Automated Diagnosis of Known and Unknown Soft-Failure in User Devices Using Transformed Signatures and Single Classifier Architecture

Chathuranga Widanapathirana, Jonathan C. Li, Milosh V. Ivanovich, Paul G. Fitzpatrick, and Y. Ahmet Şekercioğlu
Dept. of Electrical and Computer Systems Engineering, Monash University, Australia
{chathuranga.widanapathirana, jonathan.li, milosh.ivanovich, paul.fitzpatrick, ahmet.sekercioglu}@monash.edu

***Abstract*—We present an automated solution for rapid diagnosis of both known and unknown "soft-failures" in network User Devices (UDs). A multiclass classifier is first trained with the known faults and during diagnosis, the unknown faults are clustered to determine the existence of a new fault. Then, in an iterative process, the classifier is re-trained with the newly detected fault. The system relies on 410 features long Normalized Statistical Signature (NSSs) for fault characterization. Since, the high dimensionality of the NSS can create model overfitting, we propose EigenNSS, a transformed signature with lower dimensions and minimum information loss.**

The system is evaluated with live network data of 17 emulated UD faults. The results show an overall detection accuracy of 97.2% with minimum false positives and dimensionality reduction of 93.9%. Also, compared with the NSS, the EigenNSS has faster training and diagnosis times suitable for on-demand as well as real-time diagnostic applications.

I. Introduction

Root causes of the network performance problems experienced by the end-users can be traced to service provider servers, backbone networks, access network, and User Devices (UDs), i.e., devices used directly by an end-user to communicate [1]. Performance management and fault diagnosis of these network segments is a complex task and network operators have relied on Network Monitoring Systems (NMS) to detect the signs of network problems. However, the inference from such signs to diagnose the root cause requires expensive resources including intervention by skilled personnel. In recent years, researchers have proposed automated diagnosis solutions specially focused on core network, access network, and servers where a problem can potentially affect large number of users [2]. However, only a little attention has been given to develop such diagnosis techniques for UDs.

Network failures have been commonly divided into two main categories [3]: "hard-failures" and "soft-failures". A hard-failure is characterized by the loss of connectivity and inability of the network to deliver any bandwidth to the user. This type of faults are easily identified and resolved since if unnoticed by the operator, users are sure to identify the lost connection immediately. Soft-failures are less well-defined, but the prevailing opinion in most of the literature is that soft-failures are characterized by degraded performance (also ill-defined), or the loss of network bandwidth. Soft-failures are harder to detect and often require lengthy investigations to identify the root causes. In this work, we define an "healthy" UD, which is a device that receives maximum network performance level reasonably expected by a user. Degraded network quality from the baseline is defined as a soft-failure. Resolution of such soft-failures are critically important for maintaining a user's Quality of Experience (QoE) [4].

Parameter misconfiguration in various protocol layers, often as a result of overly conservative defaults in operating systems have been found to be a common issue causing performance bottlenecks [5]. In addition, hardware problems, new application installations, NIC driver issues, kernel level software problems, mismatch between system settings and the link [6], and protocol implementation errors [7] have commonly been found to cause soft-failures.

II. Related Work

As a method to automate the diagnosis of soft-failures in UDs, authors have previously proposed diagnostic systems based on supervised Machine Learning (ML) techniques and fault signatures [8], [9]. The ML algorithms in these systems are first trained using signatures called "Normalized Statistical Signatures (NSS)" generated from packet traces of controlled Transmission Control Protocol (TCP) connections between diagnostic server and UDs with known faults. Subsequently, a trace collected from a UD suspected of suffering from performance problems can be sent through the trained classifier system identify the exact root cause. However, the diagnostic capability of these systems was limited by the classes of faults used during the training. Consequently, a previously unknown fault cannot be diagnosed and often leads to a false positive. Also, the NSSs contains large number of features to capture for all possible types of UD faults. However, large feature sets in ML algorithms can lead to overfitting of the classifier models, a problem known as "curse of dimensionality"[10].

We carried out a detailed literature survey and did not find comparable solutions for UD soft-failure diagnosis. Supervised-ML has been extensively used for network related applications, specially in intrusion detection systems[11], traffic classifications[12], anomaly detection[13], and source identification and system fingerprinting [14]. Recently, unsupervised-ML has gained popularity in network applications and has been used in intrusion detection [15], IP traffic classification [16]. However, hybrid techniques that combine both supervised and unsupervised ML [17], [18] have not been studied extensively in networking domain, specially as a failure diagnosis tool.

In this paper, we propose a new single classifier architecture that combines supervised-ML with unsupervised-ML to create a known and unknown fault diagnostic system. Also, we present EigenNSS and its creation process which, transforms the NSSs to lower dimensions without losing meaningful information. We also perform a detailed analysis of system performance with data gathered from real-world networks.

The rest of this paper is organized as follows. Section III presents the operational details of the diagnostic system. Section IV discusses how the signatures of UD failures are created. In Section V, we present how the NSSs are transformed to create EigenNSS and in Section VI, we explain the training and classification criterion. Finally, Section VII contains detailed performance analysis of the system followed by the conclusions in Section VIII.

III. Operational details of the diagnostic system

A. Deployment

The diagnostic server is deployed as an application on the access router/server at the edge of the network administrator domain as shown in the Figure 1. This narrows down the path to the UD to an access link and eliminates complexities that can affect the uniformity of captured packet traces. Upon a user request, first, the modules needed for the file transfers and packet captures are loaded. Then, packets from an upload and a download file transfer is captured by the capture modules.

