Multivariate Online Anomaly Detection Using Kernel Recursive Least Squares

Tarem Ahmed and Mark Coates
Department of Electrical and Computer Engineering
McGill University
Montreal, QC, Canada
Email: tarem.ahmed@mail.mcgill.ca, coates@ece.mcgill.ca

Anukool Lakhina
Department of Computer Science
Boston University
Boston, MA, United States
Email: anukool@cs.bu.edu

***Abstract*— High-speed backbones are regularly affected by various kinds of network anomalies, ranging from malicious attacks to harmless large data transfers. Different types of anomalies affect the network in different ways, and it is difficult to know *a priori* how a potential anomaly will exhibit itself in traffic statistics. In this paper we describe an online, sequential, anomaly detection algorithm, that is suitable for use with multivariate data. The proposed algorithm is based on the kernel version of the recursive least squares algorithm. It assumes no model for network traffic or anomalies, and constructs and adapts a dictionary of features that approximately spans the subspace of normal behaviour. The algorithm raises an alarm immediately upon encountering a deviation from the norm. Through comparison with existing block-based offline methods based upon Principal Component Analysis, we demonstrate that our online algorithm is equally effective but has much faster time-to-detection and lower computational complexity. We also explore minimum volume set approaches in identifying the region of normality.**

I. Introduction

Network traffic is often seen to exhibit sudden deviations from normal behaviour. Some of these aberrations are caused by malicious network attacks such as Denial-Of-Service or viruses, whereas others are the result of equipment failures and accidental outages [1]. Network operators need to be able to diagnose anomalous behaviour in a timely manner, in order to facilitate a fast response and take precautions for the future. Most prior work in network anomaly detection has used block-based methods, which are only suitable for offline applications, requiring waits of up to hours before alerts occur [1]–[4]. We suggest an alternative approach and propose an online, recursive algorithm that detects anomalies in multivariate network-wide data within minutes.

Anomalies have historically been seen to span a wide range of types and classes, and each class may indicate its presence on raw statistics in a different manner [1], [2]. Developing widely-applicable definitions or models of normal network behaviour and anomalies is thus difficult. Our algorithm takes the alternative approach of learning the behaviour of normal traffic, and autonomously adapting to shifts in the structure of normality itself. We consider the absence of any parametric model to be a crucial feature. The disadvantage of a model is that it imposes limitations on the applicability of an algorithm, and even subtle changes in the nature of network traffic can render the model inappropriate. The choice of traffic measurement and the feature space has a strong impact on the performance of our algorithm, and determines what type of anomalies can be detected. However, we consider that this identification process is much more robust than the model specification task.

In this paper we develop a sequential, real-time anomaly detection algorithm that incrementally constructs and maintains a dictionary of input vectors which defines the region of normal behaviour. The dictionary adapts over time to address changes in the structure of normal traffic, with new elements being added obsolete members deleted as the normality region expands or migrates. We provide a comparative study on real data of our proposed Kernel-based Online Anomaly Detection (KOAD) algorithm and the block-based PCA detection algorithm described in [2]. The results indicate that the detection performance of the two are approximately equivalent, with the KOAD algorithm offering lower computational complexity and faster time-to-detection.

A. Related Work

Our work builds most closely on the series of works by Lakhina et al. in [1], [2], [5]. They demonstrate the intrinsic low-dimensionality of network flows, and the high spatial and temporal covariance structure between the flows [5]. Lakhina et al. used the technique of Principal Component Analysis (PCA) to separate the space occupied by a set of traffic metrics into two disjoint subspaces, corresponding to normal and anomalous behaviour, respectively [1], [2]. They signal an anomaly when the magnitude of the projection onto the residual, anomalous subspace exceeds an associated PCA Q-statistic threshold [6]. The PCA subspace method was shown to be more effective than EWMA and Fourier approaches in automatic diagnosis of anomalies [2].

Lakhina et al. also suggested an online formulation of the PCA-based algorithm in [5]. This involved using a sliding window implementation to identify the normal and anomalous subspaces based on a previous block of time. The variation in the structure of multivariate network traffic statistics over time is, however, non-negligible. Further, the PCA-based detection algorithm is extremely sensitive to the proper determination of the associated Q-statistic threshold. We implemented the pro-

posed online version of PCA and observed that although the anomalous and normal subspaces remained relevant over time, using stale measurements to calculate the Q-statistic threshold resulted in an unacceptable number of false positives. This indicates that straightforward extensions to the PCA-based method are not robust and motivates alternative approaches for an online application.

Much of the other previous work on online network anomaly detection has been based on network traffic models [3], [7]. Brutlag uses as an extension of the Holt-Winters forecasting algorithm, which supports incremental model updating via exponential smoothing [3]. His algorithm defines a "violation" as an observation that falls outside an interval (a confidence band), and identifies a "failure" (an anomaly) when the number of violations within an observation window exceeds a threshold. Hajji uses a Gaussian mixture model, and develops an algorithm based on a stochastic approximation of the Expectation-Maximization (EM) algorithm to obtain estimates of the model parameters [7].

A rare example of a real-time network anomaly detection method that is not based on an *a priori* model, is the time-based inductive learning machine (TIM) of Teng et al. [8]. Their machine constructs a set of rules based upon usage patterns. The detection algorithm detects a deviation when the premise of a rule occurs but the conclusion does not follow. Applying machine learning approaches to network anomaly detection is a recent phenomena. Examples include the use of statistical learning techniques to detect email worms and viruses by Martin et al. [9], and an algorithm based on Kernel PCA proposed by Heafield [10]. The learning algorithm presented in [8] is computationally intensive and the paper has subsequently had more influence in the field of intrusion detection [11], [12].

B. Outline of Paper

This paper is organized as follows. Section II reviews the concepts of kernel machines and minimum volume sets. Section III presents the KOAD algorithm, analyses computational complexity, and discusses the choice of the algorithm parameters. Section IV compares the performance of our algorithm with the block-based PCA approach, on data recorded on the Abilene backbone network. Section V provides concluding remarks and describes avenues for future research.

II. Background

A. Kernel Recursive Least Squares

Kernel machines use a kernel mapping function to produce non-linear and non-parametric learning algorithms [13]. The idea is that a suitable kernel function, when applied to a pair of input data vectors, may be interpreted as an inner product in a high-dimensional Hilbert space known as the feature space [14]. This allows inner products in the feature space (inner products of the *feature vectors*) to be computed without explicit knowledge of the feature vectors themselves, by simply evaluating the kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \tag{1}$$

where $\mathbf{x}_i, \mathbf{x}_j$ denote the input vectors and ϕ represents the mapping onto the feature space.

