# Radian: Visual Exploration of Traceroutes

Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, and Claudio Squarcella

**Abstract**—There are several projects that deploy probes in the Internet. Probes are systems that continuously perform traceroutes and other networking measurements (e.g. ping) towards selected targets. The collected measurement data are stored and then studied to gain knowledge on several aspects of the Internet. The need to understand such a huge amount of data requires suitable exploration and visualization methods and tools. We present Radian, a tool for the visualization of traceroute paths collected by the Internet probes deployed by measurement projects. Radian allows to visualize traceroute paths at different levels of detail and to animate their evolution during a selected time interval. Radian has been extensively tested on traceroutes performed by RIPE Atlas [1] Internet probes.

**Index Terms**—Computer Society, IEEE, IEEEtran, journal, LaTeX, paper, template.

## 1 Introduction

There are several organizations that distribute *probes* in the Internet with the aim of monitoring the status of the network and of measuring its performance. A probe is a typically unattended device that periodically executes popular network commands like ping, traceroute, etc. towards selected targets. A few examples of such organizations follow. SamKnows [3] probes (tens of thousands) are distributed world-wide to get broadband performance data for consumers, governments, and Internet Service Providers (*ISPs*). BISmark [4] uses probes for measuring the performance of ISPs. RIPE Atlas [1] is an open project of RIPE-NCC whose probes (almost ten thousands) can be used by anyone willing to host a RIPE Atlas probe to conduct customised measurements. MisuraInternet [5] is a probe-based project of the Italian Authority for Telecommunications that measures the quality of broadband access. Other notable examples are CAIDA Ark [6] and M-Lab [7]. Organizations providing systems of probes store the results of measures into huge repositories of data, which are hard to analyze without some visualization facility. This is especially evident for traceroute data, which combine topological and performance information. *Traceroute* is one of the most popular computer network diagnostic tools and is probably the simplest tool that can be used to gain some knowledge on the Internet topology. It can be executed to get the path of routers (*traceroute path*) traversed to reach an IP.

In this paper we present a tool, called Radian, for the visualization of traceroute data collected by a system of probes. The requirements of Radian were gathered interacting with several ISPs, within the Leone FP7 EC Project. Radian works as follows. The user selects a set $\mathcal{S}$ of probes of a certain Internet measurement project (all the experiments described in this paper were performed using RIPE Atlas [1] probes), a target IP address $\tau$, and a time interval $\mathcal{T}$, and obtains a visualization of how the traceroutes issued by the probes in $\mathcal{S}$ reach $\tau$ during $\mathcal{T}$. A snapshot of Radian is in Fig. 1. Also, a demo of the tool is available at [8]. Our tool contributes to the state of the art with: 1) a metaphor for visualizing the dynamics of Internet routing as captured by traceroutes; 2) the use of *Autonomous Systems* for allowing the user to simplify the visualization; and 3) new algorithms tailored for traceroute data.

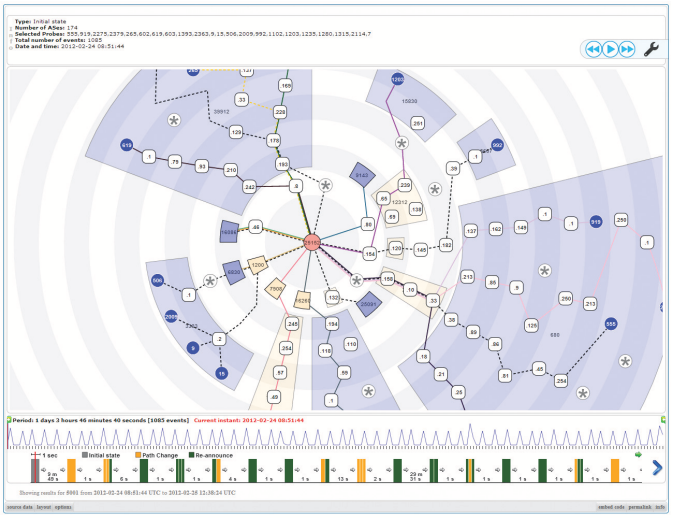

Fig. 1. The main interface of Radian.

The paper is organized as follows. Section 2 describes the scenario that originated Radian. In Section 3 we discuss how our contribution relates with the state of the art. Section 4 introduces basic terminology. In Section 5 we present an analysis of the data of interest, whose results influenced the design of our tool. Section 6 describes the adopted visual metaphor and the user interface of Radian. Section 7 illustrates the algorithmic apparatus devised for Radian. Section 8 describes a user study performed with domain experts. Finally, conclusions are in Section 9.


