A First Step Toward Understanding Inter-Domain Routing Dynamics

Kuai Xu
kxu@cs.umn.edu

Jaideep Chandrashekar
jaideepc@cs.umn.edu

Zhi-Li Zhang
zhzhang@cs.umn.edu

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN, USA

ABSTRACT

BGP updates are triggered by a variety of events such as link failures, resets, routers crashing, configuration changes, and so on. Making sense of these updates and identifying the underlying events is key to debugging and troubleshooting BGP routing problems. In this paper, as a first step toward the much harder problem of root cause analysis of BGP updates, we discuss if, and how, updates triggered by distinct underlying events can be separated. Specifically, we explore using PCA (Principal Components Analysis), a well known statistical multi-variate technique, to achieve this goal.

We propose a method based on PCA to obtain a set of clusters from a BGP update stream; each of these is a set of entities (either prefixes or ASes) which are affected by the same underlying event. Then we demonstrate our approach using BGP data obtained by simulations and show that the method is quite effective. In addition, we perform a high level analysis of BGP data containing well known, large scale events.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Monitoring*

General Terms

Measurement, Management, Design

Keywords

Routing, BGP, Root Cause Analysis

1. INTRODUCTION

BGP [1], the *de facto* Internet inter-domain routing protocol, is an *incremental* path vector protocol: routing updates (announcements and withdrawals) are generated *only* in response to *network events*, such as link or router failures (or repairs), session resets, policy changes, misconfiguration, etc. A BGP router may receive thousands of BGP updates on a daily basis, reflecting activity from all over the Internet. Given the critical nature of the Internet routing infrastructure, understanding BGP routing dynamics and the underlying "root causes" is crucial but, at the same time, extremely challenging.

A variety of events can trigger BGP updates. Some events may affect prefixes originated from only a few ASes (Autonomous Systems), generating only a small number of BGP updates. For example, a network failure in a *stub* AS will trigger route withdrawals only for destinations in that AS. On the other hand, a link or router failure within a tier-1 AS may trigger a large burst of BGP updates—the event not only affects network prefixes originated by the said AS but also those from its customers. Moreover, network events may occur concurrently or close together in time and BGP updates from these events are likely to become interleaved as they propagate across the Internet. Thus, a burst of BGP updates observed by a distant BGP vantage point can be caused by a number of unrelated events. On the flip side, a burst of *seemingly unrelated* BGP updates may actually be triggered by the *same* underlying event. For instance, a failed switch at an Internet Exchange Point may affect multiple ASes that do not have direct connections to each other, generating a burst of BGP updates whose AS Paths may not intersect. As another example, a major fiber cut (e.g., the Baltimore tunnel fire in 2001 [2]) may affect many ASes whose long haul connections happen to share a common fiber track, causing a flurry of "unrelated" BGP updates. These examples, together with the routing policy complications illustrated in [3], highlight the potential pitfalls and limitations of attempting to explain the (observed) updates. This is further underscored by recent work directed at locating routing instabilities [4, 5, 6, 7]. While these efforts have significantly advanced our understanding of routing dynamics, they also raise several questions about how to go about constructing a system that can diagnose and troubleshoot routing events on the Internet.

In this paper, we discuss the first step toward such a goal, which is also related to the problem of root cause analysis: *how to identify and separate BGP updates that are likely triggered by distinct events.* In this paper we explore a particular statistical method, Principal Components Analysis, and study its effectiveness in separating updates, collected at a vantage point, that have distinct underlying causes. Note

SIGCOMM'05 Workshops, August 22–26, 2005, Philadelphia, PA, USA.

that we do not attempt to explain or localize the underlying cause for the updates, but simply wish to group different prefixes or ASes such that all the members in a group are (most likely) affected by the *same* cause.