B. Operational overview

The Figure 2 shows the operational stages of the diagnostic system. The system operated in three main stages: (i) training stage, (ii) diagnosing stage, and (iii) new class recognition stage where in Figure 2, these stages are distinctly color coded.

During the training stage, the packet traces are collected from UDs known to suffer from a particular fault. In this system, each fault is considered a unique class. Statistical attributes of these traces are then extracted from the packet traces and these extracted features together with class label is called the "raw signature". These raw signatures are then normalized using the expected performance baseline to create the NSS. The NSSs from multiple classes are stored in a database and these NSSs are used to calculate the eigen matrix of the database. The NSSs are projected on to a selected number of principle components of the eigen matrix to create a transformed signature called "EigenNSS". The EigenNSSs of each class is then used to calculate the pattern vector (ε_f) of that particular class. Finally, two thresholds are chosen for the system, (i) λ_{nss} for determining if a particular signature is valid, and (ii) λ_{fc} for determining class associations.

Once the system is trained, traces collected from UDs suspected faulty can be diagnosed. First, a trace is collected from the UD and sent through the same feature extraction and NSS generation process. Then, the NSS is projected on the eigen matrix created during the training to create EigenNSS. EigenNSS is then used to calculate the pattern vector (ε) and the Euclidean distance (σ) of the pattern vector from each known class. Also, the NSS is used to calculate square distance (μ) from the training NSS data set. Depending on the minimum Euclidean distance (σ_{min}) and square distance (μ), the system can determine one of three outcomes.

(i) If $\mu > \lambda_{nss}$, the sample is highly distorted to the point

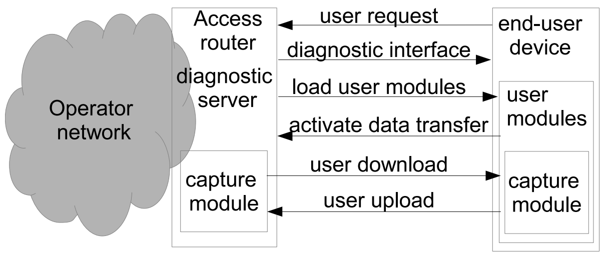
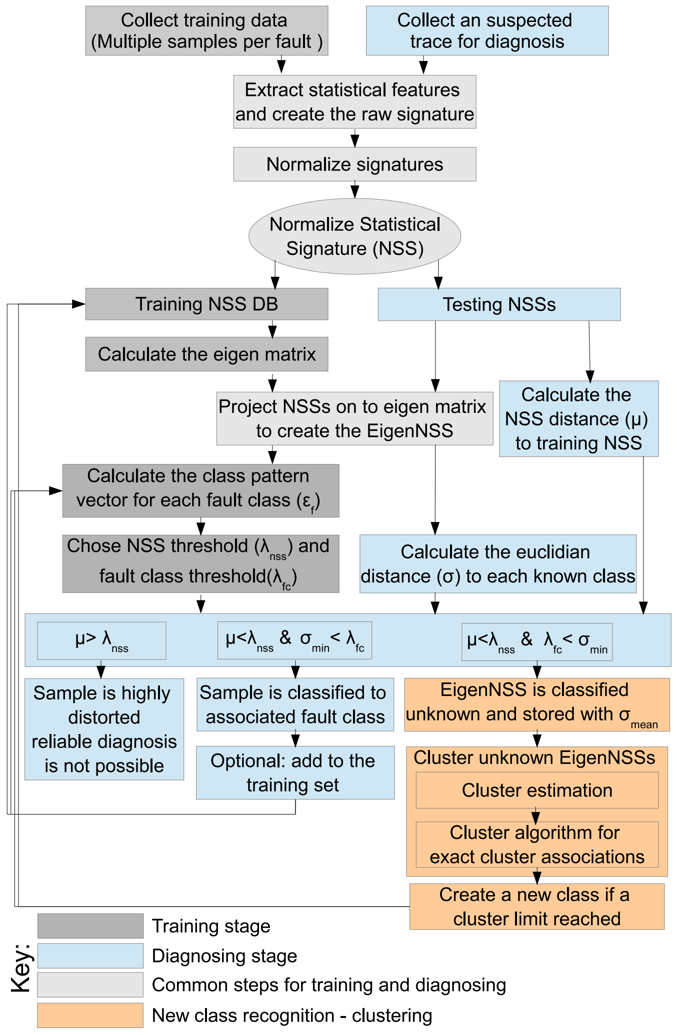


Fig. 1. Operational deployment of the diagnostic system.



Collect training data (Multiple samples per fault)
Collect an suspected trace for diagnosis
Extract statistical features and create the raw signature
Normalize signatures
Normalize Statistical Signature (NSS)
Training NSS DB
Testing NSSs
Calculate the eigen matrix
Project NSSs on to eigen matrix to create the EigenNSS
Calculate the NSS distance (μ) to training NSS
Calculate the class pattern vector for each fault class (ε_f)
Chose NSS threshold (λ_nss) and fault class threshold(λ_fc)
Calculate the euclidian distance (σ) to each known class
μ> λ_nss
μ<λ_nss & σ_min < λ_fc
μ<λ_nss & λ_fc < σ_min
Sample is highly distorted reliable diagnosis is not possible
Sample is classified to associated fault class
Optional: add to the training set
EigenNSS is classified unknown and stored with σ_mean
Cluster unknown EigenNSSs
Cluster estimation
Cluster algorithm for exact cluster associations
Create a new class if a cluster limit reached
Key:
Training stage
Diagnosing stage
Common steps for training and diagnosing
New class recognition - clustering

Fig. 2. Operational overview of the diagnostic system.

a reliable diagnosis is not possible.