- *M. Candela is with RIPE NCC, Amsterdam, Netherlands. E-mail: m.candela@ripe.net*
- *M. Di Bartolomeo and G. Di Battista are with the Department of Engineering, Roma Tre University, Italy. E-mail: giuseppe.dibattista@uniroma3.it, dibartolomeo@dia.uniroma3.it*
- *C. Squarcella is with ThousandEys, Inc. San Francisco, CA, USA. E-mail: claudio@thousandeyes.com*


*Partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) grant no. 317647 (Leone). We thank RIPE NCC for collaborating to the development of the graph animation framework used in this work. A preliminary version of this paper appeared in [2]*

## 2 Reference Scenario

### 2.1 Data of Interest and Basic Networking Concepts

The Internet is a large network that can be observed at two abstraction levels. Namely, it can be seen as a network of *routers* or, since routers are grouped into *Autonomous Systems* (ASes), as a network of ASes. An AS corresponds to an administrative authority, like an *Internet Service Provider* (ISP), a company, etc, and is identified with a number called AS Number (ASN). *Traceroute* is a standard networking tool that reports the sequence of routers followed by data to reach a given destination from a given source. The reported sequence of routers, each labelled with an *IP address*, is called *traceroute path*. Each device in the traceroute path is also labeled with the *round-trip time* from the source. Several traceroute paths collected all at the same time can be merged to form a *traceroute graph* which is a partial view, at the router level, of the topology of the traversed network at that time. Also, given that each IP address belongs to an AS, a traceroute path can be converted into a sequence of ASes. Merging such paths form a graph that is a partial view of the network at the AS level. Further, since routing continuously changes, performing the above merge at different times allows to capture different states of the network.

### 2.2 Users and Use Cases

There are several types of users who could benefit from monitoring the evolution of routing with a system of probes. ISPs are interested to check the proper operation of their networks. Government authorities of a country need to verify the quality of the Internet connection offered to the citizens. Cyber-security agencies look for abuses of the Internet that may implicate cyber attacks towards or from specific countries. Finally, researchers are interested to study the Internet routing because it is a complex, only partially understood system. We focus our attention on ISPs, because we could collect clear user requirements by interacting with a few of them within the Leone FP7 EC Project. Among the activities of an ISP, we focus on three use cases that resulted from the discussion with our partner ISPs, and that could require to check the status of the routing: *troubleshooting*, *upgrade verification*, and *inter-domain consistency check*.

*Troubleshooting* copes with any unexpected event that disrupted the normal operation of the network. For example, a router could stop forwarding packets due to an overload or a damage, making some network services unreachable or slow to respond. Or, a wrong configuration due to a human mistake could make the network be used in an inefficient way, letting many paths traverse a same node and overload it. From this perspective, following traceroute paths link-by-link is a routing analogous of low-level debugging in software engineering. Also, when troubleshooting a problem, it is often useful comparing routing changes with the variations of some metric of interest (e.g. the round-trip time or the path length). For example, a sudden increment of the latency of a link could explain why many paths abandoned that link almost at the same time. Finally, even if an ISP has an alarm system deployed on its routers, on one hand traceroutes can help trace a problem that originated in a location far from an alert. On the other hand, traceroute is one of the very few analysis tools that can be used when a problem is originated in an external network, which is not under the control of the ISP.

An *upgrade* is any modification made to the network by the ISP to improve the existing services or to add new ones. New routers and links could be added to extend the current network, or to add redundancy and make the network more resistant to failures and high loads. Also, an upgrade can involve the configuration of a device. For example, a router could get configured to distribute the incoming traffic by following some load-balancing criteria (e.g. per-source, per-destination). After an upgrade the ISP needs to verify that the desired effect was obtained. Similarly to troubleshooting, this activity benefits from analyzing the dynamics of the routing. By inspecting the paths followed by packets it is possible to check if a router was traversed by traceroutes issued by selected probes when it was expected. Also, metrics associated to the paths helps assess the outcome of the upgrade. For example, a sudden drop of the latency after a path transition tells that the change was beneficial.

*Inter-domain consistency check* consists of verifying the relationship between the intended routing established by BGP and the actual routing reported by traceroutes, where *BGP* is the protocol used by ISPs to route packets through different ASes. Checking the consistency between BGP and IP routing can be useful in a few cases. For example, the ISP must check that traffic is exchanged with other partner ISPs according to commercial agreements, or that a backup link with another AS is operating correctly.

## 3 Related Work

### 3.1 Traceroute Visualization

Because of its simplicity and effectiveness, the traceroute command attracted the interest of researchers and practitioners that developed services for visualizing the traceroute paths discovered by executing one or more traceroutes. Broadly speaking, there are two types of traceroute visualization systems: tools developed for local technical debugging purposes and tools that aim at reconstructing and displaying portions of the Internet topology.

