Identifying and Analyzing High Impact Routing Events with PathMiner

Giovanni Comarela
Boston University
Boston, USA
gcom@bu.edu

Mark Crovella
Boston University
Boston, USA
crovella@bu.edu

ABSTRACT

Understanding the dynamics of the interdomain routing system is challenging. One reason is that a single routing or policy change can have far reaching and complex effects. Connecting observed behavior with its underlying causes is made even more difficult by the amount of noise in the BGP system. In this paper we address these challenges by presenting PathMiner, a system to extract large scale routing events from background noise and identify the AS or link responsible for the event.

PathMiner is distinguished from previous work in its ability to identify and analyze large-scale events that may re-occur many times over long timescales. The central idea behind PathMiner is that although a routing change at one AS may induce large-scale, complex responses in other ASes, the *correlation* among those responses (in space and time) helps to isolate the relevant set of responses from background noise, and makes the cause much easier to identify. Hence, PathMiner has two components: an algorithm for mining large scale coordinated changes from routing tables, and an algorithm for identifying the network element (AS or link) responsible for the set of coordinated changes.

We describe the implementation and validation of PathMiner. We show that it is scalable, being able to extract significant events from multiple years of routing data at a daily granularity. Finally, using PathMiner we study interdomain routing over past 9 years and use it to characterize the presence of large scale routing events and to identify the responsible network elements.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring; C.2.5 [**Local and Wide-Area Networks**]: Internet – BGP

General Terms

Measurement

Keywords

BGP, Interdomain Routing, Boolean Tensor Factorization

IMC'14, November 5–7, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-3213-2/14/11.
http://dx.doi.org/10.1145/2663716.2663754

1. INTRODUCTION

The state of the interdomain routing system – that can be seen as the set of next-hops chosen by each AS (Autonomous System) toward each prefix – is driven by constant change, both automated (by the rules of BGP path selection) and human-mediated (by policy changes). Each routing change made by an individual AS is in response to some discrete event, such as a link failure or addition, a peer's route announcement or withdrawal, or a policy change. However the complexity of the resulting dynamics means that the causal relationship between routing changes in different parts of the system is notoriously difficult to tease out.

In this paper we present PathMiner, a system for identifying large-scale changes to the state of the routing system that are caused by individual events, and for narrowing down network elements (ASes or links) responsible for the set of changes. By 'large-scale' we mean routing changes that involve many ASes and prefixes, and may re-occur at multiple times.

The central idea behind PathMiner is that when a set of ASes change their next-hop decisions to a set of prefixes in a coordinated fashion, especially when those same changes are repeated at multiple points in time, then it is very likely that the coordinated activity is ultimately caused by actions taken by a single AS or link. This is an application of Occam's Razor: when a large set of ASes all change their next-hop decisions for a large set of prefixes, it is unlikely to be a coincidence. Rather, the simplest explanation is that all the changes were ultimately triggered by the action of a single 'actor' (AS or link). Furthermore, as the size of the AS set and prefix set involved grows, causation by a single actor becomes even more likely. Hence, PathMiner looks for significant spatio-temporal patterns in BGP routing, extracts them from background noise, and identifies the network element most likely to be responsible for generating the pattern.

We start by formalizing the concept of high impact routing events, and showing how to translate the discovery of such events into the Boolean Tensor Factorization problem. The first component of PathMiner is a new algorithm for Boolean Tensor Factorization that is well suited for the kind of data that is derived from network routing changes.

The second component of PathMiner identifies the single actor that is responsible for each event. This second step crucially depends on the fact that the first step extracts a set of coordinated routing changes. The key insight is that over the set of all paths that participate in the routing changes, the network element having highest precision and recall as a classifier for changed paths is most likely the single actor responsible for the event.

We validate PathMiner by manually inspecting the extracted events and actors. For this we developed an automated tool for graphical reconstruction of the event, which depicts the changes

made to the subgraph that is induced by the set of ASes and prefixes involved in the event. While manual inspection is time consuming and imperfect, we know of no alternative, since existing systems for root-cause analysis are not capable of working with historical data, nor with sets of large-scale routing changes. Our validation finds that the actors identified for each event almost always agree with manual analysis.

Using PathMiner we perform an initial analysis of the last 9 years of interdomain routing data, sampled at a daily granularity. We show that PathMiner is capable of extracting large events, some of which involve over 100,000 coordinated routing changes. Taken together, these events constitute between 10% and 20% of all visible routing changes over time in the datasets we analyzed. Individual events can involve tens to hundreds of ASes and prefixes, and occur tens to hundreds of times in our data. For most of these events, PathMiner is able to identify a single actor (or a small set of actors) that is likely responsible for ultimately causing this coordinated activity.

One of the main contributions of our work is to provide evidence that large-scale events do exist and they also re-occur over long periods of time. Specifically, PathMiner exposes the existence of regions of the AS-level Internet that have similar dynamics towards sets of prefixes. To the best of our knowledge PathMiner is the first tool capable of exposing such facets of the Internet at a global scale. From an engineering point of view, such information may be valuable for network administrators, when making changes in their systems, by providing historical view of events related to similar actions.

A high-level view laying out the main stages in PathMiner is shown in Figure 1. In the rest of this paper, we describe each of the stages shown in the figure. First, Section 2 uses an example to motivate the development of PathMiner. Section 3 presents a formal definition for our problem. In Section 4 we present and describe how to process raw BGP data (corresponding to the first two stages in Figure 1). In Sections 5 and 6 we describe and present results of our event detection methodology (the next two stages in Figure 1). Section 7 describes our single actor identification strategy (the last stage of the figure). Finally, we discuss related work in Section 8 and present concluding remarks and future work directions in Section 9.

2. AN EXAMPLE

Before diving into the details of PathMiner, it is helpful to examine a typical example to provide intuition and motivate our approach.

Figure 2 shows a small subgraph representing the dynamics of a portion of the network with respect to routing towards two prefixes (hosted at AS42381 and AS44173, and shown in gray at the bottom). The figure captures routing dynamics over two consecutive days (April 30, 2013 and May 1, 2013).

This subgraph is a portion of an event extracted by PathMiner. The ASes along the top row of the figure and the prefixes at the bottom of the figure constitute the output of the first step of PathMiner. From the first to the second day, all of the ASes at the top change their next-hops toward all of the prefixes at the bottom. In fact, the full event is quite large, involving dozens of ASes that all change their next-hops; we have extracted these ASes which show behavior that is typical of all the others.

A directed edge in the graph denotes the fact that the first AS uses the second AS as its next-hop for (each of) the two prefixes. Black (solid) edges refer to edges seen in both days, red (dashed) edges refer to edges seen just in the first day and green (dotted) edges refer to those seen only in the second day. Inside each node, the negative (positive) number shows the number of paths passing through the node in the first (second) day of the event (Note that the path counts reflect the full event, which involves many ASes not shown.)

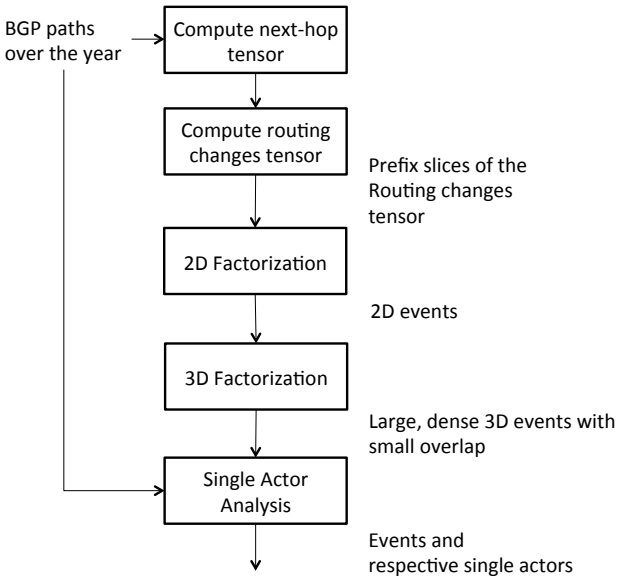


Figure 1: Stages of PathMiner.

Figure 2 shows that on April 30, most of the paths towards prefixes 1 and 2 were passing through AS6939 (Hurricane Electric). In the next day however paths are more dispersed; some pass through AS3549 (Global Crossing), some through AS3257, some only through AS174 (Cogent), and some pass only through other ASes, not in the figure, which connect directly to AS29632. In other words, ASes are switching from AS6939 to other ways to reach AS29632. We can conclude that either (a) AS6939 made its routes to the subject prefixes unattractive or unavailable, or (b) AS29632, as well the others mentioned above, took actions to make their routes more attractive or available.

This is interesting as a single event, as it shows a large-scale reorganization of the network with regard to two prefixes hosted by different ASes. However, it becomes even more interesting when we note that the same event (or its reverse) happened 28 times during 2013. From these 28 days PathMiner identified AS6969 and AS29632 as responsible by the event in 18 and 10 days respectively.

To illustrate how PathMiner finds such event, consider Figures 3(a) and 3(b). These plots show all points in time during 2013 (on the x axis) where each AS (on the y axis) changes its next-hop towards one prefix. We treat each plot as a binary matrix, in which element (j, k) is 1 if the AS represented by row j changes it next-hop towards the subject prefix between the days k and $k + 1$.[1] It is clear that each matrix consists of noise plus a strong signal; that signal is extracted and shown in Figures 3(c) and 3(d). Because Figures 3(c) and 3(d) are very similar, they together represent the fact that a group of many ASes changed their next-hops towards both prefixes synchronously, multiple times in 2013.

[1]A change is related to the observed next-hops at two instants of data collection, the first on day k, and the second on day $k + 1$. Transient changes, between the two data collection points, are not considered in this work. Details in Sections 3.2 and 4.