(ii) If $\mu < \lambda_{nss} \,\&\, \sigma_{min} < \lambda_{fc}$, the sample is classified to the class associated with the minimum distance. These samples can be added to the training NSS database or ε_f of the class can be recalculate to include the new sample.

(iii) If $\mu < \lambda_{nss} \,\&\, \sigma_{min} > \lambda_{fc}$, the sample contains a valid signature, yet does not belong to any of the known classes. The sample is classified as unknown.

The unknown classes are stored in a separate DB with their pattern vectors. These unknown classes are then sent though a cluster estimation algorithm [19] which determines (i) if the data set has samples that can be clustered within the λ_{fc} bound, and (ii) the number of clusters that can be created. Then these cluster data is then sent to a clustering algorithm which uses fuzzy C-means clustering with iterative optimization [20] to determine the exact cluster membership. A detailed discussion of [19] and [20] is not presented in this paper due to space limitations and can be found from the references. If the any of the clusters reach a minimum threshold size, this cluster is determined as a new class. The EigenNSS in the new class are sent to the class pattern vector calculation and NSS are sent to the classifier training database. Although new class can be dynamically added just by calculating class pattern vector, the system can be re-trained in a short time with the updated NSS database when it does not perform any diagnosis. Re-training includes the new classes in principal component selection and improves the final accuracy. Furthermore, the system prompts the administrators a new fault class has been detected. After an investigative analysis of the actual root cause, a class label can be added to the new fault.

IV. Normalized Network Signatures (NSS) of soft-failures

Any ML-based diagnostic system is highly dependent on the effectiveness of extracted signatures to characterize faults. We have, in previous publications, proposed a method to create signatures of UD faults that is both effective and scalable. Here, we only provide the high-level details of NSS creation and more details can be found in [21].

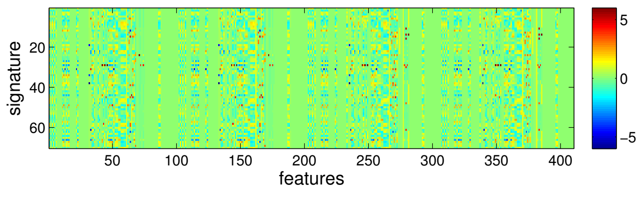
A signature is defined as a collection of features (or attributes), each representing a single aspect of runtime behavior and provide the key to differentiate normal behaviors from the abnormal or faulty ones. In our work, the features are extracted from TCP packet traces collected from controlled connections initiated between the diagnostic server and UD, on-demand by the user. Because of TCPs position in the middle of the protocol stack and reliable transport functionality, performance issues in other layers are embedded as anomalies on TCP packet streams. Also, by using TCP as the information source, data can be gathered remotely without user relinquishing control of the device, protecting the user's privacy. The connections carry a fixed size file of 20 MB as a download and an upload. The fixed size file has been used to provide an easily comparable baseline.

Feature type	Features
Cumulative totals of various packet types	Total packets SACK packets DSACK packets Retransmitted packets Triple dupACKs ⋮
Cumulative payload characteristics	Unique bytes Packets with data Data retransmitted Data out-of-order Data missing ⋮
Max, min, mean, cumulative observation frequencies of events	Out-of-order events Cumulative acknowledged segments Data retransmissions Max segment retransmissions Zero window advertisements Window probes ⋮
Max, min, mean information on variable parameters	Window advertisements Segment size ⋮
Initial state and final state parameters	Window scaling advertisements Initial window ⋮
Round trip time, arrival time progression analysis	Retransmission times-max-min-ave Idletimes-max-min-ave RTT-max-min-std ⋮
Boolean parameters of TCP settings	3 way handshake flags (SYN/FIN) TCP options Data pushed ⋮

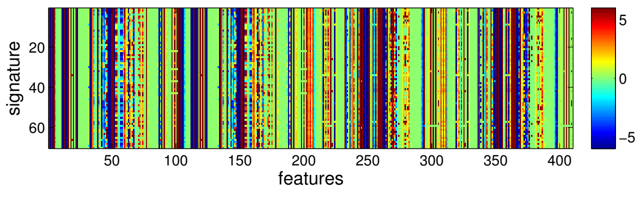
TABLE I
Types and limited examples of features extracted from the TCP trace. The same features are extracted from both upload and download packet streams. Separate cumulative features are created for each direction ($client \leftrightarrow server$) considering the bi-directional nature of the connections. Refer to [21] for further details.

The collected traces are then sent through the feature extraction module where analysis tool we developed based on tcptrace extracts aggregated statistical attributes of the traces. In this paper, we use 205 features extracted from each trace which brings the total feature set of the raw signature to 410 when both upload and download traces are combined. For examples, the features include cumulative totals of packet types, payload characteristics, observation frequencies of specific events, initial and final state parameters, delay based statistics, and TCP boolean parameters as summarized in Table I. However, an exhaustive list of features has not been presented due to space limitation of this paper.

