Mining Network Events using Traceroute Empathy

Marco Di Bartolomeo  Valentino Di Donato  Maurizio Pizzonia  Claudio Squarcella  Massimo Rimondini

Abstract—In the never-ending quest for tools that enable an ISP to smooth troubleshooting and improve awareness of network behavior, very much effort has been devoted in the collection of data by active and passive measurement at the data plane and at the control plane level. Exploitation of collected data has been mostly focused on anomaly detection and on root-cause analysis. Our objective is somewhat in the middle.

We consider traceroutes collected by a network of probes and aim at introducing a practically applicable methodology to quickly spot measurements that are related to high-impact events happened in the network. Such filtering process eases further in-depth human-based analysis, for example with visual tools which are effective only when handling a limited amount of data.

We introduce the empathy relation between traceroutes as the cornerstone of our formal characterization of the traceroutes related to a network event. Based on this model, we describe an algorithm that finds traceroutes related to high-impact events in an arbitrary set of measurements. Evidence of the effectiveness of our approach is given by experimental results produced on real-world data.

I. INTRODUCTION

A large wealth of data is available about the Internet, gathered by any sort of active and passive measurements at control plane and at data plane level. Objectives usually include measurement of the quality of service (QoS), troubleshooting, Internet mapping, and support to other research goals.

With the intent of supporting Internet Service Providers (ISPs) in their troubleshooting activities, many contributions exploit gathered data to detect anomalies and identify portions of the network supposed to be the root cause of the problem (e.g. [1], [2]). In spite of the vast literature in this field, our experience shows that network operators still prefer to analyze data without automated tools. In fact, these approaches have several limitations: they may require to set up a dedicated and costly monitoring infrastructure, the validity of results might be unclear or hard to assess for a network operator, and they cannot automatically access all the information that an operator has at disposal (e.g. routers configurations, maintenance logs, firmware bug reports, etc.). On the other hand, operators are steadily increasing their interest in visualization tools (like [3]) that support human-based analysis of data about network behavior. Such type of analysis, however, can usually handle only a limited and focused set of data.

The goal of our approach is to complement network visualization tools with a domain-specific data mining technique. We took great care in providing a formally sound technique with potential wide applicability. We focus on traceroutes performed by active measurement networks on the Internet, firmly believing that such infrastructures will become more and more available in the future: active measurement probes are indeed easy to deploy in large number without affecting configuration of production routers (like it is for BGP data collection), can run on a small dedicated hardware or embedded in existing software, and are often required by regulators as a means to check the QoS provided by network operators.

Differently from the vast literature on root-cause analysis, we do not try to spot the actual origin of the observed routing change. Our goal is to simply assert that a routing change happened, assessing its impact in terms of number of affected probes and targets and isolating the smallest interval of time that contains the identified observations. These are enough to sort observed changes based on relevance and feed a visualization tool for further human-based analysis.

Our main contributions are 1) the introduction of the concept of empathic traceroutes, and 2) a related methodology that detects routing events and infers data that is suitable as the starting point for further human-based analysis. In particular, the methodology identifies the smallest possible interval of time containing each network event, together with the set of involved probes and targets and (in certain cases) the type of event. We extensively discuss the applicability of our methodology in the real world and perform an experimental analysis that shows how well our algorithm performs in practice.

The rest of the paper is structured as follows. Section II describes the related state of the art. Section III formally introduces our model of network and measurements. In Section IV we provide definitions and assumptions on routing changes. Section V introduces the concept of empathic measurements and its main properties related to events occurring in the network. In Section VI we provide a methodology, based on the empathy theory, to infer events and report relevant data about them. Section VII discusses the impact of our assumptions on the results of our methodology. Section VIII reports an experimental analysis of the application of our methodology to real world data. Section IX presents our conclusions and ideas for future work.

II. RELATED WORK

Measurement networks. Many projects aim at performing large-scale active measurements in the Internet. Among them, SamKnows [5] intends to help regulators and operators to assess the quality of connectivity services provided to end users, RIPE Atlas [6] is mainly targeted to operators and provides active measurement tools on-demand, M-Lab [7] and Ark [8] are measurement networks mainly targeted to research. All of them use traceroutes as one of the main means of probing.

Anomaly detection and root-cause analysis based on BGP. A large number of contributions has focused on identifying the root cause of a fault. In particular, the scientific
community has mostly focused on the analysis of data related to the BGP protocol. Some initial results are provided in [9], [10], [11]. A thorough methodology to automatically determine the origin of a routing change is described in Feldmann et al. [2], where some interesting limits of the approach are described. In [11] many shortcomings of [2] are addressed at the cost of some restriction on the practical applicability of the proposed technique. Other results related to root-cause analysis are described in [12], [13], [14]. Moreover, the approach in [14] is designed to aid root cause analysis with a visual representation.

Anomaly detection using active measurements. Works related to anomaly detection with other kind of data are much less frequent. Hubble [15] is a system based on passive/active measurements to detect disconnections. Traceroutes are part of the data used by Hubble. LIFEGUARD [16] is a methodology to actively locate a failure responsible of a lack of connectivity between two autonomous systems and suggests alternative routes to restore connectivity.

Visualization. Several projects aim at providing visual means of exploring data collected by measurements networks. Among them are TPlay [17] and VisTracer [18]. Other tools targeted to visualizing control plane data are BGPlay [3] and LinkRank [19].

III. BASIC DEFINITIONS

In this section we describe the model we use to study traceroute paths. We also introduce several assumptions that make our approach easier to understand and allow us to take advantage of several properties. We illustrate in Section VIII how, even under these assumptions, interesting results can be obtained with real-world data, and discuss in Section VII their impact on the practical applicability of our approach.

Let \( G = (V, E) \) be a graph that models an IP network, where vertices in \( V \) are network devices, and edges in \( E \) are links between devices. We make the following assumption.

Assumption 1 (No aliasing): Every network device is identified by a unique IP address.

All network devices in \( V \) are capable of routing IP packets, namely behave as routers. Additionally, some devices called network probes carry out measurements of the network status on a periodical basis. Each probe, also called source, performs traceroutes from itself to a predefined set of targets called destinations, under the following condition.

Assumption 2 (Discrete time): At each time instant \( t \), every source performs an instantaneous traceroute measurement towards every configured destination.

Let \( i = (s, d) \), where \( s \in V \) is a source and \( d \in V \) is a destination. A traceroute path \( p_i(t) \) measured at time \( t \) by \( s \) towards \( d \) is a sequence \( \langle v_1, v_2, \ldots, v_n \rangle \) such that \( v_1 = s, v_j \in V \) for \( j = 1, \ldots, n \), and \( \exists (t_k, v_{k+1}) \in E \) for each pair of consecutive vertices \( v_k \) and \( v_{k+1} \) in \( p_i(t) \), \( k = 1, \ldots, n-1 \). A traceroute path \( p_i(t) \) represents the sequence of network devices reported by the traceroute tool.

We include the source in this sequence, even if it usually does not appear in the reported path. On the other hand, the target may not appear in the sequence since a traceroute path may not end at the intended destination \( d \), for example because the traceroute failed to complete successfully. In an extreme case, a traceroute path may contain only the source vertex.

Assumption 3 (Acyclicality): Traceroute paths are acyclic.