Several tools of the first type visualize a single traceroute path and display it on a map, showing the geo-location of the traversed routers. A few examples follow. Xtraceroute [9] displays traceroute paths on an interactive rotating globe as series of yellow lines between sites, shown as small spheres of different colors. GTrace [10] and VisualRoute [11] are network diagnostic tools that provide a 2D geographical visualization of paths. The latter also features non geographical representations, taking into account other information, e.g. the round-trip time between intermediate hops.

Tools of the first type are useful to analyze the path followed by the traffic from a source to a destination. In such a case, the visualization can help making sense of variations in some observed metric like the latency, by comparing it with the geographical positions of routers. However, this kind of approach does not support the exploration of data as collected from several sources at different instants, which could be exploited to find interesting routing dynamics that involved several paths. Also, drawing geolocalized graphs is particularly challenging [12]. Indeed, this kind of visualization gets easily cluttered as, for example, a traceroute

can connect two dense metropolitan networks composed of many, relatively close routers, through a long oceanic cable. Using the geographical positions to draw nodes poses strong restrictions on the layout, which cannot be further optimized for readability. In addition, ASes tend to be highly distributed, so plotting nodes at their geographical positions does not give a clear representation of the AS-structure of the portion of network traversed by traceroutes.

There are several tools of the second type that merge the paths generated by multiple traceroutes into directed graphs and show them in a drawing, focusing on the topology of the traversed network, rather than on the geographical positions of the routers. In Argus [13], the length of an edge connecting two routers depends on the one-way delay. Also, nodes belonging to a same AS tend to be visualized close to each other. Similarly to what happens with a geographical visualization, relating the edge length to the values of a metric can produce visual clutter. In the visualization provided by ThousandEys [14], paths are oriented from left to right, and nodes are positioned so that topological distances are explicit. Although the routing dynamics is not expressed in the visualization, the user can navigate through time with a slider, and and see the traceroute graph obtained at a given instant. The user can collapse parts of the topology that are not of interest, or to outline elements that experienced a degradation in performance under a metric. The information available for a node include the AS to which the node belongs. Finally, Zenmap [15] gives a radial view of the graph, with one focal node (e.g. the source of traceroutes towards several destinations, or the target of traceroutes from several sources) at the center of the drawing. Nodes are placed on concentric circles so to explicitly encode topological distances, and the thickness of an edge represents the latency. Similarly to ThousandEys, the visualization can include several paths for the same source-destination pair, but the routing dynamics are not shown. The user can collapse the "children" of a node, that is, the nodes that reach the focal node through it.

Tools of the second type have many features for supporting the use cases described in Section 2.2. First, merging traceroute paths into a graph is useful to analyze events that impacted different parts of the network. Second, these tools include layout algorithms that enhance the readability of the topology. Finally, they have the capability to reduce the visual complexity by hiding some parts of the visualization that are not of interest for a specific task and allow to correlate the traceroute graph to some metric. Some useful features, however, have been overlooked in current tools. Routing is a dynamic entity and traceroute graphs change over time, but tools present traceroute graphs more or less like static entities. Also, ASes of routers are considered only as metadata attached to nodes, not exploiting the fact that they are a natural way to cluster routers and offer a way to look at the routing at a higher level of abstraction. Finally, the graph layout algorithms are often general purpose and do not exploit the characteristics of traceroute data for improving the visualization, for example for reducing edge crossings that make the topology harder to understand.

### 3.2 Visualization of Dynamic Graphs

Merging several traceroute paths together produces a graph. If the traceroutes are collected at different time instants, the resulting graph evolves over time and is a *dynamic graph*. This is particularly relevant for our work, because visualizing the dynamics of the network helps the user understand the effect of routing events.

There is a large amount of literature on the visualization of dynamic graphs, which is well summarized in [16]. Following that work, methods for visualizing dynamic graphs can be essentially classified under: 1) visualization metaphor; 2) span of knowledge on data; 3) representation of time; 4) mental map preservation; and 5) modeling of transitions.

**Visualization Metaphor** The two principal visual metaphors used for static graphs, *node-link* and *matrices*, are also applicable to dynamic graphs. We focus on the node-link style for an important domain reason: traceroutes are paths, and following paths is easier in the node-link representation [17]. Also, this kind of diagram is intuitive for users in the networking domain.

**Span of Knowledge on Data** Visualization methods for dynamic graphs are *offline* if, at any time instant, future data are known and can be used to compose the current layout. Otherwise, they are *online*. The online setting is more versatile, because it can be applied to scenarios where data are known only up to the current instant (e.g. monitoring systems). On the other hand, the offline setting offers better opportunities to optimize the layout, by taking into account future data. Here, we focus on the offline setting, because the use cases described in Section 2.2 do not involve real-time monitoring.