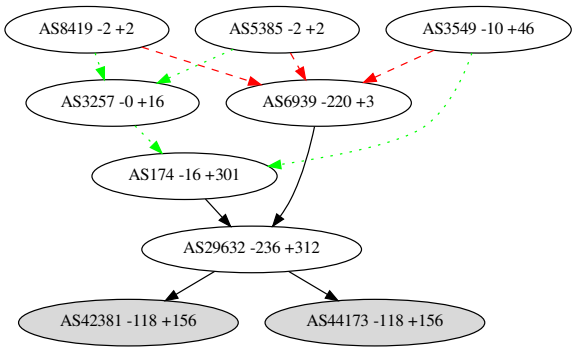


Figure 2: Path changes summary in the network towards two prefixes (hosted at gray nodes) from Apr-30-2013 to May-01-2013.

This simple example shows the nature of the kinds of events captured by PathMiner. It also illustrates the key challenges that PathMiner must overcome: *i)* how can we extract signal from noise (e.g., going from Figure 3(a) to 3(c))? *ii)* next, how can we find prefixes with similar signal matrices (e.g., matching Figure 3(c) with 3(d))? and *iii)* once multi-AS/prefix/time events are extracted, how can we identify the AS or link most likely to have triggered all of the routing changes captured in the event (as shown in Figure 2)? In the following sections we present the solutions taken by PathMiner to these challenges.

3. NOTATION AND DEFINITIONS

In this section we present the mathematical notation/definitions (Section 3.1) used throughout the paper and a formal definition of the problem we aim at solving (Section 3.2).

3.1 Notation

In this paper scalars will be denoted by lower-case letters (a), sets by upper-case letters (A), vectors by lower-case bold-face letters ($\mathbf{v}$), and matrices by upper-case bold-face letters ($\mathbf{M}$). We will also work extensively with 3-dimensional arrays, or *tensors*,[1] which we denote by upper-case calligraphic letters ($\mathcal{T}$).

A tensor may be seen as a collection of slices (matrices), fibers (vectors) or elements. More specifically, for the n-by-m-by-l tensor $\mathcal{X}$, we use $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$ and $\mathbf{X}_{::k}$ respectively to denote horizontal, lateral and frontal slices. In the same way, $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$ and $\mathbf{x}_{ij:}$ denote column, row and tube fibers respectively. Finally, x_{ijk} (and with same meaning $x_{i,j,k}$, $\mathcal{X}_{ijk}$ or $\mathcal{X}_{i,j,k}$ depending on convenience) denotes the element (i, j, k) of $\mathcal{X}$. In all cases above, $i = 1, \ldots, n$, $j = 1, \ldots, m$ and $k = 1, \ldots, l$.

Tensor $\mathcal{Y}$ is an induced tensor from $\mathcal{X}$ if there exist sets $A = \{a_1, \ldots a_{n'}\}$, $B = \{b_1, \ldots, b_{m'}\}$ and $C = \{c_1, \ldots, c_{l'}\}$ such that $y_{ijk} = x_{a_i,b_j,c_k}$. In this case we say that sets A, B and C induce $\mathcal{Y}$ in $\mathcal{X}$ and we write $\mathcal{Y} = \mathcal{X}(A, B, C)$. The same meaning holds if we use vectors instead of sets (with the provision that when using vectors, the ordering of indices matters, which is not the case when using sets).

Unless otherwise stated, operations over tensors are defined analogously as operations over matrices. Specifically, for a n-by-m-by-l tensor $\mathcal{X}$, we denote its size as (n, m, l), its volume as $vol(\mathcal{X}) = n \times m \times l$, and its Frobenius norm as $||\mathcal{X}|| = \sqrt{\sum_{i,j,k} x_{ijk}^2}$. We will also frequently refer to the *density* of a tensor, which is the fraction of its entries that are nonzero. In the particular case of binary tensors, this is $den(\mathcal{X}) = \frac{\sum_{i,j,k} x_{ijk}}{vol(\mathcal{X})}$.

[1]The word tensor is in general used to refer to N-dimensional arrays. In this paper we deal only with $N = 3$. More details about tensors can be found at [8].

3.2 Problem definition

Our starting point is the path-based nature of BGP (Border Gateway Protocol), in which ASes keep information about the preferred paths to each reachable prefix [19]. From a perspective of ASes and prefixes, at a given time t the global state of the system can be defined as: a set of ASes A, a set of prefixes P, and for each $a \in A$ a set of AS-paths, each of which allows a to reach a prefix in P. Another representation that we will also use is based on next-hops. In this representation, the state of the system consists of a set of tuples (a, b, p) where $a \in A$ uses $b \in A$ as the next-hop to reach $p \in P$. The next-hop representation of the system contains less information than the preferred-paths representation, so they are not equivalent, but each will be more convenient for certain parts of PathMiner.

To formalize the next-hop representation, at a given time t let $\mathbf{N}$ be a multivalued n-by-m matrix of next-hops in the network, where $\mathbf{N}_{ij}$ denotes the set of next-hops used by AS j to reach prefix i. Observing $\mathbf{N}$ over a discrete set of l points in time yields a n-by-m-by-l multivalued tensor $\mathcal{N}$, where $\mathcal{N}_{ijk}$ is the set of next-hops used by AS j to reach prefix i at time k. Tensor $\mathcal{N}$ represents the complete dynamics of the network over the set of measurements.

By comparing (frontal) slices of $\mathcal{N}$, we can identify next-hop changes in the network. This results in a binary tensor $\mathcal{C}$, which is the n-by-m-by-$(l-1)$ tensor of *routing changes*, defined as:

$$\mathcal{C}_{ijk} = \begin{cases} 1, & \text{if } \mathcal{N}_{i,j,k} \neq \mathcal{N}_{i,j,k+1}, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

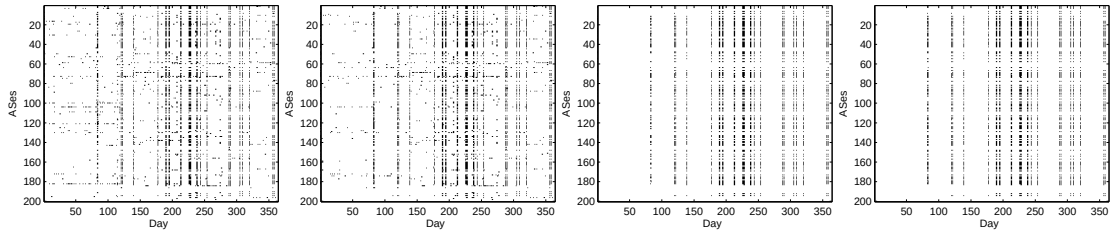
Given these definitions, we can define a *high impact event* in the global routing system as: sets I (of prefixes), J (of ASes) and K (of points in time) such that the sub-tensor $\mathcal{C}(I, J, K)$ has large volume and high density. Because $\mathcal{C}(I, J, K)$ has large volume, it has potential for high impact – many routing changes might be involved. The fact that $\mathcal{C}(I, J, K)$ has high density means that it is likely to be a singular event – that is, most ASes are changing their next-hops toward most prefixes at most timepoints, and as argued above, such unusually coordinated activity is likely due to the actions of a single network element.

To make this definition concrete, we introduce the concept of a (λ, ν)-event:

DEFINITION 1. *((λ, ν)-event) A binary tensor $\mathcal{B}$ is a (λ, ν)-event with regard to binary tensor $\mathcal{X}$ if there exist sets I, J and K such that $\mathcal{B} = \mathcal{X}(I, J, K)$; $den(\mathcal{B}) \geq \lambda$; and $vol(\mathcal{B}) \geq \nu$.*

Definition 1 still is not enough to fully characterize the events we are seeking. For instance, two distinct (λ, ν)-events $\mathcal{B}$ and $\mathcal{B}'$ may be such that $\mathcal{B}$ is a sub-tensor of $\mathcal{B}'$ and hence, the former can be viewed just as redundant information when compared with the latter. Therefore, it is also necessary to put constraints on the set of (λ, ν)-events we want to find. Thus, the final description of our problem, which we call Boolean Tensor (λ, ν)-Factorization (or (λ, ν)-BTF) is:

PROBLEM 1. *(Boolean Tensor (λ, ν)-Factorization) Given a binary tensor $\mathcal{X}$, integers r and ν and a real λ, the Boolean Tensor (λ, ν)-Factorization problem consists of finding r triples of sets (I_h, J_h, K_h), $h = 1, \ldots, r$ such that:*



(a) Prefix 1, all changes (b) Prefix 2, all changes (c) Prefix 1, signal (d) Prefix 2, signal

Figure 3: Next-hop changes towards the two prefixes of Figure 2 during 2013. An observed next-hop change, by AS j, towards the subject prefix at day k implies value 1 for the element (j, k).

i) $\mathcal{X}(I_h, J_h, K_h)$ *is a* (λ, ν)*-event in* $\mathcal{X}$*, for* $h = 1, \ldots, r$*; and*

ii) $\left|\left| \mathcal{X} - \bigvee_{h=1}^{r} \mathcal{X}^{(h)} \right|\right|$ *is minimized, where:* $\mathcal{X}^{(h)}$ *is a binary tensor with same size as* $\mathcal{X}$*, and* $x^{(h)}_{ijk} = 1$ *iff* $(i, j, k) \in I_h \times J_h \times K_h$*; and* $\bigvee_{h=1}^{r} \mathcal{X}^{(h)}$ *is defined as the elementwise logical* or *of* $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(r)}$.

The intuition behind the definition of Problem 1 is to find a set of r binary tensors that best approximate $\mathcal{X}$ as blocks of 1's. Moreover, we are only interested in tensors related to (λ, ν)-events. More formally, $\bigvee_{h=1}^{r} \mathcal{X}^{(h)}$ is the best approximation for $\mathcal{X}$ that can be obtained with a rank-r binary tensor, when each $\mathcal{X}(I_h, J_h, K_h)$ is a (λ, ν)-event.

Unfortunately, easy ways to solve Problem 1 exactly are not known. Setting $\lambda = 0$ and $\nu = 0$, the problem is equivalent to the Boolean Tensor Factorization problem studied in [9], which is known to be NP-hard [18]. Therefore, the more general problem of (λ, ν)-BTF is also NP-hard.

