Anomaly Detection for Asynchronous and Incomplete Data

John Duchi, Lester Mackey, Fabian Wauthier
University of California Berkeley
EECS
{jduchi,lmackey,flw}@cs.berkeley.edu

ABSTRACT

Traffic anomalies, node failures, and attacks are common occurrences in today's computer networks, and identifying them quickly and accurately is important for any large network. Algorithms for detection of anomalies have typically been centralized algorithms that assume synchronous measurements from a series of monitor nodes that always provide clean and complete data. This assumption is unrealistic when monitoring nodes or links fail or are not well coordinated and so provide data asynchronously. We develop a series of algorithms for anomaly detection in this incomplete and asynchronous data framework. We compare our approaches to other state of the art algorithms for detecting aberrations in network flow on a trace-based simulator for network traffic using real tier-1 ISP data.

1. INTRODUCTION

Large scale computer systems are susceptible to failure. Recovering from these failures and improving the system require knowing that, when, and where a problem occurs. Consider the example of a networked system. Requirements for additional network bandwidth can be assessed by analysis and detection of excessive network loads, while particularly low network utilization can be an indication of system malfunction. Modern systems allow the collection of amounts of data well beyond the scope of human processing, and this has motivated the application of machine learning techniques to automatically flag anomalous state for further processing by specialized methods [16, 19, 12]. Detecting anomalous system behavior at a local level can be difficult, so modern systems increasingly focus on determining anomalies globally rather than in individual components of a system.

In this work, we take as our motivating example volume anomaly detection in the network link traffic of distributed systems. Anomalous traffic can happen for many reasons: a distributed denial of service attack or the well known "Slashdot Effect" are just two of many [7]. We assume a set of monitors located across a network that continually send link volume measurements to a central coordinator for anomaly detection. Pioneering work by Lakhina et al. [16] introduced an approach for anomaly detection based on subspace projection, which essentially attempts to find the most common modes of variation in the network traffic and classifies variation outside of those principal components as anomalous. More recent work [19, 12] extends the principal component analysis anomaly detection framework. The literature to this point, however, has relied on synchronous data arrival at regularly spaced time points, and aside from [12], that data is assumed never to be missing.

We posit that these assumptions do not always hold in practice and that it is as such useful to design a system that can function in the face of missing data and asynchrony in data arrival rates from different components across the network. For example, in a setting where no global clock is available and precise synchronization between monitors is impossible, it is natural to allow asynchronous measurements. In other settings, some local measurements may be expensive to compute so that it is preferable to take measurements at varying rates. Finally, data may be intentionally filtered to reduce communication costs. In general, we focus on a scenario in which the local monitors provide data to the coordinator asynchronously and at varying rates that depend on local monitor constraints. The coordinator must combine multiple sources and detect anomalies accurately in the face of this asynchrony.

As a step toward a solution to the asynchronous and incomplete data problem, this work proposes a two-phase methodology. Phase I is the imputation—filling in missing values—of missing data. We look at and propose a variety of methods, ranging from constant propagation to combining autoregressive modeling techniques [3, 4] with function approximators such as splines [2, 8] to deal with asynchronous measurements. We use spline and function interpolation techniques to iteratively fit a model to asynchronous measurements received from a particular monitor, which we are able to use to impute data at unobserved time points. In the case of function approximators, this takes the form of resampling the fitted model. Using autoregressive

modeling, we are able to learn a joint model of the evolution of system state; we use this to impute missing data by a combination of forward prediction and statistical estimation. We estimate our model parameters efficiently and dynamically, a contrast to previous network anomaly detection work. Since imputation allows us to produce regularly spaced measurements from asynchronous data, in Phase II of our methodology we apply complete data anomaly detectors (like subspace projection and autoregression) to the imputed data of Phase I.

The remainder of this report is organized as follows. In Section 2 we detail methods for imputing missing data in asynchronous data streams and our strategies for efficient online learning of ARMA models. Section 3 outlines the subspace projection and autoregressive methods for anomaly detection. Section 4 describes a distributed implementation of our method on a set of networked computers. The experimental design and results are given in Section 5. We contrast our approach to previous approaches and give pointers to related work in Section 6, then give our conclusions.

2. IMPUTATION ALGORITHMS

Imputation is the important first step of our two-stage approach to anomaly detection in asynchronous or incomplete data settings. It is the task of filling missing values into a data stream, or imputing them. In this section, we describe several methods for imputing unobserved data, briefly describing prior techniques and expounding more on techniques we develop. Figure 1 contrasts the prototypical behaviors of some of these techniques.

2.1 Baseline Methods

Our baseline methods for imputing unobserved data are simple, low-cost, and intuitive techniques for *forward prediction*, predicting the current datapoint given past observations. Each makes the (strong) assumption that incoming data streams are independent. We will use the performance of these methods as benchmarks for more advanced solutions.

Constant propagation.

Constant propagation is the simplest of the baseline methods. For a given stream of data x_t, the value x_t for an unobserved time t is predicted as the most recent observation x_s made at a time $s < t$.

Averaging.

Averaging generalizes constant propagation. The value x_t for an unobserved time t is predicted as the average of the k most recent observations made before time t. Averaging over a short sequence of past data points incorporates more of the relevant information in the data stream and yields predictions that are more robust to noisy measurements.

Window averaging.

The window averaging method attempts to use temporal locality more strongly than does averaging, which can use uninformative very old data in rarely sampled streams. To predict a value x_t for an unobserved time t, window averaging takes the last observation time s and average over all *observed* values in the sequence $x_{s-k+1}, \ldots, x_s$. Here, k is the size of the window used for averaging. Fewer than k values contribute to the prediction when some value in $x_{s-k+1}, \ldots, x_s$ is unobserved. Any contributing datapoint is guaranteed to be within $k-1$ time steps of the most recent observation.

Linear propagation.

A slightly more advanced scheme than constant imputation estimates a missing observation by fitting a line to the most recently observed k observations and uses the linear function as a predictor for the future.