**Representation of Time** Time is a sequence of instants, each associated with a different version of the graph. *Animation* is a technique for representing the flowing of time and has been applied to dynamic graphs in, e.g., [18], [19], [20], [21], [22]. It transforms the visualization of the graph from a given instant (or frame) to another one. In its simplest form, an animation shows the evolution of a dynamic graph as a movie. The strenghts of this approach are the intuitiveness, which users tend to like, and the compactness of the representation, since the amount of used screen space does not depend on the number of graph updates. The main drawback is the difficulty for users to trace the history of an object across a long sequence of frames, because this requires them to rely on their memory of past frames [23]. Also, the time consumed by an animation increases the time necessary to perform user tasks [24]. In the *timeline* approach, time is represented through a spatial coordinate. Commonly, this is done by juxtaposing several versions of the graph, one for each time instant. The juxtaposed style is also called *small multiples*. The timeline approach has been applied to the visualization of dynamic graphs in several forms (see, e.g., [25] [26] [27] [28]). The strength of this approach is the ease of following the evolution of an object along time, since time instants can be visually compared. On the other hand, discovering differences between two frames can be hard for users [16]. Also, the scalability of the technique is limited and depends on the number of shown frames, which is a trade-off between two parameters: the

detail level of the frames, and the time granularity [21].

It is still uncertain which between animation and timeline is the best for representing dynamic graphs. Past studies gave contrasting results, depending on the experimental setting and the specific tasks (see, e.g., [29] [24] [30]). In [21] the two techniques are mixed, with small multiples used as a preview and a navigation system of an animated graph. We adopt a technique inspired by the one in [21], mixing an animation approach with a timeline that offers a way to quickly identify instants of interest.

**Mental Map Preservation** The adopted layout algorithm has an impact on the preservation of the so called user's *mental map*, which is the image that users have of the information. The preservation of the mental map has been cited, since early stages, as a desired and fundamental property of a visualization of a dynamic graph [31]. The intuition behind is that the cognitive load of the user for tracking the graph evolution is reduced by using an external visual representation, instead of relying solely on his memory. Researchers do not seem to have reached a consensus on the topic yet. While some works reported better user performance under the mental map condition [32] [33] [34], others do not report significant differences [35] or even report negative performance [36] [37]. In [16], the authors conclude that the importance of preserving the mental map may have been overestimated in the past. However, a recent review paper [38] suggests that preserving the mental map may actually support users in executing specific tasks. Depending on the layout strategy, the mental map can be preserved to various degrees. At one extreme, a *global optimization* strategy fixes the positions of all graph elements across time, perfectly preserving the mental map but possibly at the expense of a suboptimal space utilization within each frame. At the other extreme, a *local optimization* strategy changes the positions at every frame, optimizing the layout of single frames but possibly prejudicing the mental map. We adopt the global optimization strategy how explained in Section 6.

**Modeling of Transitions** Users can be helped in comparing different time instants by explicitly encoding transitions, i.e. by outlining the differences between two ore more frames. Different techniques can be used depending on the adopted representation of time. In the animation approach differences can be encoded with *staged transitions*, i.e. by executing changes in different groups depending on their type [39] [40] [41]. However, even when staged, many changes at the same time can be hard to follow [21]. The approach we use on transition encoding will be described in Section 6.

### 3.3 Layout Algorithms for Dynamic Clustered Graphs

As introduced in Section 2.1, the Internet is clustered into ASes. Since in our use cases this aspect is relevant (Section 2.2), we focus on layout algorithms that deal with node clustering. There are several definitions of a graph with clustered nodes. In a *compound graph* clusters can be nested, and edges are allowed between both nodes and clusters. A *clustered graph* is a compound graph in which edges are allowed only between nodes. Finally, a clustered graph whose clusters are not nested (i.e. there is only one level of clustering) is *flat*. Graphs produced by traceroutes fall in this last category.

The two algorithmic styles that were experimented most in this domain are *force directed* and *layered* algorithms [42]. The latter are also called algorithms for *hierarchical* layouts. Force directed algorithms have good performance on large graphs and tend to produce drawings in which dense subgraphs are well separated. Algorithms for layered drawings have lower scalability than force directed ones, but on sparse graphs may produce drawings with fewer edge crossings. Also, hierarchical relationships between nodes are explicitly represented in the layering. We focus on the case of dynamic graphs, which require stability across changes for nodes, edges, and clusters [16]. For flat clustered graphs, [43] introduces a force directed algorithm that exploits virtual nodes and cohesive forces so to keep clusters compact and well separated. [44] extends the approach of [19] to compound graphs. In particular, the stability of the cluster structure over time is obtained by merging the cluster hierarchies corresponding to all considered time instants. This aggregated information is used to produce a super-layout of the graph through a force directed algorithm, which works as a template for drawing the single time instants. In [45] a similar approach is used, but an algorithm for layered layouts is used instead of a force directed one. Also, during the animation from a frame to the next one, clusters that remain unchanged are collapsed so to focus on the parts of the graph subject to some dynamics.