The concatenation of two non-empty paths \( p' = \langle v'_1, v'_2, \ldots, v'_{m} \rangle \) and \( p'' = \langle v''_1, v''_2, \ldots, v''_{n} \rangle \) such that \( v'_m = v''_1 \) is a path \( p' \circ p'' = \langle v'_1, v'_2, \ldots, v'_m, v''_2, \ldots, v''_{n} \rangle \). For any path \( p \), also let \( \langle \rangle \circ p = p \) and \( p \circ \langle \rangle = p \). Where \( \langle \rangle \) is the empty path. It is convenient to perform set operations on the elements of a sequence. Therefore, let \( V(p) \) be the set of vertices appearing in a path \( p \) and \( E(p) \) be the set of edges \((u,v)\) such that either \((u,v)\) or \((v,u)\) appears as a subsequence in \( p \).

We call event at time \( t \) the simultaneous disappearance of a set \( E^{↓} \) of links from \( E \) (down event) or the simultaneous appearance of a set \( E^{↑} \) of links in \( E \) (up event), such that:

- either \( E^{↓} = \emptyset \) or \( E^{↑} = \emptyset \) (an event is either the disappearance or the appearance of links, not both);
- \( E^{↓} \subseteq E \) (only existing links can disappear);
- \( E^{↑} \cap E = \emptyset \) (only new links can appear);
- \( \forall v \in V \) \( \forall (u,v) \in E^{↓} : u = v \) or \( w = v \), and the same holds for \( E^{↑} \) (all disappeared/appeared edges have exactly one endpoint vertex in common).

Any vertex \( v \) that satisfies the latter condition is called hub of the event. Therefore, an event involving a single edge \((u,v)\) has two hubs: \( u \) and \( v \); any other event has a unique hub. When the type of an event is not relevant, we indicate it as \( E^{↑↓} \). We say that an event \( E^{↑↓} \) at time \( t \) is visible if there exists at least a source-destination pair (shortly sd-pair) \( i = (s, d) \) such that \( p_i(t) \neq p_i(t+1) \). Moreover, we call scope \( S(E^{↑↓}) \) of an event \( E^{↑↓} \) occurred at time \( t \) the set of sd-pairs \( i = (s, d) \) whose traceroutes have been affected by the event, namely such that \( E^{↑↓} \cap E(p_i(t)) \neq \emptyset \) or \( E^{↑↓} \cap E(p_i(t+1)) \neq \emptyset \).

This event model captures the circumstance in which one or more links attached to a network device fail or are brought up, including the case in which a whole device fails or is activated. Such events may be caused, for example, by failures of network interface cards, line cards, or routers, by accidental link cuts, by provisioning processes, and by administrative reconstructions. Failures or activations of links that do not have a vertex in common are to be considered distinct events.

IV. MODELING TRACEROUTE PATH CHANGES

We now introduce a few formal tools to examine how traceroute paths change as a consequence of network events.
We use these tools to formulate assumptions on path changes, which in turn we exploit in searching for network events.

A. Common and Changed Portions of Traceroutes

Let $i = (s, d)$, with $s \in V$ and $d \in V$, and consider traceroute paths $p_i(t) = \langle v_1, v_2, \ldots, v_n \rangle$ and $p_i(t + 1) = \langle v'_1, v'_2, \ldots, v'_m \rangle$, resulting from executing traceroutes from $s$ to $d$ at the consecutive time instants $t$ and $t + 1$. Assume that $v_1 = v'_1 = s$ and $p_i(t) \neq p_i(t + 1)$. An example of two such paths is shown in Fig. 1 where $i = (1, 9)$, $p_i(t) = \langle 1 2 3 4 5 8 9 \rangle$, and $p_i(t + 1) = \langle 1 2 6 7 8 9 \rangle$. Paths $p_i(t)$ and $p_i(t + 1)$ may have one or more vertices in common. Let $c_{\text{head}}$ denote the common prefix of $p_i(t)$ and $p_i(t + 1)$, namely the maximal subsequence $\langle v_{i1}, v_{i2}, \ldots, v_{ik} \rangle$ of $p_i(t)$ and $p_i(t + 1)$ such that $j \leq \min(n, m)$ and $v_k = v_k$ for $k = 1, \ldots, j$ (note that this is obvious for $k = 1$, since $v_1 = v'_1 = s$). For example, for the paths in Fig. 1 we have that $c_{\text{head}} = (1 2)$. A few properties apply to the common prefix. First of all, it is always $c_{\text{head}} \neq \emptyset$, because at least vertex $v_1 = s$ will always be in $c_{\text{head}}$. In addition, since $p_i(t) \neq p_i(t + 1)$, it will always be $c_{\text{head}} \neq p_i(t)$ or $c_{\text{head}} \neq p_i(t + 1)$. However, when at least one of $p_i(t)$ and $p_i(t + 1)$ does not end at the intended destination $d$, the common prefix may coincide with one of the paths, namely it can be either $c_{\text{head}} = p_i(t)$ or $c_{\text{head}} = p_i(t + 1)$.

Similarly, let $c_{\text{tail}}$ denote the common suffix of $p_i(t)$ and $p_i(t + 1)$, namely the maximal subsequence $\langle v_{il}, v_{il+1}, \ldots, v_{in} \rangle$ of $p_i(t)$ of length $n - h + 1$ and the corresponding subsequence $\langle v'_{il}, v'_{il+1}, \ldots, v'_{im} \rangle$ of $p_i(t + 1)$ of length $m - h' + 1$ such that $n - h = m - h'$ (i.e., the two subsequences have the same length), $n - h + 1 \leq \min(n, m)$, and $v_l = v'_l$ for $l = h, \ldots, n$. Considering again the example in Fig. 1 we have that $h = 6$ and $h' = 5$, because there are no more vertices before $v_6 = v'_5 = 8$ along $p_i(t)$ and $p_i(t + 1)$ that form a common subsequence ending at $v_7 = v'_6 = 9$: therefore $c_{\text{tail}} = (8 9)$. The common suffix has slightly different properties from the common prefix. Since $p_i(t) \neq p_i(t + 1)$, it must always be $c_{\text{tail}} \neq p_i(t)$ and $c_{\text{tail}} \neq p_i(t + 1)$. Moreover, it can also be $c_{\text{tail}} = \emptyset$, when at least one of $p_i(t)$ and $p_i(t + 1)$ does not end at the intended destination $d$.

Based on these definitions, we can identify the parts of a traceroute path from $s$ to $d$ that change between consecutive time instants $t$ and $t + 1$: let $\delta^\text{pre}_i(t)$ indicate the portion of path $p_i(t)$ that changes at time $t + 1$, and $\delta^\text{post}_i(t)$ indicate the portion of path $p_i(t + 1)$ that has changed since time $t$. More formally, given two traceroute paths $p_i(t)$ and $p_i(t + 1)$ such that $p_i(t) \neq p_i(t + 1)$, $\delta^\text{pre}_i(t)$ is the subsequence of $p_i(t)$ such that $p_i(t) = c_{\text{head}} \circ \delta^\text{pre}_i(t) \circ c_{\text{tail}}$, and $\delta^\text{post}_i(t)$ is the subsequence of $p_i(t + 1)$ such that $p_i(t + 1) = c_{\text{head}} \circ \delta^\text{post}_i(t) \circ c_{\text{tail}}$. Note that $\delta^\text{pre}_i(t)$ and $\delta^\text{post}_i(t)$ always include the vertices, present in both $p_i(t)$ and $p_i(t + 1)$, that delimit the changed subpath. Referring again to the example in Fig. 1 we have that $\delta^\text{pre}_i(t) = \langle 2 3 4 5 8 \rangle$ and $\delta^\text{post}_i(t) = \langle 2 6 7 8 \rangle$.