Validating *our* methodology, or any other for that matter, on actual Internet routing data is hard (the underlying events and related dynamics are unknown). We instead carry out extensive simulations to demonstrate that our method is indeed effective. The results of applying our algorithm to data thus obtained are very promising: our algorithm can identify groups of prefixes affected by the same event with high recall and precision. In addition, we also analyze real world BGP data that is known to contain large well-known events. Our initial results show that in most cases, there are other events (with smaller impact) that take place close in time to the reported events. This underscores the necessity of a methodology such as ours *prior* to applying any sophisticated root cause analysis algorithms.

A key *methodological* difference between our work and traditional efforts in root cause analysis is that ours is essentially a black-box approach, i.e., our *statistical* technique does not use information contained in the BGP updates, and instead relies on the temporal correlations between updates. We believe that such an agnostic methodology is important, at least in the initial stage, to overcome the lack of "fidelity" in BGP updates. Stated another way, information contained in BGP updates may be abstracted or even misleading (e.g., the AS triggering the BGP updates may not appear in the observed AS paths due to policies [3]). We argue that two prefixes being updated at (approximately) similar times is a stronger indicator, compared to similarities in the AS paths, that they were affected by the same underlying event. In the extreme case, the AS paths will have nothing in common if the prefixes share *physical*, but not *logical*, connectivity. We argue that the updates should be separated, at the early stage, *primarily* based on the existing temporal correlations between updates. Additional domain knowledge (and information in the updates) should be incorporated at a later stage in the root cause analysis, *after* the updates have been separated. Thus, the work that we present in this paper serves to complement the existing body of work in root cause analysis.

2. BACKGROUND

In this section, we first briefly describe BGP, focusing on details relevant to this paper. Subsequently, we motivate the reasons that lead to using PCA as a tool to study the problem of separating BGP updates.

2.1 Border Gateway Protocol

BGP is an *incremental*, *path-vector* protocol: once a session is established between neighboring routers, route updates are generated only in response to *network events*. Suppose a session between a pair of BGP routers fails, i.e., the event is a link failure, then the adjacent routers initiate BGP updates to their neighbors. These updates indicate how "reachability" to certain destinations has changed. For example, if the failure caused a loss of reachability to a destination network, the router will generate a *withdrawal* message, listing the network prefixes that have become unreachable. On the other hand, if the failure simply causes a path change (or if the router learns of a previously unknown destination), then an *announcement* is generated—containing a set of network prefixes and associated path attributes. A particularly relevant attribute is the "AS Path", which indicates both the origin AS for the prefix, as well as the sequence of ASes over which the route was propagated. Upon receiving a BGP update from a neighbor, a router might itself, after updating its own routing state, generate a *secondary* route update. Thus, by the mechanism just described, information about "events" propagates router by router through the network.

As a service to the networking community, public "collection" sites such as Route-Views [8] maintain BGP peering sessions with a number of routers in various ISPs and record the received updates. Each of these routers acts as a *vantage point* into the Internet. The logged updates reflect network events that have occurred somewhere in the Internet.

To draw an analogy, we call the time ordered sequence of updates observed at a single vantage point as a BGP *update stream*, and this will form the starting point of our methodology. Intuitively, each component in the update stream can be viewed as a "signal" associated with a particular variable (or prefix, since updates are at the level of prefixes). Seen another way, each event affects a number of prefixes, hence induces signals for a set of variables. Typically, at the location of the event (or close to it), these *related* signals are well correlated. However, due to various external effects such as network policies, timing delays, etc., when these signals are recorded at a distant vantage point, the original relationships may be slightly altered: signals for different variables induced by the same event may become less correlated, or signals for variables induced by different events may appear to be related. However, it is reasonable to expect that signals that are "related", i.e., induced by the same event will have more in common than "unrelated" signals. Our goal is to exploit correlations between these signals and identify groups of variables affected by the same event.[1]

2.2 Exploiting Correlations

As discussed previously, we expect that if a set of prefixes that are affected by the same event, updates for these prefixes would reach a given vantage point at approximately the same time. In other words, the updates for prefixes affected by the same event will share a lot of temporal similarity. There are a number of simple clustering techniques, i.e., k-means, agglomerative, etc., that may be used to this end. The drawback with these schemes is that in order to be effective, they require some a priori knowledge about the number of clusters. This is often hard to estimate; the dynamics hidden in the BGP data are exactly what we are trying to locate.

