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 continually affected by network anomalies generated by a wide range of sources, from malicious denial-of-service attacks and viruses to harmless large data transfers and accidental equipment failures. Different types of anomaly 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, suitable for use with multivariate data. The proposed algorithm is based on the kernel version of the celebrated 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 network behaviour. The algorithm raises an alarm immediately upon encountering a deviation from the norm. Through comparison with existing block-based off-line 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.**

I. Introduction

Network traffic is often seen to exhibit sudden deviations from normal behavior. Some of these aberrations are owing to 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 behavior in a timely manner, in order to facilitate a fast response and prevent such occurrences in the future. Most prior work in 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 very difficult [5]. Our algorithm learns the behavior of normal traffic, and autonomously adapts 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. Even subtle changes in the nature of network traffic can render the model inappropriate. We readily admit that the choice of traffic measurement and 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 model specification.

A. Related Work

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

Lakhina et al. [6] also suggested an online formulation of the PCA-based detection algorithm. This involved using a sliding window implementation to identify the normal and anomalous subspaces based on a previous block of time. Our proposed recursive approach is a better alternative for online applications than straightforward extensions to block-based methods. The variation in the structure of multivariate network traffic statistics over time is non-negligible. The PCA-based detection algorithm is extremely sensitive to the proper determination of the PCA Q-statistic threshold. We implemented a sliding window version of PCA and observed that although the anomalous and normal subspaces remained relevant over time, using stale measurements to calculate the PCA Q-statistic threshold resulted in an unacceptable number of false positives.

Much of the other previous work on on-line 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].

One of the few examples of real-time anomaly detection that is not based on an a priori model is the time-based inductive learning machine approach of Teng et al. [8]. The inductive learning 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 of the rule does not follow. The learning algorithm presented in [8] is computationally intensive and the results are preliminary. There does not appear to have been further development of this work, and the paper has had more influence in the field of intrusion detection [9], [10].

B. Contribution

The following are the primary contributions of this paper:

1) We describe a new recursive, learning approach to performing anomaly detection. In this approach there is no need to specify a model for normal network behaviour or anomalies. The user must simply identify a feature space in which "normal" traffic measurements or metrics display clustering behaviour.
2) 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 as the normality region expands or migrates and obsolete members being deleted. Our extended version of the Kernel Recursive Least Squares (KRLS) algorithm [11] forms the core of the algorithm.
3) We analyze computational complexity and examine relationships with minimum volume set approaches for identification of dense regions in a feature space.
4) We provide a comparative study on real data of our proposed KRLS anomaly detection algorithm and the block-based PCA detection algorithm described in [1], which indicates that the detection performance of the two is approximately equivalent.

C. Outline of Paper

This paper is organized as follows. Section II reviews the Kernel Recursive Least Squares (KRLS) algorithm, which we extend to form the core of our detection procedure. Section III presents the KRLS online anomaly detection algorithm, discussing computational complexity, the choice of algorithm parameters, and the relationship with minimum volume sets. Section IV presents and analyses the performance of our algorithm when applied to data recorded on the Abilene backbone network, and compares performance to the block-based PCA approach. Section V provides concluding remarks, and describes ongoing work and avenues for future research.

II. Background

A. Kernel Recursive Least Squares

The recursive least squares algorithm is a popular method of obtaining linear predictors of a data sequence [12]. It is suitable for on-line learning scenarios as it observes input samples sequentially and has modest storage and computational requirements. It does not need to store historical data and its computational cost per time-step is independent of time.

Kernel machines use a kernel mapping function to produce non-linear and non-parametric learning algorithms [13]. The idea behind kernel machines 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 [11]. 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:

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

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

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

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

Note that the linear kernel is simply the inner product (dot product) of its arguments.

The Kernel Recursive Least Squares (KRLS) algorithm combines the principles of recursive least squares and kernel machines, providing an efficient and non-parametric approach for performing online data mining and anomaly diagnosis. The algorithm operates on a data sequence of the form:

$$\mathcal{Z}_t = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t)\} \tag{3}$$

where the input-output pairs$(\mathbf{x}_1, y_1)$ 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 (RLS), the dimension of the space spanned by the input samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$ is constrained by the dimension of the input space, so cannot grow indefinitely over time. In contrast, kernel recursive least squares (KRLS) involves a mapping into a feature space of much higher dimensionality than the input space (and possibly of infinite dimension). The dimension of the space spanned by $\{\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_t)\}$ has the potential to increase without bound. At each timestep, the dimension will increase unless $\mathbf{x}_t$ satisfies $\phi(\mathbf{x}_t) = \sum_{i=1}^{t-1} a_i \phi(\mathbf{x}_i)$. If the dimension increases, then the new vector is providing new information and therefore adding to the predictive power, so it should be included in the predictor. This leads to the dilemma that the predictor might require storage of a very large number

of input vectors, leading to unreasonably large 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 before it is added to the predictor [11]. Feature vector $\phi(\mathbf{x}_t)$ is said to be *approximately* linearly dependent on $\{\phi(\mathbf{x}_1),\ \phi(\mathbf{x}_2),\ \ldots, \phi(\mathbf{x}_{t-1})\}$, with approximation threshold ν, if the projection error δ_t satisfies the following criterion:

$$\delta_t = \min_a \|\sum_{i=1}^{t-1} a_i \cdot \phi(\mathbf{x}_i) - \phi(\mathbf{x}_t)\|^2 < \nu. \quad (4)$$