In summary, this section sets out a definition that translates the general notion of *identifying high impact events in the global routing* into a specific problem. Unfortunately this definition highlights two difficulties: *i)* complete topologies of the global interdomain routing system over time are not available; and *ii)* computing an exact solution to Problem 1 is hard in general. In the next section we discuss the available data we used and how we address its incompleteness, and then in Section 5 we describe our heuristic algorithms for finding solutions to Problem 1.

4. DATASET DESCRIPTION

In this section we present the dataset we used, how we processed it, and its limitations. Our source of data consisted of BGP RIBs (Routing Information Bases) made available by RIPE [2] and the Route Views project [4].

We obtained data from 9 years, from the beginning of 2005 until the end of 2013, at a daily timescale. For each day and repository we obtained the RIB made available at 8am (or, if not available, the closest one). We made the arbitrary choice of 8am in order to have approximately 24 hours between routing information collected for each day. From the RIBs we extracted all records of the form *route dumping date*, *prefix* and *AS-path* (only data related to IPv4 prefixes). We stored this data on a 12-node cluster in a Hadoop Distributed File System (HDFS) and, for the most data intensive computations, we used Hadoop [1] and Spark [3]. Table 1 presents a summary of our dataset.

As described in Section 3, our first step is to obtain the next-hop tensor $\mathcal{N}$. To this end, for each entry of the form [date, prefix p, AS-path], where AS-path is given by $[AS_1, \ldots, AS_q]$, we computed the $q - 1$ 4-tuples [date, p, AS_i, AS_{i+1}], for $i = 1, \ldots, q - 1$. Each 4-tuple means that at time *date*, in order to reach prefix p, the source AS_i uses as next-hop AS_{i+1}.

Table 1: Dataset summary

Year	Prefixes	ASes	Size (GB)
2005	286723	23157	340
2006	331421	26188	470
2007	400784	29382	640
2008	438730	32929	806
2009	506978	36336	845
2010	543868	39339	939
2011	626312	43151	1154
2012	879730	46262	1397
2013	850997	49502	1745

Referring to Table 1, it is important to remark that we do not have routing information from every source to every destination at every point in time. In this context, missing data can arise mainly for two reasons: (a) data collection issues and (b) visibility problems (the RIPE and Route Views RIBs do not capture the complete AS-topology of the Internet). Note as well that the datasets considered are large – for recent years, over 1TB each in size.

Fortunately, both of these problems (missing data, and dataset size) can be significantly lessened by carefully selecting a representative subset of the data. Accordingly, we selected our data subset by greedily choosing ASes and prefixes with most of the next-hop changes in the network. To that end, using big data tools, we computed the tensor $\mathcal{C}$ ($\mathcal{C}_{ijk} = 1$ iff AS j changed next-hop towards prefix i from day k to $k + 1$) for each of the nine datasets in full. Next, we computed the total number of changes over the year for each prefix i (as a destination), each AS j (as a source) and for each pair (i, j). Figures 4(a) and 4(b) present the log-log complementary CDFs of the total number of changes for prefixes and ASes, respectively. It can be seen that for ASes there is a distinct subset of heavy-hitters that account for the majority of routing changes.

Going further, Figure 4(c) shows the cumulative number of changes for a pair (i, j) for the year of 2011. That is, value at row i and column j of the heat map represents the fraction of changes corresponding to the i prefixes and j ASes with most changes in the dataset. This figure provides information about how to greedily, with respect to the number of routing changes, obtain a sample of ASes and prefixes. With regard to ASes it is possible to see that including more than the top 200 ASes in the sample does not increase significantly the fraction of changes captured.

On the other hand, there are hundreds of thousands of prefixes that experience significant levels of routing changes. At the same time, it is also important to recognize that there exist many sets of prefixes for which routing information is essentially redundant. In

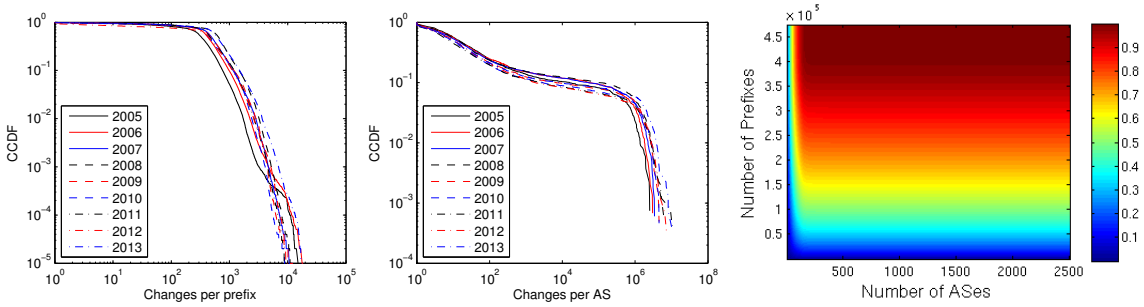


Figure 4: Distributions of the total number of next-hop changes: (a) per prefix; (b) per AS; and (c) per pair (AS, prefix), for 2011 only. In (c) value at (i, j) indicates the total number of next-hop changes from the first j ASes towards first i prefixes (ASes and prefixes are sorted, in decreasing order, by their number of changes).

Table 2: Summary of the sampled routing changes tensors

Year	Density (%)	Missing (%)
2005	0.9	7.4
2006	1.2	8.7
2007	1.5	6.7
2008	1.9	6.3
2009	1.7	8.1
2010	1.6	7.0
2011	1.7	5.9
2012	1.8	7.4
2013	2.3	5.5

particular, in many cases prefixes originated by the same AS are routed similarly [6]. For our problem, such sets of prefixes add no additional information about coordinated routing changes.

Hence, in our data subset we chose the top 20000 prefixes in terms of volume of changes, but only allowing at most one prefix hosted in each AS.[1] Since the 20000 prefixes come from 20000 distinct ASes, and there are only about 50000 ASes active in total, we expect that this subset captures a large fraction of the distinct and observable routing changes in our data. That is, the intuition behind our sampling strategy was to obtain a set of prefixes responsible for many changes, and at the same time avoiding the discovery of blocks with many prefixes belonging to the same AS and hence having identical routing changes.

Table 2 presents statistics about tensor $\mathcal{C}$ projected only over the samples; each sample tensor is of size 20000-by-200-by-364. Note that because the set of active ASes and prefixes has changed over the 9 year timespan, the sampling procedure was done independently for each year. Therefore, the sets of prefixes and ASes considered vary from year to year, in order to capture the maximum amount of routing changes, and hence results in the remaining of this work were computed independently for each sample.

Table 2 also shows the percent of missing data, i.e., missing next-hop entries in the tensor $\mathcal{N}$ (projected over the samples). Since these percentages were reasonably small, we handled missing data as follows. Suppose that at some point in time k we have $\mathcal{N}_{ijk} \neq \emptyset$ for some i, j and $\mathcal{N}_{i,j,k+1} = \emptyset$. Because data at time $k+1$ is missing, we cannot know for certain whether: *i)* AS j cannot reach prefix i at time $k+1$ (which would imply that $\mathcal{C}_{ijk} = 1$), or *ii)* AS j can reach i at time $k+1$ but our data set does not contain that information (which implies that $\mathcal{C}_{ijk}$ can be either 0 or 1). In light of this limitation we took a conservative approach and defined $\mathcal{C}_{ijk}$ to be 0. Therefore, a 1 in the tensor $\mathcal{C}$ means that we definitely observe a next-hop change in the data, but a 0 can either imply that a change did not happen, or that we simply do not know what happened. Choosing to set these entries to 0 also reflects that fact that zeros comprise the vast majority of known values in our data.

Note that the amount of uncertainty in the routing changes tensors ($\mathcal{C}$) is thus bounded by the 5 to 10% of missing data. This has implications in the next section for our event extraction. In particular, we should expect that even when a set of ASes all change their next-hops for a set of prefixes, our data may only show a portion (e.g., 90%) of those changes – i.e., the observed density of a valid event may be less than 100%.

Another aspect worth mentioning about the dataset is the presence of more than one next-hop from some ASes towards some prefixes. In fact, that was the motivation for our definition of next-hop change, given in Equation 1, where one can see that we compare sets, instead of elements. Other possible approach would be to use a finer granularity of the network, where the issue of more than one next-hop never arises. For instance, such granularity could be obtained by considering quasi-routers. Unfortunately, the use of quasi-routers would introduce problems hard to be addressed in our analyses. For example, which quasi-router observed at a day k corresponds to the quasi-routers observed at day $k+1$?

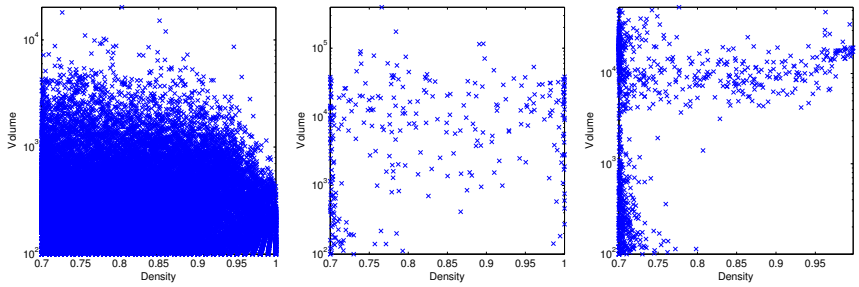
In order to provide means to understand how much the issue of more than one next-hop may affect our results, we computed the fraction of cases in which $|\mathcal{N}_{ijk}| > 1$ or $|\mathcal{N}_{i,j,k+1}| > 1$, during the computation of $\mathcal{C}_{ijk}$, in our nine datasets (before sampling). In all cases this fraction was below 8%. Furthermore, from the cases where $\mathcal{C}_{ijk} = 1$ we have that the same fraction is below 30% (in all datasets).

5. EXTRACTING EVENTS

As previously discussed, at a high level PathMiner consists of two main steps: (1) finding high-impact events, and (2) identifying the most likely network element causing the event. In this section, we describe our solution to step (1), which is an algorithm for finding good solutions to the (λ, ν)-BTF problem.