B. Path Changes and Network Events

One could argue that there could be several subsequences that are common to $p_i(t)$ and $p_i(t + 1)$ besides the common prefix and the common suffix. In the example in Fig. 1 there is a common prefix $c_{\text{head}} = (1 2)$, a common suffix $c_{\text{tail}} = (8 9)$, and an additional common subsequence $(5 6)$. In theory, our model comprises such additional subsequence within $\delta^\text{pre}_i(t)$ and $\delta^\text{post}_i(t)$. However, in this example a single event, namely the failure of link $(7, 8)$, causes changes of two non-contiguous portions of $p_i(t)$: $(2 3 4 5)$ is replaced by $(2 10 5)$ and $(6 7 8)$ is replaced by $(6 11 8)$. While the change on the latter portion is clearly induced by the link failure, the change on the first portion is only possible if any vertices among $2, 3, 4, 5,$ and $10$ implement a routing policy that determines the selected path based on the routing between $6$ and $8$, or if some policy change independent from the failure of link $(7, 8)$ has occurred. Such routing policies could cause us to improperly consider traceroute path changes as related, and are therefore ruled out by the following assumption.

Assumption 4 (Continuous changed portion): For any paths $p_i(t)$ and $p_i(t + 1)$ such that $p_i(t) \neq p_i(t + 1)$, it is always $V(\delta^{\text{pre}}_i(t)) \cap V(\delta^{\text{post}}_i(t)) = \{v_1, \ldots, v_n\}$, where $\delta^{\text{pre}}_i(t) = \langle v_1, \ldots, v_n \rangle$ and $\delta^{\text{post}}_i(t) = \langle v_1, \ldots, v_n \rangle$.

In general, multiple traceroute paths can be influenced by a single event $E_d$. In order to better isolate the impact of each event, we make the following simplifying assumption.

Assumption 5 (Non-interfering events): Consider any two visible events $E_1^d$ and $E_2^d$ occurred at the same time $t$. For every $i \in S(E_1^d)$ and $j \in S(E_2^d)$, it must be $V(\delta^{\text{pre}}_i(t)) \cap V(\delta^{\text{pre}}_j(t)) = \emptyset$ and $V(\delta^{\text{post}}_i(t)) \cap V(\delta^{\text{post}}_j(t)) = \emptyset$.

This assumption imposes that events are “independent enough”, thus enabling us to correctly handle and detect contemporary events. An immediate consequence of this assumption is that, for any two events $E_1^d$ and $E_2^d$, it must be $S(E_1^d) \cap S(E_2^d) = \emptyset$.

Real routing policies could be such that observed traceroute paths change because of an event that does not involve any edges along the paths (see for example Fig. 1, 2). While this observation is quite relevant in a strict root-cause analysis setting, we rule them out here since root-cause analysis is not among our goals. Further comments are in Section VII.

Assumption 6 (Event on path): For any paths $p_i(t)$ and $p_i(t + 1)$ such that $p_i(t) \neq p_i(t + 1)$, there exists one event $E_i^d$ or $E_i^d$, such that either $\delta^{\text{pre}}_i(t)$ contains at least a pair of consecutive vertices $(u, v) \in E_1^d$ or $\delta^{\text{post}}_i(t)$ contains at least a pair of consecutive vertices $(u, v) \in E_1^d$. A direct consequence of this assumption is that traceroute paths can only change if at least one event has occurred.
Fig. 3 shows a situation which is ruled out by Assumptions [5] and [6]. This configuration cannot be produced by more than one event, by Assumption [5]. On the other hand, by Assumptions [6] the only candidate for being a hub of a single event is vertex 5. Edges (3, 5) and (5, 6) cannot be part of any event because they appear in traceroute paths both at time t and at time t + 1. The remaining edges cannot both belong to the same event, since (4, 5) is used a time t + 1 and (5, 7) is used a time t.

V. THE EMPATHY RELATION

Now that we have defined $\delta^\text{pre}_i(t)$, and $\delta^\text{post}_i(t)$, we can exploit them to introduce a relation, called empathy, that determines when traceroute paths between different sd-pairs exhibit a similar behavior over time.

A. Pre-Empathy and Post-Empathy

Let $p_1(t)$ be a traceroute path measured by $s_1$ towards $d_1$ at time t and $p_2(t)$ be a traceroute path measured by $s_2$ towards $d_2$ at the same time. Also let $p_1(t + 1)$ and $p_2(t + 1)$ be the traceroute paths measured at time t + 1 between the same sd-pairs. We say that $(s_1, d_1)$ and $(s_2, d_2)$ are pre-empathic at time t, indicated with $(s_1, d_1) \sim_\text{pre} (s_2, d_2)$, if:

1. the two traceroute paths $p_1$ and $p_2$ change between t and $t + 1$, namely $p_1(t) \neq p_1(t + 1)$ and $p_2(t) \neq p_2(t + 1)$;
2. the portions of $p_1(t)$ and $p_2(t)$ that change at $t + 1$ overlap, namely $V(\delta^\text{pre}_i(t)) \cap V(\delta^\text{post}_i(t)) \neq \emptyset$.

Similarly, we say that $(s_1, d_1)$ and $(s_2, d_2)$ are post-empathic at time t, indicated with $(s_1, d_1) \sim_\text{post} (s_2, d_2)$, if:

1. the two traceroute paths $p_1$ and $p_2$ change between t and $t + 1$, namely $p_1(t) \neq p_1(t + 1)$ and $p_2(t) \neq p_2(t + 1)$, and $V(\delta^\text{pre}_i(t)) \cap V(\delta^\text{post}_i(t)) \neq \emptyset$.

Relations $\sim_\text{pre}$ and $\sim_\text{post}$ are, trivially, commutative and reflexive.

Intuitively, distinguishing the pre-empathy from the post-empathy allows us to get a more accurate picture of how the traceroute paths between two sd-pairs change between t and $t + 1$: if two sd-pairs are pre-empathic, their traceroute paths stop traversing a portion that they shared before the event occurred; if two sd-pairs are post-empathic, their traceroute paths start traversing a common portion that they did not use before the event occurred. An example showing how to determine empathies is shown in Fig. 4. There are two traceroute paths $p_1$, from $s_1$ to $d_1$, and $p_2$, from $s_2$ to $d_2$, and the event that causes these paths to change between t and $t + 1$ is the failure of link (5, 6). In this example we have $\delta^\text{pre}_1(t) = \{(5, 6)\}$, $\delta^\text{post}_1(t) = \{(5, 6)\}$, $\delta^\text{pre}_2(t) = \{(4, 5, 6, 8)\}$, and $\delta^\text{post}_2(t) = \{(4, 10, 8)\}$. It is now easy to observe that $(s_1, d_1) \sim_\text{pre} (s_2, d_2)$, because $V(\delta^\text{pre}_1(t)) \cap V(\delta^\text{pre}_2(t)) = \{(5, 6)\}$. On the other hand, despite the fact that $p_1(t + 1)$ and $p_2(t + 1)$ have common subpaths, $(s_1, d_1)$ and $(s_2, d_2)$ are not post-empathic, because $V(\delta^\text{post}_1(t)) \cap V(\delta^\text{post}_2(t)) = \emptyset$.