Principal Components Analysis is a well studied statistical technique which operates by analyzing variance between a given set of variables, which may be correlated and produces a new set of "latent", uncorrelated variables, which are called the principal components. Geometrically, PCA is an orthonormal transformation of a set of points where the variance of the original data in each direction is maximized. PCA is well suited to unsupervised tasks since it relies completely on the variance existing in the data. One of the classical applications of PCA is in "dimensionality reduction". In a typical dataset, few principal components

[1]For our purpose, an event is simply any underlying dynamic that affects a set of prefixes. Thus, a single instance of a link-failure is termed an event. So is a flapping link (affecting a different set of prefixes).

account for almost all the variance in the original data. This aspect makes it particularly attractive for the problem we study in this paper—we expect that few (underlying) events affect a large number of prefixes, generating a large number of updates. Moreover, since PCA "concentrates" variance in each independent dimension, this property can be effectively exploited to cluster the prefixes. In this paper, we explore the application of this technique toward the problem of separating updates triggered by different events. In the following, we provide a brief overview of the PCA method before we go on to describing how it is actually used (next section). A comprehensive description of PCA can be found in [9].

Suppose we have a time-series of n variables (sampled at t discrete time steps), $\mathbf{X} = [\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_n]^T$, represented as a $n \times t$ matrix. Each row corresponds to a variable and each column to a sample across the n variables. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be the rank ordered *eigenvalues* associated with the matrix $\mathbf{X}\mathbf{X}^T$ and $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ be the associated *eigenvectors*. Then the i-th principal component, denoted PC_i of $\mathbf{X}$, is obtained as $PC_i = \alpha_i^T \mathbf{X}$. Importantly, the variance captured by the i-th principal component is exactly described by its corresponding eigenvalue, i.e., $var(PC_i) = \lambda_i$. Also, α_1 is the direction along which the original data has the largest variance; the fraction of variance captured by this component is $\frac{|\lambda_1|}{\sum_i |\lambda_i|}$.

Since $\mathbf{PC} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T \mathbf{X}$, PCA transforms the space containing the "samples" of the n *observable* variables $\{X_i\}$ into a new space of n principal components (or *latent* variables) denoted $\{PC_i\}$, where the first variable PC_1 contains the most variance inherent in the original data, and PC_j, for $j = 2 \ldots n$, captures the maximal variance in the remaining data (after removing the contributions of the previous $j-1$ principal components). Informally, PCA re-expresses the original data in an *efficient* manner, i.e., *most* of the information in the original data is captured in a few PCs.

3. METHODOLOGY

In this section, we describe a methodology that takes an update stream, i.e., updates collected at a vantage point, as input and produces a set of "clusters", each of which is a set of prefixes or ASes that are *all* affected by the same underlying network event. First, we explain how to construct a time-series representation of the update stream. Given this representation, a number of statistical techniques may be applied. In this paper we primarily explore the use of PCA to cluster prefixes (or ASes) based on the raw temporal similarity in their updates.

3.1 Constructing the BGP Update Matrix

Given a stream of BGP updates recorded over an interval at a single vantage point, we first convert it to a (BGP) *update matrix*, $\mathbf{X}$, as follows. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of *all* prefixes for which at least one update (announcement or withdrawal) was observed in the interval. Also, let $A = \{a_1, a_2, \ldots, a_m\}$ be the set of corresponding *origin* ASes that these prefixes belong to.

To construct the (prefix based) time-series, we divide the observation interval into discrete bins of size δ. In this paper, we select $\delta = 30$, which corresponds to a popular choice in routers for the announcement rate limiting timer.