KRLS uses (4) to obtain a *dictionary* of input vectors $\mathcal{D} = \{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_m\}$, where $m < t$, such that $\phi(\tilde{\mathbf{x}}) = \{\phi(\tilde{\mathbf{x}}_1), \phi(\tilde{\mathbf{x}}_2), \ldots, \phi(\tilde{\mathbf{x}}_m)\}$ 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:

$$\hat{y} = \sum_{j=1}^{m} \alpha_j \cdot \langle \phi(\tilde{\mathbf{x}}_j), \phi(\mathbf{x_t}) \rangle = \sum_{j=1}^{m} \alpha_j \cdot kernel(\tilde{\mathbf{x}}_j, \mathbf{x_t}) \quad (5)$$

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

III. The KRLS Anomaly Detection Algorithm

Traffic flows in backbone networks have low intrinsic dimensionality, and demonstrate strong spatial and temporal covariance [6]. Consider a set of multivariate "normal" traffic measurements $\{\mathbf{x}_t; t = 1, 2, \ldots, T\}$. In an appropriately chosen feature space $\mathcal{F}$, with an associated kernel function $kernel(\mathbf{x}_i, \mathbf{x}_j)$, the features corresponding to the normal traffic measurements should *cluster*. That is, it should be possible to describe the region occupied by the traffic features using a relatively small dictionary of linearly independent elements $\{\phi(\mathbf{x}_i)\}$. We can construct an approximately equivalent description by forming a dictionary of elements selected from the input measurements $\{\mathbf{x}_t\}$. The feature vectors corresponding to this dictionary will only form an approximately linearly independent basis, and therefore be somewhat redundant, but the dictionary can be learned on-line directly from the data and its size m will be much less than T, leading to computational and storage savings.

We use an extended version of the kernel recursive least squares (KRLS) algorithm to learn and adapt this dictionary from a set of traffic measurements. 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, KRLS offers a very natural way of assessing this distance: the projection error δ_t of (4).

Algorithm 1: Outline of KRLS anomaly detection

```
1  Set thresholds: ν1, ν2 ;
2  for t = 1, 2, ... do
       Data: (x_t, y_t)
       /* Evaluate current measurement */
3      Compute projection error δ_t ;
4      if δ_t > ν2 then
5          Raise Red1 Alarm ;
6      else if ν1 < δ_t < ν2 then
7          Raise Orange Alarm(x_t) ;
8          Temporarily add x_t to dictionary D;
9      else
10         Green-light x_t, leave D unchanged;
11     endif
       /* Process previous orange alarm */
12     if Orange Alarm(x_{t-ℓ}) then
13         Evaluate usefulness of x_{t-ℓ} over previous ℓ
           measurements;
14         if NOT useful then
15             Raise Red2 Alarm(x_{t-ℓ});
16             Remove x_{t-ℓ} from dictionary D;
17         else
18             Lower Orange Alarm(x_{t-ℓ}).
19         endif
20     endif
       /* Remove obsolete elements */
21     Evaluate usefulness of each dictionary element over
       previous L measurements;
22     Remove any useless element from dictionary D;
23 endfor
```

A. The Algorithm

Algorithm 1 provides a high-level overview of our KRLS anomaly detection algorithm. We include a more complete description in the Appendix. The algorithm operates at each timestep t on a traffic measurement vector $\mathbf{x}_t$ and an associated scalar y_t. 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. In accordance with standard KRLS [11], the algorithm begins by evaluating the error, δ_t, in projecting the arriving $\mathbf{x}_t$ onto the 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 (i.e. explained by) 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 behavior, and immediately raise a "Red1" alarm to signal an anomaly.

In the remaining case where δ_t lies between ν_1 and ν_2, we infer that $\mathbf{x}_t$ is sufficiently linearly independent from the dictionary to be added, but not far enough away to be immediately declared an anomaly. In this case, we do the following:

temporarily expand the dictionary to include the new input vector, declare an "Orange" alarm, track the "usefulness" of this newly-added dictionary member for a short interim period (ℓ timesteps), and then make a firm decision on the orange alarm.