Popular kernel functions include the Gaussian kernel with variance σ^2: $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}\}$, and the polynomial kernel of degree p: $k(\mathbf{x}_1, \mathbf{x}_2) = (a\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + b)^p$ [13]. A special case of the polynomial kernel is the linear kernel:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle. \tag{2}$$

The Kernel Recursive Least Squares (KRLS) algorithm combines the principles of kernel machines and the popular Recursive Least Squares (RLS) algorithm [15], and provides an efficient and non-parametric approach for performing online data mining. The KRLS algorithm operates on a data sequence of the form $\mathcal{Z}_t = \{\mathbf{x}_i, y_i\}_{i=1}^t$, where the input-output pairs $(\mathbf{x}_i, y_i)$ are assumed to be independent, identically distributed samples from some distribution $p(Y, \mathbf{X})$. The objective is to obtain the best predictor $\hat{y}_t$ of y_t, given $\mathcal{Z}_{t-1} \bigcup \{\mathbf{x}_t\}$.

In conventional Recursive Least Squares, the dimension of the space spanned by the input samples $\{\mathbf{x}_i\}_{i=1}^t$ is constrained by the dimension of the input space. In contrast, Kernel Recursive Least Squares involves a mapping onto a feature space of much higher dimensionality than the input space, and the dimension of the space spanned by $\{\phi(\mathbf{x}_i)\}_{i=1}^t$ has the potential to increase without bound. At each timestep, the dimension will increase unless $\mathbf{x}_t$ satisfies $\phi(\mathbf{x}_t) = \sum_{i=1}^{t-1} a_i \cdot \phi(\mathbf{x}_i)$. If the dimension increases, then the new vector is providing new information and adding to the predictive power, and so should be included in the predictor. This leads to the dilemma that the predictor may require the storage of a large number of input vectors, leading to unreasonable memory and computational requirements.

In defining KRLS, Engel et al. address this problem by imposing a minimum threshold on the amount of new information an input vector must provide in order to be added to the predictor [14]. Feature vector $\phi(\mathbf{x}_t)$ is said to be *approximately* linearly dependent on $\{\phi(\mathbf{x}_i)\}_{i=1}^{t-1}$, with approximation threshold ν, if the projection error δ_t satisfies the following criterion:

$$\delta_t = \min_a \left\| \sum_{i=1}^{t-1} a_i \cdot \phi(\mathbf{x}_i) - \phi(\mathbf{x}_t) \right\|^2 < \nu. \tag{3}$$

KRLS uses (3) to obtain a *dictionary* of input vectors $\mathcal{D} = \{\tilde{\mathbf{x}}_j\}_{j=1}^m$, where $m < t$, such that $\phi(\tilde{\mathbf{x}}) = \{\phi(\tilde{\mathbf{x}}_j)\}_{j=1}^m$, approximately spans the feature space. The best predictor $\hat{y}$ of y in the feature space of the sparse set, $\phi(\tilde{\mathbf{x}})$, can then be evaluated:

$$\hat{y} = \sum_{j=1}^{m} \alpha_j \cdot \langle \phi(\tilde{\mathbf{x}}_j), \phi(\mathbf{x_t}) \rangle = \sum_{j=1}^{m} \alpha_j \cdot k(\tilde{\mathbf{x}}_j, \mathbf{x_t}) \tag{4}$$

The weights $\{\alpha_j\}_{j=1}^m$ are learned by KRLS over time through successive minimization of prediction errors in the least-squares sense.

B. Minimum Volume Sets

We expect the *region of normality* to correspond to a high-density region of the space. That is, it should contain the vast majority of the encountered measurement vectors. It is thus natural to compare the outcome of our anomaly detection algorithm to other approaches for determining high-density regions. One common approach is the estimation of minimum volume sets (MVSs). Given data drawn from some underlying probability distribution, the MVS estimation problem is to find the minimum volume subset $\mathcal{S}$ of the input space, such that the probability that a test point drawn from the distribution lies outside $\mathcal{S}$ equals a pre-specified value μ [13]. These sets are known in the MVS literature as density contour clusters.

Muñoz and Moguerza propose the One-Class Neighbour Machine (OCNM) algorithm for estimating minimum volume sets [16]. The OCNM algorithm is a block-based procedure that provides a binary decision function indicating whether $\mathbf{x}_t$ is a member of the MVS or not. The algorithm requires the choice of a *sparsity measure*, which relaxes the density estimation problem by replacing the task of estimating the density function at each data point by a simpler measure that asymptotically preserves the order induced by the density function. We have implemented the OCNM algorithm for comparative purposes using the n-th nearest neighbour distance as the sparsity measure.

III. The Kernel-based Online Anomaly Detection (KOAD) Algorithm

Traffic flows in backbone networks have low intrinsic dimensionality, and demonstrate strong spatial and temporal covariance [5]. Consider a set of multivariate "normal" traffic measurements $\{\mathbf{x}_t\}_{t=1}^{T}$. In an appropriately chosen feature space $\mathcal{F}$, with an associated kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, the features corresponding to the normal traffic measurements should *cluster*. That is, given the inherent low dimensionality of network traffic, it should be possible to describe the region occupied by the traffic features using a relatively small dictionary of linearly independent elements $\{\phi(\tilde{\mathbf{x}}_j)\}$. We can construct an approximately equivalent description by learning a dictionary of elements from the arriving input measurements $\{\mathbf{x}_t\}$, and the size of the dictionary (m) will be much less than T, leading to computational and storage savings.

The set of measurements contains not only "normal" traffic, but also anomalous measurement vectors. We therefore require a procedure for identifying when a measurement vector is anomalous and excluding it from the dictionary. This procedure is based on the intuition that an anomaly should be distant in the feature space from the cluster of normal traffic. Once a suitable dictionary has been trained to capture normal traffic, the projection error δ_t from (3) offers a very natural way of assessing this distance.

A. The Algorithm

Algorithm 1 provides a high-level overview of the KOAD algorithm; a more complete description is presented in the Appendix. The KOAD algorithm operates at each timestep t on a traffic measurement vector $\mathbf{x}_t$. In accordance with the standard KRLS algorithm [14], we also define a scalar y_t associated with the measurement vector $\mathbf{x}_t$. Although KOAD does not use y_t in its present version, we intend to explore its utility in discovering additional anomalies in our future work. An example choice for $\mathbf{x}_t$ is the *flow vector* (the number of packets in each source-destination flow, normalized to the unit hypersphere), and for y_t the total number of packets in the network, as recorded during the measurement interval corresponding to timestep t.