In each time slot j, we calculate the number of *distinct* updates associated with a prefix p_i and denote this as X_{ij}. In other words, X_{ij} is the number of updates seen in time j that affected prefix p_i. Furthermore we convert the matrix entries into standard units with the transformation: $\mathbf{X} = (\widetilde{\mathrm{X}}_{ij})$ with $\widetilde{\mathrm{X}}_{ij} = (\mathrm{X}_{ij} - \mu_i)/\sigma_i$. Then, each row $\widetilde{\mathrm{X}}_i$ is a time series with *zero* mean and *unit* variance; this represents the "relative" signal strengths associated with each prefix p_i. Also, the absolute value of $\widetilde{\mathrm{X}}_{ij}$ indicates how much the observed update signal in bin j differs from the overall (mean) signal strength seen over the entire stream.

Note that, with few exceptions, each prefix is uniquely associated with an "origin" AS. Thus, we can also construct the matrix $\mathbf{X}$ at a more aggregate level, in terms of the origin ASes, rather than prefixes. In this case, each row corresponds to a distinct origin AS, and each entry represents the number of updates for any prefix *belonging* to the AS.

The BGP update matrix thus constructed, which we can view as a discrete time series, is in a very general form. There are a number of statistical methods that may be applied to understand different relationships between the prefixes (or ASes). However, for reasons discussed previously, PCA seems particularly well suited for our purpose. In the next part we describe how we apply this particular technique to obtain the clusters described previously.

3.2 Using PCA to Extract AS Clusters

Applying the PCA algorithm to the BGP update matrix $\mathbf{X}$ returns a set of "latent" variables, which (abstractly) correspond to distinct events. However, we are more interested in obtaining the set of *original* variables, i.e., objects from the set P (or even A) that are affected by the same event.

Applying the PCA procedure to $\mathbf{X}$, we obtain n eigenvectors, and consequently the n (principal) components. Our first observation is that all the n PCs extracted from $\mathbf{X}$ are not equal. Those associated with larger eigenvalues contain more information (hence the term "principal") and are more "significant" than others. Deciding on the specific number of PCs depends to a large extent on the particular application and several heuristics are described in the literature [9]. Here, we wish to focus on events that have a *large* impact (affect a large number of prefixes or ASes and trigger a large number of updates). The heuristic that suggests itself is the so called Kaiser's criterion, stated as follows: select a principal component, PC_i, as being significant if $\lambda_i \geq \kappa$. Note that the set $\{\lambda_i\}_{1 \leq i \leq n}$ is rank ordered; thus if $j \leq n$ is the largest integer such that $\lambda_j \geq \kappa$, we have that $PC_i : i \leq j$ are significant, while $PC_i : i > j$ are not, and can be discarded. The intuition behind Kaiser's criterion is as follows: since $var(X_i) = 1$, $\lambda_j \geq \kappa$ implies that the corresponding principal component "explains" the variance of at least κ original variables. Having decided on the number of principal components to be retained, for convenience we refer to each component as a *major* or *significant* event.

The next step is to actually group the variables (prefixes or ASes). Our "clustering" algorithm uses the fact that each PC is simply a linear combination of the original variables (rows in the BGP update matrix), i.e.,

$$PC_i = \alpha_i^T \mathbf{X} = [\alpha_{i1} X_1 + \ldots + \alpha_{in} X_n]^T = [\sum_{j=1}^{n} \alpha_{ij} X_j]^T$$

For a *significant* principal component, say PC_i, the coefficient (PC loading) α_{ij} reflects the influence of PC_i on the

variance contributed by $\widetilde{X}_j$, i.e., the update signals from AS j. Let $\hat{a}_i = \max_{1 \le j \le n} \alpha_{ij}$ be the maximal value of the coefficients. We then select ASes that contribute approximately the same loading and place them in the same cluster. The underlying intuition is that signals that contribute (approximately) the same loading to a particular PC are most likely related, i.e., affected by the same event. To do this, we select all coefficients that are "close" to the maximal value, grouping the corresponding ASes that satisfy this condition into the AS cluster associated with PC_i. Following our earlier intuition, the underlying "event" captured by PC_i is most likely to affect those variables whose corresponding PC loadings are close to the maximal value; thus updates associated with these ASes are likely to be highly correlated.