The usefulness of the added dictionary vector is assessed by tracking $kernel(\mathbf{x}_t, \mathbf{x}_j)$ over the time period $i = t+1, \ldots, t+\ell$. If the kernel is large (greater than a threshold d), then $\phi(\mathbf{x}_t)$ is close to $\phi(\mathbf{x}_j)$. If a significant number of the kernel values are large, then $\mathbf{x}_t$ cannot be considered anomalous; normal traffic has just migrated into a new portion of the feature space and we should lower the orange alarm. 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:

$$\left[\sum_{j=t+1}^{t+\ell} \mathbb{I}(kernel(\mathbf{x}_t, \mathbf{x}_j) < d) \right] > \epsilon\ell, \tag{6}$$

where $\mathbb{I}$ is the indicator function and $\epsilon \in (0, 1)$ is a selected constant. If (6) evaluates true, then we turn the orange alarm to green (no anomaly). If not, we elevate it to a "Red2" alarm, and remove the relevant input vector x_t from the dictionary.

Removal of elements from the dictionary occurs when a Red2 alarm is raised or when a dictionary element is declared obsolete. The latter event occurs after a similar test to (6) above. We periodically perform the same check, but replace ℓ by L, a much larger number. It is important to keep ℓ relatively small, because it determines the lag-time before an anomaly is detected. 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 short periods of obsolescence to be ignored. We describe in the appendix the mechanics of removing an element from the dictionary. This removal of dictionary elements, together with the use of dual thresholds and the incorporation of exponential forgetting (see Appendix) are extensions of KRLS beyond the original version in [11].

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 one *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 (30-50 elements is typical).

The computational bottlenecks in the KRLS anomaly detection algorithm are the matrix multiplications (see Algorithm 2 in the Appendix). When no element is dropped, there are 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 [1]. The key point to note here is that the complexity of PCA is a function of time, whereas the KRLS complexity is not.

C. Parameter Selection

The 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. In addition, we set the value of ℓ, the time allowed before a decision must be made on an orange alarm, to be large. During the training period, we are not interested in rapid declaration of anomalies and care more that the space of normality is correctly determined. By considering a larger set of vectors for determining the "usefulness" of an input vector, we improve the robustness of the learning algorithm.

During normal operation, the two parameters that have the most direct effect on the detection performance are the thresholds ν_1 and ν_2. Our experiments have shown that optimal settings for the thresholds 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 addresses). However, for the same metric, the performance of a setting remains approximately the same across widely-separated time periods. In Section IV, we analyse 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 time-lag to anomaly declaration and false alarm rate. From a practical point of view, as statistics are often exported by network monitoring devices every 5 minutes, a value of $\ell = 20$ evaluates to under 2 hours. The 20 input vectors usually provide more than enough data to assess the usefulness of a new dictionary element. Indeed, our experiments have shown that for almost all anomalies that are 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 choose $L = 100$ in the experiments, but any choice in the range $40 - 200$ results in similar performance.

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 region of usefulness. If an element explains more than 10 percent of the data it should not be considered an anomaly (corresponding to $\epsilon = 0.9$). However, if ℓ is small, then setting a slightly smaller value, e.g., $\epsilon = 0.8$, eliminates misses when a number of similar anomalous events occur in succession or if the anomaly persists over several timesteps.

D. Relationship with Minimum Volume Sets

The anomaly detection algorithm we have described effectively identifies a *region of normality* that corresponds to a high-density region of the space, i.e., it contains the vast majority of the encountered measurement vectors. It is natural to compare the outcome of the KRLS approach to other approaches for determining high-density regions. One common approach is the estimation of minimum volume sets (MVSs) [13]. 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 prespecified value μ [13]. These sets are known in the MVS literature as density contour clusters.

Muñoz and Moguerza propose the One-Class Neighbor Machine (OCNM) algorithm for estimating minimum volume sets in [14]. The OCNM algorithm is a block-based procedure that provides a binary decision function $h(\mathbf{x}_t)$ indicating whether $\mathbf{x}_t$ is a member of the MVS. The algorithm requires the choice of a sparsity measure (a distance function) $g(\mathbf{x}_t)$. We have implemented the OCNM algorithm for comparative purposes using the n-th nearest neighbour distance as the sparsity measure. Our comparison in this paper is purely empirical (see Section IV).

IV. Experiments

A. Data

To evaluate our algorithm, we examined performance on network-wide traffic datasets analyzed by Lakhina et al. in [1]. This data was collected from 11 core routers in the Abilene backbone network for a week (December 15 to December 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 5 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.

The anomalies in this dataset were manually identified in [1]. Thus we have "ground truth" anomaly annotations against which to compare the output of our KRLS detection algorithm. We were able to manually identify 19 different anomalies for the packet-counts timeseries and 21 different anomalies for the IP flow-counts timeseries.