An attractive solution to the (λ, ν)-BTF problem would be to start from an existing algorithm for standard Binary Tensor Factorization. Unfortunately, the large size (combined with the density) of our binary tensors was too much for existing algorithms (e.g., in [9]) to handle. As a result, we developed a scalable heuristic to find (λ, ν)-events in the routing changes tensor $\mathcal{C}$.

[1]We considered the *host* of a prefix to be the AS advertising the prefix for the first time in each year.



(a) (b) (c)

Figure 5: Density *versus* Volume for 2D (λ, ν)-events obtained from slices of tensor $\mathcal{C}$, 2013 only. Prefix (horizontal) slices in (a), AS (lateral) slices in (b) and time (frontal) slices in (c). Experiments performed using Algorithm 1 with $\lambda = 0.7$, $\nu = 100$ and $\epsilon = 1\%$.

Our approach has two steps. First we look at individual slices of $\mathcal{C}$ and extract (λ, ν)-events within slices (Section 5.1). Then, we aggregate similar slice events in order to form events that span multiple slices (Section 5.2).

The algorithmic approach we adopt is well suited to the case where significant large, dense events exist in the data (as those presented in Figure 3). Thus it is well suited to the current problem, while other algorithms (including [9]) may be more appropriate when events are smaller and rarer. For the sake of brevity and due to space limitations we present only a high level description of our algorithms.[1]

5.1 2D Factorization

Let $\mathbf{X}$ be a slice of $\mathcal{C}$. It can be a prefix (horizontal), AS (lateral) or a time (frontal) slice. For example, $\mathbf{X}$ may be the prefix slice shown in Figure 3(a). The goal of 2D Factorization is to extract from $\mathbf{X}$ a set of large volume, high density patterns such as shown in Figure 3(c). For simplicity of notation we assume from now on that $\mathbf{X}$ is a prefix slice of $\mathcal{C}$ for prefix i. Description for AS and time slices proceeds analogously.

To start, we define a rank-1 binary matrix: given binary vectors $\mathbf{a}$ and $\mathbf{b}$, a rank-1 binary matrix $\mathbf{X}$ is given by $\mathbf{X}_{ij} = \mathbf{a}_i \times \mathbf{b}_j$. That is, a rank-1 binary matrix is the outer product of two binary vectors and can be thought of as a 'block' of 1s (after reordering of rows and columns).

The 2D Factorization step repeats the following as long as a significant fraction of 1s can be obtained from $\mathbf{X}$: Find a good rank-1 binary approximation for $\mathbf{X}$, denoted $\mathbf{M}$. If the set of rows J (representing ASes) and columns K (representing days) with non zero elements of $\mathbf{M}$ induces a submatrix $\mathbf{B}$ with volume at least ν and density at least λ, then label the triple $(\{i\}, J, K)$ as a (λ, ν)-event. Next, independently of volume and density of $\mathbf{B}$, remove all ones captured by the set of rows J and the set of columns K. This strategy is described more precisely in Algorithm 1.

The key challenge is to obtain the rank-1 approximation $\mathbf{M}$ (Line 5). In fact, this is equivalent to the Frequent Itemset Mining problem, which currently has not a known easy way to be exactly solved. Furthermore, the tensor $\mathcal{C}$ may have many thousands of slices, so the algorithm we use must run quite quickly. Hence we use the strategy of *relaxing* the discrete problem into a continuous one, solving the continuous problem, and thresholding the result to find an approximate discrete solution.

[1]Our implementation is publicly available and can be obtained at `http://cs-people.bu.edu/gcom/bgp/imc2014` or by direct request to the authors.

Algorithm 1: 2D-FACTORIZATION

Data: n by m binary matrix $\mathbf{X}$ related to prefix i, λ, ν and convergence parameter ϵ

1 $F \leftarrow \{\}$
2 $\mathbf{Z} \leftarrow \mathbf{X}$
3 **repeat**
4 $\quad \mathbf{Z}' \leftarrow \mathbf{Z}$
5 $\quad \mathbf{M} \leftarrow$ RANK-1-APPROXIMATION$(\mathbf{X}, \mathbf{Z}, \lambda, \nu)$
6 $\quad J \leftarrow \{j : m_{jk} = 1\}$
7 $\quad K \leftarrow \{k : m_{jk} = 1\}$
8 $\quad \mathbf{B} \leftarrow \mathbf{X}(J, K)$
9 $\quad$ **if** $den(\mathbf{B}) \geq \lambda$ **and** $vol(\mathbf{B}) \geq \nu$ **then**
10 $\quad\quad F \leftarrow F \cup \{(\{i\}, J, K)\}$
11 $\quad$ **for** $(j, k) \in J \times K$ **do**
12 $\quad\quad z_{jk} \leftarrow 0$
13 **until** $\|\mathbf{Z}\| = 0$ **or** $\frac{\|\mathbf{Z}-\mathbf{Z}'\|^2}{\|\mathbf{Z}\|^2} < \epsilon$;
14 **return** F

Our relaxation seeks a real-valued rank-1 approximation to $\mathbf{Z}$ (which initially is a copy of $\mathbf{X}$). For this we use Non-Negative Matrix Factorization (NNMF) [5]. Using NNMF we find real nonnegative vectors $\mathbf{w}$ (n-by-1) and $\mathbf{h}$ (1-by-m) such that $\mathbf{wh}$ is a real nonnegative rank-1 matrix approximating $\mathbf{Z}$. We then threshold $\mathbf{w}$ and $\mathbf{h}$ independently, obtaining the binary vectors $\mathbf{w}'$ and $\mathbf{h}'$, such that $\mathbf{M} = \mathbf{w}'\mathbf{h}'$ minimizes $\|\mathbf{X} - \mathbf{M}\|$ (see that the error is computed considering the original matrix). We note that once $\mathbf{w}$ and $\mathbf{h}$ are computed, computing $\mathbf{M}$ (by finding the optimal thresholding) can be performed in time $O(mn)$.

For the results in this paper, we ran Algorithm 1 for each of the 9 datasets with $\lambda = 0.7$, $\nu = 100$ and $\epsilon = 1\%$. We considered as input three different sets of slices: prefix slices, AS slices and time slices. Figure 5 summarize the results by presenting a scatterplot of density *versus* volume for the year of 2013 (same general comments apply for other datasets). While prefix slices give us events with a wide range of density and volume, AS and time slices behave differently. Basically, most of the AS and time events have low density or volume close to 10^4. The explanation is twofold: first, AS and time slices are harder to mine, since their size is significantly larger than the size of prefix slices; and second, many of the events on AS and time slices consist of one AS changing its next-hop towards all (or almost all) prefixes in the network. Hence, because prefix slices offer the best quality results (in terms of density) and reveal the most interesting structure in the data, we use only pre-

fix slices to generate the (λ, ν)-events used in the next stage of the algorithm.

5.2 3D Factorization

The next step is to find 3 dimensional (λ, ν)-events in the tensor $\mathcal{C}$. The input S, obtained using the method in the last section, is a set of triples of the form $(\{i\}, J, K)$, where i represents a prefix, J a set of ASes and K a set of points in time. The basic idea is to find triples having similar blocks (given by sets J and K) and to group them to create 3D events.

We consider that there may be two ways of combining a pair of blocks. First, they may be nearly identical – that is, their intersection may be nearly as large as their union. In that case, we merge them by constructing the block that contains them both. Second, their intersection may be much smaller than their union, but still sufficiently large in terms of absolute volume. In that case, we merge them by constructing the block that is their intersection.

To evaluate these two cases, we define two functions over pairs of triples $x = (I, J, K)$ and $x' = (I', J', K')$ as:

$$d_B(x, x') = 1 - \frac{|J \cap J'| \times |K \cap K'|}{|J \cup J'| \times |K \cup K'|} \quad (2)$$

and

$$s_B(x, x') = |J \cap J'| \times |K \cap K'|. \quad (3)$$

The distance function d_B measures the volume of the intersection of two blocks divided by the volume of their union. If close to zero, then J is similar to J' and K is similar to K'. Hence, it is natural to assume that $(I \cup I', J \cup J', K \cup K')$ is a larger event. We extend this strategy to merge multiple triples in a single step as follows: given a triple x, look for a set S' (which contains x) such that $\max_{y,z \in S'} d_B(y, z) \leq \gamma$, for some $0 \leq \gamma \leq 1$. Once S' is found we combine all of its elements at once using the following operator:

$$\textsc{Combine-Union}(S') = \left(\bigcup_{y \in S'} y_1, \bigcup_{y \in S'} y_2, \bigcup_{y \in S'} y_3 \right),$$

where y_1, y_2 and y_3 represent, for triple y, the sets of prefixes, ASes and points in time respectively.

The similarity function s_B captures the case when the intersection of two blocks is large enough by itself to merit merging the blocks. That is, it may be the case that triple x is not nearly the same as any other triple in S, but there still exists x' such that $s_B(x, x')$ is significantly large (larger than some threshold β). In this case, we may conclude that $(I \cup I', J \cap J', K \cap K')$ is an event. In fact, the second part of the algorithm looks for elements $x' \in S$ such that $s_B(x, x') \geq \beta$ and then combines them using the following operator:

$$\textsc{Combine-Inter}(x, x') = (I \cup I', J \cap J', K \cap K').$$

Alternating the test for maximum distance and minimum similarity and using COMBINE-UNION and COMBINE-INTER operators iteratively yields the discovery of new triples (possible representing new (λ, ν)-events) and is the core of our strategy to find 3 dimensional blocks in $\mathcal{C}$. The procedure is summarized in Algorithm 3 (presented in Appendix A).

One may note that Algorithm 3 does not check the density of new formed 3D blocks. This is because extracting the relevant portions of $\mathcal{C}$ for this check is so time consuming as to be prohibitively expensive for our datasets. Of course this absence of verification may lead to triples that induce blocks with low density. A second problem that may arise is that the extracted 3D blocks may still show significant overlap. In order to address these two issues we added a final stage that discards blocks with low density and blocks that do not add new information to the results because it overlaps with many others. This process is described in Algorithm 2, which greedily (by decreasing order of volume) selects only triples with a minimum density and that captures significant information not contained in blocks previously selected. Results and parameters set up related to the methodology presented in this section will be discussed in Section 6.