The algorithm begins by evaluating the error δ_t in projecting the arriving $\mathbf{x}_t$ onto the current dictionary (in the feature domain). This error measure δ_t is then compared with two thresholds ν_1 and ν_2, where $\nu_1 < \nu_2$. If $\delta_t < \nu_1$, we infer that $\mathbf{x}_t$ is sufficiently linearly dependent on the dictionary, and represents normal traffic. If $\delta_t > \nu_2$, we conclude that $\mathbf{x}_t$ is far away from the realm of normal behaviour, and immediately

Algorithm 1: Outline of Kernel-based Online Anomaly Detection (KOAD) algorithm

```
Set thresholds: ν1, ν2 ;
for t = 1, 2, ... do
    Data: (x_t, y_t)
    /* Evaluate current measurement */
    Compute projection error δ_t for x_t using D_t ;
    if δ_t > ν2 then
        Raise Red1 Alarm ;
    endif
    if δ_t > ν1 then
        Raise Orange Alarm ;
        Store x_t in Θ ;
    endif
    /* Process previous orange alarm */
    if Orange Alarm(x_{t-ℓ}) then
        Re-evaluate projection error δ for x_{t-ℓ} using D_t
        if δ > ν1 then
            Evaluate usefulness of x_{t-ℓ} over previous ℓ
            measurements ;
            if NOT useful then
                Raise Red2 Alarm(x_{t-ℓ}) ;
            else
                Add x_{t-ℓ} to dictionary D ;
                Lower Orange Alarm(x_{t-ℓ}) ;
            endif
        else
            Lower Orange Alarm(x_{t-ℓ}) ;
        endif
        Remove Θ{1} ;
    endif
    /* Remove obsolete elements */
    Evaluate usefulness of each dictionary element over
    previous L measurements;
    Remove any useless element from dictionary D;
endfor
```

raise a "Red1" alarm to signal an anomaly.

If $\delta_t > \nu_1$, we infer that $\mathbf{x}_t$ is sufficiently linearly independent from the dictionary to be considered an unusual event. It may indeed be an anomaly, or it may represent an expansion or migration of the space of normality. We raise "Orange" alarm, store the relevant input vector in data structure Θ to keep track of its contribution over the next ℓ timesteps, and then resolve the orange alarm.

At timestep $t + \ell$, we re-evaluate the error δ in projecting $\mathbf{x}_t$ onto dictionary $\mathcal{D}_{t+\ell}$. Note that the dictionary may have changed between timesteps t and $t + \ell$, and the value of δ at this re-evaluation may consequently be different from the δ_t at timestep t. If the value of δ after the re-evaluation is found to be less than ν_1, we lower the orange alarm and keep the dictionary unchanged.

If the value of δ is found instead to be greater than ν_1 after the re-evaluation at timestep $t + \ell$, we perform a secondary "usefulness" test to resolve the orange alarm. The usefulness of $\mathbf{x}_t$ is assessed by observing the kernel values of $\mathbf{x}_t$ with $\mathbf{x}_i$, $i = t + 1, \ldots, t + \ell$. If a kernel value is high (greater than a threshold d), then $\phi(\mathbf{x}_t)$ is deemed close to $\phi(\mathbf{x}_i)$. If a significant *number* of the kernel values are high, then $\mathbf{x}_t$ cannot be considered anomalous; normal traffic has just migrated into a new portion of the feature space and $\mathbf{x}_t$ should be entered into the dictionary. Contrarily if almost all kernel values are small, then $\mathbf{x}_t$ is a reasonably isolated event, and should be heralded as an anomaly. We evaluate:

$$\left[\sum_{i=t+1}^{t+\ell} \mathbb{I}(k(\mathbf{x}_t, \mathbf{x}_i) > d) \right] > \epsilon\ell, \tag{5}$$

where $\mathbb{I}$ is the indicator function and $\epsilon \in (0, 1)$ is a selected constant. If (5) evaluates true, then we lower the relevant orange alarm to green (no anomaly) and add $\mathbf{x}_t$ to the dictionary. If (5) evaluates false, we elevate the relevant orange alarm to a "Red2" alarm. Once an orange alarm is resolved, we remove the stored input vector from Θ.

Removal of elements from the dictionary occurs when a dictionary element is declared obsolete. This event occurs after a test similar to (5) above. We periodically perform the same check, but replace ℓ by L, a much larger number. In addition, we replace ϵ by 0 and turn the inequality into an equality comparison. It is important to keep ℓ relatively small, because it determines the time lag before an orange alarm is resolved. On the other hand, we do not wish to declare an element obsolete if a short time period occurs where no traffic lies in its vicinity. Therefore L should be relatively large, allowing for short periods of uselessness to be ignored. The KOAD algorithm also incorporates exponential forgetting, so that the impact of past observations is gradually reduced.

B. Complexity Analysis

Storage and complexity issues are paramount to online applications. In terms of storage requirements, the maximum dimensions of the variables that we have to store are $m \times m$, where m represents the size of the dictionary. We also store the input vectors that raise orange alarms for ℓ timesteps, and an additional binary $L \times m$ matrix. Our experiments have shown that high sparsity levels are achieved in practice, and the dictionary size does not grow indefinitely.

The computational bottlenecks in the KOAD algorithm are the matrix multiplications. When no element is dropped, there is a constant number of multiplications of an $m \times m$ matrix with an $m \times 1$ column vector. In the rare case that an element is removed from the dictionary, the algorithm must perform a multiplication of two $m \times m$ matrices. The complexity of the algorithm is thus $\mathrm{O}(m^2)$ for every standard timestep and $\mathrm{O}(m^3)$ for timesteps when element removal occurs.

The complexity using PCA over a block of data of length t is $\mathrm{O}(tR^2)$, where R is the number of principal components [2]. The key point to note here is that the complexity of PCA is a function of time, whereas the KOAD complexity is not.

C. Parameter Selection

The KOAD algorithm requires the setting of a number of constant parameters. When the algorithm commences, the dictionary has not been formed so there is no definition of normality. For this reason, every vector should be considered for addition to the dictionary, i.e., there should be no Red1 alarms. During this initial training period (we use 300 training samples in the experiments), the value of ν_2 is set to 1.

During normal operation, the parameter that has the most direct effect on the detection performance is the threshold ν_1. Threshold ν_2 determines the instant flagging of an anomaly (the Red1 alarms). Whenever $\delta_t > \nu_1$, the algorithm signals an orange alarm which it processes ℓ timesteps later.

Our experiments have shown that optimal settings for ν_1 and ν_2 vary for different traffic metrics (such as number of packets, number of bytes, number of flows, or entropies of destination IP addresses). However, for the same metric, the performance of a setting remains approximately the same across widely-separated time periods. In Section IV, we analyze the performance variation obtained by different choices. Currently, we do not offer an automatic approach for setting the thresholds — this is an area of future research. Instead, we recommend the procedure of running the algorithm over a training set of data with known anomalies and then setting the values to achieve an acceptable compromise between detection and false alarm rates.

Experiments indicate that the algorithm performance is not particularly sensitive to the choice of ℓ, d, ϵ or L. The choice of ℓ governs a compromise between the lag-time to anomaly declaration and the false alarm rate. From a practical point of view, as statistics are often exported by network monitoring devices every five minutes, a value of $\ell = 20$ evaluates to under two hours. The 20 input vectors usually provide more than enough data to assess the usefulness of a potential dictionary element, and the time-to-detection is still faster than using block-based methods. Indeed, our experiments have shown that for almost all *bona fide* anomalies that were initially identified as orange alarms, the kernel value drops immediately, similar to the example of Fig. 2(c). The

parameter L exerts a similar influence to ℓ, but determines when obsolete vectors are removed from the dictionary. In this case, there is not such a pressing motive to keep L small, since it is not critical to remove obsolete elements immediately. We use $L = 100$ in the results presented, but any value in the range $40 - 200$ resulted in similar performance in our experiments.