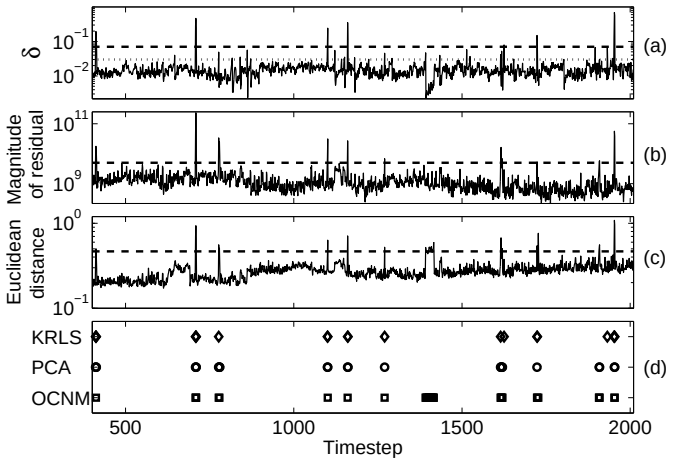


Fig. 1. (a) KRLS projection error δ with dotted line indicating ν_1 and dashed line indicating ν_2; (b) magnitude of PCA projection onto the residual subspace (dashed line indicates PCA Q-statistic threshold); (c) OCNM using nth nearest-neighbor distance ($n = 50$, dashed line indicates anomaly distance threshold); and (d) positions at which anomalies are detected by ($\diamond$) KRLS, (o) PCA, and ($\square$) OCNM, all as functions of time. Dataset: Abilene packet-counts timeseries (see Section IV-A).

B. Results

In our experiments, we set $\mathbf{x}_t$ to be the *flow vector* at timestep t, normalized to the unit hypersphere. The flow vector is defined as $\mathbf{X}_t^{(i)}$, where $i = 1, 2$ indicates whether the number of packets or IP flows is being measured. We chose as the associated y_t the total amount of traffic in the network:

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

This choice exploits clustering due to spatial correlations in network traffic [6]. We also performed experiments where $\mathbf{x}_t$ was generated by concatenation of several successive flow vectors. This choice also allows detection of anomalous departures from the usual temporal correlation observed in "normal" network traffic [6]. We did not detect additional temporal anomalies, so we do not report the results here.

We ran our KRLS 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 $l = 20$ and $\epsilon = 0.8$, and pertain to the no-forgetting ($\gamma = 1$) case (see Appendix). 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 $\mu = 0.98$.

For the dataset comprising the Abilene packet-counts timeseries, Figs. 1(a) and 1(b) show the variations in δ_t obtained using KRLS with $\nu_1 = 0.03, \nu_2 = 0.07$, and the magnitude of the energy in the residual components from PCA using 4 principal components, respectively. For PCA-based anomaly detection, we use the Q-statistic threshold from [1]. 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)

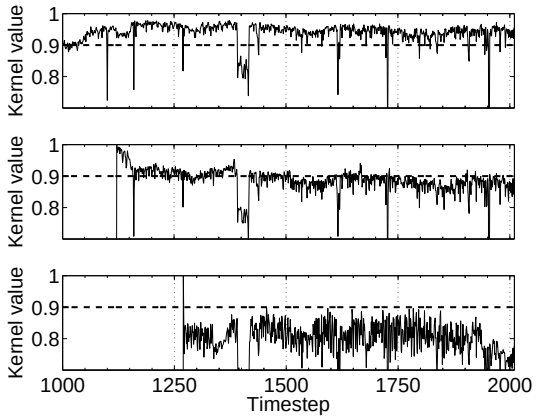


Fig. 2. Behaviour over time of $kernel(\tilde{\mathbf{x}}_j, \mathbf{x}_t)$, when $\tilde{\mathbf{x}}_j$ is (top) a normal $\mathcal{D}$ member, (middle) a $\mathcal{D}$ member that eventually becomes obsolete, and (bottom) an anomalous $\mathcal{D}$ member. In each figure, the dashed line indicates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

indicate the anomalies signalled by KRLS, PCA and OCNM. Fig. 1(d) compares the positions of the detected anomalies, indicating that, for the most part, the three methods detect the same anomalous events.

Using these settings, OCNM flags 14 of the 19 different anomalies in the packet timeseries. The anomalous input vectors thus also exhibit high Euclidean nth neighbour distances in the input 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 KRLS δ_t is unusually low for a block of time. This is indicative of the subtle differences in the three approaches. The block of 26 input vectors forms a small cluster. The cluster is sufficiently numerous (and energetic) for PCA to dedicate a principal component to it, so PCA does not recognize it as an anomaly. KRLS signals the first vector in the cluster as an orange alarm, but because there are sufficient similar vectors immediately thereafter, the usefulness test is passed, and the orange alarm is rendered green. The remaining vectors in the cluster are very similar to this first vector, which results in the unusually low δ_t values. In contrast, the nth nearest-neighbour distance for every vector in the cluster is large (n is larger than 26, the number of vectors in the cluster). It is interesting to note that if we set $\ell = 125$ and $\epsilon = 0.75$, KRLS also flags an anomaly. Alternatively, if we reduce n to 25, OCNM declares *none* of these timesteps as anomalous.