Algorithm 1 EXTRACTCLUSTER($PC_i, \{\alpha_{ij}\}, \kappa, \epsilon$)

```
PC_i is such that λ_i > κ; CLUS = {};
a^+ = max_{1≤k≤n} α_ik; a^- = max_{1≤k≤n} |α_ik|
if a^+ ≠ a^- then
  for j = 1 to n do
    α_ij = −α_ij
  end for
  â = a^-
else
  â = a^+
end if
for j = 1 to n do
  if α_ij ≥ (1 − ε) * â and α_ij ≤ â then
    CLUS = CLUS ∪ {j}
  end if
end for
if |CLUS| > κ then
  return CLUS
end if
```

The detailed algorithm for extracting the cluster from a PC is given in Algorithm 1. The parameter ϵ is used to control the correlation "tolerance" among prefixes in the same cluster: smaller values of ϵ will admit more prefixes, with less strongly correlated updates, into a cluster.

The parameter κ, which controls which PCs are run through the algorithm, is used to control the "significance" of (inferred) events. For example, with $\kappa = 1$, only dominant PC_i's with $\lambda_i > \kappa = 1$ with an extracted cluster of size at least two are considered as "significant" events.

In general, for a given κ, we regard a PC_i (and its associated AS cluster) as *an inferred event (with respect to κ) if $\lambda_i > \kappa$ and the size of the resulting cluster $> \kappa$*. This definition reflects our intuition that "significant" events cause a larger number of updates (thus, larger variance) and affect more than a few prefixes (or ASes). While these notions are subjective, our intent is simply to provide a framework to differentiate events that are significant from those which constitute the data noise. In the next section, we apply our methodology to "simulated" update streams to verify its effectiveness in identifying the significant events.

4. SIMULATIONS

In this section, we describe details of how simulated BGP update traces were obtained, and the results of applying our algorithm on them. Testing our approach in a controlled setting helps us determine the "optimal" parameter settings, and also enables us to validate, and verify the *effectiveness*, of our methodology. The latter is much harder to carry out with real BGP data — there is little information about the underlying dynamics responsible for the updates. The key question we attempt to answer in this section is whether the events "inferred" by our (PCA based) algorithm do, in fact, correspond to actual events. As we show shortly, the answer is affirmative.

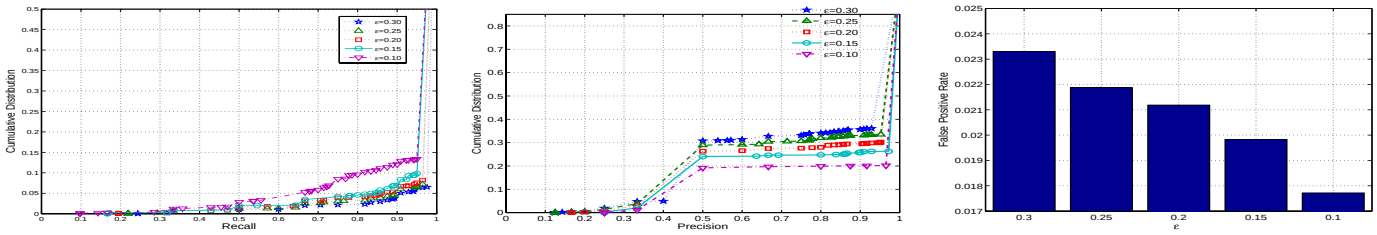
We carried out a large number of simulations, using different topology families, with the SSFNet simulation package [10]. In this section, we only discuss the results from a fixed 400 node power-law AS topology [11]. Each node in the topology represents an AS with a single router and originating a single destination prefix. Also, routers prefer routes with shorter AS paths. Finally, we attach a *vantage point* to a node in the graph which records all updates.