The choice of d determines how close the dictionary element must be to an input vector before it is considered useful, and therefore defines a region of usefulness in the feature space. The appropriate value is dependent on the kernel being used (the kernel implicitly defines a distance measure), and should lie below the long-term average kernel value of any genuine dictionary element. The choice can be made based on an inspection of kernel values (as depicted in Fig. 2). The value of ϵ determines what fraction of input vectors must lie within the defined region of usefulness.

IV. Experiments

A. Data

To evaluate our algorithm, we examined performance on network-wide traffic datasets analyzed by Lakhina et al. in [2]. This data was collected from 11 core routers in the Abilene backbone network for a week (Dec. 15 to Dec. 21, 2003). It comprises two multivariate timeseries, one being the number of packets and the other the number of individual IP flows in each of the Abilene backbone flows (the traffic entering at one core router and exiting at another), binned at five minute intervals. Both datasets, $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$, are of dimension $F \times T$, where $T = 2016$ is the number of timesteps and $F = 121$ is the number of backbone flows.

We manually identified the anomalies present in our datasets. Thus we have "ground truth" anomaly annotations against which to compare the output of our KOAD detection algorithm. We were able to manually identify 34 anomalies in the packet-counts timeseries and 44 anomalies in the IP flow-counts timeseries.

B. Results

In our experiments, we set $\mathbf{x}_t$ to be the *flow vector* at timestep t normalized to the unit hypersphere, and as the associated y_t the total amount of traffic in the network:

$$\mathbf{x}_t = \frac{\mathbf{X}(1:F,t)}{\|\mathbf{X}(1:F,t)\|} \quad , \quad y_t = \sum_{f=1}^{F} \mathbf{X}(f,t).$$

Here $i = 1, 2$ indicates whether the number of packets or IP flows is being measured. This choice exploits clustering due to spatial correlations in network traffic [5].

We ran our KOAD algorithm for various combinations of the thresholds ν_1 and ν_2. For the results presented in this paper, the default settings for the dropping parameters were $d = 0.9$ and $L = 100$, the tolerance for resolving orange alarms were $\ell = 20$ and $\epsilon = 0.20$, and pertain to the no-forgetting ($\gamma = 1$) case. We used a linear kernel, as defined in (2). We implemented the OCNM algorithm using the n-th nearest neighbour distance as the sparsity measure with n set to 50 and $\mu = 0.98$ to select the 98% minimum volume set.

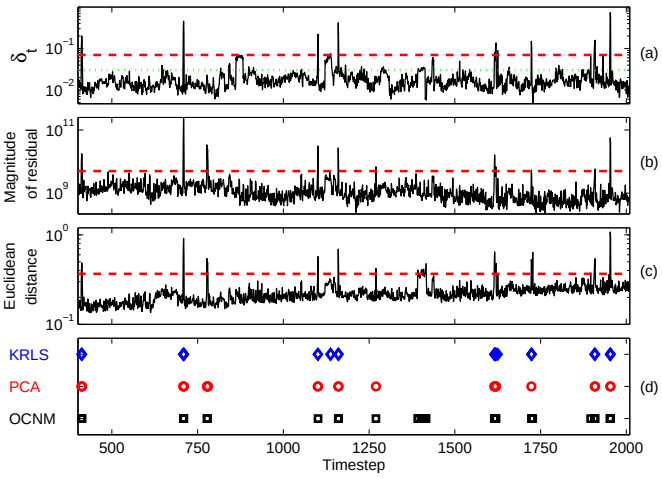


Fig. 1. (a) KOAD projection error δ_t, with dotted line indicating ν_1 and dashed line indicating ν_2; (b) magnitude of PCA projection onto residual subspace, with dashed line indicating associated Q-statistic threshold; (c) OCNM using n-th nearest-neighbour distance, with dashed line indicating anomaly distance threshold; and (d) positions at which anomalies are detected by (◇) KOAD, (o) PCA, and (□) OCNM, all as functions of time. Dataset: Abilene packet-counts timeseries.

For the dataset comprising the Abilene packet-counts timeseries, Figs. 1(a) and 1(b) show the variations in δ_t obtained using KOAD with $\nu_1 = 0.03$ and $\nu_2 = 0.07$, and the magnitude of the energy in the residual components from PCA using $R = 4$ principal components, respectively. For PCA-based anomaly detection, we use the Q-statistic threshold [6] as in [2]. Fig. 1(c) shows the distance measures obtained using the OCNM algorithm, together with the threshold indicating the 98% minimum volume set. The spike positions in Figs. 1(a-c) indicate the anomalies signalled by KOAD, PCA and OCNM. Fig. 1(d) compares the positions of the detected anomalies, indicating that, for the most part, the three methods signal the same events.

Using these settings, OCNM flags 26 of the 34 anomalies in the packet timeseries. The anomalous input vectors thus also exhibit high Euclidean n-th neighbour distances in the input space. Recall that KOAD deals with distance in the feature space. One noticeable difference in the results of the three different algorithms occurs around $t = 1400$. Here OCNM detects a series of anomalies, PCA indicates nothing abnormal, and the KOAD δ_t is initially high for a block of time and then suddenly dips. This is indicative of the subtle differences in the three approaches. The block of 25 input vectors forms a small cluster. This cluster is sufficiently numerous and energetic for PCA to dedicate a principal component to it, so PCA does not consider it an anomaly. KOAD signals the first vector in the cluster as an orange alarm, tracks it for $\ell = 20$ subsequent timesteps during which δ_t remains high, then because there are sufficient similar vectors immediately thereafter the usefulness test is passed and the orange alarm is rendered green. The remaining 5 vectors in the 25-vector cluster are very similar to this first vector, which results in the sudden dip in δ_t. In contrast, the n-th nearest-neighbour distance for every vector in the cluster is large (n is larger than 25). It is interesting to note that if we reduce n to 25 from 50, OCNM declares none

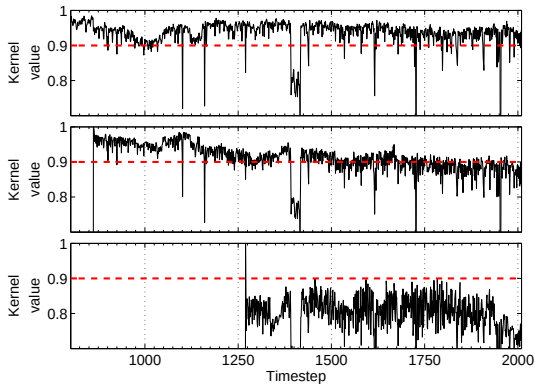


Fig. 2. Behaviour over time of $k(\tilde{\mathbf{x}}_j, \mathbf{x}_t)$, when $\tilde{\mathbf{x}}_j$ is (top) a normal $\mathcal{D}$ member, (middle) a $\mathcal{D}$ member that eventually becomes obsolete, and (bottom) an anomalous $\mathcal{D}$ member. Dashed line in each figure indicates usefulness threshold d. Dataset: Abilene packet-counts timeseries.