Fig. 2 depicts the changing behaviour over time of the kernel values $kernel(\tilde{\mathbf{x}}_j, \mathbf{x}_t)$ for three different types of dictionary members $\tilde{\mathbf{x}}_j$. Fig. 2(a) depicts the behaviour for a consistently useful dictionary element; many input vectors are close to this element resulting in high kernel values above the threshold d. Sudden drops below the threshold correspond primarily to anomalies. Fig. 2(b) shows the behaviour for a dictionary element that gradually becomes obsolete and is eventually discarded. Fig. 2(c) illustrates the case of a Red2 alarm. The kernel values drop significantly below the threshold *immediately* after this element is initially entered into the dictionary. This input vector is not close to any subsequent vectors, so it is flagged as an anomaly after ℓ timesteps.

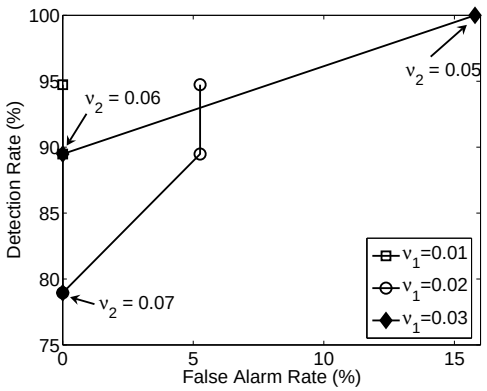


Fig. 3. Detection versus false-alarm rate tradeoff as a function of ν_2, for various sets of fixed ν_1. Dataset: Abilene packet-counts timeseries.

TABLE I

DETECTION OF THE 19 ANOMALIES IN THE ABILENE PACKET-COUNTS DATASET FOR VARIOUS REPRESENTATIVE SETTINGS OF ν_1 AND ν_2.

ν_1 setting	ν_2 setting	Red1	Red2	Missed	False
$\nu_1 = 0.02$	$\nu_2 = 0.05$	12	3	2	3
$\nu_1 = 0.02$	$\nu_2 = 0.06$	11	3	3	1
$\nu_1 = 0.02$	$\nu_2 = 0.07$	9	2	6	0
$\nu_1 = 0.03$	$\nu_2 = 0.05$	16	1	0	5
$\nu_1 = 0.03$	$\nu_2 = 0.06$	13	3	1	1
$\nu_1 = 0.03$	$\nu_2 = 0.07$	12	3	2	0
PCA		16		1	0

C. Detection Performance

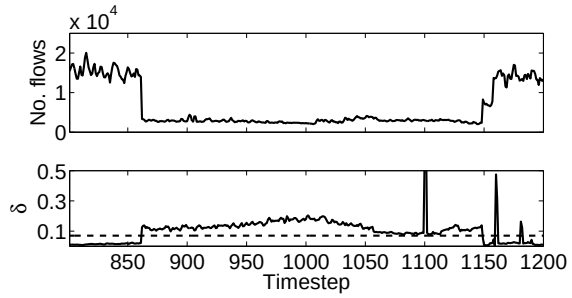
Our objective is to detect the 19 unique anomalies in the packet-counts timeseries, and the 21 unique anomalies in the flow-counts timeseries. Note that several of these anomalies persist over several measurement intervals. Fig. 3 illustrates the detection versus false-alarm tradeoff for the packet-counts dataset, for various settings of ν_1 and ν_2.

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. Most of the anomalies are flagged as Red1 alarms. The detection rate does not monotonically increase as ν_1 decreases because ν_1 does not solely act as a detection threshold, but also determines, in a non-linear fashion, the size of the dictionary and hence the region of normality.

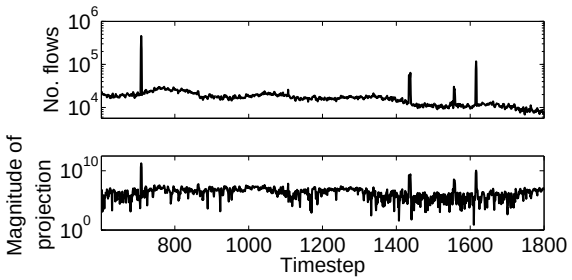
TABLE II

DETECTION OF THE 21 ANOMALIES IN THE ABILENE FLOW-COUNTS DATASET FOR VARIOUS REPRESENTATIVE SETTINGS OF ν_1 AND ν_2.

ν_1 setting	ν_2 setting	Red1	Red2	Missed	False
$\nu_1 = 0.01$	$\nu_2 = 0.05$	13	3	4	1
$\nu_1 = 0.02$	$\nu_2 = 0.06$	13	2	5	1
$\nu_1 = 0.03$	$\nu_2 = 0.05$	18	0	2	1
$\nu_1 = 0.04$	$\nu_2 = 0.07$	18	0	2	1
PCA		16		4	0



(a) Top panel: Variation in number of IP flows in Abilene Los Angeles-Chicago backbone flow over time. Bottom panel: Variation in δ_t during the same time-period for $\nu_1 = 0.03$ and $\nu_2 = 0.07$ (shown as 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 second principal component over the same interval.