2.2 Splines

Splines are piecewise polynomial functions that can either interpolate or smooth irregularly sampled data. A spline is defined by a knot sequence $t_1, \ldots, t_k$ of k (not-necessarily uniformly spaced) points that breaks up an interval of $\mathbb{R}$ into subintervals $[t_i, t_j)$, and the spline uses one parametric function for each subinterval. In many applications the knot sequence is taken to be the times at which measurements are made, and the purpose of the piecewise parameterization is to fill in the gaps; however, most applications of splines focus exclusively on situations in which all the available data is already known rather than being iteratively observed [10, 17]. Splines permit us to efficiently make more informed predictions of past unobserved data given newer observations, and because they do not assume uniformity of the knot sequence, allow flexibility in dealing with asynchrony. As such, we use splines to integrate information from points before and after a missing measurement into improved estimates. Splines in our framework primarily serve to update our predictions of past datapoints, and we leave the task of forward prediction to one of the other presented methods.

Linear splines.

A very simple family of splines is the linear spline, which interpolates between neighboring points though a linear function. Given two measurements x_r and x_t, for $r < t$, the estimate x_s for the unobserved time s, with $r < s < t$ is calculated simply as $x_r + \frac{x_t - x_r}{t-r}(s-r)$.

Cubic splines.

A flexible family of splines that is frequently used in

computer graphics is the cubic spline. Cubic splines are popular in many fields due to their simplicity of construction, modelling flexibility, and ease of evaluation. It can be shown that cubic splines can describe the minimum-curvature curve that runs through a set of data points. Heuristically, cubic splines can model data streams in the smoothest way possible [10]. Cubic splines (also called splines of order $d = 4$) use a set of cubic functions on knot intervals. To smoothly join the cubics at knot points, constraints are imposed between cubics lying on neighboring intervals. Different flavours of cubic splines differ in the kinds of contraints that are imposed and the type of continuity that can be achieved globally, and it is straightforward to efficiently compute and update cubic splines [8].

B-splines.

Cubic splines, because they often interpolate the observation sequence exactly, can be sensitive to outliers and hence are unsuitable for data interpolation or imputation. In this case an alternative cubic spline, such as a B-Spline, that smooths the data while retaining continuity through the 2^{nd} derivative may be more desirable. Smoothing introduces robustness into the imputation process by requiring the spline to fit the data only approximately. In addition to this smoothing property, B-Splines provide local control, which means that changes to individual measurements change the estimated function only locally. As noted before, this property is key for more efficient spline update schemes—in the sequential setting where new measurements are only added to the end of an observation sequence, re-estimation of imputed values is limited to the last few time points. All points beyond a time horizon in the past cannot be affected by future measurements and are effectively frozen.

B-splines of order d are defined through a blending function $B_{k,d}(t)$ which specifies how observed measurements should be mixed together to produce a smoothed estimated at an unobserved time t. The estimate for the unobserved measurement x_t using a B-spline of order d and a knot sequence $t_1, \ldots, t_k$ of length k is $x_t = \sum_{k=0}^{n} x_k B_{k,d}(t)$. De Boor [6] showed that the blending function for B-splines of order d on a knot sequence of length k satisfies the following recursive expression:

$$\begin{aligned} B_{k,1}(t) &= \begin{cases} 1 & \text{if} \quad t_k \le t < t_{k+1} \\ 0 & \text{otherwise} \end{cases} \\ B_{k,d}(t) &= \frac{t - t_k}{t_{k+d-1} - t_k} B_{k,d-1}(t) \\ &\quad + \frac{t_{k+d} - t}{t_{k+d} - t_{k+1}} B_{k+1,d-1}(t). \end{aligned}$$

The key propety of the blending function alluded to earlier is that it encodes local control during the imputation process. The estimate of an unobserved measurement can only be affected by measurements were taken near the unobserved time.

Figure 1 illustrates several imputation techniques on a sample dataset.

2.3 ARMA Modeling

The Box-Jenkins methodology, or AutoRegressive Integrated Moving Average (ARIMA) modeling technique [3, 4] is a class of linear time-series forecasting techniques that capture linear dependency of future variables on the past. An ARIMA model includes three different order parameters: p, the autoregressive parameter; q, the moving average parameter; and d, the differencing parameter. We focus on $d = 0$ models, which means that we perform ARMA modeling directly on the sequence of values we receive.

When we receive points in a time series $x_t \in \mathbb{R}^d$, the mean-0 ARMA(p, q) model can be written as

$$\begin{aligned} x_t &= A_1 x_{t-1} + A_2 x_{t-2} + \ldots + A_p x_{t-p} \\ &\quad + B_1 z_t + B_2 z_{t-1} + \ldots + B_q z_{t-q+1} \end{aligned} \tag{1}$$

where $z_t \sim \mathcal{N}(0, \Sigma)$ are normally-distributed noise parameters independent of x_t. The standard ARMA procedure, employed, for example, in [19], is to estimate A_i, B_i, and Σ, usually via a procedure such as maximum likelihood or moment-matching via the Yule-Walker equations [4]. Solving for A_i, B_i, and Σ above, however, is non-convex, so it is effectively impossible to get the true maximum likelihood estimates. Further, estimating the parameters requires maintaining all data from previous times $x_1, x_2, \ldots$, and repeated estimation passes over the entire time series can be prohibitively expensive. Many approaches to learning ARMA parameters are thus done offline rather than dynamically, which is undesirable (see our experiments in Sec. 5.1). For these reasons, we use an AR(p) model with no moving-average parameters. As we show in the sequel, this allows efficient and exact computation of the A_i and noise covariance Σ.