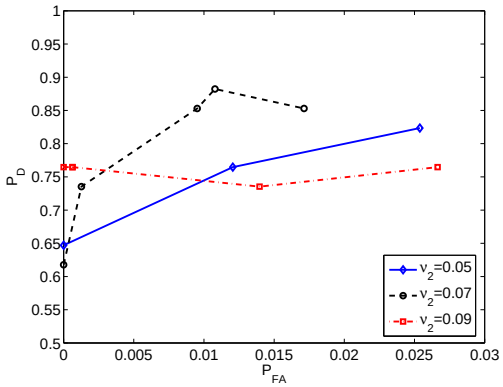


Fig. 3. Receiver Operating Characteristics (ROC) curves showing the trade-off between the probability of detection (P_D) and the probability of false alarms (P_{FA}). Dataset: Abilene packet-counts timeseries.

of these timesteps as anomalous.

Fig. 2 depicts the changing behaviour over time of the kernel values $k(\tilde{\mathbf{x}}_j, \mathbf{x}_t)$ for three different types of dictionary members $\tilde{\mathbf{x}}_j$. Fig. 2(a) shows a consistently useful dictionary element. Many input vectors are close to this element, resulting in high kernel values. Sudden drops below the threshold d correspond primarily to anomalies. Fig. 2(b) shows the behaviour of a dictionary element that gradually becomes obsolete. This figure motivates the need to eventually discard such dictionary members. Fig. 2(c) illustrates the case of a Red2 alarm. The kernel values drop significantly below the threshold *immediately* after this input vector arrives and produces a $\delta_t > \nu_1$. The algorithm signals an orange alarm in this case, tracks it for ℓ timesteps, upon which it performs the usefulness test and elevates the orange alarm to a Red2 alarm. This input vector is never entered into the dictionary.

C. Detection Performance

Our objective is to detect the 34 anomalies in the packet-counts timeseries, and the 44 anomalies in the flow-counts timeseries. Fig. 3 presents the trade-off between the probability of detection (P_D) and the probability of false alarms (P_{FA}) as Receiver Operating Characteristics (ROC) curves for the packet-counts timeseries. Curves are presented for various settings of ν_2, and each point corresponds to a choice of ν_1. Fig. 3 shows that the detection rate does not monotonically increase as ν_1 decreases. This is because ν_1 does not solely act as a detection threshold, but also determines, in a non-linear fashion, the size and nature of the dictionary and hence the identified region of normality. The ν_2 setting determines the (instant) declaration (correctly or falsely) of Red1 alarms.

Table I provides a breakdown of detection performance for the packet-counts dataset and Table II presents the same information for the flow-counts dataset. We see that KOAD provides comparable detection rates, with the added advantage of lower lag and complexity.

TABLE I

DETECTION OF THE 34 ANOMALIES IN THE ABILENE PACKET-COUNTS DATASET WITH KOAD USING VARIOUS REPRESENTATIVE SETTINGS OF ν_1 AND ν_2, COMPARED WITH PCA AND OCNM.

ν_1 setting	ν_2 setting	Detected	Missed	False
$\nu_1 = 0.01$	$\nu_2 = 0.05$	22	12	0
$\nu_1 = 0.02$	$\nu_2 = 0.05$	26	8	19
$\nu_1 = 0.02$	$\nu_2 = 0.07$	21	13	0
$\nu_1 = 0.05$	$\nu_2 = 0.07$	30	4	17
$\nu_1 = 0.04$	$\nu_2 = 0.09$	26	8	1
PCA with $R = 4$		25	9	0
OCNM with $n = 50$		26	8	14

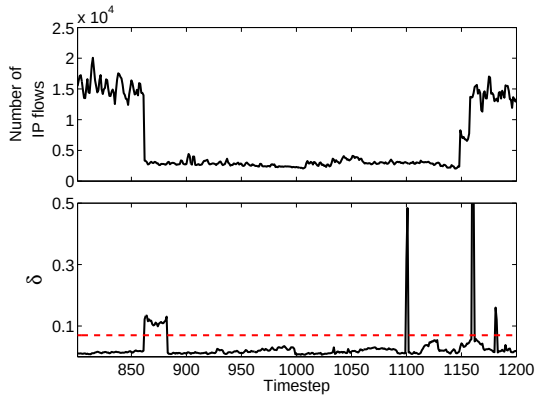
D. Anomaly Analysis

We now examine two types of anomalies in more detail to highlight the differences between the block-based approach of PCA and the sequential, learning approach of KOAD. In our first example, we consider a 240-timesteps (20-hour) period during which approximately 20 backbone flows experience step-changes in behaviour and the nature of the network traffic changes fundamentally. This is most evident in the flow-counts dataset. The top panel of Fig. 4(a) depicts, as an example, the variation in the number of IP flows over time in the Seattle-Chicago backbone flow. When the first input vector corresponding to this shift arrives, KOAD signals a Red1 alarm (as $\delta_t > \nu_2$), and keeps tracking it for ℓ timesteps, during which all input vectors produce significantly high δ_t values and are all signalled as Red1 alarms (Fig. 4(a), bottom panel). Once the first input vector corresponding to this shift is entered into the dictionary, it explains all subsequent input vectors that belong to this temporary shift, and δ_t becomes low. The spikes

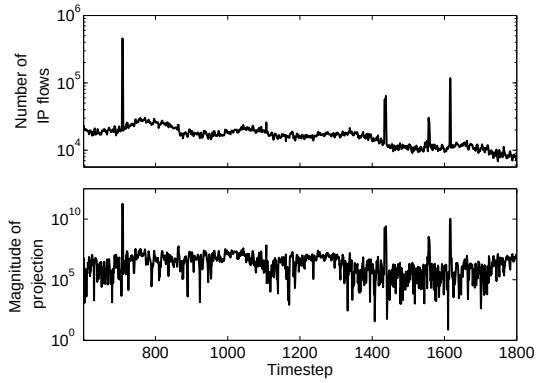
TABLE II

DETECTION OF THE 44 ANOMALIES IN THE ABILENE FLOW-COUNTS DATASET WITH KOAD USING VARIOUS REPRESENTATIVE SETTINGS OF ν_1 AND ν_2, COMPARED WITH PCA AND OCNM.

ν_1 setting	ν_2 setting	Detected	Missed	False
$\nu_1 = 0.03$	$\nu_2 = 0.07$	37	7	21
$\nu_1 = 0.01$	$\nu_2 = 0.08$	28	16	5
$\nu_1 = 0.04$	$\nu_2 = 0.08$	39	5	29
$\nu_1 = 0.04$	$\nu_2 = 0.09$	38	6	25
PCA with $R = 4$		22	22	0
OCNM with $n = 50$		37	7	3



(a) Top panel: Variation in number of IP flows in Abilene Seattle-Chicago backbone flow over time. Bottom panel: Variation in the KOAD δ_t during same period for $\nu_1 = 0.03$ and $\nu_2 = 0.07$ (dashed line).