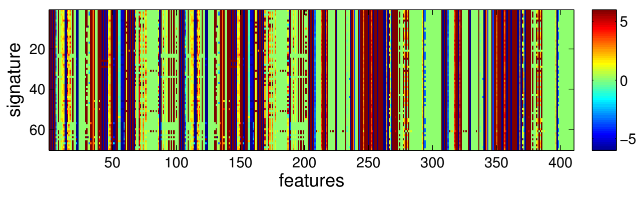
When deployed in a network, the traces captured from a healthy UD with optimum system settings provide the baseline performance signature. Each of the raw signatures is normalized against the healthy baseline and the resultant feature vector is called the Normalized Statistical Signature (NSS). The Figure 3 shows NSSs collected from 4 types (classes) of UDs, a healthy and three types of faults. The



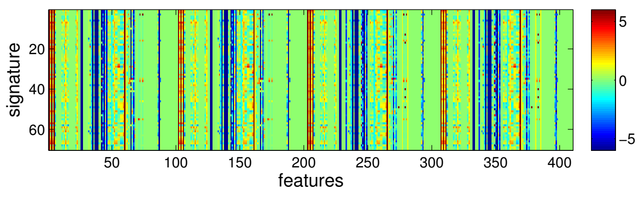
(a) Healthy UD



(b) Disabled SACK error



(c) Duplex mismatch



(d) Window scaling error

Fig. 3. Comparison of NSSs for four common UD soft-failures. Here, the x-axis (columns) of each figure represents 410 features of the 70 NSSs shown on y-axis (rows). Each of the features has been normalized and scaled [-6,6] and each of the features is represented by colored vertical line projecting the scaled feature value to RGB space. This representation offers easy visualization and comparison of the total NSS. As visible here, the combination of features uniquely represents each fault and can be used as the "fingerprint" for a diagnosis.

figures clearly show that each type of NSS has a unique feature pattern throughout the 70 samples and this pattern provides the distinction needed for an effective diagnosis.

Having a large feature set enables the NSS to characterize a range of faults through a single representation. However, as evident from the Figure 3, not all features contribute equally for the separability of the classes. Also, even within the same class, the NSSs show variations due to the stochastic nature of the link. When training a ML-based system, a large feature set can lead to over-fitting of the data and ultimately lead to poor generalization and poor classification accuracy. Furthermore, the feature variations within a class can compound this problem as ML-algorithm tries account for all the variations in the data by using more features. Consequently, the dimensionality of the NSSs should be reduced while preserving important information to build an effective diagnostic system.

V. EigenNSS: Transformed signatures for reduced dimensionality

To overcome the aforementioned challenges, we look at the signatures from information theoretic point of view emphasizing on the significant global features that contain a maximum amount of information. In our method, information is extracted from the NSSs that contributes to maximum amount of class variations. Then, the NSSs are encoded using this information to reduce the dimensionality of an NSS. We achieve this by finding the principle components (i.e eigenvectors of the covariance matrix) of the distribution of NSSs. These eigenvectors are then ordered, each one accounting for a different amount of variation among the NSSs. Finally, the NSSs are projected on to selected number (D) of eigenvectors that account for the highest variation. These projections are called "EigenNSS" and represent the original samples in a D-dimensional space.

A. *Calculating EigenNSS*

The calculation of EigenNSS is as the following. Assume we have training NSSs, $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p$ where $\mathbf{x}$ is an m-dimensional feature vector. For an example, the set of NSSs shown in Figure 3 has 70 samples per class and 4 classes ($p = 70 \times 4 = 280$) each with 410-dimensional feature vector ($m = 410$). Mean NSSs of the data set can be found by

$$\Phi = \frac{1}{p} \sum_{k=1}^{p} \mathbf{x}_k$$

The training NSSs matrix is then mean centered by

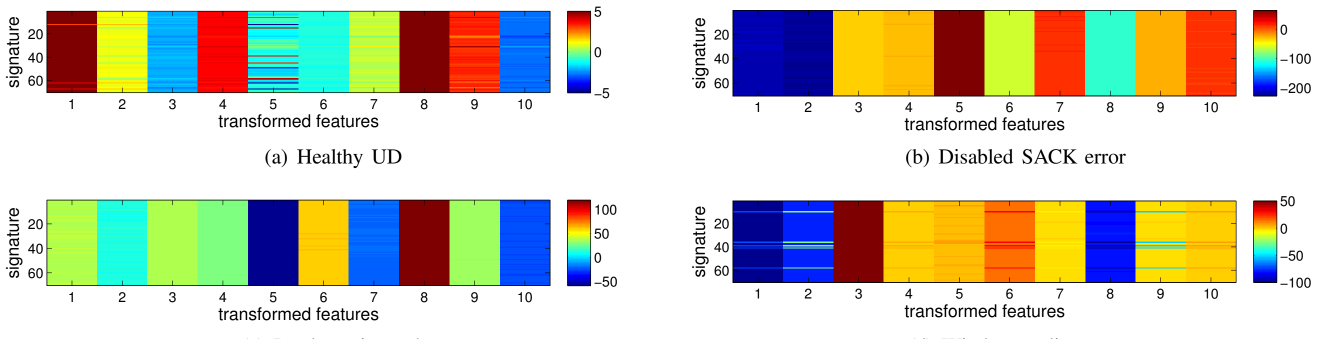
$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \Phi$$

where $\bar{\mathbf{x}}_i$ is the shifted m-dimensional feature vector of the i^{th} instance. The resultant matrix $\Delta = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_1, ..., \bar{\mathbf{x}}_p\}$ has the dimensions of $p \times m$ and subjected to principal component analysis where eigenvectors v_i and the corresponding eigenvalues η_i of Δ are determined by solving well-known singular value decomposition problem [22]. However, only a pre-determined D number of eigenvectors are selected with the highest η to create the eigen matrix which has the dimensions of $m \times D$. EigenNSSs are then created by projecting the shifted NSS matrix Δ on to the eigen matrix Ψ as

$$\Theta = \Psi \Delta$$

where $\Theta = \{\mathbf{e}_1, \mathbf{e}_1, ..., \mathbf{e}_p\}$ and $\mathbf{e}$ is an D-dimensional EigneNSS.