Fig. 4. Example anomalies in the Abilene flow-count data set. (a) The structure of normal traffic shifts dramatically for approximately 240 time steps (20 hours). The KRLS anomaly algorithm flags all vectors during this time period as red alarms; this behaviour can only be avoided by resetting the dictionary or setting ν_2 very high. (b) Three anomalies occur in the IP flow-count dataset on the New York-Chicago backbone flow. Due to the PCA anomaly algorithm's block-based analysis, it clusters these together and forms a principal component to describe them. The anomalies go undetected. The KRLS anomaly algorithm adapts over time and identifies all three as anomalies for most threshold settings.

D. Anomaly Analysis

We now examine two types of the anomaly in more detail to highlight the differences in behaviour between the block-based approach of PCA and the on-line approach of KRLS. In the first example, we consider a a 20-hour (240 time step) period, during which the nature of the network traffic changes fundamentally. This is most evident in the flow-count data set; approximately 20 backbone flows experience step-changes in behaviour. The top panel of Figure 4(a) depicts, as an example, the number of IP flows in the Abilene Los Angeles-Chicago backbone flow over time. This change in normality is too radical for the KRLS algorithm to make changes to its dictionary. It flags all the vectors, which are very similar in nature, as red alarms. In contrast, PCA dedicates one principal component to this block of time and does not indicate any anomalous behaviour. In the second example, we examine the New York-Chicago backbone flow. In this case, there are three sudden spikes where the number of IP flows increases dramatically (each lasting 3-6 time steps, see top panel of Fig. 4(b)). The PCA-based anomaly algorithm clusters these anomalies together and dedicates a principal component to describing them (bottom panel of Fig. 4(b)). The KRLS anomaly algorithm adapts over time and detects the three anomalies.

V. Discussion and Future Work

We have described an on-line anomaly detection algorithm based on an extended version of Kernel Recursive Least Squares (KRLS) that is able to detect anomalous events immediately and in real-time. Through analysis of datasets recorded on the Abilene network, we have demonstrated that the algorithm achieves similar detection performance to the most effective block-based approaches, but has a faster time-to-detection and lower computational complexity.

The nature of our results raises interesting questions about what should be reported as an anomaly. For example, the input cluster discussed in Section IV can be viewed as a large time-scale anomaly (in that the traffic is very different from anything seen during the rest of the week). At the time-scale used for analysis in this paper (5-minute intervals) it persists for 26 timesteps (or two hours), rendering debatable its labelling as anomalous. In the flow-count data set, we observe a 20-hour period during which traffic is fundamentally different from the rest of the week. The algorithm presented in this paper usually flags such events as orange alarms (unless the change is radical), which at least draws attention to the traffic for further analysis, but a multiscale anomaly detection approach might prove advantageous.

In this paper, we identify anomalies solely based on the nature of $\mathbf{x}_t$. One of our initial motivations in selecting KRLS as the core of our algorithm was to allow joint detection of two types of anomalies. The first type would be flagged when $\mathbf{x}_t$ lay far away from the dictionary-defined normality and the second when the KRLS prediction error $|\hat{y}_t - y_t|$ was unusually large. To date, we have not incorporated the latter type of anomaly detection because our preliminary analysis did not indicate that it improved performance. However, further investigation is required to determine whether it can prove useful for alternative choices of y_t.

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 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 can be incorporated to provide pseudo-labels for the data. We are also exploring the possibility of dynamically adjusting the thresholds.

It is interesting to consider how combinations of the three approaches we have discussed might perform. For example, PCA analysis could be performed on an initial block of training data and the most significant components could be used as the initial KRLS 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 KRLS dictionary and thresholds.

REFERENCES

[1] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proc. ACM SIGCOMM*, Portland, OR, Aug. 2004.

[2] ——, "Mining anomalies using traffic feature distributions," in *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005.

[3] J. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *Proc. USENIX Fourteenth System Admin. Conf. LISA XIV*, 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] L. LaBarre, "Management by exception: OSI event generation, reporting, and logging," in *Proc. Int. Symp. Integrated Network Management*, Washington, DC, Apr. 1991.

[6] 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.

[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] T. Lane, "Machine learning techniques for the computer security domain of anomaly detection," Ph.D. dissertation, Purdue University, W. Lafayette, IN, Aug. 2000.

[10] 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.

[11] 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.

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

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

[14] A. Muñoz and J. Moguerza, "Estimating of high-density regions using one-class neighbor machines," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 28, no. 3, pp. 476–480, Mar. 2006.