(b) Top panel: Variation in number of IP flows in Abilene New York-Chicago backbone flow over time. Bottom panel: Projection of $\mathbf{x}_t$ onto the second principal component over the same interval.

Fig. 4. Example anomalies in the Abilene flow-counts dataset. (a) The structure of normal traffic shifts dramatically for approximately 240 timesteps. KOAD flags the first $\ell = 20$ arrivals as Red1 alarms. (b) Three anomalies on the New York-Chicago backbone flow that are not detected by PCA.

at $t = 1100$ and 1160 represent different, definite anomalies. PCA dedicates one principal component to this block of time and does not indicate any anomalous behaviour.

In our second example, we examine the New York-Chicago backbone flow. Here there are three instances at around $t = 710$, 1434 and 1615 persisting for $3-5$ timesteps each, where the number of IP flows increases dramatically (Fig. 4(b), top panel). The PCA-based algorithm clusters these anomalies together and dedicates a principal component to describing them (Fig. 4(b), bottom panel). The anomalies consequently go undetected. The KOAD algorithm in contrast adapts over time and detects all three anomalies.

V. Discussion and Future Work

We have described a kernel-based online anomaly detection algorithm that is able to detect anomalous events in real-time. Through analysis of datasets recorded on the Abilene network, we have demonstrated that the proposed algorithm achieves similar detection performance to the most effective block-based approaches, but has a faster time-to-detection and lower computational complexity. Anomalies are either flagged within the same timestep of its occurrence, or after the orange alarm is resolved a short interval later. In both cases, the time-to-detection is faster than with block-based methods.

The nature of our results raises interesting questions about what should be reported as an anomaly. For example, the 25-vector input cluster around $t = 1400$ (Section IV-B) can be viewed as a large time-scale anomaly, in that the traffic here is very different from anything seen during the rest of the week. On the other hand, it persists for two hours, rendering debatable its labelling as anomalous. In the flow-counts dataset, we observe a 20-hour period during which traffic is fundamentally different from the rest of the week. The algorithm presented in this paper flags the first ℓ measurements as red alarms, then incorporates this shift into its designation of the space of normality.

The experimental results presented in this paper pertain only to volume anomalies, which are evident in packet-counts and flow-counts timeseries. The KOAD algorithm is also able to detect anomalies in distributions (such as arising from DoS attacks) which are evident in timeseries of header entropies [1], and in a variety of other applications such as camera networks.

We do not currently present an automatic procedure for setting the thresholds ν_1 and ν_2. Our experiments indicate that optimal settings for them vary for different traffic metrics, as evident from Tables I and II, but remain similar over time for the same metric. Both supervised and unsupervised learning approaches can be adopted to train the parameters. In the supervised setting, the algorithm can be run repeatedly over a set of training data with adjustments to the thresholds until the detection rate is maximized for a specified maximum false alarm rate. In the unsupervised setting, a minimum volume set approach may be incorporated to provide pseudo-labels for the data. Our future research involves making ν_1 and ν_2 adaptive, and designing supplementary algorithms for autonomously setting the other KOAD algorithm parameters.

It is interesting to consider how combinations of the three approaches might perform. For example, PCA analysis could be performed on an initial block of training data and the most significant components used as the initial KOAD dictionary members. Alternatively, the OCNM algorithm could be applied to provide an initial definition of the region occupied by normal traffic. Further study is required to develop a better understanding of the relationships between minimum volume sets and the KOAD dictionary and thresholds.

Appendix: Algorithmic Details

Here we present a more detailed description of the KOAD algorithm. Matlab code implementing the algorithm, our datasets and instructions on replicating our experiments, are available at [17]. Algorithm 2 presents pseudocode.

Notation and Definitions: We use subscripts with variables enclosed in square brackets to denote a subset of the rows and columns of a matrix. Thus $[\Lambda]_{2:5,1:5}$ refers to the second to fifth rows, and first to fifth columns, of matrix Λ. $\tilde{\mathbf{K}}_t$ represents the $m_t \times m_t$ kernel matrix for the elements $\{(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_t}$ that are in the dictionary at time t. Thus the matrix entry $[\tilde{\mathbf{K}}_t]_{i,j}$

Algorithm 2: Kernel-based Online Anomaly Detection

1 Set thresholds: ν_1, ν_2 ;
2 Choose γ, d, L, l, ϵ ;
3 Initialize: $t = 1$, $\mathcal{D} = \{\mathbf{x}_1\}$, $m_1 = 1$, $\tilde{\mathbf{K}}_1 = [k_{11}]$, $\tilde{\mathbf{K}}_1^{-1} = [\frac{1}{k_{11}}]$, $\tilde{\alpha}_\mathbf{1} = \frac{y_1}{k_{11}}$, $\mathbf{P}_1 = [1]$, $\Lambda = [1]$;
4 **for** $t = 2, 3, \ldots$ **do**
  Data: $(\mathbf{x}_t, y_t)$
  `/* Evaluate current measurement */`
5 Compute $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$ using (6) ;
6 Set $\mathbf{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \cdot \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$;
7 Calculate projection error $\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \mathbf{a}_t$;
8 Update Λ using (7) ;
9 **if** $\delta_t > \nu_2$ **then**
10 Raise Red1 Alarm ;
11 **endif**
12 **if** $\delta_t > \nu_1$ **then**
13 Raise Orange Alarm ;
14 Set $\Theta = \Theta \bigcup \mathbf{x}_t$ and append Λ with $(\ \mathbf{0}\ \ 1\ )^T$;
15 **endif**
16 Compute $\mathbf{q}_t$, $\mathbf{P}_t$ and $\tilde{\alpha}_t$ using (8)-(10) ;
  `/* Process previous orange alarm */`
17 **if** *Orange Alarm($\mathbf{x}_{t-\ell}$)* **then**
18 Re-evaluate projection error δ for $\mathbf{x}_{t-\ell}$ using $\mathcal{D}_t$;
19 **if** $\delta_t > \nu_1$ **then**
20 **if** $\mathrm{sum}([\Lambda]_{L-\ell+1:L\,,\,m_{t-1}+1}) > \epsilon\ell$ **then**
21 Set $\mathcal{D} = \mathcal{D} \bigcup \Theta\{1\}$ and $\tilde{\mathbf{a}}_t = \mathbf{a}_t$;
22 Compute $\tilde{\mathbf{K}}_t$, $\tilde{\mathbf{K}}_t^{-1}$ using (11), (12)
23 Set $\mathbf{a}_t = (\ \mathbf{0}\ \ 1\ )^T$;
24 Compute $\mathbf{P}_t$ and $\tilde{\alpha}_t$ using (13) and (15) ;
25 $m_t = m_{t-1} + 1$;
26 Lower Orange Alarm($\mathbf{x}_{t-\ell}$) to Green ;
27 **else**
28 Elevate Orange Alarm($\mathbf{x}_{t-\ell}$) to Red2 ;
29 **endif**
30 **else**
31 Lower Orange($\mathbf{x}_{t-\ell}$) to Green ;
32 **endif**
33 Remove $\Theta\{1\}$;
34 **endif**
  `/* Remove obsolete elements */`
35 **for** $j = 1, \ldots, m_t$ **do**
36 **if** $\mathrm{sum}([\Lambda]_{1:L\,,\,j}) = 0$ **then**
37 `DropElement(j)` ;
38 **endif**
39 **endfor**
40 **endfor**