The Figure 4 shows the EigenNSSs with $D = 10$ created from NSSs in Figure 3. As can be seen, EigenNSSs has drastically reduced the dimensionality while preserving the most important information for class separation. This is clearly visible from the range of EigneNSS feature values in each class which shows significant differences compared to the NSSs.



(a) Healthy UD

(b) Disabled SACK error

(c) Duplex mismatch

(d) Window scaling error

Fig. 4. EigenNSSs generated from the NSSs shown in Figure 2. Here, each class has 70 EigenNSS with $D = 10$. Each class has been scaled differently to reflect the feature values range in each class.

VI. System Training and Classification Criteria

The EigenNSS matrix Θ contains samples from multiple classes, each associated with a fault (f). Assuming that each class has n EigenNSSs, the D-dimensional class pattern vector ε_f is calculated by averaging the EigenNSSs as

$$\varepsilon_f = \frac{1}{n} \sum_{k=1}^{n} \mathbf{e}_k$$

The simplest method of determining the class association of an NSS with EigenNSS $\mathbf{e}$ is by calculating the Euclidean distance σ between $\mathbf{e}$ and each of the class pattern vectors as

$$\sigma_f = \|\mathbf{e} - \varepsilon_f\|^2$$

The sample NSS is classified to as belonging to class f when the minimum σ_f is below a chosen threshold λ_{fc}.

Due to errors in data collection and random nature of networks, some collected packet traces can be distorted. However, due to the low dimensionality of the EigenNSS, there is a chance that erroneous NSSs generated from these distorted packet traces could project into a valid, but wrong class leading to a false detection. We define another distance threshold λ_{nss}, where squared distance between mean centered NSS and the training NSSs must be less than the λ_{nss} for the sample to be considered valid.

Once the class pattern vectors ε_f, λ_{fc}, and λ_{nss} are determined, the system is ready to perform diagnosis of UDs suspected faulty. First, a packet trace is collected from the UD and sent through the feature extraction, normalization to generate the NSS ($\mathbf{y}$). Then, the NSS is mean adjusted by

$$\overline{\mathbf{y}} = \mathbf{y} - \Phi$$

and projected on to the eigen matrix Ψ created during training as

$$\mathbf{E} = \Psi \overline{\mathbf{y}}$$

where, $\mathbf{E}$ is the D-dimensional EigenNSS of the sample. The Euclidean distance from the sample to each class is calculated by

$$\sigma_f = \|\mathbf{E} - \varepsilon_f\|^2$$

The distance from NSS feature vector $\mathbf{y}$ to training NSS are calculated as

$$\mu = \|\overline{\mathbf{y}} - \Phi\|^2$$

Depending on the minimum σ_f and μ, the sample is determined (i) distorted and discarded, (ii) classified into the class f, (iii) classified as a known class and added to unknown DB following the criteria previously mentioned in Section III.

VII. Performance Analysis

The performance of the system was evaluated experimentally and this section provides an analysis of the diagnostic capability.

A. Data set

The data required to train and test the system was collected over the live campus network. Collecting data from actual user complaints was challenging considering the amount of time required to collect a sufficient number of samples for a statistically robust analysis, and resources needed to manually identify the issues and label the signatures. Instead of using data from real users, we created a fault emulator module to recreate commonly found problems in UDs. The fault emulator was installed in test computers connected to the live university network in multiple locations. The campus network was used to demonstrate the viability of the system in a real computing environment with cross traffic and congestion. Emulation of faults offered an efficient way of collecting accurate data with minimal resources.

We emulated 16 common UD faults that can affect the network performance as listed in Table II. The faults included hardware platform problems and span multiple layers of the network stack. The 16 fault cases together with the "Healthy" case created the 17 classes used in this evaluation. Over 6 weeks period of time, we collected 12685 traces from UDs emulating these 17 cases. These traces contained approximately equal amounts of samples from healthy as well as the faulty classes and included UDs that ran 8 different flavors of TCP. Multiple flavors of TCP was used since in the real world networks, implementation of TCP changes between devices resulting in subtle differences in connection behavior [7]. We

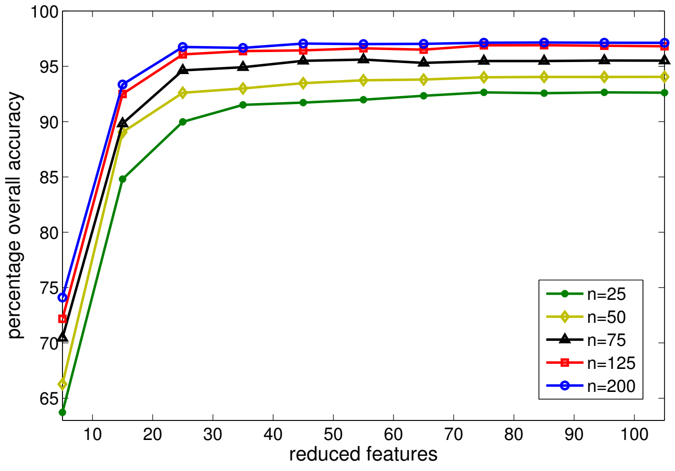


Fig. 5. Overall accuracy of the system against dimensionality (D) of the EigenNSS. Each graph represents a different per-class training dataset size.