In relation to the problem of drawing graphs produced from traceroutes, some considerations are necessary when choosing an algorithm. While force directed ones pose few or no constraints on the input graph, they are iterative algorithms that converge at a local minimum, giving few guarantees on the quality of the produced drawing. Also, the direction of edges is usually not considered in the process. On the other hand, layered drawings exploit edge directions and use them to make node hierarchies explicit, which can support user tasks. Also, layering gives an explicit visualization of topological distances in a network, making possible, to some extent, to compare the length of different paths. But these algorithms are also known to have lower scalability than force directed ones, and require the input graph to be acyclic. When this is not the case, the graph must be transformed to become acyclic, which may negatively influence the representation of hierarchies. We chose the layered style for our problem, because the representation of hierarchies and path directions is relevant in the use cases. Further motivations are given in Section 5, following the results of experiments that we performed on our data of interest.

## 4 TERMINOLOGY

This section introduces notation that will be used in the rest of the paper.

Consider a time interval $\mathcal{T}$ and a set of probes $\mathcal{S}$. During $\mathcal{T}$ each probe periodically issues a traceroute towards a target IP address $\tau$. A traceroute from probe $\sigma \in \mathcal{S}$ outputs a directed path on the Internet from $\sigma$ to $\tau$, called *traceroute path* or simply *traceroute*. If a traceroute is available in Internet at time $t \in \mathcal{T}$, then it is *valid* at time $t$. By convention, we consider traceroutes as paths that are directed from the target to the source. Each vertex of a traceroute originated

from $\sigma \in \mathcal{S}$ is either a router or a computer. Vertices are identified as follows: (1) $\sigma$ has an identifier assigned by the RIPE NCC; (2) vertices with a *public* IP address [46] are identified by such an address; (3) vertices with a *private* IP address [46] are identified by a pair composed of their address and the identifier of $\sigma$; (4) the remaining vertices are labeled with a "*" (i.e. an unknown IP address). For the sake of simplicity, consecutive vertices labeled with "*" are merged into one vertex.

A digraph $G_t$ is defined at each instant $t \in \mathcal{T}$ as the union of all the traceroute paths valid at $t$ produced by the traceroutes issued by the probes of $\mathcal{S}$. A digraph $G_{\mathcal{T}}$ is defined as the union of all graphs $G_t$ with $t \in \mathcal{T}$.

Each vertex of $G_{\mathcal{T}}$ is assigned to one *cluster* as follows. (1) Each probe is assigned to the cluster that corresponds to the AS where it is hosted. (2) Each vertex identified by a public IP address is assigned to a cluster that corresponds to the AS announcing that address on the Internet. This information is extracted from the RIPEstat [47] database and for some public IP addresses may occasionally be missing. (3) For each vertex $v$ that is not assigned to a cluster after the previous steps, let $\mu$ be the cluster assigned to the nearest predecessor of $v$ with an assigned cluster and let $\nu$ be the cluster assigned to the nearest successor of $v$ with an assigned cluster. If $\mu = \nu$ then $v$ is assigned to $\mu$. (4) Each remaining vertex is assigned to a corresponding *fictitious* cluster. We define $V_\mu$ as the set of vertices assigned to cluster $\mu$. The set of all clusters assigned in $G_{\mathcal{T}}$ is denoted by $\mathcal{C}(G_{\mathcal{T}})$, or simply $\mathcal{C}$ when there are no ambiguities.

For any $t \in \mathcal{T}$, $G_t$ can be visualized at different abstraction levels. Namely, the user can select a set $\mathcal{E}$ of clusters that are fully visualized and each cluster that is in the complement $\bar{\mathcal{E}}$ of $\mathcal{E}$ is contracted into one vertex. More formally, given the pair $G_t, \mathcal{E}$ the visualized graph $G_{t,\mathcal{E}}(V, E)$ is defined as follows. $V$ is the union of the $V_\mu$ for all clusters $\mu \in \mathcal{E}$, plus one vertex for each cluster in $\bar{\mathcal{E}}$. $E$ contains the following edges. Consider edge $(u, v)$ of $G_t$ and clusters $\mu$ and $\nu$, with $u \in \mu$ and $v \in \nu$. If $\mu \neq \nu$, $\mu \in \mathcal{E}$, and $\nu \in \mathcal{E}$, then add edge $(u, v)$. If both $\mu$ and $\nu$ are in $\bar{\mathcal{E}}$ then add edge $(\mu, \nu)$. If $\mu \in \mathcal{E}$ ($\mu \in \bar{\mathcal{E}}$) and $\nu \in \bar{\mathcal{E}}$ ($\nu \in \mathcal{E}$) then add edge $(u, \nu)$ ($(\mu, v)$). We define $G_{\mu,t}$ as the subgraph of $G_t$ induced by $V_\mu$. Analogously, we define $G_{\mu,\mathcal{T}}$ as the subgraph of $G_{\mathcal{T}}$ induced by $V_\mu$ and define $G_{\mathcal{T},\mathcal{E}}$ as the union of the $G_{t,\mathcal{E}}$ for each $t \in \mathcal{T}$. Note that $G_{\mathcal{T},\emptyset}$ denotes the graph in which all clusters are contracted, while $G_{\mathcal{T},\mathcal{C}}$ denotes the graph in which all clusters are expanded and it is equivalent to $G_{\mathcal{T}}$. We assume that $G_{\mathcal{T},\mathcal{E}}$ is an acyclic graph. However, cycles can exist in single traceroute paths or can be created by merging several paths. The origin and the incidence of those cycles is discussed in Section 5, while Section 7.2 describes an algorithm to deal with them.