Algorithm 2: EVENT-SELECTION

Data: Tensor $\mathcal{C}$, S, set of triples of sets of the form (I, J, K) and thresholds λ and ν

```
L ← sort triples (I, J, K) ∈ S in decreasing order of volume (|I| × |J| × |K|)
S' ← ∅
for i = 1 to |L| do
    (I, J, K) ← L_i
    B ← C(I, J, K)
    if vol(B) ≥ ν and den(B) ≥ λ then
        foreach (i, j, k) ∈ I × J × K do
            C_ijk ← 0
        S' ← S' ∪ {(I, J, K)}
return S'
```

6. CHARACTERIZING EVENTS

After the execution of the set of algorithms presented in the previous section, we have a collection of 3D events extracted from from the routing changes tensor $\mathcal{C}$, each one with large volume and high density. In this section, we briefly pause to characterize the events found by PathMiner in the 9 years of routing data.

After experimentation we settled on the following parameters for the algorithms presented in Section 5.2: $\lambda = 0.8$, $\nu = 100$, $\gamma = 0.1$ and $\beta = 100$.[1] Table 3 summarizes the overall performance of PathMiner when using these parameter settings on our data. The column '# 1's Retrieved' is the total number of routing changes that were contained in events, and the 'Percent' column is the fraction of all routing changes in our data that were contained in events.

Table 3: Summary of (λ, ν)-events

Dataset	#Events	#1's Retrieved	Percent
2005	5255	1107109	8.2
2006	6823	1689299	9.5
2007	8252	2504558	11.1
2008	7996	2411041	8.3
2009	8602	2466807	9.7
2010	9646	2952688	12.6
2011	12042	3991264	16.3
2012	13910	4611049	17.8
2013	13992	5880885	17.7

The table shows that PathMiner is able to find many blocks, comprising a significant fraction of the routing changes contained in the datasets, ranging from 8.2% in 2005 up to 17.8% in 2012. Figures

[1]For instance, using $\beta = 200$ and $\gamma \in \{0.2, 0.05\}$ did not change the results significantly in terms of percentage of retrieved next-hop changes, average volume or average density. Tests performed with 2013 dataset.

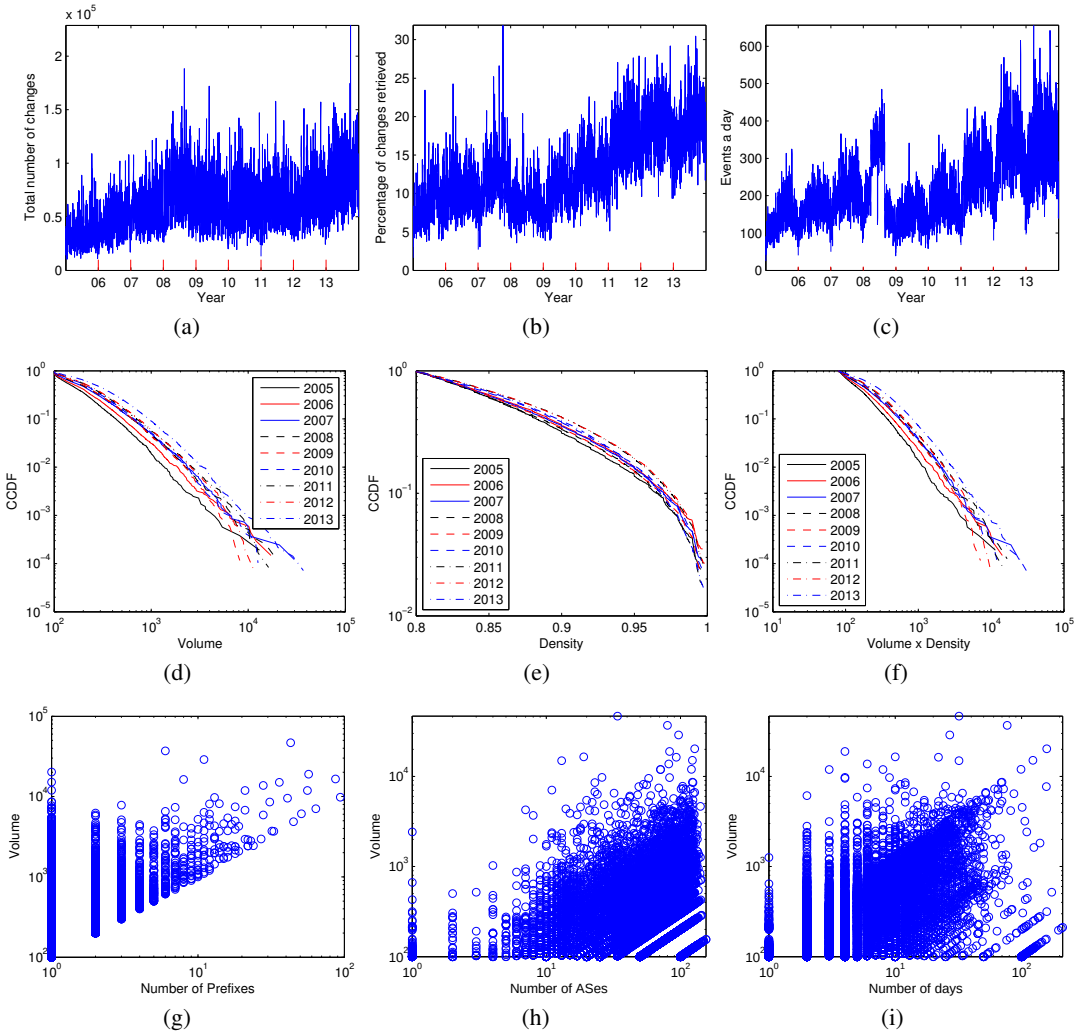


Figure 6: Basic statistics about (λ, ν)-events found by PathMiner (λ = 0.8 and ν = 100). Time series of: (a) total number of next-hop changes in each yearly sample; (b) percentage of these changes captured in events by PathMiner; (c) number of events found in each day. Complementary CDFs of: volume of events – volume is defined as the product of the number of prefixes, ASes and days in a event; (e) density of events – density is the number of changes captured by an event over its volume; (f) product volume $\times$ density of events. Event scatterplots (2013 only): (g) number of prefixes *versus* volume; (h) number of ASes *versus* volume; (i) number of days *versus* volume.

6(a), 6(b) and 6(c) present the same statistics on a daily basis as a time series over the 9 years of data. Note that the samples for each year are distinct so trends should not be inferred across years. From Figures 6(b) and 6(c) we can see that PathMiner performs better in tensors with higher density and that overall, the shape of the time series related to the fraction of changes and the number of events a day is similar to the shape of the time series related to the total number of changes (higher values in 2008 and in the last three years, 2011, 2012 and 2013).

Next, we move to analyzing density and volume of events. As an algorithmic constraint we have that each block has density higher than 0.8 and volume greater than 100. But, in general, how large and dense are the events? Figures 6(d), 6(e) and 6(f) present the log-log complementary CDF for volume, density and their product, respectively, for all 9 datasets. In general the distributions are long-tailed, with many small events and a few very large events. In terms of volume we can see that for 2005 more than 10% of all blocks have volume greater than 500 and that this number goes up to 1000 in 2013. In the tail of the curves we can see some huge events with volume greater than 10^4. Similar comments can be made for Figure 6(f). In terms of density, although 0.8 can be considered a high threshold, there are many blocks that are much denser. For instance, approximately 30% of all events (in all datasets) have density higher than 0.9 and 10% are higher than 0.95 in density.

We also explore the relationship between the volume of events and the number of prefixes, ASes and days that the event contains. Figures 6(g), 6(h) and 6(i) present the results for the year of 2013 (results for other years follow the same trend). It can be seen that the extracted events include large numbers of days and ASes (obtained via the 2 dimensional factorization algorithm) and that events with many prefixes are formed as a result of the 2D event aggregation step.

Finally, Table 4 presents the 5 largest events, in terms of volume for each year. Columns 2 to 4 represent the number of prefixes, ASes and days in each event (thus the volume of the event can be obtained by their product). The table shows that PathMiner was able to find events of remarkable scope, some of which involve dozens of ASes, prefixes, and days. The largest events (in

2007 and 2008) involve over 100,000 individual routing changes. In this respect it is important to recall as well that each prefix in our datasets is originated in a distinct AS, so the actual number of routing changes per event in the full data is much larger because of the similar routing behavior of co-originated prefixes.

Table 4: Description of top-5 events over 9 years

Dataset	#Prefixes	#ASes	#Days	Density
2005	2	54	200	0.90
	2	60	109	0.88
	1	59	138	0.83
	1	57	95	0.87
	2	52	48	0.91
2006	3	68	168	0.86
	82	11	19	0.83
	1	66	183	0.86
	3	58	61	0.91
	1	65	156	0.85
2007	140	49	20	0.80
	13	52	44	0.81
	75	8	35	0.89
	77	5	28	0.80
	45	37	6	0.91
2008	156	28	25	0.88
	79	41	6	0.85
	15	44	23	0.83
	22	37	13	0.83
	10	103	10	0.88
2009	45	102	2	0.80
	1	90	91	0.87
	13	51	11	0.81
	6	89	13	0.80
	14	48	10	0.83
2010	49	38	16	0.90
	28	41	11	0.97
	60	101	2	0.88
	98	15	8	0.94
	17	43	13	0.80
2011	81	23	16	0.80
	37	23	19	0.90
	21	43	14	0.88
	46	53	5	0.90
	47	20	11	0.95
2012	1	87	152	0.82
	4	65	44	0.86
	1	98	101	0.90
	1	94	96	0.90
	31	36	8	0.87
2013	43	34	32	0.82
	6	80	77	0.82
	11	92	27	0.81
	1	128	158	0.80
	57	83	4	0.86

7. SINGLE ACTOR ANALYSIS

In this section we present the second component of PathMiner, an algorithm to identify the network element that is most likely to have caused a large event, given that the ASes, prefixes and days involved in the event are known. We start by explaining our identification methodology, and then we present results.