B. Empathy Graphs

Being a natural representation for a relation, we conveniently model empathies between sd-pairs using a graph. We call $G^\text{pre}(t) = (V^\text{pre}(t), E^\text{pre}(t))$ the pre-empathy graph at time t, where each vertex $v = (s, d) \in V^\text{pre}(t)$ is a sd-pair and there is an edge between $(s_1, d_1) \in V^\text{pre}(t)$ and $(s_2, d_2) \in V^\text{pre}(t)$ if and only if $(s_1, d_1) \sim_\text{pre} (s_2, d_2)$. Likewise, we define the post-empathy graph at time t, $G^\text{post}(t) = (V^\text{post}(t), E^\text{post}(t))$, relying on $\sim_\text{post}$.

We exploit the empathy graph in order to single out network events, but a few additional properties are needed.

Property 1: For any visible event $E^\uparrow$ (respectively, $E^\downarrow$) occurs at time t, then for every $i, j \in S(E^\uparrow)$ (respectively, $i, j \in S(E^\downarrow)$), we have that $i \sim_\text{pre} j$ (respectively, $i \sim_\text{post} j$).

Proof: The property holds because, by definition, the hubs of $E^\uparrow$ appear in $\delta^\text{post}_i(t)$ for any $i \in S(E^\uparrow)$, and the hubs of $E^\downarrow$ appear in $\delta^\text{pre}_j(t)$ for any $j \in S(E^\downarrow)$.

C. Cliques in the Empathy Graphs

Property 1 suggests an interesting observation: if a visible event $E^\uparrow$ occurs at time t in the network, then a clique is formed in an empathy graph at time t (we recall that a clique is a structure in a graph such that there is an edge between any two distinct vertices).
In order to turn this observation into a statement, we introduce the following concept: given a set $C \subseteq V_{\mathrm{pre}}(t)$ of vertices of $G_{\mathrm{pre}}(t)$, we call pivot set of $C$ a set $\Pi_{\mathrm{pre}}(C) \subseteq V$ such that, for every $v \in \Pi_{\mathrm{pre}}(C)$, it is $v \in \delta_u^{\mathrm{pre}}(t)$ for all $i \in C$. By construction, the pivot set of a set $C$ of vertices in $V_{\mathrm{pre}}(t)$ can only be non-empty if vertices of $C$ form a clique in the pre-empathy graph, namely $(u, v) \in E_{\mathrm{pre}}(t)$ for any $u, v \in C$. We also naturally define a pivot set $\Pi_{\mathrm{post}}(C')$ for every set $C'$ of vertices in the post-empathy graph $G_{\mathrm{post}}(t)$. Intuitively, given a group of traceroute paths that behave similarly to each other, the pivot is the set of vertices that appear in the changed portions of all these paths. We now state the most important properties of the empathy graph, which we use in Section VI to devise an algorithm that searches for events based on the observed traceroute paths.

Theorem 1: For every visible event $E^1$ at time $t$ the following conditions hold: i) there exists one clique in $G_{\mathrm{pre}}(t)$, namely a set $C \subseteq V_{\mathrm{pre}}(t)$ of vertices such that, for every $u_1, u_2 \in C$ there is an edge $(u_1, u_2) \in E_{\mathrm{pre}}(t)$; ii) $C = S(E^1)$; iii) $\Pi_{\mathrm{pre}}(C) \neq \emptyset$; iv) $C$ forms an isolated connected component in $G_{\mathrm{pre}}(t)$, namely for any two vertices $v \in C$ and $w \in V_{\mathrm{pre}}(t) \setminus C$ it is $(v, w) \notin E_{\mathrm{pre}}(t)$. The statement also applies to an event $E^2$, by replacing $G_{\mathrm{pre}}(t)$ by $G_{\mathrm{pre}}(t^1, E_{\mathrm{pre}}(t))$ with $G_{\mathrm{post}}(t) = (V_{\mathrm{post}}(t), E_{\mathrm{post}}(t))$. 

Proof: Suppose there is a visible event $E^1$ at time $t$. By definition of visible event and by Property 1 there must be a $C$ of vertices in $V_{\mathrm{pre}}(t)$ that form a clique. As stated in the proof of Property 1, the hub of $E^1$ must appear in $\delta_u^{\mathrm{pre}}(t)$ for every $u \in C$, which also implies that $\Pi_{\mathrm{pre}}(C) \neq \emptyset$. By construction, every vertex in $S(E^1)$ is part of the clique, namely $C = S(E^1)$, and for every other vertex $w \in V_{\mathrm{pre}}(t) \setminus C$, for which by Property 2 there exists an event $E^1$ at time $t$ such that $w \in S(E^1)$, it must be $E^1 \neq E^2$. By Property 3 this also excludes the existence of any edges $(v, w) \in E_{\mathrm{pre}}(t)$, with $v \in C$. The same arguments apply for an event $E^2$. □

The following theorems establish a relationship between the structures of $G_{\mathrm{pre}}(t)$ and $G_{\mathrm{post}}(t)$, which we also use in the algorithm in Section VI.

Theorem 2: If there exists a visible event $E^1$ (respectively, $E^2$) at time $t$, then every set $C$ of vertices in $G_{\mathrm{post}}(t)$ (respectively, $G_{\mathrm{pre}}(t)$) that form a connected component with at least one vertex in $S(E^1)$ (respectively, $E^2$) is such that $C \subseteq S(E^1)$ (respectively, $C \subseteq S(E^2)$).

Proof: Assume that there is a visible event $E^1$ at time $t$ and suppose, by contradiction, that one of the connected components in $G_{\mathrm{post}}(t)$ is formed by a set $C$ of vertices such that one vertex $v \in C$ is in $S(E^2)$ and another vertex $w \in C$ is not in $S(E^2)$. By Property 2 there must be another event $E^2 \neq E^1$ such that $w \in S(E^2)$, leading to an absurd because, by Property 3, edge $(v, w) \in E_{\mathrm{post}}(t)$ should not exist. Similar arguments can be applied for an event $E^2$. □

From Theorems 1 and 2 it is possible to deduce that the structure of $G_{\mathrm{pre}}(t)$ and $G_{\mathrm{post}}(t)$ in the presence of network events is the following: for each event $E^1$ there is an isolated clique in $G_{\mathrm{pre}}(t)$ that spans all sd-pairs in $S(E^1)$, and one or more connected components in $G_{\mathrm{post}}(t)$ that are formed by sd-pairs in $S(E^2)$. Our algorithm in Section VI is based on recognizing such patterns in $G_{\mathrm{pre}}(t)$ and $G_{\mathrm{post}}(t)$.

The following property is another direct consequence of Theorems 1 and 2.

Property 4: Given any two events $E^1$ and $E^2$ occurred at time $t$, it is always $\Pi_{\mathrm{pre}}(S(E^1) \cap S(E^2)) = \emptyset$ and $\Pi_{\mathrm{post}}(S(E^1) \cap S(E^2)) = \emptyset$.

Proof: The statement immediately follows by considering that, by Theorem 1 cliques in $G_{\mathrm{pre}}(t)$ or in $G_{\mathrm{post}}(t)$ corresponding to distinct events are isolated from each other and, by Theorem 2, the same isolation applies to any other connected components in $G_{\mathrm{pre}}(t)$ or in $G_{\mathrm{post}}(t)$. □

The following two theorems state that network events can be pointed out by searching for cliques in empathy graphs.