We now derive the maximum likelihood estimates for our AR(p) model. We can write the sequence from Eq. (1) as $x_t = A_1 x_{t-1} + A_2 x_{t-2} + \ldots + A_p x_{t-p} + z_t$, wrapping B_1 into the covariance for z. We can now write the likelihood of a sequence $x_p, \ldots, x_t$ given the first observations $x_1, \ldots, x_{p-1}$ as

$$f(x_p, \ldots, x_T) = \prod_{t=p}^{T} f(x_t \mid x_{t-1}, \ldots, x_{t-p})$$

where $f(x_t \mid x_{t-1}, \ldots, x_{t-p})$ is the Gaussian density with mean $A_1 x_{t-1} + \ldots A_p x_{t-1}$ and covariance Σ. The log-loss for one term in the sequence (ignoring constants not dependent on Σ or the A_i) is

$$\begin{aligned} l_t(A, \Sigma) &= \log\det\Sigma + \operatorname{tr}\left(\Sigma^{-1}(x_t - A_1 x_{t-1} - \ldots - A_p x_{t-p}) \right. \\ &\quad \left. (x_t - A_1 x_{t-1} - \ldots - A_p x_{t-p})^\top\right). \end{aligned}$$

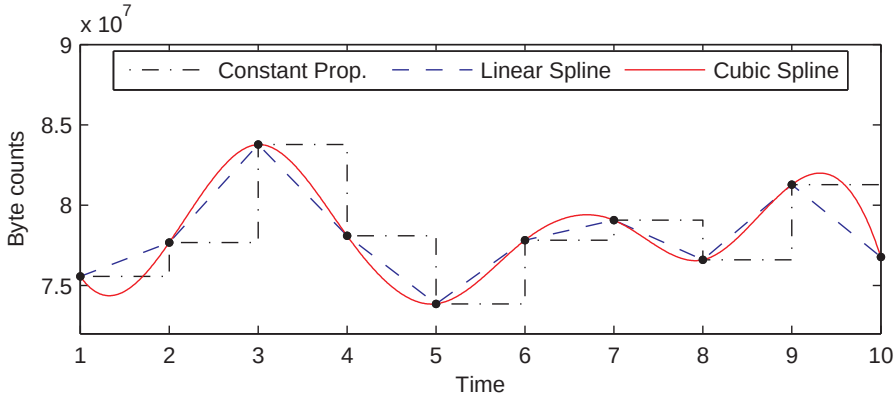


Figure 1: Prototypical imputation behavior of constant propagation, linear splines, and cubic splines.

For notational convenience in the rest of this section, we define the following variables:

$$x(t) \triangleq \begin{bmatrix} x_{t-1} \\ \vdots \\ x_{t-p} \end{bmatrix}, \quad A \triangleq [A_1 \dots A_p],$$

$$X_r(T) \triangleq \sum_{t=p}^{T} x(t)x(t)^\top, \quad X_l(T) \triangleq \sum_{t=p}^{T} x(t)x_t^\top.$$

The derivative of the log-loss with respect to A_i is

$$\nabla_{A_i} l_t(A, \Sigma) = \Sigma^{-1}(Ax(t) - x_t)x_{t-i}^\top.$$

We can thus sum over all time steps and set the result to 0 to find the maximum likelihood parameters for A_i:

$$0 = \sum_{t=p}^{T} Ax(t)x_{t-i}^\top - x_t x_{t-i}^\top, \quad \text{so} \quad X_l(T) = X_r(T)A^\top. \tag{2}$$

Estimating Σ is similarly straightforward given that the estimation of A can be done without consideration of Σ. Summing the log-loss per-example and letting $K = \Sigma^{-1}$, we have

$$\sum_{t=p}^{T} l_t(\Sigma, A) = -(T - p + 1)\log\det K + \sum_{t=p}^{T} \operatorname{tr}\left(K(x_t - Ax(t))(x_t - Ax(t))^\top\right)$$

and by taking derivatives with respect to K, we find that

$$\Sigma = \frac{1}{T-p+1}\sum_{t=p}^{T}(x_t - Ax(t))(x_t - Ax(t))^\top. \tag{3}$$

Fast Parameter Updates.

Using an AR(p) model, it is straightforward to quickly update the set of learned matrices A using new observations of the time series. Indeed, because $A^\top = X_r(T)^{-1}X_l(T)$ by Eq. (2), we would like to efficiently update $X_r(T)^{-1}$ given a new point (it is clear that $X_l(T)$ is trivial to update). The Sherman-Morrison-Woodbury identity [9] shows that given a new point x_{T+1},

$$X_r(T+1)^{-1} = X_r(T)^{-1} - \frac{X_r(T)^{-1}x(T+1)x(T+1)^\top X_r(T)^{-1}}{1 + x(T+1)^\top X_r(T)^{-1}x(T+1)}$$

As we see in the sequel, we often multiply vectors by the matrix A, but it is really unnecessary to keep the matrix A at all; we can simply multiply any vector by $X_r(T)^{-1}$ and $X_l(T)$. This means that we do not need the $O(d^3p^3)$ operations to recompute A per iteration—we only need d^2p^2, a significant savings when d is large. Similar updates can be derived for Σ and $K = \Sigma^{-1}$ if we only iteratively update the sum with each new point in Eq. (3) rather than recalculating it every time we update A. Admittedly, this is incorrect, but it is possible to show asymptotic stability as the AR model continues to run.

Imputation with AR.

AR modeling is useful in that it allows (often accurate) predictions of future variables and gives a framework for joint inference over unobserved variables. We consider prediction of the linear process in two modes. For the first, we assume that we have observed all past data and are only attempting to predict future data. Starting with prediction of x_t from $x_{t-1}, x_{t-2}, \dots$, we have

$$\begin{aligned} &\mathbb{E}[x_t \mid x_{t-1}, \dots] \\ &\quad = \mathbb{E}\left[A_1 x_{t-1} + \dots + A_p x_{t-p} + z_t \mid x_{t-1}, \dots, x_{t-p}\right] \\ &\quad = \sum_{i=1}^{p} A_i x_{t-i} \end{aligned}$$

Denoting by $\hat{x}_t$ the prediction for time step t, it is

straightforward to derive the recursive update

$$\mathbb{E}\left[x_{t+j} \mid x_{t-1}, \ldots\right] = A_1\hat{x}_{t+j-1} + \ldots + A_p\hat{x}_{t+j-p},$$

which shows that we can simply compute the 1-step predictions, followed by the 2-step predictions, etc.