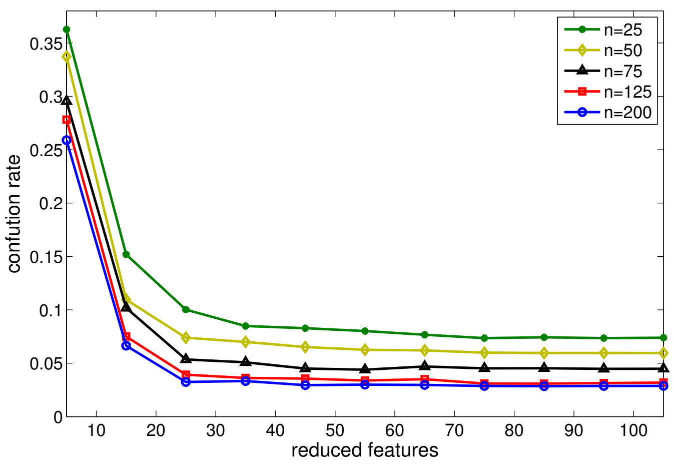


Fig. 6. Overall confusion rate of the diagnostic system against dimensionality (D) of the EigenNSS for different per-class training dataset size (n).

wanted to demonstrate the robustness of the system against such differences.

B. System performance

For the initial system training, we only chose 13 classes as our evaluation needed to consider the unknown faults. The faults CF2, CF8, CF10, CF17 were kept as the unknown faults (refer to Table II). The data set of the first 13 classes was randomly divided to training and testing groups. The unknown class data was sent through the system at random intervals during the testing stage. Also, the cluster threshold to add an unknown fault as a known fault was set as equal to the number of per-class training samples used to initiate the system (e.g. if system was initially trained with 50 samples class, an unknown class is added as a known fault when a cluster membership reaches 50). To achieve a statistically robust results, we conducted the experiment for 8 iterations and averaged the results.

The two threshold parameters λ_{nss} was set to 1.57×10^{-9} and λ_{fc} was set to 0.06. λ_{nss} value was chosen by analyzing the whole training dataset and λ_{fc} was chosen by an iterative process that maximized the correct class associations. These two parameters can be carried to a new system, given the network environment does not significantly differ from the one being used during the experiment.

The Figure 5 shows the overall accuracy of the system in identifying the testing samples which were previously unseen. Here, n represents the number of samples per class used for training and x-axis shows the dimensionality (D) of the EigenNSS. Similarly, Figure 6 shows the overall confusion rate of the system, which indicate the ratio of misclassified and total samples. From the Figures 5 and 6, we can see that the system achieved a high overall performance with 25 EigenNSS features for all n-values. Any additional features added only a marginal performance gain. Given that original NSS contained 410 features, these results show a successful dimensionality reduction of 93.9%. Also, the figures show that with increasing training sample size, the system performance improves. This indicated that since, the system has been designed to iteratively add and re-train the system with correctly classified samples, when deployed in a network, the overall accuracy of the system continues to improve with time. However, as the figures show, the performance gain diminishes with the increasing sample size. We managed to achieve a 97.1586% overall accuracy and 0.0284 confusion rate with 200 samples per class and 25 EIgenNSS features for the 17 class system.

The Table II summarizes performance of the 17 classes used in our EigenNSS-based multiclass classifier design. Matrices used in the table are as following:

(i) True-Positive Rate (TPR): Members of class X correctly classified as belonging to class X.
(ii) False-Positive Rate (FPR): Members of other classes incorrectly classified as belonging to class X.
(iii) True-Negative Rate (TNR): Members of other classes

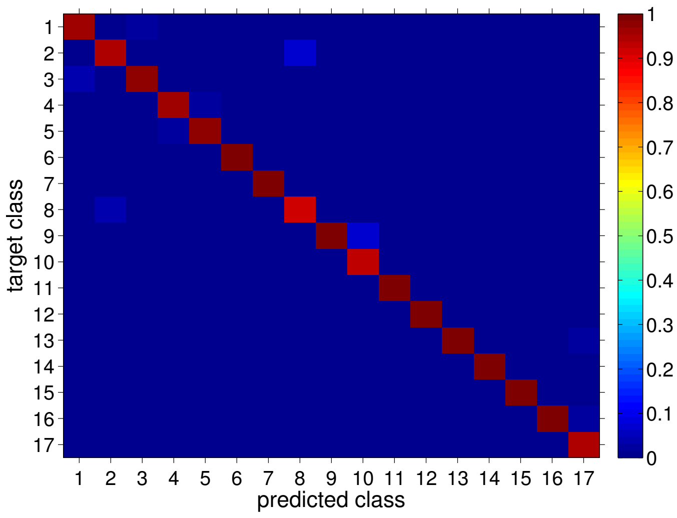


Fig. 7. Confusion matrix for the 17 classes in EigenNSS-based diagnostic system.

Fault	Description	TPR (%)	FPR (%)	TNR (%)	FNR (%)
CF1	Healthy	95.5202	4.4798	99.8254	0.1746
CF2	Disabled SACK error	94.1799	5.8201	99.4724	0.5276
CF3	Insufficient write buffer	97.6994	2.3006	99.5482	0.4518
CF4	Insufficient read buffer	96.5517	3.4483	99.7679	0.2321
CF5	Simultaneously insufficient read & write buffer	97.4203	2.5797	99.7913	0.2087
CF6	TCP timestamps are not working/in error	98.7952	1.2048	99.9536	0.0464
CF7	Window scaling error	98.9426	1.0574	99.942	0.058
CF8	Limited reordering threshold	91.2587	8.7413	99.7016	0.2984
CF9	Link-UD speed mismatch level 1	99.1886	0.8114	99.5678	0.4322
CF10	Link-UD speed mismatch level 1 & duplex mismatch	92.9204	7.0796	99.9771	0.0229
CF11	Link-UD speed mismatch level 2	99.0494	0.9506	99.9201	0.0799
CF12	Link-UD speed mismatch level 2 & duplex mismatch	98.4906	1.5094	99.9429	0.0571
CF13	UD firewall causing packet loss	100	0	99.8977	0.1023
CF14	UD firewall causing packet delay	100	0	100	0
CF15	Overloaded UD CPU	99.7403	0.2597	99.8652	0.1348
CF16	Overloaded UD memory	98.6111	1.3889	99.8532	0.1468
CF17	UD HDD faulty or i/o overloaded	94.198	5.802	99.9333	0.0667

TABLE II
PER-CLASS ESTIMATION PERFORMANCE OF THE EIGENNSS-BASED DIAGNOSTIC SYSTEM AT $D = 25$ AND $n = 200$.

correctly classified as not belonging to class X

(iv) False-Negative Rate (FNR): Members of class X incorrectly classified as not belonging to class X.