Theorem 3: If a set $C \subseteq V_{\mathrm{pre}}(t)$ of at least 2 sd-pairs forms a maximal clique in $G_{\mathrm{pre}}(t)$ and it is $\Pi_{\mathrm{pre}}(C) = \emptyset$, then $\Pi_{\mathrm{pre}}(C) \neq \emptyset$ and there is a unique visible event $E^3$ at time $t$ whose hubs are in $\Pi_{\mathrm{pre}}(C)$. The theorem can be restated by swapping $G_{\mathrm{pre}}(t)$ with $G_{\mathrm{post}}(t)$, $V_{\mathrm{pre}}(t)$ with $V_{\mathrm{post}}(t)$, $\Pi_{\mathrm{pre}}(C)$ with $\Pi_{\mathrm{post}}(C)$, and $E^3$ with $E^4$.

Proof: Consider a clique $C \subseteq V_{\mathrm{pre}}(t)$ consisting of at least 2 sd-pairs and such that $\Pi_{\mathrm{pre}}(C) = \emptyset$. By Assumption 6 at least one event $E^3$ whose scope involves sd-pairs in $C$ must have occurred at time $t$, and this event is obviously visible. In addition, because sd-pairs in $C$ form a clique in $G_{\mathrm{pre}}(t)$, there can be no more than one event $E^3$ whose scope involves sd-pairs in $C$, otherwise Assumption 5 is violated. This, together with the fact that the clique is maximal, implies that $S(E^3) = C$. Since $\Pi_{\mathrm{post}}(C) = \emptyset$, there is no way of constructing a single up event $E^3$ such that $S(E^3) = C$, therefore there must exist a unique down event $E^3$ at time $t$ such that $S(E^3) = C$. Finally, since $E^3$ is unique, $\Pi_{\mathrm{pre}}(C)$ cannot be empty and must contain the hubs for $E^3$. □

Theorem 4: If a set $C$ of at least 2 sd-pairs forms a maximal clique in $G_{\mathrm{pre}}(t)$ such that $\Pi_{\mathrm{pre}}(C) \neq \emptyset$ and a clique in $G_{\mathrm{post}}(t)$ such that $\Pi_{\mathrm{post}}(C) \neq \emptyset$, there exists a unique visible event $E^5$ at time $t$ whose hubs are in $\Pi_{\mathrm{pre}}(C) \cup \Pi_{\mathrm{post}}(C)$.

Proof: By exploiting Assumptions 5 and 6 and applying arguments similar to those used in the proof of Theorem 3 we can deduce that there exists exactly one event in $E^3$ at time $t$ and, since $E^3$ is unique, the hubs of $E^3$ must be contained in $\Pi_{\mathrm{pre}}(C) \cup \Pi_{\mathrm{post}}(C)$. Differently from Theorem 3 it is not possible to disambiguate the type of event, because $\Pi_{\mathrm{pre}}(C) \neq \emptyset$ and $\Pi_{\mathrm{post}}(C) \neq \emptyset$. □

We now exploit Fig. 4 to make an example of application of Theorem 5.

In this figure we only have a clique $C = \{(s_1, d_1), (s_2, d_2)\}$ in $G_{\mathrm{pre}}(t)$ and no cliques in $G_{\mathrm{post}}(t)$, therefore it is $\Pi_{\mathrm{post}}(C) = \emptyset$. Indeed, in this example we have $\Pi_{\mathrm{pre}}(C) = \{5, 6\}$, and it can be easily checked that both 5 and 6 are hub vertices for the event $E^5 = \{(5, 6)\}$ which has actually occurred at time $t$.

In order to find traceroute paths that behave similarly over time, we need to search for events that may have influenced these paths. The above theorems suggest that hubs for these
events can be searched in the pivot set of maximal cliques in $G^{\text{pre}}(t)$ or $G^{\text{post}}(t)$. Vice versa, IP addresses occurring in $\delta_i^{\text{pre}}(t)$ or $\delta_i^{\text{post}}(t)$ of many sd-pairs $i$ provide strong hints about the existence of maximal cliques. These properties are at the basis of the methodology described in Section VI.

VI. SEEKING EMPATHY: METHODOLOGY AND ALGORITHM

In this section, we describe an inference algorithm for detecting network events and reporting traceroute paths that, in consequence of these events, behave similarly to each other. The algorithm takes as input a set of traceroute paths, and produces as result a list of inferred events, each equipped with an interval of time in which the event is supposed to be happened, a set of sd-pairs affected by the event, the inferred type of event, and a set of IP addresses that appeared in, or disappeared from, traceroutes of all affected sd-pairs.

We refer to the model illustrated in the previous sections. However, for the sake of applicability we relax Assumption 2 in the sense that we consider a continuous time model and allow non-synchronized measurements. Indeed, misalignments in time between traceroute measurements can improve the accuracy of the interval that our algorithm reports for an inferred event. Also, we consider routing changes as instantaneously propagated in the network.

We assume that, for an sd-pair $i$, traceroute paths $p_i(t)$ are only available at specific time instants $t \in \mathbb{R}$ that depend on $i$. A transition $\tau_i$ for $i$ is a pair of consecutive traceroutes $p_i(t_1), p_i(t_2)$ such that $t_1 < t_2$ and $p_i(t_1) \neq p_i(t_2)$. We say that $\tau_i$ is active between $t_1$ and $t_2$. We call $t_1$ and $t_2$ the endpoints of the transition.

Assuming that instant $\bar{t}$ corresponds to $t_1$ and $\bar{t} + 1$ corresponds to $t_2$, we naturally extend the definition of the changed portions of path $p_i$ by indicating with $\delta_i^{\text{pre}}(\tau_i)$ the contents of sequence $\delta_i^{\text{pre}}(\bar{t})$. Similarly for $\delta_i^{\text{post}}(\tau_i)$. Moreover, for a transition $\tau_i$ we define a set $\eta(\tau_i)$ of extended addresses, consisting of IP addresses in $V(\delta_i^{\text{pre}}(\tau_i))$ labeled with a tag $\text{pre}$ and IP addresses in $V(\delta_i^{\text{post}}(\tau_i))$ labeled with a tag $\text{post}$.

Our algorithm is divided in three phases.

Phase 1 – Transitions identification: in this phase, for each sd-pair $i$, input samples $p_i(t)$ are scanned and all transitions $\tau_i$, with the corresponding $\eta(\tau_i)$, are identified.

Phase 2 – Cliques extraction: in this phase, the algorithm tracks the evolution of cliques in empathy graphs which, by Theorems 3 and 4, correspond to network events. This phase is detailed in Fig. 5. At a certain time instant $t$, the structure of $G^{\text{pre}}(t)$ and $G^{\text{post}}(t)$ is determined by the sd-pairs corresponding to the transitions that are active at time $t$. Therefore, the algorithm keeps track of the size of cliques by sweeping all endpoints of transitions ordered by time (line 5) and by maintaining, for each extended address $n$, the set of sd-pairs $(C_n^{\text{now}})$ at lines 9 and 19 associated with transitions that are active at time $t$ and in which $n$ is involved. In particular, the composition of these sets is updated depending on the fact that transitions end (line 7) or start (line 17) at time $t$. In order to be able to ascribe transitions with a common endpoint to different events, the algorithm must consider any transitions starting at $t$ (line 17) only after all transitions ending at $t$ (line 7). The algorithm points out temporally local maxima in the size of these sets (line 10) and returns a tuple for each such maximum corresponding to a clique in an empathy graph and specifying the interval of validity of the clique, the set of involved sd-pairs, and the related extended address. In general, if more than one extended address has a local maximum at time $t$, many tuples are reported. Also, differently from what is indicated in Theorems 3 and 4, not only maximal cliques are detected in this phase, but all cliques whose pivot set is not empty. We argue that this improves the effectiveness of our algorithm when applied to real-world data, because multiple simultaneous events that interfere with each other (i.e., violate Assumption 5) are also detected, even if their scope can only be identified with a limited precision.