Now suppose that we observe only parts of a previous sample. In particular, we want to predict x_t based on $x_{t-1}, \ldots$, but we observe only part of x_{t-1}. Denote the observed portion of x_{t-1} by o_{t-1} and the unobserved by u_{t-1}. We have

$$\begin{aligned}
&\mathbb{E}\left[x_{t+1} \mid o_t, x_{t-1}, \ldots\right] \\
&= \mathbb{E}\left[A_1 x_t + \ldots + A_p x_{t-p+1} + z_t \mid o_t, x_{t-1}, \ldots\right] \\
&= A_1\mathbb{E}\left[x_t \mid o_t, x_{t-1}, \ldots\right] + A_2 x_{t-1} + \ldots + A_p x_{t-p+1}.
\end{aligned}$$

It is straightforward to see that u_t is Gaussian given $o_t, x_{t-1}, \ldots$. Let $K = \Sigma^{-1}$ for notational convenience, and note that our likelihood term in the expectation of x_t is now proportional to

$$\exp\left(-\frac{1}{2}\left(\begin{bmatrix} o_t \\ u_t \end{bmatrix} - Ax(t)\right)^\top K \left(\begin{bmatrix} o_t \\ u_t \end{bmatrix} - Ax(t)\right)\right).$$

Expanding the terms dependent on u_t and ignoring the exp, we have

$$\begin{aligned}
&\begin{bmatrix} o_t \\ u_t \end{bmatrix}^\top \begin{bmatrix} K_{oo} & K_{ou} \\ K_{uo} & K_{uu} \end{bmatrix} \begin{bmatrix} o_t \\ u_t \end{bmatrix} - 2\begin{bmatrix} o_t \\ u_t \end{bmatrix}^\top KAx(t) \\
&= o_t^\top K_{oo} o_t + 2o_t^\top K_{ou} u_t + u_t^\top K_{uu} u_t \\
&\quad -2u_t^\top [K_{uo}\ K_{uu}](A_1 x_{t-1} + \ldots + A_p x_{t-p}) + C(o_t)
\end{aligned}$$

where $C(o_t)$ is not dependent on u_t. To find the expectation (and mode) of the above with respect to u_t, we take the derivative and find

$$K_{uo}o_t + K_{uu}u_t - [K_{uo}\ K_{uu}](A_1 x_{t-1} + \ldots + A_{t-p}x_{t-p}) = 0,$$

or that

$$u_t = K_{uu}^{-1}[K_{uo}\ K_{uu}]Ax(t) - K_{uu}^{-1}K_{uo}o_t.$$

Once we have estimated the unobserved part of x_t, we can use it to easily predict x_{t+1}. It is possible to derive similar exact updates for computations with more unobserved variables, but we focused on efficient updates in our system and so, if there are multiple unobserved variabels in sequence, we simply perform one (inexact) pass from the most recently fully observed variables, estimating as we propagate from x_{t-k} to x_t.

In this paper, we also combine splines and AR modeling to do imputation—we can use past predicted values of the AR process as "hints" to the spline's interpolation of previously observed points. Effectively, we give the spline slope information from the AR predictions to shape the function we use for interpolation. In all our experiments with AR imputation in Sec. 5, we use the joint interpolation.

3. DETECTION ALGORITHMS

The second stage of our approach to detecting anomalies in asynchronous and incomplete data settings consists of applying a complete data anomaly detector to our imputed data. In this section, we describe our adaptations of two leading methods for complete data anomaly detection.

3.1 Subspace Projection

3.1.1 Original Formulations

Subspace projection for anomaly detection [16, 19, 12] operates by dividing the input space of data points into "normal" and "abnormal" subspaces, projecting each new data point onto the abnormal subspace, and declaring that point to be anomalous if the magnitude of its abnormal projected component exceeds a threshold. The division into normal and abnormal subspaces is achieved by Principal Component Analysis (PCA) [14], a popular and powerful technique for finding the directions of maximum variation in a dataset.

More formally, the subspace projection detector of [16] receives a data matrix $Y \in \mathbb{R}^{n\times d}$ as input (where row i is an observation at time t_i and each column corresponds to a single monitor) and performs PCA on Y, extracting the first k directions of maximum variance $\{p_1, \ldots, p_k\}$. These first k principal components directions are assumed to be a basis for the normal subspace, so we obtain the abnormal component a_t of a given data point $y_t \in \mathbb{R}^d$ by projecting y_t onto the orthocomplement of the space spanned by $\{p_1, \ldots, p_k\}$. That is, $a_t = (I - PP^T)y_t$ where $P = [p_1, \ldots, p_k]$. We then compare the magnitude of a_t to Q_α, the $1-\alpha$ confidence level Q-statistic [13], and declare y_t an anomaly if $||a_t||^2 > Q_\alpha$. An explicit procedure for computing Q_α is given in [16].

There are two principal drawbacks of this original subspace projection formulation. First, the authors of [16] compute the principal components of Y only once, offline, prior to any detection, and do not update the components over time. Thus, the system does not adapt to changes in normalcy over time. Second, and perhaps more seriously, the algorithm as presented only works in a batch setting, where the data from all time points are observed prior to performing any detection. This restriction is due to the matrix Y, which contains observations from all timepoints of interest. The principal components used to classify each time point are therefore based on future information, making this procedure infeasible in an online detection setting.

The subspace projection formulation of [12] solves both of these problems by computing at each time point t the principal components of $Y_t = [y_{t-m}, \ldots, y_{t-1}]$, a window containing the previous m observations, and using these dynamic components to classify the current

vector y_t. This method, however, suffers from using only m observations for the principal component computation at each time step; all other observations are ignored.

3.1.2 Alternative Online Formulation

We present here an online bookkeeping method to help subspace projection that adapts dynamically to changes in normalcy while utilizing all data seen so far in computing principal components. The key is to maintain at each time step the outer product matrix $A_t = \sum_{i=1}^{t} y_i y_i^\top$ and the vector sum $z_t = \sum_{i=1}^{t} y_i$. Given these quantities, the covariance matrix of the first t datapoints can be computed as $C_t = \frac{1}{t-1}(A_t - \frac{1}{t} z_t z_t^\top)$, and PCA can be peformed directly on the covariance matrix C_t. This formulation yields efficient updates for stored quantities $A_t = A_{t-1} + y_t y_t^\top$ and $z_t = z_{t-1} + y_t$ with update time and space complexities independent of the number of time steps.