Procedure `DropElement(p)`

1 Move pth rows & columns of $\tilde{\mathbf{K}}_t$, $\tilde{\mathbf{K}}_t^{-1}$ to ends ;
2 Set $\delta_p = 1/[\tilde{\mathbf{K}}_t^{-1}]_{m_t,m_t}$ and $\tilde{\mathbf{a}}_p = -\delta_p[\tilde{\mathbf{K}}_t^{-1}]_{1:m_t-1,m_t}$;
3 Calculate $\tilde{\mathbf{K}}_t^{-1}$ and $\tilde{\alpha}_t$ using (16) and (17) ;
4 Set $\tilde{\mathbf{K}}_t = [\tilde{\mathbf{K}}_t]_{1:m_t-1,1:m_t-1}$, $m_t = m_{t-1} - 1$ and $\mathbf{P} = 10000 \cdot \mathbf{I}_m$;
5 Remove pth element from $\mathcal{D}$ and pth column from Λ_t ;

is $k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. $\tilde{\mathbf{K}}_t^{-1}$ is the inverse of $\tilde{\mathbf{K}}_t$ and $k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t)$. $\mathbf{P}_t$ is known as the covariance matrix in RLS literature and equals $[\mathbf{A}^{\mathrm{T}}\mathbf{A}]^{-1}$ where $\mathbf{A}_t$ is the full $t \times m_t$ matrix of least squares coefficients $\mathbf{a} = (a_1, a_2, \ldots, a_{m_t})$. Structure Θ stores the input vectors corresponding to unresolved orange alarms. At any time t, Θ will contain G input vectors, where G denotes the number of orange alarms that occurred between timesteps $t - \ell$ and $t - 1$.

Lines 1-2: The fixed parameters of the algorithm were discussed in Section III with the exception of γ, the forgetting factor. Our KOAD algorithm gradually disregards old data through exponential forgetting, which puts time-dependent weights on old observations.

Line 3: At timestep 1, the first input vector is added to the dictionary, and all variables are initialized.

Lines 5-7: Upon receiving each input vector, the projection error δ_t is evaluated. The first step in this process is the calculation of the kernel values for the current input measurement:

$$[\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)]_j = k\left(\mathbf{x}_t, \mathcal{D}\{j\}\right) \ \text{ for } \ j = 1, \ldots, m_{t-1} \tag{6}$$

This allows computation of the sparsification vector $\mathbf{a}_t$ and subsequently δ_t.

Line 8: The binary matrix Λ is concatenated from two submatrices of sizes $L \times m_{t-1}$ and $L \times G$. The columns of Λ indicate whether the kernel values of $\tilde{\mathbf{x}}_j$ with $\mathbf{x}_t$ for the $j = 1, \ldots, m_{t-1}$ members of the dictionary, and those of $\Theta\{g\}$ with $\mathbf{x}_t$ for the $g = 1, \ldots, G$ unresolved orange alarms, exceeded d for the previous L timesteps.

Aside from the additional effects of changes to the dictionary and the number of unresolved orange alarms, the Λ matrix is updated each timestep as follows:

$$\Lambda = \begin{cases} \begin{pmatrix} [\Lambda]_{2:end,1:end} \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} > d, k(\Theta\{g\}, \mathbf{x}_t) > d \end{pmatrix} & \text{for } t > L, \\ \begin{pmatrix} [\Lambda] \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} > d, k(\Theta\{g\}, \mathbf{x}_t) > d \end{pmatrix} & \text{otherwise.} \end{cases} \tag{7}$$

Lines 9-15: These lines address the signalling of alarms. If $\delta_t > \nu_2$, we instantly declare a Red1 alarm. If $\delta_t > \nu_1$ then we raise an orange alarm, store $\mathbf{x}_t$ in Θ, and append Λ with $(\ \mathbf{0}\ 1\ )^{\mathrm{T}}$.

Line 16: The covariance matrix $\mathbf{P}$ is updated using:

$$\mathbf{P}_t = \frac{1}{\gamma}\left(\mathbf{P}_{t-1} - \mathbf{q}_t \mathbf{a}_t^{\mathrm{T}} \mathbf{P}_{t-1}\right) \tag{8}$$

where $\mathbf{q}_t$ is known as the Kalman gain in the RLS literature:

$$\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\mathbf{a}_t}{\gamma + \mathbf{a}_t^{\mathrm{T}}\mathbf{P}_{t-1}\mathbf{a}_t}. \tag{9}$$

The least-squares vector $\tilde{\alpha}$ is also updated:

$$\tilde{\alpha}_t = \tilde{\alpha}_{t-1} + \tilde{\mathbf{K}}_{t-1}^{-1}\mathbf{q}_t\left(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1}\right). \tag{10}$$

Lines 17-34: These lines address the resolution of orange alarms observed ℓ timesteps ago. First, we re-evaluate the error δ in projecting $\Theta\{1\} = \mathbf{x}_{t-\ell}$ on to the dictionary at time t.

If the re-evaluated value of δ is found to be greater than ν_1, we perform a secondary "usefulness" test to resolve the orange alarm. We evaluate the "usefulness" of $x_{t-\ell}$ by summing the entries in the $(m_{t-1}+1)$-th column of Λ for the previous ℓ timesteps and comparing with $\epsilon\ell$. Notice that the $(m_{t-1}+1)$-th column of Lambda is always associated with $\Theta\{1\} = \mathbf{x}_{t-\ell}$, which is always the orange alarm that is to be resolved.

If the re-evaluated value of δ exceeds ν_1 and the usefulness test fails, we elevate the orange alarm to Red2. If the re-evaluated value of δ is below ν_1, we lower the orange alarm. In both cases, the dictionary is unchanged.

