Learn to support the spliting of categorical feature as mentioned in Section 3.

We first conduct experiments to choose the proper model scale for forest-based approach. As mentioned in Section 4, we choose OOB score and average prediction accuracy to determine the number of subtrees. We conduct experiments on LTE USA dataset and Wi-Fi USA dataset respectively, and the results are shown in Figure 7. No matter Wi-Fi or LTE, we can see these two metrics are both tending towards stability as the number of subtrees becomes larger than about 25. To minimize the computation complexity, we finally choose 25 as the number of subtrees.

B. Detection of Synthetic Anomalies

First, we evaluate the proposed method by the performance of network anomaly detection. Since the data of MopEye is not labeled, this means that we cannot know whether the detected network anomaly is true. In order to better verify the effectiveness of the proposed method, we manually construct anomaly rules with synthetic instances.

Specifically, 10 synthetic anomaly rules are introduced into the LTE-USA dataset and Wi-Fi USA dataset respectively: five one-dimensional anomaly rules, three two-dimensional anomaly rules, and two three-dimensional anomaly rules. The dimension here represents the anomaly cause. Each rule is constructed with 300 instances, where the ratio of anomaly instances varies between 91% and 100%. All synthetic instances are mixed with the orignial dataset.

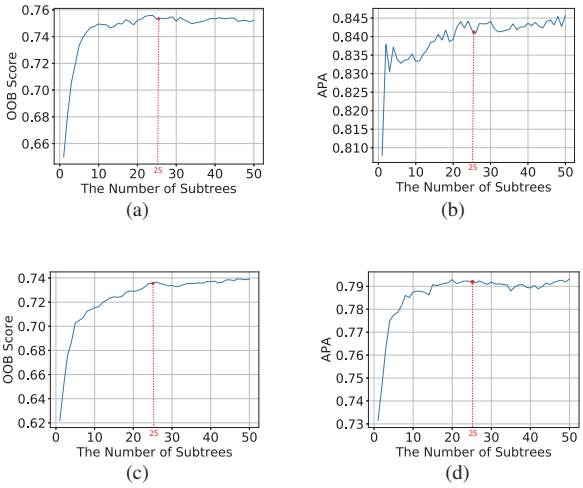


Fig. 7. OOB score and average prediction performance under different number of subtrees. (a) OOB score on LTE USA dataset. (b) Average prediction accuracy on LTE USA dataset. (c) OOB score on Wi-Fi USA dataset. (d) Average prediction accuracy on Wi-Fi USA dataset.

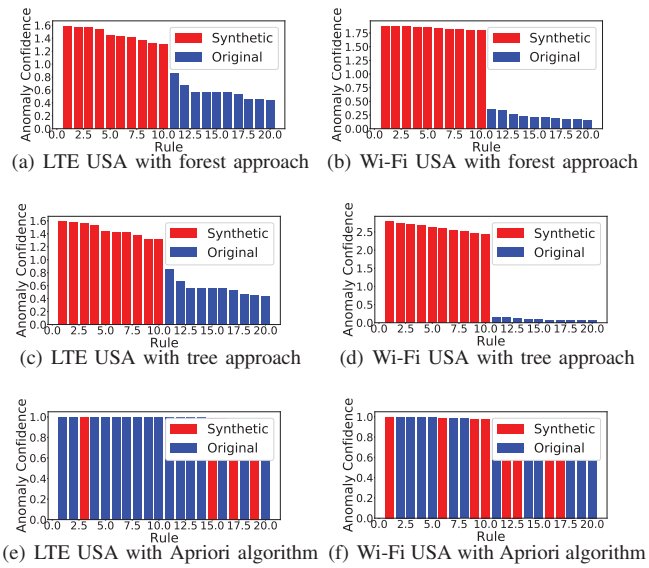


Fig. 8. Three approaches' anomaly detection performance of synthetic anomalies. Red bar stands for synthetic anomaly while blue bar stands for original anomaly detected.