[15] 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

APPENDIX: ALGORITHMIC DETAILS

In this appendix, we present a more detailed description of the KRLS anomaly detection algorithm. Matlab code implementing the algorithm, together with the datasets, is available at [15]. 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 vectors $\{(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_t}$ that are in the dictionary at time t, i.e., $\tilde{\mathbf{K}}_t$. Thus the matrix element $[\tilde{\mathbf{K}}_t]_{i,j} = kernel(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. $\tilde{\mathbf{K}}_t^{-1}$ is the inverse of $\tilde{\mathbf{K}}_t$ and $k_{tt} = kernel(\mathbf{x}_t, \mathbf{x}_t)$. $\mathbf{P}_t$ 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 $\mathbf{a} = (a_1, a_2, \ldots, a_{m_t})$.

Algorithm Discussion:

Lines 1-2: The fixed parameters of the algorithm were discussed in Section III, with the exception of γ, the forgetting factor. Our KRLS algorithm gradually disregards old data through exponential forgetting, which puts time-dependent weights on old observations, leading to a weighted least squares problem. The forgetting factor γ must be chosen keeping in mind that KRLS effectively uses a data window of size $\frac{1}{1-\gamma}$, so γ is usually very close to 1.

Line 3: At timestep 1, initialization occurs. The first input vector is added to the dictionary, the kernel matrix and its inverse are initialized, the initial α value is calculated, and the covariance and Λ matrices are set to 1.

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:

$$[\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)]_j = kernel\left(\mathbf{x}_t, \mathcal{D}\{j\}\right) \text{ for } j = 1, \ldots, m_{t-1} \quad (7)$$

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

Line 8: The j-th column of the binary matrix Λ indicates whether the kernel values $kernel(\tilde{\mathbf{x}}_j, x_t)$ exceeded d for the previous L timesteps. Aside from the additional effects of changes to the dictionary, the Λ matrix (of size $L \times m_{t-1}$) is updated each timestep as follows:

$$\Lambda = \begin{cases} \begin{pmatrix} [\Lambda]_{2:end,1:end} \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} > d \end{pmatrix} & \text{for } t > L, \\ \begin{pmatrix} [\Lambda] \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} > d \end{pmatrix} & \text{otherwise.} \end{cases} \quad (8)$$

Lines 9-15: These lines address the occurrence of an orange alarm. The current vector $\mathbf{x}_t$ is added to the dictionary. The optimum $\tilde{\mathbf{a}}_t$ is now given by $\mathbf{a}_t$. The kernel matrix and its inverse are updated as follows:

$$\tilde{\mathbf{K}}_t = \begin{pmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} & k_{tt} \end{pmatrix} \quad (9)$$

$$\tilde{\mathbf{K}}_t^{-1} = \frac{1}{\delta_t} \begin{pmatrix} \delta_t \tilde{\mathbf{K}}_{t-1} + \tilde{\mathbf{a}}_t \tilde{\mathbf{a}}_t^{\mathrm{T}} & -\tilde{\mathbf{a}}_t \\ -\tilde{\mathbf{a}}_t^{\mathrm{T}} & 1 \end{pmatrix}. \quad (10)$$

A column of ones is appended to Λ to represent the new dictionary vector. The entries are set to one so that the vector cannot be deemed obsolete until (approximately) L timesteps have passed. 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:

$$\mathbf{P}_t = \frac{1}{\gamma} \begin{pmatrix} \mathbf{P}_{t-1} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \gamma \end{pmatrix}. \quad (11)$$

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

$$\tilde{\alpha}_t = \tilde{\mathbf{K}}_t^{-1} \mathbf{P}_t \mathbf{A}_t^{\mathrm{T}} \mathbf{Y}_t \quad (12)$$

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:

$$\tilde{\alpha}_t = \begin{pmatrix} \gamma^{-\frac{1}{2}} \tilde{\alpha}_{t-1} - \frac{1}{\delta_t} \tilde{\mathbf{a}}_t \left(y_t - \gamma^{-\frac{1}{2}} \cdot \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1} \right) \\ \frac{1}{\delta_t} \left(y_t - \gamma^{-\frac{1}{2}} \cdot \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1} \right) \end{pmatrix}. \quad (13)$$