To allow our imputation methods the flexibility of updating their past imputations, our subspace detector also stores the window Y_t of the past m imputed vectors received. When an imputer module provides an update for any of the m past vectors at time t, the original contribution of that vector is subtracted from A_{t-1} and z_{t-1}, and the new vector's contribution is added in its place: e.g., $A_{t-1} = A_{t-1} - y_{\text{old}} y_{\text{old}}^\top + y_{\text{new}} y_{\text{new}}^\top$ and $z_{t-1} = z_{t-1} - y_{\text{old}} + y_{\text{new}}$.

3.1.3 Redundancy Filtering

A key contribution of [12] was its demonstration that the communication cost of anomaly detection can be drastically reduced without a significant sacrifice of detection accuracy over standard subspace projection. This reduction is achieved by filtering out redundant data values from each monitor node's datastream. Monitor node i maintains a prediction value R_i and only reports its statistic v_t to the detector node at time t if the value v_t falls outside of the window $[R_i - \delta, R_i + \delta]$. The value δ is known as the monitor slack. The slack is automatically selected by the subspace detector node to ensure that the reduction in detection accuracy is bounded and is then propagated to all monitor nodes.

When a monitor node reports a statistic, it updates its prediction value to equal the average of its last five statistics and reports the new R_i value to the detector node as well. The detector then uses the value R_i to impute the missing values in datastream i. Thus, the redundancy filtering for reduced communication scenario of [12] is a special case of our incomplete data framework. We call this method of imputation *monitor imputation*. We implement monitor imputation along with our other imputation methods to compare their performances in the redundancy filtering setting. Note that monitor imputation enjoys a significant advantage over our methods: it computes its prediction value using complete data, while our methods must infer predictors from an incomplete data stream. On the other hand, monitor imputation must transmit twice as much data in the redundancy filtering setting as one of our imputation methods, as R_i must be packaged along with v_t. Hence, improved imputation methods on the detector node's end could halve the communication cost of network anomaly detection.

3.2 AR Detection

As suggested in [19], we can use the AR(p) models described in Sec. 2 to perform anomaly detection as well as imputation of time series data. The basic idea is to use the previous p steps, $x_{t-1}, \ldots, x_{t-p}$ to form the prediction of x_t, $\hat{x}_t = A_1 x_{t-1} + \ldots A_p x_{t-p}$. Anomalous data is then the error in prediction $z_t = x_t - \hat{x}_t$. We now derive a statistic for detection of anomalies with an AR(p) model; we are unsure whether this particular test has been considered in the network anomaly detection literature. According to our modeling assomputions,z_t should be distributed as $\mathcal{N}(0, \Sigma)$. To check if z_t is "anomalous" at a probability level α, we find the c such that the probability a point lies outside the ellipsoid given by $\{x : x^\top \Sigma^{-1} x \geq c\}$ is equal to α—these are the level curves of the $\mathcal{N}(0, \Sigma)$ distribution. To that end, let $\mathcal{E}_c$ be the ellipsoid set for level c and let L be the lower Cholesky decomposition of Σ. Now note that if $w \sim \mathcal{N}(0, I)$, $z = Lw$ is distributed as $\mathcal{N}(0, \Sigma)$. Likewise, $w = L^{-1} z$ is distributed as $\mathcal{N}(0, I)$. By a straightforward change of variables,

$$\mathbb{P}(w^\top w \geq c) = \mathbb{P}(z^\top L^{-\top} L^{-1} z \geq c) = \mathbb{P}(z^\top \Sigma^{-1} z \geq c).$$

By inspection, $w^\top w$ is distributed as a χ^2 random variable with n degrees of freedom. Thus, let F_n be the cumulative distribution function for a $\chi^2(n)$ random variable. To say that a point z_t is anomalous with probability α, we set $c = F_n^{-1}(1 - \alpha)$, and we classify the point x_t as anomalous at if $z_t^\top \Sigma^{-1} z_t > c$.

4. SYSTEM DETAILS

To simulate a true networked anomaly detection system, we implemented a full trace-driven distributed system. Monitor nodes run on networked machines and read in the appropriate data stream from a trace file populated across the network. The nodes then send messages with monitor statistics to a centralized detector according to a protocol for filtering trace data. The detector node, at some fixed interval, reads in the messages received from each monitor (and their time stamps), imputes missing data according to the imputation mode being used, and performs anomaly detection on the imputed data. We implemented each of the imputation and detection modes described in previous sections to run on the detector node.

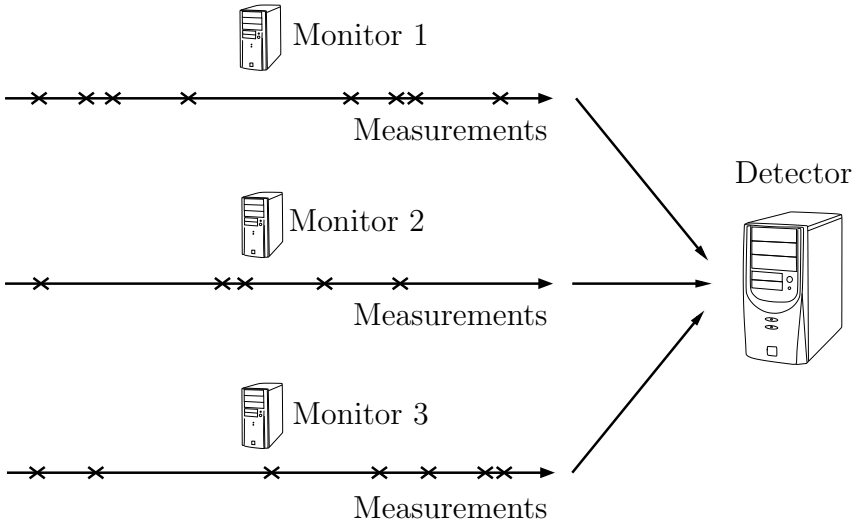


Figure 2: Networked anomaly detection with asynchronous data arrival.