Now if the re-evaluated value of δ exceeds ν_1 and the usefulness test passes, we must lower the orange alarm and *also* add $\Theta\{1\}$ to the dictionary. Adding an element to the dictionary involves the following steps. First, the optimum $\tilde{\mathbf{a}}_t$ is now $\mathbf{a}_t$. The kernel matrix and its inverse are updated as:

$$\tilde{\mathbf{K}}_t = \begin{pmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} & k_{tt} \end{pmatrix} \tag{11}$$

$$\tilde{\mathbf{K}}_t^{-1} = \frac{1}{\delta_t} \begin{pmatrix} \delta_t \tilde{\mathbf{K}}_{t-1} + \tilde{\mathbf{a}}_t \tilde{\mathbf{a}}_t^{\mathrm{T}} & -\tilde{\mathbf{a}}_t \\ -\tilde{\mathbf{a}}_t^{\mathrm{T}} & 1 \end{pmatrix}. \tag{12}$$

Once $\mathbf{x}_t$ is added to $\mathcal{D}_{t-1}$, $\mathbf{x}_t$ becomes perfectly representable by the elements in $\mathcal{D}_{t-1}$, so $\mathbf{a}_t = (\ \mathbf{0}\ 1\ )$. The update equation for the covariance matrix $\mathbf{P}_t$ is:

$$\mathbf{P}_t = \frac{1}{\gamma} \begin{pmatrix} \mathbf{P}_{t-1} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \gamma \end{pmatrix}. \tag{13}$$

The least squares weights $\tilde{\alpha}_t$ are:

$$\tilde{\alpha}_t = \tilde{\mathbf{K}}_t^{-1} \mathbf{P}_t \mathbf{A}_t^{\mathrm{T}} \mathbf{Y}_t \tag{14}$$

where $\mathbf{Y}_t$ denotes the full history of y_t for timesteps $i = 1 : t$. The recursive update equation for $\tilde{\alpha}_t$ in terms of $\tilde{\alpha}_{t-1}$ is then:

$$\tilde{\alpha}_t = \begin{pmatrix} \gamma^{-\frac{1}{2}} \tilde{\alpha}_{t-1} - \frac{1}{\delta_t} \tilde{\mathbf{a}}_t \left(y_t - \gamma^{-\frac{1}{2}} \cdot \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1} \right) \\ \frac{1}{\delta_t} \left(y_t - \gamma^{-\frac{1}{2}} \cdot \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1} \right) \end{pmatrix}. \tag{15}$$

Lines 35-39: These lines evaluate whether any dictionary element has become obsolete. The test is similar to the usefulness test, except it is performed over L timesteps, and the criterion for declaring a dictionary element obsolete is made stricter. An element is discarded from the dictionary if the relevant column of Λ contains all zeroes, indicating that the particular element has been consistently useless.

Procedure DropElement(p): This procedure removes the p-th element from the dictionary. The first step is to re-organize the rows and columns of $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{K}}^{-1}$, such that kernel values of every other element with the p-th element is associated with the last row and column of $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{K}}^{-1}$. We then evaluate δ_p and $\tilde{\mathbf{a}}_p$, and use them to update $\tilde{\mathbf{K}}_t^{-1}$ according to:

$$\tilde{\mathbf{K}}_t^{-1} = [\tilde{\mathbf{K}}_t^{-1}]_{1:m_t-1,1:m_t-1} - \frac{\tilde{\mathbf{a}}_p \tilde{\mathbf{a}}_p^{\mathrm{T}}}{\delta_p}. \tag{16}$$

The update equation for $\tilde{\alpha}_t$ is:

$$\tilde{\alpha}_t = \tilde{\alpha}_t - \frac{1}{\delta_p} \begin{pmatrix} \tilde{\mathbf{a}}_p \tilde{\mathbf{a}}_p^{\mathrm{T}} & -\tilde{\mathbf{a}}_p \\ -\tilde{\mathbf{a}}_p^{\mathrm{T}} & 1 \end{pmatrix} \tilde{\mathbf{K}}_t \tilde{\alpha}_t. \tag{17}$$

The new optimum weight vector $\tilde{\alpha}_t$ must explain the estimate $\hat{y}$ using one less component. Equation (17) ensures that the value of the last coefficient in the updated $\tilde{\alpha}_t$ is 0. The last component of the updated $\tilde{\alpha}_t$ may then be truncated, along with the last row and column of $\tilde{\mathbf{K}}_t^{-1}$. We now delete the pth element from $\mathcal{D}$ and decrement m. Recalculation of the covariance matrix $\mathbf{P}$ requires full access to the historical data; instead we choose to reset $\mathbf{P}$ to a large constant times the appropriately-sized identity matrix every time an element is dropped. Our experiments have shown that a value of $10,000$ for the constant allows $\mathbf{P}$ sufficient variation in subsequent timesteps, and the predictive component of the algorithm (as given by (4)) stabilizes within a few timesteps.

ACKNOWLEDGEMENTS

This research was supported by the Natural Sciences and Engineering Research Council (NSERC) and industrial and government partners, through the Agile All-Photonic Networks (AAPN) project.

REFERENCES

[1] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," in *Proc. SIGCOMM*, Philadelphia, PA, Aug. 2005.

[2] ——, "Diagnosing Network-Wide Traffic Anomalies," in *Proc. SIGCOMM*, Portland, OR, Aug. 2004.

[3] J. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," in *Proc. USENIX System Admin. Conf. (LISA)*, New Orleans, LA, Dec. 2000.

[4] P. Barford, J. Kline, D. Plonka, and A. Ron, "A Signal Analysis of Network Traffic Anomalies," in *Proc. Internet Measurement Workshop*, Marseille, France, Nov. 2002.

[5] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. Kolaczyk, and N. Taft, "Structural Analysis of Network Traffic Flows," in *Proc. ACM SIGMETRICS*, New York, NY, Jun. 2004.

[6] J. Jackson and G. Mudholkar, "Control Procedures for Residuals Associated With Principal Component Analysis," *Technometrics*, vol. 21, no. 3, pp. 341–349, Aug. 1979.

[7] H. Hajji, "Statistical Analysis of Network Traffic for Adaptive Faults Detection," *IEEE Trans. Neural Networks*, vol. 16, no. 5, pp. 1053–1063, Sep. 2005.

[8] H. Teng, K. Chen, and S. Lu, "Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns," in *Proc. IEEE Comp. Soc. Symp. Research in Security and Privacy*, Oakland, CA, May 1990.

[9] S. Martin, A. Sewani, B. Nelson, K. Chen, and A. Joseph, "A Two-Layer Approach for Novel Email Worm Detection," University of California, Berkeley, Berkeley, CA, 2005. [Online]. Available: http://www.cs.berkeley.edu/~anil/papers/SRUTI_submitted.pdf

[10] K. Heafield, "Detecting Network Anomalies With Kernel Principal Component Analysis," Pasadena, CA, May 2006, research report. [Online]. Available: http://kheafield.com/professional/netlab/final.pdf

[11] T. Lane, "Machine Learning Techniques For The Computer Security Domain of Anomaly Detection," Ph.D. dissertation, Purdue University, W. Lafayette, IN, Aug. 2000.

[12] K. Ilgun, R. Kemmerer, and P. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," *IEEE Trans. Software Eng.*, vol. 21, no. 3, pp. 181–199, Mar. 1995.

[13] B. Schölkopf and A. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, Dec. 2001.

[14] Y. Engel, S. Mannor, and R. Meir, "The Kernel Recursive Least Squares Algorithm," *IEEE Trans. Signal Proc.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.

[15] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, Sep. 2001.

[16] A. Muñoz and J. Moguerza, "Estimation of High-Density Regions Using One-Class Neighbor Machines," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 3, pp. 476–480, Mar. 2006.

[17] T. Ahmed and M. Coates. Online Sequential Diagnosis of Network Anomalies. Project Description. [Online]. Available: http://www.tsp.ece.mcgill.ca/Networks/projects/projdesc-anom-tarem.html