7.1 Algorithm

We start with some basic observations. Consider an action taken by some network element a (say, an AS) that causes a path from AS b towards prefix p to change. We observe that a is, in general, either on the path from b to p before the change, or after [10]. For example, if a link fails or comes up, or an AS announces a new route and/or withdraws an existing route, these events can cause changes to many paths, but the paths involved will all pass through the link or AS either before or after the change.[1]

Thus, if we are interested in some event that happened on day k, then we may want to compare the set of paths seen in the network on day k with those seen on day $k+1$. Counting paths in order to identify (or narrow down) a root cause for a routing change has been explored before, e.g. [16]. However, PathMiner differs from such previous work in terms of goals (since it particularly focuses on large events that may re-occur over time) and in methods (since it carefully chooses the set of paths to analyze).

PathMiner takes advantage of the fact that starting from a collection of (λ, ν)-events, found by its first component, is an effective way of isolating the set of paths to study, thus avoiding interference of paths that are changing for unrelated reasons. Specifically, when analyzing an event that on day k has set of ASes J and set of prefixes I, we consider only paths seen on days k and $k+1$ passing through ASes in J towards prefixes in I. If one of such path start at an AS which is not in J and passes through $a \in J$, then PathMiner takes such path in account, but it ignores the portion before a. Accordingly, we define P_k to be the multiset of all paths that are found starting at an element of J and ending at an element of I on day k.

Considering the above observations we have that a good candidate for the single network element responsible for the entire set of observed path changes has the following properties: on either day k or day $k+1$, (a) most of the changed paths pass through the network element and (b) most the paths going through the network element change. We call these rules *single actor rules* and the network element so identified the *single actor.*

For concreteness, assume that network element e is the single actor, and the changed paths go through e on day k but not on day $k+1$. Among all the paths being considered, define $D = P_k \setminus P_{k+1}$ as the paths that disappeared; D_e as the paths in D passing through e; and $P_{k,e}$ as the paths in P_k that pass through e. If e is the single actor, by the single actor rules one concludes that $r_e^D = \frac{|D_e|}{|D|}$ should be close to 1, and $p_e^D = \frac{|D_e|}{|P_{k,e}|}$ should be close to 1. In other words, r_e^D captures the fraction of disappeared paths passing through e (rule (a)) and p_e^D captures the fraction of paths passing through e that disappeared (rule (b)).

For intuition, we note that p_e^D and r_e^D correspond respectively to the definitions of *precision* and *recall* for a particular classifier. This classifier is the one that declares that a path will disappear if the path passes through e on day k. In other words, maximizing p_e^D and r_e^D with regard to e yields the network element that best classifies the paths as changing versus unchanging. Following common practice in machine learning, we combine precision and recall into the F-score. In fact, in our case we use the F_2 score defined by $F_2^D(e) = \frac{5p_e^D r_e^D}{4p_e^D + r_e^D}$ to place higher emphasis on the role of recall.

We can make analogous definitions covering the set of paths that appear on day $k+1$. In that case, we define $A = P_{k+1} \setminus P_k$, and A_e as the paths in A that pass through e. Hence, $p_e^A = \frac{|A_e|}{|P_{k+1,e}|}$, $r_e^A = \frac{|A_e|}{|A|}$, $F_2^A(e) = \frac{5p_e^A r_e^A}{4p_e^A + r_e^A}$ and the same interpretation given above is valid here as well.

In practice one does not know a priori whether e will be found using the single actor rules applied to disappeared paths on day k, or appearing paths on day $k+1$. To overcome this problem we

[1]The assumption that the cause of the event will be in the new or old path is not always true [14]. However, PathMiner is built under such assumption because it deals with large-scale events, where the likelihood of observing paths passing through the cause increases.

define the candidate single actor to be the network element (or set of elements) that maximizes $\Delta F(e) = \max\{F_2^D(e),\ F_2^A(e)\}$. From this analysis we exclude AS-links that are seen on days k and $k+1$ and ASes which all incoming and outgoing AS-links are also seen on days k and $k+1$. This decision is justified by the fact that an AS-link that has not been disrupted and an AS that, explicitly, has not changed its local preference and/or export policies are unlikely the element that triggers a large-scale event.

This algorithm is effective, but can some times return multiple elements as candidate single actors for any given day k in an event. However, one can bring one more observation to bear: the single actor should be the same over all days k in the event. This offers an additional opportunity to winnow the set of candidate single actors. So the final step of the single actor analysis is: for events that reoccur over a set of days K, repeat the single-day strategy for every $k \in K$. Then, define as the final single actor of the event the element (or elements) that is observed in the candidate single actors set in at least the majority of the days in K. If such element is not found (no element is a candidate single actor in the majority of days) the algorithm declares that it was not able to identify the cause of the event.

7.2 Performance

PathMiner is able to identify the actors responsible for most of the events presented in Table 4. Table 5 shows the ASes identified as the single actor for each event, and the number of days that the element was classified as single actor (note that in the table we present only the most frequent one and that other elements, present in the majority of days, also have to be considered as possible actors). The boldface numbers indicate the cases where PathMiner identified the same element(s) as cause(s) for a majority of days in the event. It can be seen that PathMiner has more difficulty identifying single actors for events spanning many days and/or involving few ASes.

In order to validate the actors identified by PathMiner for events in Table 4 we performed a visual inspection of each event as follows[4]: for each day of each event we looked at the network element identified as actor, the graphical representation of the event, and asked two questions. First, "would an action of that network element (e.g. an AS changing its local preferences and/or its export policies) explain the occurrence of the event?" Second, "Do the actions of this network element provide the simplest explanation among all elements involved in the event?" If the answers for those question were affirmative in the majority of days of the event we considered such event validated.

We also conducted analyses to understand how often PathMiner is able to identify a single actor as cause of an event. Our first analysis is related to the maximum value of $\Delta F(e)$ obtained for each day in an event and for all events that PathMiner found in the 9 datasets. The results, presented in Figure 7(a) by a CDF, indicate that large values are the predominant case. For instance, the figure shows that the maximum $\Delta F(e)$ is equal to 1 in 40% of the cases and is greater than 0.8 in more than 80%. These numbers suggest that in most of the days within events PathMiner is capable of finding a network element (or a set of) that, in fact, can explain the massive amount of changes related to the event.

These results are promising, but note that they refer to individual days within events. It is important to also ask how consistent the identification is over the days of an event, and how often PathMiner is able to find the same actor over the set of days of the event. To answer those questions we identified for each event the element that appears as cause of the event most frequently over the days of the event and the fraction of days this single actor has been identified. Figure 7(b) presents the results. It can be seen (with exception of 2005) that in more than 40% of the events a single network element has been observed by PathMiner as an event cause over all days of the event. Furthermore, we can see that in more than 90% of the events there is a single actor that is present over the majority of days. For the cases where majority has not been found we refer to Figure 7(c) to show that in some of them the event contains many days, which naturally increases the complexity of the problem. However, there are many small (in terms of days) events for which PathMiner was not able to identify a single actor. Initial

Table 5: Single actor analysis of Top-5 events of each year

Dataset	Most Frequent Actor	Days
2005	31050	76
	23918	38
	1299	38
	8342	**57**
	20485	**30***
2006	23918	50
	3257	**13**
	33697	**103**
	20485	**58**
	33697	**93**
2007	174	**14**
	1273	18
	4637	**35**
	3257	**16***
	7575	**6**
2008	9121	**16***
	25462	3
	174	**12**
	25462	5
	3303	**8**
2009	3216	**2***
	15412	**62**
	3216	**7**
	8359	**10**
	29049	5
2010	30890	**14**
	3491	**11**
	21219	**2**
	5588	**8***
	13249	**8**
2011	5588	**16**
	3549	**17**
	3491	**13**
	12989	**4***
	174	**10**
2012	38312	**142**
	29632	**37**
	4755	**96**
	56209	**91**
	4651	**8**
2013	7713	**28**
	12880	**52**
	8529	**19**
	10029	**116**
	21219	**4**

* The actor found by PathMiner provides a simple explanation for the event. However, other elements providing a simple explanation could be identified by visual inspection.

[4]Graphical representations for these events and actor identification summary are available at `http://cs-people.bu.edu/gcom/bgp/imc2014`. We also make available the same representation and summary for another 500 randomly selected events for the year of 2013.

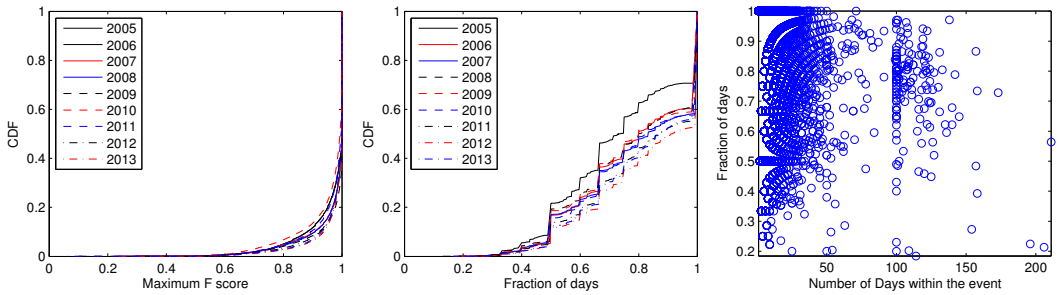


Figure 7: Summary of our methodology for Single Actor identification: (a) CDF of ΔF, computed over each day of each event; (b) CDF of the fraction of days that the same AS (or AS-link) is reported as actor (has maximum ΔFscore) over all days of the event; (c) scatterplot of the number of days within an event *versus* the fraction of days that the same AS (AS-link) is reported as actor – same variable as in (b).

investigation reveals that those events are related to a small number of ASes, and as consequence to a small number of paths, which is not an ideal situation to work with measures such as precision and recall. We emphasize that further investigation is necessary in that direction as future work.