Phase 3 – Cleanup: in this phase, the cliques recognized in the previous phase are sieved to build a set of inferred events, each characterized by a time interval, a scope (a set of affected sd-pairs), a set of involved IP addresses, and a type (up/down/unknown). This phase is detailed in Fig. 6. Set $K$ in Fig. 6 is the set of tuples generated in Phase 2 such that $t_{\text{start}} \leq t < t_{\text{end}}$. In lines 4 and 10 all cliques whose set of sd-pairs is contained in the set of sd-pairs of another clique are discarded, a step that is useful because Phase 2 also outputs non-maximal cliques. Intuitively, in this step we prevent cliques from improperly being inferred as events if their set of sd-pairs is covered by the scope of another event. Additionally, for a down (up) event at time $t$ this step leaves out all connected components in $G^{\text{post}}(t)$ ($G^{\text{pre}}(t)$), which are
Input: a set \( \mathcal{C} \) of tuples produced as output in phase 2 (see Fig. 3).

Output: a set \( \mathcal{E} \) of tuples \((t_{\text{start}}, t_{\text{end}}, S, \Pi, \text{type})\), where each tuple is an inferred event happened between \( t_{\text{start}} \) and \( t_{\text{end}} \), whose scope is \( S \), which involved the IP addresses in \( \Pi \), and whose type is \text{type}.

1: \( K \leftarrow \emptyset \)
2: for every time \( t \) appearing as \( t_{\text{start}} \) or \( t_{\text{end}} \) in tuples of \( \mathcal{C} \) do
3: Let \( \mathcal{C}^{\text{t}} \) be the set of tuples in \( \mathcal{C} \) that end at \( t \)
4: for every \( e^{\text{c}} = (t_{\text{start}}, t_{\text{end}}, C^{\text{c}}, n) \in \mathcal{C}^{\text{t}} \) do
5: \( K \leftarrow K \cup \{e^{\text{c}}\} \)
6: if \( C \subset C^{\text{t}} \) then
7: \( K \leftarrow K \setminus \{e^{\text{c}}\} \)
8: end if
9: end for
10: end for
11: For every arbitrary set \( C \) of sd-pairs, let \( P[C] = \emptyset \)
12: for \( e^{\text{c}} = (t_{\text{start}}, t_{\text{end}}, C, n) \in C \cap K \) do
13: \( T[C] \leftarrow t_{\text{start}} \); \( P[C] \leftarrow P[C] \cup \{n\} \)
14: \( K \leftarrow K \setminus \{e^{\text{c}}\} \)
15: end for
16: for every key \( C \) of \( P \) do
17: if all addresses in \( P[C] \) are tagged as \text{pre} then
18: \( \text{type} \leftarrow \text{down} \)
19: else if all addresses in \( P[C] \) are tagged as \text{post} then
20: \( \text{type} \leftarrow \text{up} \)
21: else
22: \( \text{type} \leftarrow \text{unknown} \)
23: end if
24: \( \mathcal{R} \leftarrow \mathcal{R} \cup \{(T[C], t, C, P[C], \text{type})\} \)
25: end for
26: Add to \( K \) all tuples of \( C \) such that \( t_{\text{start}} = t \)
27: end for

Fig. 6. Phase 3 of our algorithm: it reports inferred events starting from a set of cliques produced in Phase 2.

Formed according to Theorems 1 and 2. For every clique \( C \) that passed this step, in lines 11-25 Phase 3 of our algorithm defines an inferred event, builds the corresponding set \( P[C] \) of involved extended addresses, and determines the event type based on the composition of this set.

Fig. 7 graphically describes sample outputs of the three phases of our algorithm. This example illustrates the detection of a down event whose hub is vertex 1. At the end of Phase 1, three transitions are singled out, indicated in the figure with segments associated with the corresponding endpoints and sets \( \delta_{\text{pre}}(\tau) \) and \( \delta_{\text{post}}(\tau) \). \( a, b, \) and \( c \) represent the sd-pairs of the paths of each transition. After Phase 2, three cliques are constructed: \( \{a, b, c\} \) with extended address \( \delta_{\text{pre}} \), \( \{a, b\} \) with extended address \( \delta_{\text{post}} \), and \( \{b, c\} \) with extended address \( \delta_{\text{post}} \). Intervals of validity of each clique, where each extended address is shared by a maximal number of sd-pairs, are indicated with segments. After Phase 3, clique \( \{a, b, c\} \) is discarded because it overlaps in time with \( \{a, b, c\} \) and all its sd-pairs are contained in \( \{a, b, c\} \). The same happens to clique \( \{b, c\} \). Finally, the only remaining clique is reported as an event occurred between \( t_3 \) and \( t_4 \), affecting the paths between sd-pairs \( a, b, \) and \( c \), involving vertex 1 and with type down.

The output of our algorithm is provably correct and complete, as stated by the following theorems.

Theorem 5 (Correctness): Each event inferred by our algorithm corresponds to one visible event.

Proof: Suppose tuple \((t_1, t_2, S, \Pi, \text{type})\) is part of the output of Phase 3. For this to happen, a tuple \( \theta = (t_1, t_2, S, \cdot) \), where \( \cdot \in \Pi \), must be part of the output \( C \) of Phase 2. Additionally, \( S \) must not be a subset of any other set of sd-pairs in other tuples of \( C \), otherwise \( \theta \) would have been discarded by Phase 3. By construction, for each \( i \in S \), transition \( \tau_i \) overlaps with interval \((t_1, t_2)\), it is \( \eta(\tau_i) \), and, for every \( i \in S \), it can only be either \( v \in V(\delta_{\text{pre}}(\tau_i)) \) or \( v \in V(\delta_{\text{post}}(\tau_i)) \): since Phase 3 of our algorithm only preserves maximal cliques, by Theorems 3 and 4 clique \( \theta \) correctly corresponds to an event.

Theorem 6 (Completeness): For each visible event, an inferred event is reported by our algorithm.

Proof: Suppose a visible event \( E \) occurs at time \( t \), with a hub \( v \). Under Assumption 2, all sd-pairs \( S(E^\downarrow) \) are therefore pre-empathic. In the relaxed scenario adopted in this section each sd-pair \( i \in S(E^\downarrow) \) has a transition whose interval contains \( t \), therefore the time intervals of all transitions caused by \( E^\downarrow \) intersect in a common interval \([t_1, t_2]\) containing \( t \). Moreover, by construction vertex \( v \) appears in \( \delta_{\text{pre}}(\tau_i) \) for each \( i \in S(E^\downarrow) \), which means that the size of the clique that has \( v \) as extended address will reach its maximum in \([t_1, t_2]\). Our algorithm detects such maximum in Phase 2 when the sweep is at \( t_2 \), because at least one transition caused by \( E^\downarrow \) ends at \( t_2 \), producing a drop in the number of sd-pairs associated with \( v \). The output of Phase 2 therefore includes a tuple \((t_1, t_2, S(E^\downarrow), v)\). Moreover, this tuple will not be discarded in Phase 3 because \( S(E^\downarrow) \) is the largest clique induced by \( E^\downarrow \), and any other events induce disjoint cliques by Property 3. Finally, set \( S(E^\downarrow) \) includes at least \( v \), which means that the algorithm infers a visible down event. Similar arguments can be applied for a visible up event.