In each simulation, we generate a set of dynamic events (between six and eight) of two types: *major* events, which affect a large number of ASes; and *minor* events, which have a relatively small impact (and act as the noise). For *major* events, which are 60% of all events, a node is failed (once) and subsequently restored, i.e., a single on-off instance. In contrast, *minor* events are *persistent*; selected nodes are periodically failed and restored. Clearly, a *major* event affects reachability to many ASes, triggering a large number of updates, while a minor event affects only one or at most a few ASes (and this happens multiple times). The rationale for choosing this event-configuration comes from the analysis of Internet prefixes carried out by Ceasar et. al. [5].

To select candidate nodes for each type of event, we first construct the *path-set* for the vantage point. Simply, this is the set of paths used by the vantage point to reach each of the 400 destination nodes in the topology. Given this *path-set*, we compute the *transit cost* of each node as the number of "paths", in the *path-set*, that traverse it. Intuitively, the *transit cost* of a node is the smallest number of destinations that will be affected by the failure. If node x lies on the path, from the vantage point, to the set of destinations $\{a, b, c, d\}$, the transit cost for x is 4. Consequently when node x fails, we expect to see updates for each of the nodes in the set $\{a, b, c, d\}$. Clearly, the transit cost for a stub node (with degree 1) is 1. Thus, nodes that have a higher transit cost will have a greater impact on the topology when they fail or are repaired. To simulate *major* events, we select nodes with a high *transit cost* (and a low *transit cost* nodes for the *minor* events). The inter-arrival times between events are exponentially distributed (the mean is set to the convergence time for the topology). For minor events, the number of on-off cycles is selected such that there is at least one minor event occuring around the same time as a major event. However, the duration between cycles is fixed.

We performed over 300 simulation runs with this topology. In each instance we generated the required number of events, and at the end of the simulation, fed the updates collected at the vantage point into our algorithm. The output is a set of *AS clusters* which are, informally, the "inferred" events. We use the *recall* and *precision* metrics, described in the following, to evaluate our algorithm on the simulations. Note that in the simulations, since each node originates exactly one prefix and hence the BGP update matrix, whether constructed in terms of prefixes or origin ASes, is the same; we use the terms AS clusters and prefix clusters interchangeably.

From the topology itself, corresponding to each event, we determine the *expected* set of affected ASes. This is simply the set of ASes for which we expect to see updates for the event (when a node fails). Let a particular "observed" set of



(a) *Recall* (b) *Precision* (c) *False Positive Rate*

Figure 1: Performance of algorithm with varying ϵ, which controls tolerance during clustering.

ASes, i.e., an inferred AS cluster be denoted as S_o. With this set, we can associate an expected set S_e that best matches the observed set. Then we can define the recall as $\frac{|S_e \cap S_o|}{|S_e|}$ and precision as $\frac{|S_o \cap S_e|}{|S_o|}$, for each observed event.

4.1 Simulation Results

Next, we briefly discuss the *overall* results of applying our methodology on data obtained from simulation. In the results that follow, we set $\kappa = 2$, so that the algorithm only reports events that affect at least two ASes (or prefixes, equivalently). The generated simulation traces correspond to about 663 events, half of which were major events (as defined in the setup).

In Fig. 1(a), we plot the cumulative distribution of recall for different values of ϵ. The x-axis in the figure corresponds to different recall values. Notice in the figure that the overall recall is quite good (in the worst case, i.e., with $\epsilon = 0.10$, less then 15% of events have a recall lower than 0.95), and there is some variation across the different ϵ values). When $\epsilon = 0.10$, about 0.05% of events have recall lower than 0.7, while the recall is about 0.9 for the same fraction when $\epsilon = 0.30$. Also, at least 99.5% of events have a recall greater than 0.9 when $\epsilon = 0.30$. Clearly, larger ϵ translates into higher recall. Note that ϵ affects the "tolerance" of the factor loading, i.e., how much the loading for a variable can differ from the the maximal loading, and still be included in the cluster. Thus by increasing ϵ, we allow greater latitude in the factor loadings while including ASes in the cluster. Including more ASes, by increasing ϵ, cannot decrease the recall. However, the flip side of this effect is that precision decreases by allowing more ASes into the cluster.