As a final analysis, we list the ASes which, as actors had the greatest impact on the network. We considered the year of 2013, for all events that PathMiner was able to identify a single actor and that involved at least 50 ASes (to avoid the problem of few paths mentioned previously). Table 6 shows the top-20 ASes ranked in three different ways: first, by the number of events that the AS is an actor; second, by the aggregate volume of all events that the AS is an actor; and third, by the aggregate number of days of all events that the AS is an actor. In all three cases, if more than one AS was identified as actor, or if there was an AS-link, we counted each AS involved individually. Therefore, events with more than one AS as actor are counted more than once in Table 6.

Table 6: Three different rankings for the Top-20 Single Actors

#Events	Total volume	Total days
AS174	AS9498	AS9498
AS9498	AS4755	AS4755
AS9002	AS6453	AS6453
AS3356	AS12880	AS174
AS6939	AS6939	AS3549
AS3549	AS9002	AS3216
AS12389	AS174	AS6939
AS3216	AS3549	AS3356
AS20485	AS3216	AS20485
AS6453	AS12389	AS15412
AS31133	AS15412	AS12389
AS7018	AS20485	AS9002
AS701	AS3356	AS701
AS4755	AS3491	AS3491
AS12880	AS55410	AS18101
AS8167	AS10029	AS8167
AS6461	AS12989	AS12880
AS209	AS4651	AS7018
AS3491	AS197556	AS18881
AS8359	AS8167	AS55410

First, it can be seen that the way we ranked the ASes change the order in the three columns of the table, but there is a significant overlap between the top-20 of each rank, showing that, in general, there are heavy-hitter actors that trigger many large, re-occurring events. Second, in all three ranks, it is possible to identify some large ASes, which peer other large ASes and are important parts of the Internet core. For example, AS174 (Cogent), AS6939 (Hurricane Electric), AS3549 (GBLX) and AS3356 (Level3) are in the top-20 of each list. This suggests that some ASes in the core are responsible for a large amount of the reorganization that happens in the network. However, it is important to remember that our strategy to sample ASes may have influence in this result, and ultimately that our original dataset does not cover the complete AS-level topology of the Internet (which in fact motivated our sampling strategy).

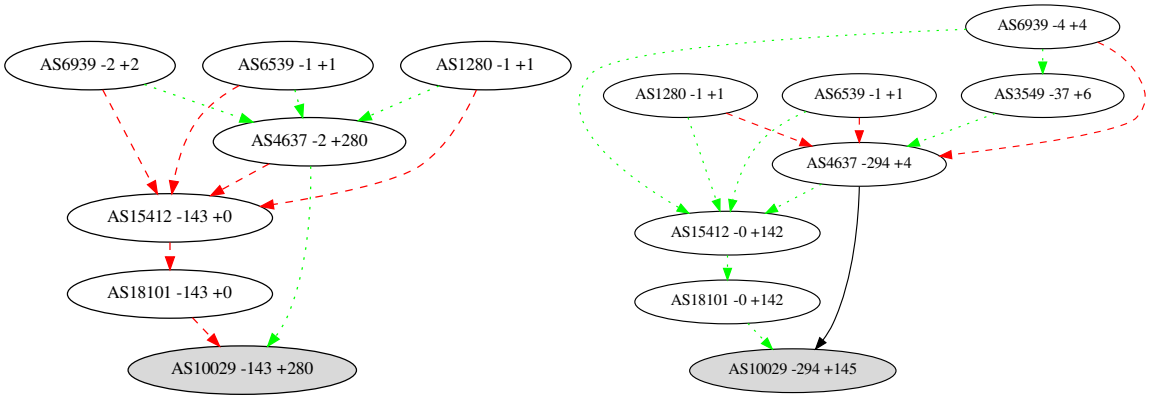
7.3 Case studies

In this section we present three case studies in order to show the ability of PathMiner to identify actors of high impact events. Case studies I and II were chosen to illustrate typical scenarios where PathMiner is able to identify a network element (or set of elements) whose actions would explain, in a simple way, the occurrence of the event. We remark that many events share the same structure (this can be seen in the online supplementary material). Case study III, on the other hand, shows that PathMiner is not guaranteed to always work. In that case, we discuss the reason and show that the assumptions under which PathMiner was build are not satisfied.

In the representation of each case study, Figures 8, 9 and 10, ASes in gray hosts a destination prefix. Red (dashed) edges are those present only at day k, green (dotted) edges are present only at day $k+1$ and black (solid) edges are present in both days. Inside each node we have: the AS identifier; the number of paths passing through the AS at day k, with a negative sign; and the number of paths passing through the AS at day $k+1$, with a positive sign. Only paths passing through an AS in the event towards a prefix in the event are counted.

7.3.1 Case study I

Our first case study is related to the second event for the year of 2009 presented in Table 4. In this event we have 90 ASes changing their next-hops towards one prefix over 91 days during the year. Among those, we picked 2 pairs of consecutive days in order to illustrate the nature of the event. Following the same convention used in Section 2, Figures 8(a) and 8(b) picture the most important part of the network for our needs. From Feb-08 to Feb-09 many ASes stopped using AS15412 as next-hop and started using



(a) Path changes between Feb-08-2009 and Feb-09-2009 (b) Path changes between Feb-10-2009 and Feb-11-2009

Figure 8: Representation of case study I.

AS4637 instead. Just from that observation it is possible to conjecture that one of these two ASes was responsible for the changes, and in fact, considering Feb-08 and Feb-09, both ASes have the same ΔF value. On the next day almost the reverse event happens, with ASes leaving AS4637 and starting to use AS15412 as next-hop. But it can be seen that AS4637 still has a path towards the prefix and it still is used by some ASes as next-hop. This fact indicates that AS4637 is probably not the actor responsible for the event.

It is also necessary to observe that AS15412 and AS18101 are not seen in our dataset at Feb-09 and Feb-10 (no paths passing through them). Hence we cannot say exactly if the cause is one or the other or even a link leaving one of them, narrowing down the set of candidates to AS15412, AS18101 and AS10029 (and links). However, in other days of the event the same structure of the figures was observed, but there were paths passing through AS15412 and AS18101. This indicates that in fact many paths left (preferred) AS15412 as next-hop due its actions, not AS18101 or AS10029.

Following the majority rule, PathMiner selected AS15412 as single actor of the event and therefore, it was able to capture the discussion presented above.

7.3.2 *Case study II*

In this case study we explore the second event for the year of 2012 shown in Table 4. The event involves 4 prefixes, 65 ASes, and occurs on 44 days during the year. Figures 9(a) and 9(b) present the simplified dynamics of the changes for 2 days, Jan-10-2012 and Jan-11-2012. This case study seems similar to the previous one, since we can observe that over the time ASes are alternating their next-hops (towards the four prefixes) between AS35320 and AS29632. Simply looking at those figures it is possible to narrow down the cause to one of these two ASes (or the links between them and AS42418). The question that arises is: why did PathMiner chose AS29632? It turns out that for many of the 44 days of the event, some of ASes leaving next-hop AS29632 started reaching two of the four destinations by paths not passing through AS35320. Therefore, either AS29632 or two other ASes (at the same time) are the causes. PathMiner identifies AS29632 as actor, capturing the idea that the simplest explanation is the most likely one.

7.3.3 *Case study III*

Our last case study discusses an event for which PathMiner was not able to identify a single network element responsible for the whole event. The event is the one on the first row of Table 4 for the year of 2005. The event involves 2 prefixes, 54 ASes and it happens on 200 days of the year. However PathMiner was not able to find a single network element responsible for changes in at least 100 days. Why did that happen? A closer look at the event shows us that the assumptions over which PathMiner is built are not valid. In fact, this is a strange case. Figures 10(a) and 10(b) present two typical subgraphs describing the changes over the days of the event. First, we note that the structure of the graph is completely different of our two previous case studies. The second, and key, fact is that the event contains only two prefixes, but we can see three gray nodes in the figures (gray nodes are those hosting the prefixes of the event). Denoting those two prefixes by prefix 1 and prefix 2, through data examination it was possible to see that: *i)* at Jan-11 prefixes 1 and 2 were being hosted at AS23918 and AS29257 respectively; *ii)* at Jan-12 prefix 1 was being hosted at AS29257 and at AS31050 (which does not seem to be a normal situation) and prefix 2 was being hosted at AS23918; and *iii)* at Jan-13 prefixes are hosted as at Jan-11.

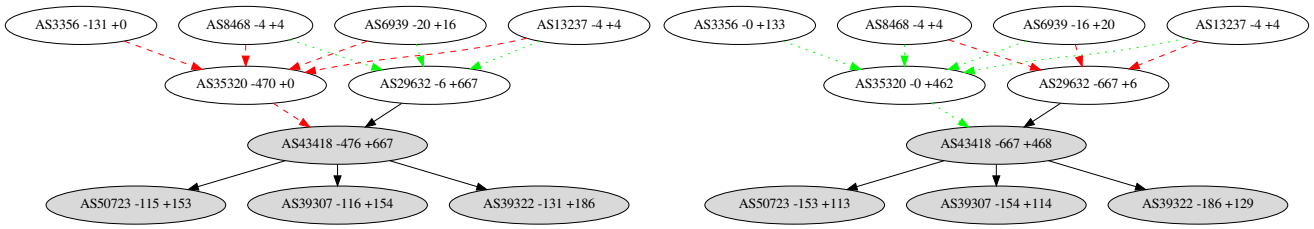
Over the 200 days of the event this alternating state was repeated. In summary, it can be seen that is hard to find one network element that can be responsible for these changes. More specifically, our assumption of a single actor causing the event does not seem to be valid here – it appears that a set of coordinated changes is being implemented through actions of multiple ASes.

8. RELATED WORK

There is a considerable amount of literature about BGP due to its importance to the global Internet. In this section we discuss how PathMiner differs from existing work.

BGP event detection: BGP event detection has been studied in a variety of ways. Most work on event detection relies on BGP update messages in order to analyze path changes. For example, the authors in [20] proposed a methodology to identify high impact BGP routing changes. However their scope is different from ours, since in their context the term *high impact* is related to the impact in the network traffic leaving a specific ISP and not in terms of changes propagating throughout the network.