The table shows that all different types of faults can be uniquely identified independently with high-level of accuracy as suggested by high TPR, TNR. Also, low FPR, FNR rates indicate that the classifier has a low false detection rate for all the classes. The unknown classes, CF2, CF8, CF10, and CF17 have a slightly lower accuracy compared to other classes because the first observations of these classes are classified unknown prior to recognizing them as a new class. Even with this delay in creating the class, the previously unknown classes managed to achieve >90% TPR.

The Figure 7 shows the confusion matrix of the EigenNSS-based system. Confusion matrix is a typical representation used in evaluating multiclass classifiers where the rows represent target or expected (actual) class and the columns represent the predicted classes. The correct classifications lie on the diagonal of the matrix and all the other indices are misclassified instances. The ratio of instances in every element and column sum (total samples actually belongs to the class) is color coded. In real world diagnosis of UD faults, misclassifications could lead to false conclusions. However, the Figure 7 shows that our system was successful in avoiding large misclassifications satisfying a primary requirement of such a diagnostic solution.

The proposed EigenNSS-based Euclidean distance classifier system (EigenNSS-ED) was also tested against an Euclidian Distance classifier (NSS-ED) and a Naive Bayes multiclass classifier [23] (NSS-NB) that used the original full length NSS. The system was trained with all 17 classes at the beginning and used the same training and testing data. The Figure 8 shows the comparison of overall system accuracy with increasing number of training samples between EigenNSS-ED, NSS-ED

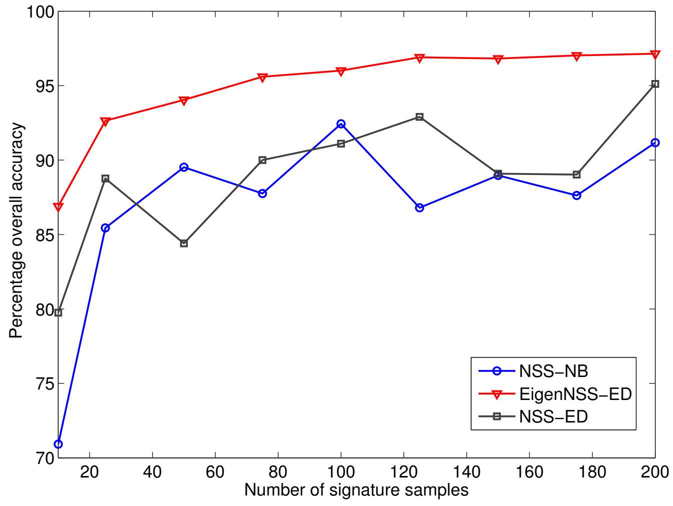


Fig. 8. Overall accuracy of the EigenNSS-based diagnostic system vs NSS-NB system.

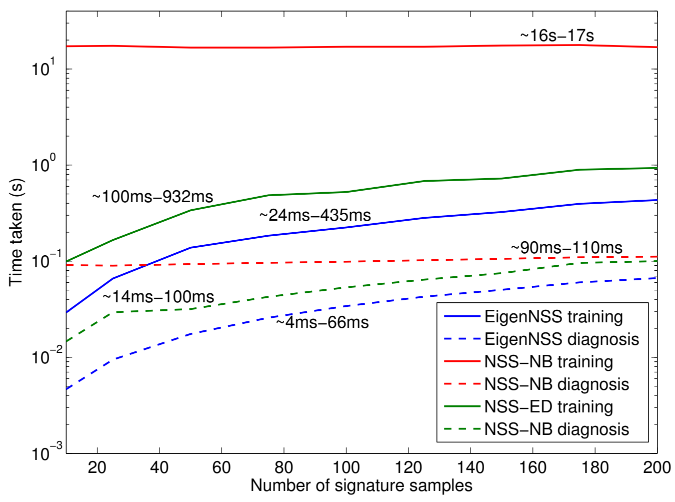


Fig. 9. Training and single sample diagnosis time variations against increasing training dataset size for EigenNSS-ED and NSS-NB classifiers.

and NSS-NB systems. With any given training dataset, the figure shows that EIgneNSS-ED system outperforms NSS-ED and NSS-NB systems. This mainly a consequence of classifier overfitting with the 410-feature NSS.

Time taken to train the system and evaluate a new sample (diagnosis) are important performance criteria, specially considering the iterative training process of EigenNSS-ED system. We tested and compared how both the total training time and single diagnosis time vary with training samples per class for all the classifiers. The test were carried out in a PC with Intel Core i7 4.4GHz CPU, 16GB 1600MHz RAM and using Matlab R2012b. The Figure 9 shows that NSS-NB and NSS-ED classifiers have longer training time compared to EigenNSS-ED classifier. Furthermore, the training time for EigenNSS-ED system was in the order of milliseconds (24 ms-435 ms when $n = 10 - 200$) which suggests iterative training does not impact the practical usability of the EigenNSS-ED system. The diagnosis time also shows that EigenNSS sample can be diagnosed faster compared to NSS, despite the fact that EigenNSS calculation involves more steps. Diagnosis times of the EigenNSS-ED (4 ms - 66 ms) indicate that, once trained, such a system will not be limited to on-demand diagnosis, but also can be used for real time diagnosis applications.