Figure 1(b) plots the cumulative distribution of the precision metric for different ϵ values. Overall, the precision is still quite high–about 80% of inferred events have a preicision greater than 0.95 in the best case (with $\epsilon = 0.10$). In the worst case, i.e., $\epsilon = 0.30$, the precision is greater than 0.90 in more than 65% of all inferred events. However, changing ϵ has a larger impact on precision as compared to the recall. To see this in more detail, notice that when $\epsilon = 0.30$, 10% of the events have precision less than 0.35, and 70% of the events have precision greater than 0.5; in contrast, when $\epsilon = 0.10$, precision is less than 0.4 for 10% of the events, and at least 80% of the events have precision greater than 0.5. The explanation is that increasing ϵ makes it more likely that ASes are incorrectly included into a cluster. While these (erroneously) included ASes do not impact the recall, the additions increase the observed event size, which is the denominator in the expression for *precision*.

The corollary to this observation, i.e., increasing ϵ lowers precision, is that the false positive rate also increases the same way. An inferred (observed) event is a *false positive* if it cannot be matched with any real event. Fig. 1(c) plots the fraction of events that are false positives when different ϵ values are used in the algorithm. Clearly, larger ϵ allows ASes to be *incorrectly* included into an event cluster, i.e., ASes with weaker temporal similarity may be grouped together. In such cases, while the dominant AS cluster (which is the inferred cluster that best "covers" the expected set) has high recall. However, smaller events are broken up and do not match any expected event.

From the results presented in this section, we see that: first, our algorithm performs very well in separating the ASes affected by distinct events, and secondly, the algorithm is not overly sensitive to the parameter ϵ. Thus, while there is a tradeoff between recall and precision for different ϵ, a reasonable value would be in the range $[0.75, 0.9]$. In practice, we expect that a network operator would subjectively decide on a value depending on whether recall or precision is more important.

5. CASE STUDY

To further validate our approach, we perform several detailed case studies of BGP routing data containing reported routing events. In almost all cases, we identified several smaller events having occurred around the same time. This clearly illustrates the pitfalls in applying simplistic methods to perform root cause analysis, without first separating the updates for different events.

In each case study instance, corresponding to large scale events reported on NANOG [12], we construct the BGP update stream for the interval containing the event. Subsequently, we ran our algorithm on the update stream. Unlike in the previous section, here we construct the BGP update matrix based on the origin ASes (rather than prefixes). The reason for this is that it is relatively hard to analyze topological similarity and spatial relationships between prefixes. Such information is key to understanding if the prefixes in a cluster are indeed related. On the other hand, spatial relationships about ASes are somewhat easier to understand from the Internet AS graph.

Table 1 summarizes six different case studies. The first two columns describe the subject header in the NANOG posting and when it was posted. The third column lists

Known events	Date	View	# ASes observed during the window	# of inferred events	Size of the most significant event
Network outage	07/21/2003	AS1221	487	11	182
Northeast blackout	08/14/2003	AS11608	587	15	118
Hardware problem	02/23/2004	AS6539	607	8	385
Peering link instability	05/25/2004	AS11608	225	12	31
Network unreachable	06/12/2004	AS1239	781	10	662
Route leaking	09/17/2004	AS6539	1333	14	1168

Table 1: Summary of six events obtained from NANOG used for case studies.

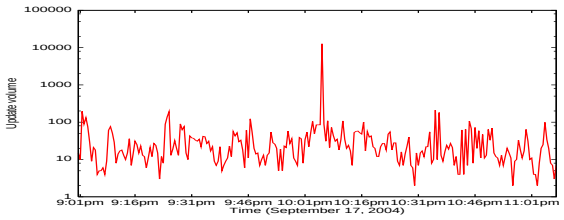


Figure 2: Update stream around the time of "AS22534 route leaking" event.

the particular vantage point used to construct the stream. Column 4 lists the number of origin ASes for which updates were observed over the interval of the event. Column 5 lists the number of events that were "inferred" as *major* by our algorithm (with $\kappa = 2$). Finally, the last column indicates the size of the most significant event, i.e., the size of the AS cluster extracted from PC_1. Note that in all of these cases, there are a multiplicity of events in the observation interval. Due to a lack of space, we only describe the detailed analysis for the last event in Table 1.