We further implemented four data filtering protocols for the monitor nodes, corresponding to natural data arrival scenarios:

1. Filter mode NONE: All data is reported.
2. Filter mode RANDOM(p): Each node statistic is reported independently with probability p.
3. Filter mode PERIODIC(f): Monitor i reports statistics with frequency f_i.
4. Filter mode REDUNDANT: Each monitor reports its statistics according to the decreased redundancy filtering method of Sec. 3.1.3.

Filter mode NONE provides a complete data benchmark against which the peformance on other filter modes can be compared. Filter mode RANDOM models data that is lost or corrupted at random or that exhibits some nondeterministic arrival rate. Filter mode PERIODIC models monitors with characteristic frequencies of reporting data and allows those frequencies to vary from node to node. This model is especially appropriate when the computation time of statistics varies from node to node or when monitor nodes realize different costs for transmitting data across the network. Filter mode REDUNDANT implements the deliberate filtering scheme detailed in Sec. 3.1.3. The goal is to reduce the amount of monitor/detector communication while maintaining reasonable detection accuracy. The deliberate filtering scheme requires some moderate additional infrastructure for communicating permissible slack values in the monitor nodes. To handle this, we implemented a second signalling mechanism that on each time step broadcasts the new slack value computed by the detector to all monitor nodes.

Note that the purpose of this work is to investigate the efficacy and accuracy of our methods for handling asynchronous latent data. Hence, we report system performance mainly in terms of anomaly detection accuracy and not in terms of run time or communication overhead.

5. EXPERIMENTS AND RESULTS

In this section we describe the series of experiments that we performed using the system we built. Our experiments were all run using our trace-driven simulator on real network data. We used four one-week traces collected from the Abilene network[1]. Each of the Abilene traces consists of measurements for the 41 links in the network, which aggregate data from 121 incoming flows as specified by the routing matrix. Network data is collected every 10 minutes for all 41 links and 121 incoming flows, and a complete datapoint consists of the number of bytes flowing over a link in 10-minute window. To develop more statistically significant tests, we used the method of [16] to inject synthetic anomalies into the datasets. We use the strategy of [12] and inject multiple anomalies into each trace.

5.1 Dynamic AR Models

Before we describe our results on anomaly detection, we make a brief detour to demonstrate the benefits of dynamically learning an autoregressive model of our sequence data rather than learning an offline or rarely updated model. In Fig. 3, we plot the ℓ_2-norm of the prediction errors of a dynamically updated AR(2) process versus the prediction errors over time of an AR(2)

[1]An Internet2 high-performance backbone network connecting a large number of universities and a few research institutions

process learned on the previous week's data. The figure makes clear that the model learned offline has far inferior performance for tracking the process (and one can argue that the two points at which the dynamic model has larger error are likely anomalies in network traffic).

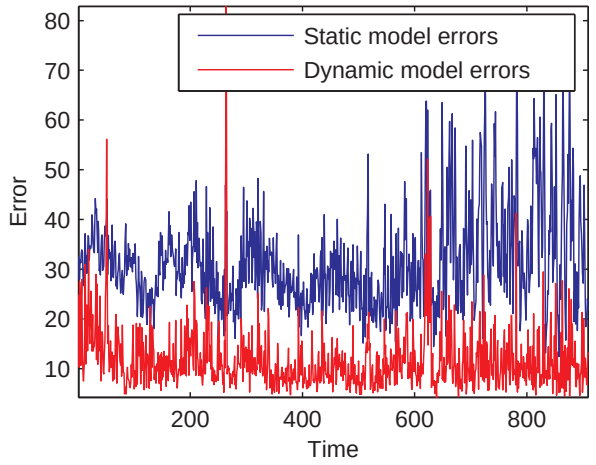


Figure 3: Comparison of dynamically learned AR(2) model on Abilene flow versus statically learned AR(2) model.

5.2 Imputation and Detection Methods

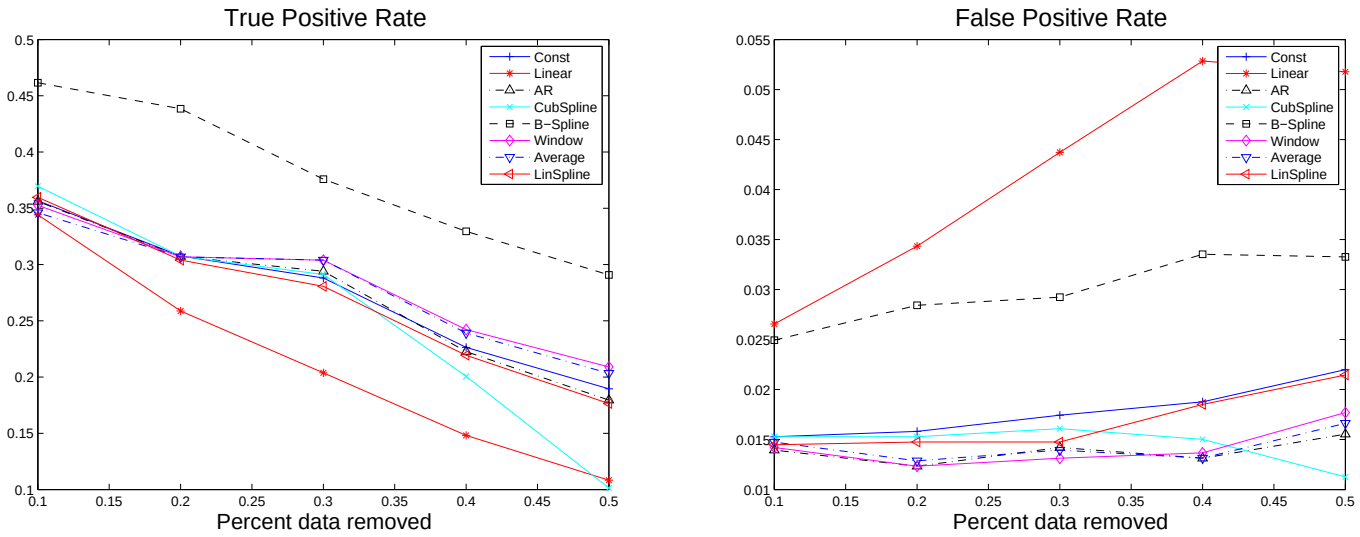
We ran a suite of experiments to test the various methods for imputation of missing and asynchronous data presented in Sec. 2. Each imputer-detector combination was evaluated on each trace under various parameterizations of each of the four modes of data filtering (see Sec. 4): NONE, RANDOM, PERIODIC, and REDUNDANT. Aside from NONE, each of these reduces the amount of data observed by the central detector, and each captures a different operating regime from the others. The results under filter mode PERIODIC were similar to those obtained under filter mode RANDOM, so we only report findings from RANDOM and REDUNDANT modes. For each trace dataset, we further ran each experiment with 0, 15, 35, and 70 synthetic anomalies injected into the trace. Results were qualitatively similar for differing injection levels, so we report results only for tests in which we inserted 70 anomalies or no anomalies.