Algorithm 2: KRLS 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 (7) ;
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 (8) ;
9 **if** $\nu_1 < \delta_t < \nu_2$ **then**
10 Raise Orange Alarm($\mathbf{x}_t$) ;
11 Set $\mathcal{D} = \mathcal{D} \bigcup \mathbf{x}_t$ and $\tilde{\mathbf{a}}_t = \mathbf{a}_t$;
12 Compute $\tilde{\mathbf{K}}_t$ and $\tilde{\mathbf{K}}_1^{-1}$ using (9) and (10) ;
13 Set $\Lambda = (\ \Lambda\ \ \mathbf{1}\ )$ and $\mathbf{a}_t = (\ \mathbf{0}\ \ 1\ )^T$;
14 Compute $\mathbf{P}_t$ and $\tilde{\alpha}_t$ using (11) and (13) ;
15 $m_t = m_{t-1} + 1$;
16 **else**
17 **if** $\delta_t > \nu_2$ **then**
18 Raise Red1 Alarm ;
19 **endif**
20 $\mathcal{D}_t = \mathcal{D}_{t-1}$, $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$, $\tilde{\mathbf{K}}_t^{-1} = \tilde{\mathbf{K}}_{t-1}^{-1}$, $m_t = m_{t-1}$;
21 Compute $\mathbf{q}_t$, $\mathbf{P}_t$ and $\tilde{\alpha}_t$ using (14)-(16) ;
22 **endif**
  `/* Process previous orange alarm */`
23 **if** *Orange alarm($\mathbf{x}_{t-\ell}$)* **then**
24 Identify j such that $\tilde{\mathbf{x}}_j = \mathbf{x}_{t-\ell}$;
25 **if** $\mathrm{sum}([\Lambda]_{L-\ell+1:L\,,\,j}) < \epsilon\ell$ **then**
26 Convert Orange Alarm to Red2 Alarm ;
27 `DropElement(j)` ;
28 **else**
29 Convert Orange Alarm to Green ;
30 **endif**
31 **endif**
  `/* Remove obsolete elements */`
32 **if** $t > L$ **then**
33 **for** $j = 1, \ldots, m_t$ **do**
34 **if** $\mathrm{sum}([\Lambda]_{1:L\,,\,j}) < \epsilon L$ **then**
35 `DropElement(j)` ;
36 **endif**
37 **endfor**
38 **endif**
39 **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 (17) and (18) ;
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 ;

Lines 17-21: These lines correspond to the case of a Red1 alarm ($\delta_t > \nu_2$) or green-lighted traffic ($\delta_t < \nu_1$). In either case the dictionary does not change. The kernel matrix ($\tilde{\mathbf{K}}$) and its inverse ($\tilde{\mathbf{K}}^{-1}$) remain the same. The covariance matrix $\mathbf{P}$ is updated using the following equation:

$$\mathbf{P}_t = \frac{1}{\gamma}\left(\mathbf{P}_{t-1} - \mathbf{q}_t \mathbf{a}_t^{\mathrm{T}} \mathbf{P}_{t-1}\right) \tag{14}$$

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

$$\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\mathbf{a}_t}{\gamma + \mathbf{a}_t^{\mathrm{T}}\mathbf{P}_{t-1}\mathbf{a}_t}. \tag{15}$$

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

$$\tilde{\alpha}_t = \tilde{\alpha}_{t-1} + \tilde{\mathbf{K}}_{t-1}^{-1}\mathbf{q}_t\left(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^{\mathrm{T}} \cdot \tilde{\alpha}_{t-1}\right). \tag{16}$$

Lines 23-30: These lines process an orange alarm raised ℓ timesteps ago. We evaluate the "usefulness" of the vector $x_{t-\ell}$ by determining its position j in the dictionary and summing the values of the j-th column of Λ for the previous ℓ timesteps. If the sum exceeds the threshold $\epsilon\ell$, then we green-light the traffic, otherwise we convert the orange alarm to a Red2 alarm and drop element j from the dictionary.

Lines 32-38: These lines evaluate whether any dictionary element has become obsolete. The test is the same as for the orange alarm, except it is performed over L timesteps. The sum of each column of Λ measures the usefulness of the corresponding dictionary element; if it drops below ϵL, the element is removed from the dictionary.

Procedure DropElement(p): This procedure removes the p-th element from the dictionary and makes the necessary modifications to the variables. The first step is the re-organization of the rows and columns of $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{K}}^{-1}$, such that kernels of every other element with the pth 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:

$$\tilde{\mathbf{K}}_t^{-1} = [\tilde{\mathbf{K}}_t^{-1}]_{1:m_t-1,1:m_t-1} - \frac{\tilde{\mathbf{a}}_p\tilde{\mathbf{a}}_p^{\mathrm{T}}}{\delta_p}. \tag{17}$$

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

$$\tilde{\alpha}_t = \tilde{\alpha}_t - \frac{1}{\delta_p}\begin{pmatrix} \tilde{\mathbf{a}}_p\tilde{\mathbf{a}}_p^{\mathrm{T}} & -\tilde{\mathbf{a}}_p \\ -\tilde{\mathbf{a}}_p^{\mathrm{T}} & 1 \end{pmatrix}\tilde{\mathbf{K}}_t\tilde{\alpha}_t. \tag{18}$$

The new optimum weight vector $\tilde{\alpha}_t$ must explain the estimate $\hat{y}$ using one less component. Equation (18) 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 thus 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. A value of $10,000$ allows $\mathbf{P}$ sufficient variation in subsequent timesteps. This leads to faster stability of the algorithm. We have chosen a value of $10,000$ for the constant. Our experiments have shown that the prediction component of the algorithm stabilizes within a few timesteps.