## 5 Analysis of Data

This section describes preliminary experiments that we conducted to characterize the data of interest for Radian. The results of these experiments deeply influenced the design of our tool. We collected traceroutes executed in nine months (from March 20th, 2012 to December 20th, 2012) by about 200 world-wide distributed RIPE Atlas probes towards 5 public targets. The probes performed measurements for the whole period, with variable frequencies for the execution of traceroutes ranging between one every minute and one every 30 minutes. Then, we processed the data and generated $12,500$ random visualization scenarios of the kind to be displayed by Radian. Each visualization scenario is identified by a tuple $\langle \tau_j, \mathcal{S}_i, t_i, d_k \rangle$, where $\tau_j$ is a target, $\mathcal{S}_i$ is a set of 50 probes, $t_i$ is a time instant, and $d_k$ is a time duration expressed in number of hours. Such a tuple identifies a set of traceroute paths collected by probes $\mathcal{S}_i$ towards target $\tau_j$, starting from instant $t_i$ and for a period of $d_k$ hours. We constructed the visualization scenarios as follows. We selected uniformly at random: 100 sets $\mathcal{S}_i$ $i = 1 \dots 100$ each consising of 50 probes, and 100 time instants $t_i$ $i = 1 \dots 100$ in the above nine months interval. By repeating this for the five targets $\tau_j$ and for durations $d_k = k - 1, k = 1 \dots 25$, we obtained the $12,500$ visualization scenarios. A visualization scenario of this type is an overestimation of real use cases, in terms of number of probes and length of the analyzed time interval. Indeed, often an ISP owns sufficient information collected from other sources that allow to restrict the set of probes and the length of the analyzed time interval. For each scenario we also computed graphs $G_{\mathcal{T},\mathcal{C}}$ and $G_{\mathcal{T},\emptyset}$ (see Section 4). In this section we will refer to $G_{\mathcal{T},\mathcal{C}}$ simply as the *graph*, and to $G_{\mathcal{T},\emptyset}$ as the *contracted graph*. Note that graphs with a duration equal to 0 have special semantics, since they approximate the state of the routing at a fixed time instant. On the other hand, graphs with a duration greater than 0 are composed of several states.

In Fig. 2a we plot a cumulative distribution function of the length of the traceroute paths of the dataset. It gives an indication on the maximum distance between a probe in $\mathcal{S}$ and $\tau$. The plot shows that traceroutes with more than 15 vertices are rare, which implies that a traceroute path can reasonably be visualized on the screen in its full extent. Figs. 2b-f are related to the graphs computed for our visualization scenarios. On the $x$ axis there is the time duration $d_k$ ranging from 0 to 24 hours, while the $y$ axis shows the value of a metric averaged over all graphs. Fig. 2b shows that our graphs are quite sparse, even for large durations. In particular, graphs with duration equal to 0 are almost trees (density $\sim$ 1), which was expected, since routing protocols forward data through a network by computing a spanning tree. Figs. 2c and 2d show the number of vertices and the number of ASes in a graph (both order of hundreds), respectively. These amounts depend on the number of probes in our tests and on the average length of a traceroute path. Visualizing a network with hundreds of vertices and ASes on a screen is challenging. Even if the amount of screen space is enough to avoid clutter in the visualization, understanding such a network is a demanding cognitive task for a user. Therefore, an effective visualization requires a way to reduce the amount of details shown on the screen. From Figs. 2c and 2d we also draw conclusions on the impact that dynamics has on the size of a graph. We can see that the size of a graph increases with time, but such increment is not dramatic. Namely, passing from a duration of 0 hours to 24 hours the average number of vertices increases from about 400 to about 500.