VII. Applicability Considerations

In this section we discuss some hypothesis we rely on and their impact on the results of our approach.

A. Time-Related Assumptions

According to Assumption 2 at each time instant \( t \), every source performs an instantaneous traceroute measurement towards every configured destination. We relax this assumption in the algorithm described in Section VI by assuming that time is continuous while retaining the assumption on the
instantaneous traceroutes which is a good approximation of what happens in a real network.

There are two more assumptions that have been implicitly made regarding time: 1) the internal clock of the probes is properly set, and 2) an event is instantly propagated to the entire network. The internal clock of the probes is usually kept synchronized by NTP, whose precision is quite high with respect to the needs of our methodology, even in the case of asymmetric bandwidth lines like ADSL. So, we think this is not a relevant issue.

Regarding the delay in the dissemination of routing messages, it is well known that certain routing events take some time to propagate (e.g. BGP advertisements might be delayed by the MRAI timer, by 30 seconds according to the standard). On the other hand, link state routing protocols, like OSPF, are much faster to converge. So, when BGP is involved, at a given time some routers may see (and propagate) a stale version of the routing. In this case, it might happen that not all transitions overlap on a common interval (see Section VI), with the effect of having multiple inferred events usually with many sd-pairs in common.

Assumption 5 also involves timing. It mandates that two distinct events should neither overlap in time nor in the changes they induce on traceroute paths.

It is clear that in real world data this assumption may not hold. Two interfering events may lead to several effects. First, it is no longer true that distinct down (up) events lead to distinct connected components in $G_{\text{pre}}^t$ ($G_{\text{post}}^t$), as stated in Theorem 1 but, since our methodology starts from elements of the pivot set, it is likely to still infer the right events. However, IP addresses involved in more than one event induce inference of fictitious events having as inferred scope the interfering sd-pairs. Second, it is no longer true that for each down (up) event with scope S there are in $G_{\text{post}}^t$ ($G_{\text{pre}}^t$) several connected components contained in S (Theorem 2). E.g., two down events can have sd-pairs with interfering $d_{\text{post}}^t$. This results in fictitious up events for the interfering sd-pairs.

Another relevant effect of interference is the skewing of the timings of the inferred events. It might happen that a large number of sd-pairs agree on an event $E_{\text{dt}}$ with interval $(t_1, t_2)$ and just a single distinct sd-pair interferes, because of another event, with interval $(t_2 - \epsilon, t_3)$. In this case our methodology reports only one event with interval $(t_2 - \epsilon, t_2)$ which might be quite distant from the instant in which the real event happened. This kind of behavior can be observed in the experimental results in Section VII-B.

B. Handling Load Balancers

Load balancers cannot be ignored in our analysis, since they are the cause of many routing changes which are reported as events by our algorithm while they are usually not interesting. Our approach for handling load balancers is to suppose they are known and to pre-process the traceroutes to substitute each IP address belonging to a load balancer with one fixed representative address.

We claim that such assumption is not dramatic. First, an ISP willing to apply our methodology already knows the load balancers in its network. Second, Paris Traceroute [20] is a well-known variation of the traceroute measurement that is able to find load balancers by exhaustively exploring routing paths in a network. This kind of measurement can be potentially implemented in the same probes that perform traceroutes to gather this information.

The probes used in the experiment considered in Section VIII do not support Paris Traceroute, so we applied a very rough heuristic to detect load balancers using data produced by standard traceroutes, which performed well enough for our evaluation purposes. We analyzed traceroutes in a time-ordered manner. For each destination we learned the routing of all nodes that appear in each traceroute and their evolution over time. We computed some simple statistics on route changes for each node. Neighbors that are not “stable” enough (in our case when more than 20% of the samples are changes) are considered to belong to a load balancer.

C. Non-Ideal Traceroutes

According to Assumption 1 every network device (e.g. router) is identified by a unique IP address. In practice a router can send replies with different source IP addresses, a phenomenon known as aliasing [21, 22]. If a router replies with a different IP address each time it is probed, then it is considered as a load balancer by our heuristic. If a router replies with deterministically chosen IP address we view that single router as several different routers. This can lead us to failure when flagging two sd-pairs as empathic if their changed portions only share the router affected by aliasing. In our experience aliasing turned out not to be a relevant problem.

According to Assumption 3 traceroutes are acyclic. This is a reasonable assumption for a working network. Performing traceroutes may still erroneously report a cyclic path in several peculiar cases. We simply discard all cyclic traceroutes.

According to Assumption 4 an event can have impact on at most one contiguous portion of a traceroute. This is quite reasonable when load balancers are properly handled as described in Section VII-B.

According to Assumption 6 a traceroute path can change from time instant $t$ to $t + 1$ only due to an event that is either in $p_i(t)$ or in $p_i(t + 1)$. This is a well known and discussed hypothesis in the context of root-cause analysis (see [1, 2]). Since our approach does not aim at detecting the root cause of an event, we think that this is not a problem in our context.

VIII. Experimental Results

A. Detection of BGP Reconfigurations

In this section we present one experiment conducted on publicly available data collected by RIPE Atlas [6] probes. The measurements were performed to validate some hypotheses on the reachability of an Italian ISP under different BGP announcement settings. We reuse such experiment to validate our technique, comparing the results of our inference algorithm with the chronological sequence of BGP reconfigurations.
The involved ISP has BGP peerings with three main upstream providers and with a number of ASes in three Internet eXchange Points (IXPs), i.e. MIX and NaMeX (the main IXPs in Italy) and AMS-IX. An IP subnet was reserved for the purposes of the experiment. In six consecutive 4-hour windows the subnet was announced via BGP to different subsets of peers: see the second column of Table 1 for further details. During the experiment, 89 RIPE Atlas probes located in Italy were instructed to perform traceroutes every 10 minutes (between 2014-05-02 13:00 UTC and 2014-05-03 15:00 UTC) targeting a host inside the reserved subnet. We fed the algorithm described in Section VII with the collected traceroute measurements, after applying the load balancers cleanup procedure described in Section VII-B. Finally, we applied a filter to the computed involved events, discarding those involving less than 5 source-destination pairs.

Table 1 presents a summary of the output of the whole procedure, split in different phases. Each odd row represents a set of events that happened within a 2-minute window centered at the time of the actual BGP announcement, to account for potential synchronization issues between probes and MRRAI intervals (note that such time interval is much shorter than the 10-minute period of traceroutes). The remaining rows contain events happening between two of such announcements. We obtained 33 events (144 before the load balancers cleanup). Of these, 22 belong to the first category. A closer analysis of the remaining 11 events easily reveals their nature. In particular, one is caused by intra-domain changes in a specific AS, hence completely independent on the experiment. All the other events happen in a range of 10 minutes from a BGP announcement and are the effect of the interference between intra-domain changes independent from the experiment and the crafted BGP announcements. These events are wrongly inferred as a single event due to the violation of Assumption 5 (see the discussion in Section VII for details).