After introducing the synthetic anomaly rules, we use the new dataset to evaluate the network anomaly detection performance of the above three algorithm. The experiment result is shown in Figure 8. For each scenario, we display 20 detected anomalies with the highest confidence. Red bar stands for synthetic anomaly while blue bar stands for original anomaly detected.

From above results, we can see no matter forest-based approach or tree-based approach, could both successfully

TABLE V
LEVENSHTEIN DISTANCE BETWEEN NOISY RESULTS AND ORIGINAL RESULTS

Levenshtein distance	Forest-based	Tree-based	Apriori
LTE USA	1083	1497	661
Wi-Fi USA	1316	1576	884

extract all synthetic anomalies.Apriori algorithm can only extract the synthetic anomaly rules with high proportion of anomaly instances. This is because apriori algorithm pays too much attention on the proportion of anomaly instance, while ignoring the rule's occurrence frequency among the dataset. Our approaches would take both anomaly proportion and occurrence frequency into consideration. To summarize, this experiment verify the anomaly detection effectiveness of our approaches under the scenario of crowdsourcing, compared to the apriori algorithm.

C. Generalization Performance

Here we define generalization performance of anomaly detection as the similarity of rule mining results. If the rule mining result of a certain approach can keep consistent, even some minor changes like noise are introduced into the original dataset, we can say this approach has good generalization performance.

Specifically, levenshtein distance is chosen to measure the proximity of rule mining results. Levenshtein distance is generally defined as "the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other" [14]. First of all, top 300 rules are selected from each rule mining result. Then we flatten the 300 rules' feature value to form a one-dimension vector. The vector can be viewed as a "word" while each feature value can be treated as a "character". Therefore, the levenshtein distance of the two vectors can be regarded as the similarity of the two rule mining results. The shorter the levenshtein distance, the more similar the two results are, thus indicating a better generalization performancer. Based on this idea, we evaluate the generalization performance of the three anomaly detection algorithms from two aspects: roubustness of noise and performance of cross validation

1) Robustness of Noise: We first evaluate the generalization performance of the three approaches by measuring the robustness of noise. The basic idea is to introduce random gaussian noise into the original dataset to get a new noisy dataset. Then we calculate levenshtein distance of the two datasets' rule mining results. Concretely, we add gaussian noise ($Mean = 0, StandardError = 20$) into the original LTE USA dataset and Wi-Fi USA dataset respectively and group them into instances again. The result is shown as Table V.

We can observe that apriori algorithm has the minimal levenshtein distance because apriori algorithm is a relatively simple algorithm which underfits the training data, thus leading to a good generalization performance. However, the price of underfitting is the poor performance on the anomaly detection, as the previous experiment shows. Although forest-based approach has a bit smaller levenshtein distance than apriori algorithm, it still enhances about 25 percent of generalizaion performance compared with the tree-based approach.

TABLE VI
LEVENSHTEIN DISTANCE BETWEEN CROSS VALIDATION RESULTS OF LTE USA DATASET

Levenshtein distance	Forest-based	Tree-based	Apriori
LTE USA 1 - 23	1149	1464	782
LTE USA 2 - 13	1183	1571	784
LTE USA 3 - 12	1122	1437	791
Average	1151	1491	786

TABLE VII
LEVENSHTEIN DISTANCE BETWEEN CROSS VALIDATION RESULTS OF WI-FI USA DATASET

Levenshtein distance	Forest-based	Tree-based	Apriori
Wi-Fi USA 1 - 23	1315	1762	1066
Wi-Fi USA 2 - 13	1255	1679	1027
Wi-Fi USA 3 - 12	1298	1674	1073
Average	1289	1705	1055

2) Performance of Cross Validation: Next we evaluate the generalization performance through 3-fold cross validation. The reason for choosing 3-fold is to achieve the balance between accuracy and runtime. Specifically, we shuffle the original dataset and cut it equally into three parts. We apply three approaches to the six datasets and get corresponding mining rule results. Finally, levenshtein distances are derived from the mining results of dataset-1 and dataset-23, dataset-2 and dataset-13, dataset-3 and dataset-12. We execute above procudures on LTE USA dataset and Wi-Fi USA dataset respectively.