Since we deal with routing data we expected to find only acyclic graphs, since a cycle in an IP network originates an incorrect routing. One surprising result of our analysis

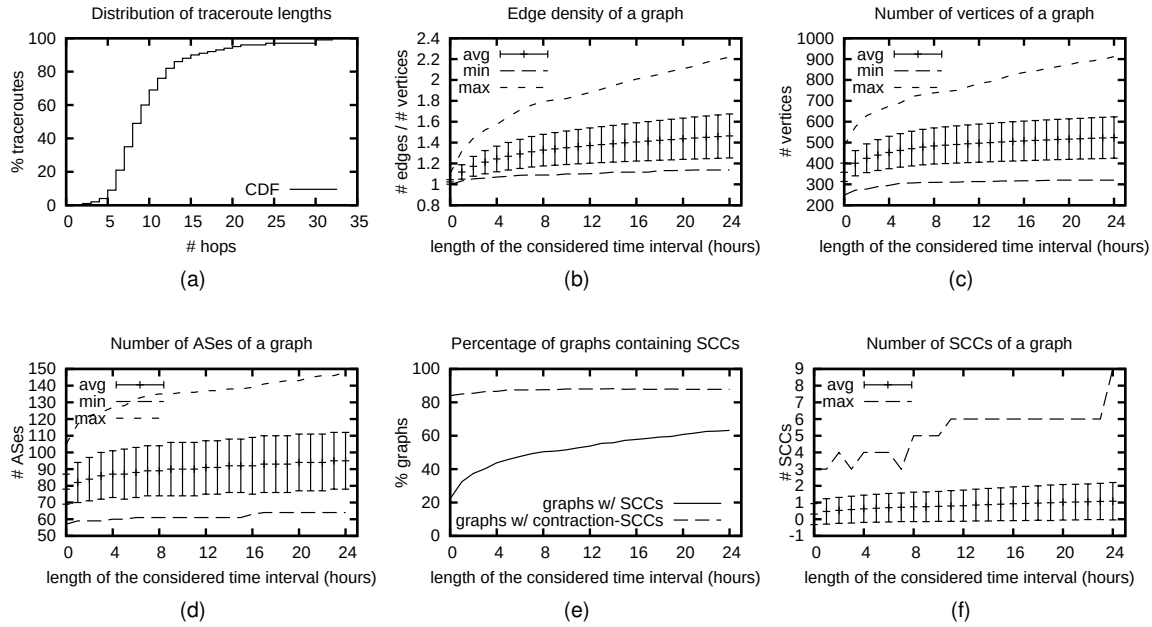


Fig. 2. Statistics on traceroute data. Error bars report the standard deviation. (a) Cumulative distribution function (CDF) of the length of traceroute paths in our dataset. (b) Edge density of a graph in our dataset. (c) Number of vertices of a graph in our dataset. (d) Number of ASes of a graph in our dataset. (e) Percentage of graphs that contain SCCs, and that contain contraction-SCCs when they are contracted. (f) Number of SCCs in a graph. The plot for minimum values is not shown, it is always equal to $0$.

is that, in graphs produced from traceroutes, cycles exist with non negligible probability. Fig. 2e shows that, in our dataset, $20\%$ of graphs with $d_k = 0$ (i.e. almost static routing graphs) contain cycles. The percentage grows to $60\%$ for graphs with $d_k = 24$. Note that Figs. 2e and 2f count the *strongly connected components* (SCC) of a graph, which imply the existence of cycles and can be efficiently detected. A SCC is a subset of the vertices such that there exists a directed path between any pair of them. Even if a graph has a high probability to contain a cycle, Fig.2f shows that such cycles are few. While cycles in a graph with some dynamics ($k > 0$) can be explained from the fact that they result from the merge of several routing states, cycles in a static graph ($k = 0$) are more obscure. By analysing our dataset we discovered that such cycles depend on two factors. First, a static graph is only an approximation of the routing at a given time instant, since for each probe we select the latest measurement available and therefore the graph can actually span several time instants. Second, measurement errors can happen. The execution of the traceroute command is a process that takes a time in the order of seconds, and the routing can change during the execution. A reported traceroute path can therefore be the fusion of two unrelated paths, which do not exist at the same time and induce the presence of a cycle. One additional property of SCCs in our dataset is that even if they are small on average, outliers can be large, up to $180$ vertices. Such big numbers are due to the fact that SCCs are formed by interconnecting traceroute paths, whose length can reach $30$ vertices. Interconnecting a few of them is sufficient to create large SCCs. It is easy to see that a SCC that spans more than one AS induces a SCC also in the contracted graph. What is less immediate is that the other way around does not necessarily hold. In fact, it is possible to produce cycles by contracting clusters of an acyclic graph. We call *contraction SCCs* the SCCs of the contracted graph that are produced by contracting clusters. Note that these SCCs do not have corresponding SCCs in the original graph. Fig. 2e shows that contraction SCCs exist in the 90% of graphs in our dataset. Differently from the case of the graph, we do not consider the presence of contraction SCCs only a measurement error, because the policy-based nature of BGP actually allows cycles.