VIII. Conclusion

We have proposed and evaluated an automated UD soft-failure diagnostic system based on single multiclass classifier design. Combining supervised and unsupervised ML techniques, the system is capable of diagnosing known and previously unknown faults. We also presented a signature transformation technique to reduce the dimensionality of NSS while preserving important information and created EigenNSS. The EigenNSS reduces the complexity of the classifiers and avoids model overfitting to achieve a higher system accuracy.

The system was evaluated by diagnosing 17 UD faults collected over live campus network which showed overall accuracy of up to 97.2%. We also achieved a dimensionality reduction of 94% and low confusion between classes. To our knowledge, the proposed system is the first framework for automating the known and unknown UD soft-failure diagnosis. This work provides the foundation to extend the system to more complex network environments with thousands of users, diverse client platforms, and complex traffic patterns.

References

[1] S. Sundaresan, W. de Donato, and N. Feamster, "Broadband Internet Performance: A View From the Gateway," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 134–145, Aug. 2011.

[2] M. Thottan and C. Ji, "Anomaly Detection in IP Networks," *IEEE T. Signal. Proces.*, vol. 51, no. 8, pp. 2191–2204, 2003.

[3] R. Maxion and F. Feather, "A case study of Ethernet anomalies in a distributed computing environment," *IEEE Trans. Rel.*, vol. 39, no. 4, pp. 433 –443, Oct. 1990.

[4] ITU-T Rec. P.10/G. 100, "Vocabulary and effects of transmission parameters on customer opinion of transmission quality," International Telecommunications Union, Geneva, Switzerland, Tech. Rep., 2006.

[5] M. Mathis, J. Heffner, and R. Reddy, "Web100: extended tcp instrumentation for research, education and diagnosis," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 69–79, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.957002

[6] S. Shalunov and R. Carlson, "Detecting Duplex Mismatch on Ethernet," in *Proceedings of PAM 05*. Boston, MA: Springer-Verlag, Berlin, Oct. 2005, pp. 135–148.

[7] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Behavior Analysis of TCP Linux Variants," *Comput. Netw.*, vol. 56, no. 1, pp. 462–476, Jan. 2012.

[8] C. Widanapathirana, Y. Şekercioğlu, M. Ivanovich, P. Fitzpatrick, and J. Li, "Automated Inference System for End-To-End Diagnosis of Network Performance Issues in Client-Terminal Devices," *Int. Jour. of Comput. Netw. & Comm. (IJCNC)*, vol. 4, no. 3, pp. 37–56, 2012.

[9] C. Widanapathirana, J. Li, Y. Sekercioglu, M. Ivanovich, and P. Fitzpatrick, "Intelligent Automated Diagnosis of Client Device Bottlenecks in Private Clouds," in *Proc. IEEE UCC 11*. Melbourne, Australia: IEEE, Dec. 2011, pp. 261 –266.

[10] P. Sterlin, "Overfitting prevention with cross-validation," Master's thesis, University Pierre and Marie Curie (Paris VI), Paris, France, 2007.

[11] B. Zhang, J. Yang, J. Wu, and Z. Wang, "MBST: Detecting Packet-Level Traffic Anomalies by Feature Stability," Comp. J., Advance Access, published January 5, 2012, Oxford, UK, 2012.

[12] T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," *IEEE Commun. Surv. Tutor.*, vol. 10, pp. 56 –76, 2008.

[13] A. Koufakou and M. Georgiopoulos, "A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes," *Data Min. Knowl. Discov.*, vol. 20, no. 2, pp. 259–289, Mar. 2010.

[14] J. V. Gomes, P. R. Incio, M. Pereira, M. M. Freire, and P. P. Monteiro, "Exploring Behavioral Patterns Through Entropy in Multimedia Peer-to-Peer Traffic," *Comp. J.*, vol. 55, no. 6, pp. 740–755, 2012.

[15] J. Zhang and M. Zulkernine, "Anomaly based network intrusion detection with unsupervised outlier detection," in *IEEE International Conference on Communications, 2006. ICC '06.*, vol. 5, 2006, pp. 2388–2393.

[16] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks, 2005.*, 2005, pp. 250–257.

[17] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Information Sciences*, vol. 177, no. 18, pp. 3799–3821, 2007.

[18] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-supervised network traffic classification," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 369–370, Jun. 2007.

[19] S. L. Chiu, "Fuzzy model identification based on cluster estimation," *Journal of intelligent and Fuzzy systems*, vol. 2, no. 3, pp. 267–278, 1994.

[20] J. C. Bezdek, "Fuzzy mathematics in pattern classification," *PhD Dissertion, Applied mathematics center, Cornell University*, 1973.

[21] C. Widanapathirana, J. Li, M. Ivanovich, P. Fitzpatrick, and Y. Şekercioğlu, "Adaptive Signatures of Soft-Failures in End-User Devices Using Aggregated TCP Statistics," in *Proceedings of IEEE/IFIP IM 13*. Ghent, Belgium: IEEE, New York, May 2013.

[22] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, 1970.

[23] B. Cestnik, I. Kononenko, and I. Bratko, "Assistant 86: A knowledge elicitation tool for sophisticated users," *Progress in machine learning*, pp. 31–45, 1987.