We focused on the 22 events with compatible times and manually went through them to verify the correctness of the inference. Generally, we expected at most one inferred event for each upstream or IXP involved in each of the BGP announcements. For example, after the first announcement we expected at most five inferred events, of which two caused by probes disconnecting from the two IXPs and three by the same probes redistributing between the three available upstreams. That is motivated by the fact that the experiment intrinsically violates Assumption 5 because BGP policies in each of the six phases are mirrored at each BGP-speaking router of the announcer, causing many events to occur at the same time. In one case (third phase, fifth row in Table 1) we see exactly one event that precisely describes the migration of 29 probes from MIX to NaMeX, so the detection is optimal. In the first phase we see 7 inferred events instead of the expected 5: two of them are basically the same (two probes present asterisks in their traceroutes that cause the split in two events), while two others seem to hint at a backup peering with one of the three upstreams that was not declared by the partner ISP at the time of the experiment. Afterwards, the ISP has confirmed the existence of that backup peering by private communication. In each of the remaining phases we see exactly one “duplicate” event, caused by either asterisks in traceroutes or the upstream provider mentioned before. Apart from such exceptions, all inferred events precisely meet our expectations by pointing at representative IPs that appear or disappear in the traceroutes.

Further, we expected consistency across different events, e.g. a substantial overlap between sets of probes disconnecting and reconnecting to the same IXP or upstream, under the reasonable assumption that the BGP policies of their hosting ASes were not modified during the experiment. Our analysis fully confirms the hypotheses with respect to at least the two Italian IXPs and the two upstreams for which we made no further hypothesis. For example, MIX and NaMeX are respectively seen by the same 29 and 28 probes in all their disconnections and reconnections. Conversely, we report that the third IXP and the upstream with a backup peering are somehow correlated, in that the latter seems to attract most of the traffic in the fifth phase, thus making it harder to evaluate the consistency.

B. Detection of Accidental Outage

To validate our inference algorithm we also used measurements performed during a real-world Internet outage happened within the network of a single provider. Let us call such provider X and let us say that the outage happened on d/m/y from t₁ to t₂.

We built our ground truth as follows. We preliminarily scanned the BGP activity for hundreds of prefixes announced by X and seen by RIPE RIS ¹ in month m. We found out that many were involved in hundreds of announcements and withdrawals in the day d of the outage. We then studied the most active prefixes with BGPlay ² and confirmed a partial disconnection of X.

We then fed our algorithm with traceroutes performed by 20 probes inside X towards a number of destinations

¹Routing Information Service RIPE NCC (http://ris.ripe.net)
²BGPlay RIPE NCC (https://stat.ripe.net/widget/bgplay)
outside X, during the outage. Measurements for each sd-pair are continually performed with periodicity of one hour. The granularity is therefore quite coarse, but still enough to detect the disconnection. We computed the events and filtered out those involving less than 5 probe-destination pairs.

We counted 31 inferred events between \( d - 1 \) and \( d + 1 \). Of these, two small batches (of 6 and 5 events, respectively) had timing compatible with the outage observed on BGP, involved 11 probes, and were accompanied by dramatic differences in the related traceroutes. In particular, the first batch of events contains traceroutes that only receive replies from hosts inside X and fail to reach the target. That is easily explained by the lack of a reverse path that would allow routers in other ASes to send ICMP replies to the probe. The second batch confirms the restoration of connectivity with traceroutes equivalent to the ones performed before the outage. To better understand the difference between the two identified sets of probes, we used BGPlay to study the connectivity of prefixes including their public IP during the outage. We found out that those including the affected probes lost connectivity to a large number of ASes, while the remaining managed to survive through backup links.

This experiment allowed us to build a very good evidence of the effectiveness of our algorithm in establishing strong initial hypotheses about a real-world outage. Indeed, based on the events inferred by the algorithm we could perform several observations on the nature and possible timeline of the outage, which would have been extremely difficult to derive without such an automated tool and which were later confirmed against available information.

IX. Conclusions and Future Work

We have presented a model and methodology for the identification and analysis of network events based on the notion of empathic traceroute measurements. We have translated our theoretical approach into an algorithm and applied it to real-world data, proving the effectiveness of our methodology.

We plan to further validate our approach with other measurement platforms (see Section II for examples), topologies, and network events. We will focus in particular on intra-domain routing events as opposed to BGP routing changes, which we proved to be more likely to break our theoretical assumptions. Further, we will study heuristics to merge two or more inferred events that are likely to represent one single network event.

REFERENCES

[1] U. Javed, I. Cunha, D. Choffnes, E. Katz-Bassett, T. Anderson, and A. Krishnamurthy, “Postout: Investigating the root cause of interdomain path changes,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 183–194, Aug. 2013.
[2] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, “Locating Internet routing instabilities,” SIGCOMM Comput. Commun. Rev., vol. 34, no. 4, pp. 205–218, Aug. 2004.
[3] RIPE NCC, “Bgpplay,” http://stat.ripe.net/widget/bgpplay
[4] L. Colitti, G. Di Battista, F. Mariani, M. Patrignani, and M. Pizzonia, “Visualizing interdomain routing with BGPlay,” J. Graph Alg. and App., vol. 9, no. 1, pp. 117–148, 2005.
[5] “SamKnows,” https://www.samknows.com.
[6] “RIPE Atlas,” http://atlas.ripe.net.
[7] “MLab,” http://www.measurementlab.net.
[8] “Ark,” http://www.caida.org/projects/ark.
[9] M. Caesar, L. Subramanian, and R. H. Katz, “Towards localizing root causes of BGP dynamics,” Computer Science Division, University of California, Tech. Rep. CSD-03-1292, 2003.
[10] D.-F. Chang, R. Govindan, and J. Heidemann, “The temporal and topological characteristics of BGP path changes,” in Proc. ICNP, 2003.
[11] M. Lad, A. Nanavati, D. Massey, and L. Zhang, “An algorithmic approach to identifying link failures,” in Proc. IEEE Symposium on Dependable Computing, 2004.
[12] J. Zhang, J. Rexford, and J. Feigenbaum, “Learning-based anomaly detection in BGP updates,” in Proc. ACM SIGCOMM workshop on Mining network data, 2005.
[13] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, “Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network,” in Proc. NSDI, 2005.
[14] S. T. Teoh, S. Ranjan, A. Nucci, and C.-N. Chau, “BGP eye: a new visualization tool for real-time detection and analysis of BGP anomalies,” in Proc. workshop on Visualiz. for computer security, 2006.
[15] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. E. Anderson, “Studying black holes in the Internet with Hubble,” in Proc. NSDI, 2008.
[16] E. Katz-Bassett, C. Scott, D. R. Choffnes, I. Cunha, V. Valucius, N. Fearmster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy, “Lifeguard: Practical repair of persistent route failures,” ACM SIGCOMM Comput. Commun. Rev., vol. 42, no. 4, pp. 395–406, 2012.
[17] M. Candela, M. Di Bartolomeo, G. Di Battista, and C. Sguarcia, “Dynamic traceroute visualization at multiple abstraction levels,” in Proc. 21st Int. Symp. on Graph Drawing, ser. LNCS, vol. 8242, 2013.
[18] F. Fischer, J. Fuchs, P.-A. Vervier, F. Mansmann, and O. Thonnard, “Vistracer: a visual analytics tool to investigate routing anomalies in traceroutes,” in Proc. Int. Symp. on Visualiz. for Cyber Security, 2012.
[19] M. Lad, L. Zhang, and D. Massey, “Link-rank: A graphical tool for capturing BGP routing dynamics,” in Proc. NOMS, 2004.
[20] B. Augustin, T. Friedman, and R. Teixeira, “Measuring load-balanced paths in the Internet,” in Proc. IMC, 2007.
[21] P. Marchetta, V. Persico, and A. Pescaè, “Pythia: yet another active probing technique for alias resolution,” in Proc. CoNEXT, 2013.
[22] J. Sherry, E. Katz-Bassett, M. Pimenova, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy, “Resolving IP aliases with prespecified timestamps,” in Proc. IMC, 2010.