A "route leaking" event was reported (on NANOG) on Sept. 17, 2004. It was specifically reported AS22534 was leaking transit routes from AS3356 (Level 3) to AS6461 (Metromedia Fibre Network). The probable cause was a misconfiguration at a router inside AS22534. Figure 2 is a time series of the number of updates received at the vantage point. The dramatic spike soon after 10:00 p.m. was blamed on the reported event. However, our algorithm identified 14 distinct events in the same time interval, only one of which corresponds to the reported event. Upon further examination, we noted that the "spike" contains updates from 6 of the 14 events. In other words, there were 5 other events that occurred (very) near the time of the reported event. We can be quite certain that these events are unrelated since the update patterns for these events are distinct, and different from that seen for the route leaking event.

The take away message from here is that even though a single event can account for most of the updates observed at a given time, there may be other unrelated events around the time that contribute a relatively smaller amount. Analyzing these updates without first separating them may lead to incorrect inference of the root causes. However, as we have shown in this paper, we can use statistical correlation methods to separate updates associated with distinct events.

6. CONCLUSIONS

In this paper, we explored the use of Principal Components Analysis as a way to separate separate BGP updates triggered by distinct events. We tested our approach extensively with simulation traces and showed that our simple algorithm can separate events with a high degree of accuracy. In addition, we examined several case studies to further investigate the soundness of our approach.

An implicit assumption that we made in modelling the BGP update stream was that distinct events affect disjoint sets of prefixes or ASes, i.e., it is unlikely that two distinct events will affect the same prefix (or AS) in the same observation interval. While we believe this to hold in the majority of cases, there may be instances where this is not true. In such cases, our PCA based algorithm does not perform as well as if the events affect disjoint sets; this is related to the orthogonality of the new axes determined by the eigen vectors. We are currently investigating other techniques such as Independent Component Analysis and Kernel PCA, which can relax the requirement for orthogonality.

The paper describes a specific method to address, what is essentially the first step in the much harder problem of root cause analysis (of BGP updates). Thus, our work complements the existing techniques in the area. In the immediate future, we are trying to evaluate the improvement achieved by using our techniques prior to applying a more fine grained analysis as is describe in previous work [5, 6].

7. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under grants ITR-0085824 and CNS 0435444. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

8. REFERENCES

[1] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," Mar. 1995, rFC 1771.
[2] "2001 Baltimore tunnel fire," http://www.usfa.fema.gov/downloads/pdf/publications/tr-140.pdf.
[3] R. Teixeira and J. Rexford, "A measurement framework for pin-pointing routing changes," in *Proc. of ACM SIGCOMM Network Troubleshooting Workshop*, 2004.
[4] Di-Fa Chang and Ramesh Govindan and John Heidemann, "The temporal and topological characteristics of BGP path changes," in *Proc. of ICNP*, 2003.
[5] M. Caesar, L. Subramanian, and R. Katz, "Root cause analysis of Internet routing dynamics," U.C. Berkeley Technical Report UCB/CSD-04-1302, Tech. Rep., Nov. 2003.
[6] A. Feldmann and O. Maennel and Z. Mao and A. Berger and B. Maggs, "Locating Internet routing instabilities," in *Proc. ACM SIGCOMM*, 2004.
[7] M. Lad, D. Massey, and L. Zhang, "Link-rank: A graphical tool for capturing bgp routing dynamics," in *Proc. of NOMS*, Apr. 2004.
[8] "Routeviews," http://archive.routeviews.org.
[9] I. Jolliffe, *Principal Component Analysis (2nd edition)*, ser. Spinger Series in Statistics. Springer, 2002.
[10] "Scalable Simulation Framework," http://www.ssfnet.org.
[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-law relationships of the Internet topology," in *Proc. ACM SIGCOMM*, Aug. 1999.
[12] NANOG, "NANOG mailing list," http://www.nanog.org.