Our primary evaluation metrics are the true positive rate (the fraction of true anomalies detected) and the false positive rate (the fraction of non-anomalies marked as anomalies). To evaluate whether a detected anomaly is a true anomaly, we compare to the set of anomalies we synthetically inject as well as the original set of anomalies in trace data. We extract the original set of anomalies by performing subspace anomaly detection on the full origin-destination flow matrices.

Fig. 4 shows the anomaly detection performance of the different imputation methods we consider, where the anomaly detection is done using the subspace projection algorithm of Sec. 3 with significance level $\alpha = .001$. On the left side of the figure, we plot the true positive rate for each of the different imputation algorithms as we increase the amount of data missing from 10% to 50%. On the right, we plot the false positive rate, which gives the fraction of points each algorithm erroneously flags as anomalous. For these experiments, we randomly remove monitor measurements. From the figure, we see that most of the imputation methods exhibit similar performance with a few notable exceptions. B-splines imputation records the highest true-positive rate, but its significantly higher false-positive rate is troubling. At a false positive rate of just 2.5%, B-splines makes 25 errors on a 1000 sample set of data. Thus, with a true positive rate of 45%, B-splines classifies correctly about 32 of 70 anomalies and incorrectly identifies 25. Linear propagation was the worst performer at this task, with the lowest true positive rate and highest false positive rate. Among the remaining imputation methods, window average most consistently demonstrates the lowest false positive rate and highest true positive rate.

Fig. 5 shows similar results to Fig. 4, but instead of using subspace projection to detect anomalies, we use AR detection (see Sec. 3.2) with significance level $\alpha = .05$. In this case, the choice of imputation method had little effect on the true positive rate, while false positive rate varied considerably from method to method. In particular, spline-based methods and constant propagation methods are outperformed significantly by AR imputation, windowing, averaging imputation. It is also worth noting that the true positive rate for the detection using an autoregressive model is around 20% more than that for the subspace projection methods, and the false positive rates are only slightly (less than .5%) worse. At least in this regime, then, it seems that principled detection of anomalies via an AR model is more effective than examination of the principal components of variation in the data.

Finally, Table 1 shows the performance of our various imputation methods under subspace projection, without anomalies injected, and using REDUNDANT filtering. Included also is the monitor imputation mode of [12] described in Sec. 3.1.3. We see that monitor imputation and averaging exhibit the best performance with identical true positive rates and the lowest false positive rates. Notably, monitor imputation enjoys a significant advantage over our methods: it computes its prediction value using complete data, while our methods must infer predictors from an incomplete data stream. On the other hand, monitor imputation must transmit twice as much data in the redundancy filtering setting as one of



(a) (b)

Figure 4: Subspace projection detection. (a) True positive rates of different imputation algorithms as percentage of missing data grows. (b) False positive rates as missing data grows.

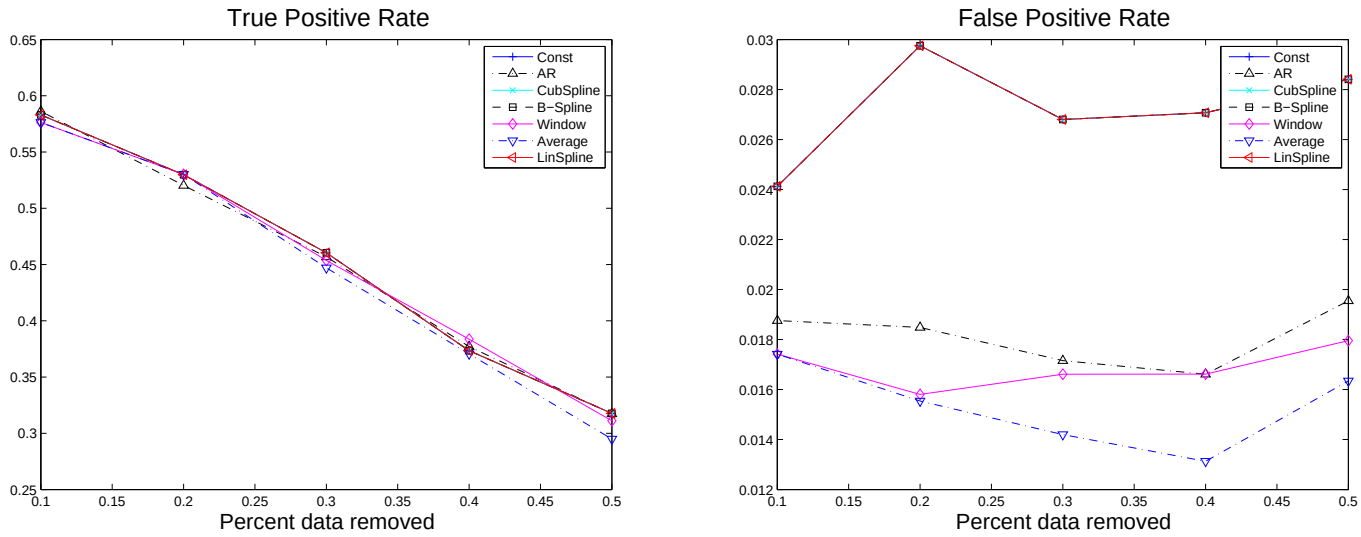
our imputation methods, as R_i must be packaged along with v_t. Hence, utilizing averaging or one of the other high performing methods under the redundancy filtering framework could halve the communication cost of network anomaly detection with little loss of accuracy.