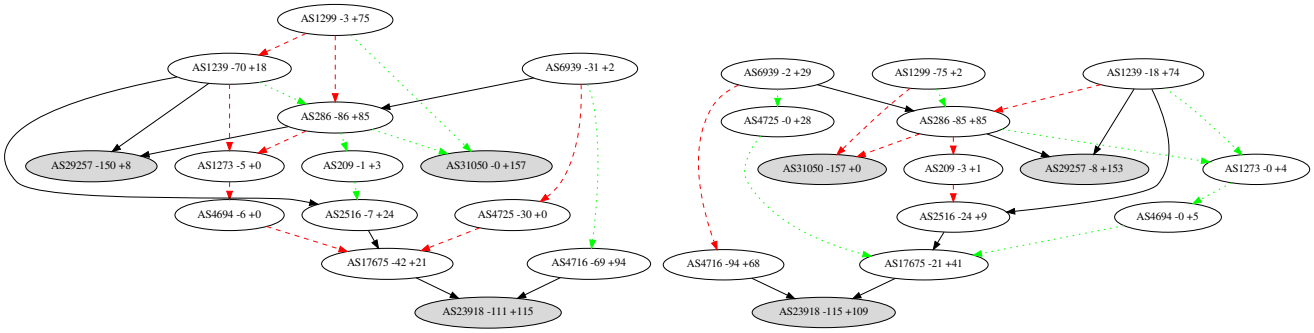
In [17, 16] the authors propose a way to identify temporal event boundaries in the stream of update messages and a visualization tool that allows users to narrow down and infer root causes. Our work, on the other hand, does not rely on inferring the state of the routing system by using update messages (in fact this is a hard task, since in most cases internal AS policies are unknown). Instead, we



(a) Path changes between Jan-10-2012 and Jan-11-2012

(b) Path changes between Jan-11-2012 and Jan-12-2012

Figure 9: Representation of case study II.



(a) Path changes between Jan-11-2005 and Jan-12-2005

(b) Path changes between Jan-12-2005 and Jan-13-2005

Figure 10: Representation of case study III.

use real daily snapshots of the network to perform a large scale event identification.

The authors of [11] use tensor factorization techniques (real, not boolean) to infer events in the stream of BGP updates. However, since they use a path-based representation they are restricted to a small set of monitors. Moreover, since they work with data obtained at the granularity of minutes their strategy is not able to identify large scale events involving hundreds of sources, possibly months apart.

Finally, it is important to mention that working with BGP updates demands extra processing in order to clean the data. In our approach we use BGP RIBs, i.e., snapshots of the interdomain routing system , which avoids the complex process of update messages cleaning and allows PathMiner to scale over long timescales.

Root cause analysis: the second component of PathMiner, described in Section 7, consists of a technique to identify (or at least narrow down) possible ASes (or links) triggering large scale events. Although sounding related to root cause analysis, as in [14] and [10], it is important to remark that our requirements and assumptions differ significantly. On one hand, general root cause identification systems are real time, work with BGP update messages (sometimes with information coming from external sources) and are interested at identifying causes of any path changes. On the other hand, differently of PathMiner, those systems are not capable, of identifying large events and narrowing down causes using just routing tables.

BGP path discovery: as mentioned in Section 8 our datasets do not contain paths from every source to every destination and even for those we have, the issue of missing data arises. Therefore, we note that PathMiner could benefit from techniques that allow it to have better snapshots of the global interdomain routing system. Among recent works on this direction are [15], [13] and [12].

Boolean Tensor Factorization: in this work we did not aim at a general approach for solving instances of the BTF problem coming from an abitrary application. In fact, among recent works trying to solve the general BTF problem are [7] and [9]. The former is designed to find blocks of closed relations (i.e., equivalent to blocks with density 1), which is too strong a requirement for our application. The latter would suit our needs, but although it does not impose any formal requirement for volume and density, unfortunately the available implementation did not scale for our datasets due to the amount of data analyzed by PathMiner.

9. CONCLUSIONS AND FUTURE WORK

In this work we presented PathMiner, a system capable of identifying and analyzing high impact events in the interdomain routing system of the Internet. We started by giving a formal definition of the problem and discussing how to prepare the data and how to deal with missing data. In its first phase PathMiner has the advantage of working with a next-hop representation of the AS-level Internet, which allows PathMiner to naturally combine paths obtained from multiple sources.

In order to identify events we proposed a new heuristic to solve the Boolean Tensor Factorization problem, since algorithms currently available in the literature were not scalable to our datasets. Using datasets spanning 9 years of routing in the Internet we showed that PathMiner is able to find many large and dense events.

Next, we addressed the challenge of discovering network elements (ASes or AS-links) that were likely the actors responsible for the event occurrence. Our methodology, based on concepts borrowed from classification theory, was able to identify possible event causes in most of the events we found. One key aspect of PathMiner is its ability to look at the same event re-occurring many times, which helps the process eliminating event cause candidates. Finally, we presented 3 case studies, for large events, exemplifying all these aspects of PathMiner.

To the best of our knowledge, PathMiner is the first system capable of exploring the amount of information we analyzed, finding daily large routing events and recognizing that such events re-occur many times during the period of one year, in some cases months apart.

In future work we intend to address the need to better understand cases where PathMiner is not capable of finding a single actor responsible for an event (as exemplified in our third case study). Moreover, we are interested in the possibility of using PathMiner to explore persistent routing changes, i.e., considering only changes that were not quickly undone in the system. Such studies might have the potential to expose interesting structural reorganization of the AS-level Internet.

10. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under grant numbers CNS-0905565, CNS-1018266, CNS-1012910, and CNS-1117039, and supported by the Army Research Office under grant W911NF-11-1-0227. We are grateful to the anonymous reviewers and to our shepherd, Alex Snoeren, for their valuable comments. We also thank Evimaria Terzi and Dora Erdos for the helpful discussions.

11. REFERENCES

[1] Hadoop. `http://hadoop.apache.org`.

[2] Ripe's routing information service raw data project. `http://www.ripe.net/data-tools/stats/ris/ris-raw-data`.

[3] Spark. `http://spark.apache.org`.

[4] University of oregon route views project. `http://www.routeviews.org`.

[5] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1):155–173, September 2007.

[6] A. Broido and kc claffy. Analysis of RouteViews BGP data: policy atoms. In *NRDM*, 2001.

[7] Loïc Cerf, Jérémy Besson, Céline Robardet, and Jean-François Boulicaut. Closed Patterns Meet n-ary Relations. *ACM Trans. Knowl. Discov. Data*, 3(1), 2009.

[8] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley Publishing, 2009.

[9] Dóra Erdös and Pauli Miettinen. Walk 'n' Merge: A Scalable Algorithm for Boolean Tensor Factorization. In *Proceedings of the 13th International Conference on Data Mining*, ICDM '13, pages 1037–1042. IEEE, 2013.

[10] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet Routing Instabilities. In *Proceedings of SIGCOMM Conference*, SIGCOMM '04, pages 205–218. ACM, 2004.

[11] K. Glass, R. Colbaugh, and M. Planck. Automatically identifying the sources of large Internet events. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on*, pages 108–113, 2010.

[12] Enrico Gregori, Alessandro Improta, Luciano Lenzini, Lorenzo Rossi, and Luca Sani. On the incompleteness of the as-level graph: A novel methodology for bgp route collector placement. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 253–264. ACM, 2012.

[13] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. Lord of the Links: A Framework for Discovering Missing Links in the Internet Topology. *IEEE/ACM Trans. Netw.*, 17(2):391–404, April 2009.

[14] Umar Javed, Italo Cunha, David Choffnes, Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy. PoiRoot: Investigating the Root Cause of Interdomain Path Changes. In *Proceedings of SIGCOMM Conference*, SIGCOMM '13, pages 183–194. ACM, 2013.

[15] Akmal Khan, Taekyoung Kwon, Hyun-chul Kim, and Yanghee Choi. AS-level Topology Collection Through Looking Glass Servers. In *Proceedings of the 2013 Internet Measurement Conference*, IMC '13, pages 235–242. ACM, 2013.

[16] Mohit Lad, Dan Massey, and Lixia Zhang. Visualizing Internet Routing Changes. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1450–1460, November 2006.

[17] Mohit Lad, Lixia Zhang, and Daniel Massey. Link-rank: A Graphical Tool for Capturing BGP Routing Dynamics. In *Proceedings of the Network Operations and Management Symposium*, NOMS '04, pages 627–640. IEEE, 2004.

[18] Pauli Miettinen. Boolean Tensor Factorizations. In *Proceedings of the 11th International Conference on Data Mining*, ICDM '11, pages 447–456. IEEE, 2011.

[19] Y. Rekhter and T. Li. *A Border Gateway Protocol (BGP-4)*. RFC Editor, 1995.

[20] Jian Wu, Zhuoqing Morley Mao, Jennifer Rexford, and Jia Wang. Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network. In *Proceedings of the Symposium on Networked Systems Design & Implementation*, NSDI'05, pages 1–14. USENIX Association, 2005.

Appendix A - 3D-FACTORIZATION Algorithm

Algorithm 3: 3D-FACTORIZATION

Data: S, set of triples of the form (I, J, K), and thresholds γ and β

$F \leftarrow \emptyset$
while $S \neq \emptyset$ **do**
  $s \leftarrow$ pick and **do not** remove an element of S
  $S' \leftarrow \{x : x \in S \text{ and } d_B(x, s) \leq \gamma\}$
  find maximal $S'' \subseteq S'$ such that $\max_{x,y \in S''} d_B(x, y) \leq \gamma$
  if $|S''| > 1$ **then**
    $y \leftarrow$ COMBINE-UNION(S'')
    $S \leftarrow S \backslash S''$
    $S \leftarrow S \cup \{y\}$
  else
    $F \leftarrow F \cup \{s\}$
    $S \leftarrow S \backslash \{s\}$
    $S' \leftarrow \{x : x \in S, s_B(x, s) \geq \beta$ and COMBINE-INTER$(s, x) \neq s\}$
    if $|S'| > 0$ **then**
      $s' \leftarrow \arg\max_{x \in S'} s_B(s, x)$
      $y \leftarrow$ COMBINE-INTER(s, s')
      $S \leftarrow S \cup \{y\}$
return F