The result is shown in Table VI and Table VII, which is similar to the result of noise robustness evaluation. Apriori algorithm has the minimum levenshtein distance. And forest-based approach still has a shorter levenshtein distance than tree-based approach, which leads to about 25 percent enhancement on generalization performance.

D. Discussion of Experiments

From the results of experiments, we obtain the comprehensive performance of our approaches and apriori algorithm. Apriori algorithm has the best generalization performance due to the simplicity of its model while performing much worse as to anomaly detection. Tree-based approach gets a good anomaly detection performance while it is too sensitive to noise thus leading to the worst generalization performance. Forest-based approach strikes the best balance between anomaly detection performance and generalization performance. While successfully detecting all synthetic anomlies, forest-based approach also achieves about 25 percent higher generalization performance than single tree-based approach.

VI. RELATED WORK

Traditional methods to detect network anomalies can be classified as threshold detection, statisical analysis and wavelet

transform. Frank [15] design an adaptive threshold detection mechanism. [16] Based on the statistical method of Generalized Likelihood Ratio (GLR) to detect anomalies in network servers while the work in [17] is based on wavelet transform for anomaly detection.

A number of specific measurement tools have been developed to do the network peformance analysis. For example, in the field of active measurement, Ferlin *et al.* implemented a bottleneck detection method for multipath transmission control protocols [18], while Li *et al.* modeled the repetitive behavior of network packet arrivals [19]. In the field of mobile network monitoring applications, Mobilyzer [20], MobiePerf [21] and MobileInsight [22] worth attentions. Since the active measurement is too costly and the application of mobile network monitoring is concentrated on the single device, it is impossible to comprehensively analyze the performance of a large-scale network.

In order to cope with the analysis of large-scale data, machine learning methods are often adopted in the design of network anomaly detection. Pajouh *et al.* utilize naive bayes, KNN and linear discriminant analysis to build a two-layer classification model [23]. Zhao *et al.* proposes a new framework for real-time anomaly detection of network traffic with the help of machine learning [24]. Association rule mining algorithms are also used to locate the causes of network performance degradation [13]. The Opprentice system [25] performs anomaly detection by training random forests to automatically combine traditional anomaly detectors and automatically choose appropriate relevant parameters. [26] uses decision trees to model the metrics of the event network, and then manually analyzes the structural information of the tree.

VII. Conclusion

In this work we propose a tree-based data mining method for large scale network anomaly detection. We determine the root cause of network anomaly with the help of decision trees, which is utilized as an analysis model rather than a prediction model. By analyzing the tree structure and tree nodes' information, we extract the potential cause of abnormal RTT behavior. Moreover, we set up an evaluation criteria called "confidence" to measure the severity of each anomaly. To achieve a better robustness, we build random forest and compact subtrees' outputs to produce a generalized result.

In the furture work, we would explore the enhancement of anomaly detection and decision. First, as to the input data, we would enlarge the scale and coverage of crowdsourcing. Second, as to the modeling approach, we would like to extend the granularity of anomaly detection, e.g., extending the binary classification to multi-classification during the labeling of instances.

VIII. Acknowledgment

The work in this paper is supported by the National Natural Science Foundation of China (NSFC) 61872233, 61572319, 61829201, 61532012, 61325012, 61428205, the Science and Technology Innovation Program of Shanghai under Grant 17511105103 and Shanghai Key Laboratory of Scalable Computing and Systems. Xiaohua Tian is the corresponding author.

References

[1] D. Wu, R. K. Chang, W. Li, E. K. Cheng, and D. Gao, "Mopeye: Opportunistic monitoring of per-app mobile network performance," in *Proc. USENIX ATC*, 2017, pp. 445–457.