6. RELATED WORK

Lakhina et al. [16] presented the original work on PCA-based anomaly detection. Zhang et al. [19] further extend the framework, showing how to infer network anomalies in both spatial and temporal domains. Similar to our work, they assume a distributed set of monitors that send link volume measurements to a central coordinator for anomaly detection. In principle, this approach works very well, and both [16] and [19] give experimental results validating their approaches. A limitation of the continuous synchronous update is its lack of scalability. In order to function, monitors must continually send measurements to the central coordinator, which increases network load when many monitors are present, or when measurements occur on small time scales. Huang et al. [12] address the scalability issue. They describe a method that allows monitors to withhold measurements that have not changed significantly. The quantization is achieved through parametrised local sliding filters at monitors. An analysis based on stochastic matrix perturbation theory is then used to determine a local sliding filter parameter that bounds the detection error rate.

A limitation of the monitors communicating to a centralized server not addressed by any of [16, 19, 12] is the method's reliance on synchronous measurements at regularly spaced time points. In [12] for example, the data are link-wise byte counts measured at 10-minute intervals. While [12] shows how local nodes can withold measurements such that the global detection error is bounded, their method relies on discrete time points and the assumption that monitors withhold measurements because the measurements are not changing significantly; we avoid both of these limitations through interpolation and constructing a joint autoregressive model of the system. Beyond this, our asynchronous framework subsumes issues of missing and censored data.

Certainly there is a large literature for anomaly or outlier detection in time-series in statistics and machine learning. Many sophisticated (such as [1], which used a wavelet-based detection method) and unsophisticated (i.e. [5], which uses simple Holt-Winters forecasting) methods have been applied to network anomaly detection. See also, for example, [11, 15, 18]. All these assume a standard time-series model of synchronous data arrivals with full data. Sketch-based change detection [15] aggregates data from multiple streams into a probabilistic summary, or sketch, and might in principle be able to handle asynchronous data by hiding it within a sketch; the authors do not address this, however. To our knowledge, previous papers on network anomaly detection such as the above and [19] that use autoregressive modeling techniques do so in an offline, non-dynamic fashion, which we handle via a few restrictions to our model class and give very efficient updates for.



(a) (b)

Figure 5: AR detection. (a) True positive rates of different imputation algorithms as percentage of missing data grows. (b) False positive rates as missing data grows.

Imputation Method	Const	Linear	AR	B-spline	Window	Avg.	Lin-Spline	Monitor
True Pos. Rate	0.4028	0.2500	0.4778	0.3292	0.4569	**0.4986**	0.3361	**0.4986**
False Pos. Rate	0.0175	0.1093	0.0268	0.0240	0.0178	**0.0173**	0.0178	**0.0170**

Table 1: Redundant mode error rates for imputation methods with Subspace Projection anomaly detection.

Finally, there are additional problems within the sphere of anomaly detection that we do not address. In particular, *anomography* [19] attempts to use routing matrices and information to infer true underlying system state. This could potentially allow localization of anomalies within a network, but we do not address this problem in the interests of keeping this paper manageable.

7. CONCLUSION

In this paper, we introduced the problem of anomaly detection in streaming data settings with frequent data loss and incomplete data. We proposed a general framework for solving the anomaly detection problem: first, to impute missing data, and second, to apply complete data anomaly detection methods. We believe there is ample future work to be done in this vein. The time-series literature is relatively mature, and seems to have a good hold of synchronous data; however, very little analysis exists for the case when data arrival is more sporadic yet still coupled. While our focus was on network traffic anomaly detection throughout this paper, there is a significant amount of sequential data that falls into our framework. For example, intrusion detection often requires monitoring a wide range of asynchronous states, patient-state assessment in the medical domain includes a huge number of signals with varying frequencies and interdepenence between signals. While our approaches clearly do not yet have the accuracy to be applied to medicine, we hope this will serve as a stepping stone toward future work on anomaly detection in asynchronous data settings.

References

[1] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *ACM Internet Measurement Workshop*, 2002.

[2] Garrett Birkhoff and Carl de Boor. Piecewise polynomial interpolation and approximation. In *General Motors Symposium of 1964*, pages 164–190. Elsevier, 1965.

[3] George Box, Gwilym Jenkins, and Gregory Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, fourth edition, 2008.

[4] Peter Brockwell and Richard Davis. *Introduction to Time Series and Forecasting*. Springer, second edition, 2002.

[5] J. Brutag. Aberrant behavior detection and con-

trol in time series for network monitoring. In *14th Systems Administration Conference*, 2000.

[6] Carl de Boor. *A Practical Guide to Splines*. Springer, revised edition, 2001.

[7] Jeremy Elson and Jon Howell. Handling flash crowds from your garage. In *USENIX Annual Technical Conference*, 2008.

[8] Michael Gleicher. A curve tutorial for introductory computer graphics. Notes for CS559, Wisconsin University, 2004.

[9] Gene Golub and Charles Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[11] C. Hood and C. Ji. Proactive network fault detection. *IEEE Transactions on Reliability*, 46(3), 1997.

[12] Ling Huang, XuanLong Nguyen, Minos Garofalakis, Michael I. Jordan, Anthony Joseph, and Nina Taft. In-network pca and anomaly detection. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 617–624, Cambridge, MA, 2007. MIT Press.

[13] J. E. Jackson and G. S. Mudholkar. Control procedures for residuals associated with principal component analysis. *Technometrics*, 21(3):341–349, 1979.

[14] I. T. Jolliffe. Principal component analysis. In *Principal Component Analysis*. Springer Verlag, New York, 1986.

[15] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation and applications. In *ACM Internet Measurement Conference*, 2003.

[16] Anukool Lakhina, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. In *ACM Internet Measurement Conference*, pages 201–206, 2004.

[17] J. Ramsay and B. Silverman. *Functional Data Analsysis*. Springer, 2005.

[18] A. Ward, P. Glynn, and K. Richardson. Internet service performance failure detection. *ACM SIGMETRICS Performance Evaluation Review*, 26(3), 1998.

[19] Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In *ACM Internet Measurement Conference*, pages 317–330, 2005.