[2] W. Huang, C. Zhou, and Y. Li, "cnicloud: Querying the cellular network information at scale," in *Proc. ACM WiNTECH*, 2017, pp. 17–24.

[3] N. Larson, D. Baltrunas, A. Kvalbein, A. Dhamdhere, A. Elmokashfi *et al.*, "Investigating excessive delays in mobile broadband networks," in *Proc. ACM AllThingsCellular*, 2015, pp. 51–56.

[4] A. Nikravesh, D. R. Choffnes, and E. Katz-Bassett, "Mobile network performance from user devices: A longitudinal, multidimensional analysis," in *Proc.Springer PAM*, 2014, pp. 12–22.

[5] Y. Liu and Z. Li, "aleak: Privacy leakage through context-free wearable side-channel," *Proc. IEEE INFOCOM*, 2018.

[6] K.-H. Kim, H. Nam, and V. Singh, "Dyswis: crowdsourcing a home network diagnosis," in *Proc. IEEE ICCCN*, 2014, pp. 1–10.

[7] F. Espinet, D. Joumblatt, and D. Rossi, "Framework, models and controlled experiments for network troubleshooting," *Computer Networks*, vol. 107, pp. 36–54, 2016.

[8] Z. S. Bischof, J. S. Otto, and M. A. Sánchez, "Crowdsourcing isp characterization to the network edge," in *Proc. ACM W-MUST*, 2011, pp. 61–66.

[9] "Mobiperf on google play," 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.mobiperf.

[10] R. O. L. Breiman, J. Friedman and C. Stone., *Classification and Regression Trees*. New York, NY: CRC Press, 1984.

[11] R. T. B. Efron, *An Introduction to the Bootstrap*. New York, NY: Chapman Hall, 1993.

[12] L. Breiman, *Machine Learning*, 1996, name of chapter: Bagging predictors, pp. 123–140.

[13] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, and H. Yan, "Detecting and localizing end-to-end performance degradation for cellular data services," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[14] Wikipedia, "Levenshtein distance," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Levenshtein_distance

[15] F. Frank, S. Dan, and A. Roy, "Fault detection in an ethernet network using anomaly signature matching," *Proc. ACM SIGCOMM*, pp. 279–288, 1993.

[16] T. Marina and J. Chuanyi, "Statistical detection of enterprise network problem," *Journal of Network and Systems Management*, pp. 47–58, 1999.

[17] L. WEI and A. A. GHORBANI, "Network anomaly detection based on wavelet analysis," *EURASIP Journal on Advances in Signal Processing*, pp. 1234–1249, 2009.

[18] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath tcp with shared bottleneck detection," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[19] J. Li, J. Tao, X. Ma, J. Zhang, and X. Guan, "Modeling repeating behaviors packet arrivals: Detection and measurement," in *Proc. IEEE INFOCOM*, 2015, pp. 2461–2469.

[20] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao, "Mobilyzer: An open platform for controllable mobile network measurements," in *Proc. ACM MobiSys*, 2015, pp. 389–404.

[21] J. Huang, C. Chen, and Y. Pei, "Mobiperf: Mobile network measurement system," 2011.

[22] Y. Li, C. Peng, Z. Yuan, J. Li, H. Deng, and T. Wang, "Mobileinsight: Extracting and analyzing cellular network information on smartphones," in *Proc. ACM MOBICOM*, 2016, pp. 202–215.

[23] H. H. Pajouh, G. Dastghaibyfard, and S. Hashemi, "Two-tier network anomaly detection model: a machine learning approach," *Journal of Intelligent Information Systems*, vol. 48, no. 1, pp. 61–74, 2017.

[24] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *Proc. IEEE DRCN*, 2015, pp. 267–270.

[25] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proc. ACM IMC*, 2015, pp. 211–224.

[26] A. P. Iyer and L. E. Li, "Automating diagnosis of cellular radio access network problems," in *Proc. ACM MOBICOM*, 2017, pp. 79–87.