We draw several conclusions from the analysis of data. First, the average length of a traceroute allows visualizing it on the screen in its full extent. The size of a routing graph depends mainly on the number of selected probes, and in a realistic setting it can be large enough to make cognitive tasks hard. For this reason, an effective visualization must allow to reduce the amount of details shown on the screen. On the other hand, the maximum level of detail is still required by the use cases that we considered in Section 2.2 to analyze specific parts of the network. Therefore, the user must be able to choose the amount of displayed details on specific parts of the visualization. Second, the sparsity of the graphs suggests using graph drawing algorithms for layered layouts. Hence, we performed preliminary experiments with algorithms for layered layouts, discovering that crossing-reduction heuristics like those in [48], [49] applied to our graphs are quite effective. However, in our case the graph density is so low that very often graphs are *planar*, i.e. can be drawn without crossings, or quasi-planar. Unfortunately, the heuristics in [48], [49] do not guarantee the absence of edge crossings for planar graphs, and it is quite disappointing for the user to see crossings that can be easily avoided. So, we decided to devise new algorithmic techniques, based on planarity, suitable in this

scenario. Also, in Section 7.2 we describe special algorithms for dealing with cycles in our data.

## 6 User Interface

This section presents the user interface of Radian and the user tasks, along with the motivations behind the design.

### 6.1 Overview of the Interface

The user interface is shown in Fig. 1. It is composed of four main panels: the *graph*, the *timeline*, the *controller*, and the *info*. The user can select the data to visualize, identified by a target $\tau$, a set of probes $\mathcal{S}$, a time interval $\mathcal{T}$, and a set of expanded clusters $\mathcal{E}$. The interface presents at any moment the state of the data at a time instant $t \in \mathcal{T}$.

The *graph panel* displays the graph $G_{t,\mathcal{E}}$ with a radial drawing centered in $\tau$. All vertices and clusters that appear in at least one traceroute in $\mathcal{T}$ are in the drawing, including those that at time $t$ are not traversed by any traceroute. Probes in $\mathcal{S}$ are represented as blue circles labeled with their identifier. Vertices are represented as white rounded rectangles labeled with the last byte of their IP address, or with a "*". Also, they are placed on concentric circles, with probes located at the periphery of the drawing. Clusters are represented as annular sectors labeled with their AS number. Each cluster in $\mathcal{E}$ encloses the nodes that are assigned to it, while clusters not in $\mathcal{E}$ are empty and have a fixed size. Clusters containing probes in $\mathcal{S}$ are light blue, the cluster containing $\tau$ is light red, and the remaining clusters are light yellow. Fictitious clusters are not displayed. Each traceroute path from a probe $\sigma \in \mathcal{S}$ to $\tau$ is represented as a colored curve from $\sigma$ to $\tau$ passing through intermediate vertices. For each probe the latest traceroute available before $t$ is shown, and it corresponds to a path in the graph $G_{\mathcal{T}}$. Traceroute paths are not simply merged and displayed in an aggregate fashion, since each of them has its own informative value and can change over time independently. For this reason, we explicitly represent all paths adopting a metro-line metaphor [50], and draw them using different colors. Paths that change in $\mathcal{T}$ are represented with solid lines. For paths that do not change and thus represent static routes, we borrow a technique from [51]. These paths are partitioned into sets such that each set determines a tree on the graph, and each tree is depicted with dashed lines and a distinctive color. This has the effect of reducing the number of lines in the drawing, while preserving the routing information of each probe. The user can interact with the graph in several ways. First, he can change the current time instant $t$ that is visualized. Path differences between the two instants are shown with an animation, which continuously morphs each path from its initial position to the final one. Paths that are morphing become thicker during the animation, so to be more visible. A new route that was previously unavailable is shown in the animation with a gradually appearing path, while a disconnection is shown with a gradually fading path. A path is also outlined when the pointer hovers on it, and, similarly, hovering on a vertex outlines all paths passing through it. Finally, the user can arbitrarily change the set $\mathcal{E}$ of expanded clusters. Double-clicking an expanded cluster collapses it and removes if from $\mathcal{E}$, and vice versa. Vertices in a collapsed cluster are not shown.

The *timeline panel* is in the lower part of the window and provides an overview of the trend in the number of route changes over time. It features a red cursor that points at the current time instant $t$. The timeline panel and the graph panel are linked views, and the visualized graph corresponds to the time instant selected in the timeline. The user can move the cursor to change the current time instant. Additional timelines can be added below the main one, each regarding the trend in the value of a *metric* with respect to a probe. The user can add and remove the metric timeline of a probe by double-clicking that probe in the graph panel. All timelines have the same width and represent the same time interval $\mathcal{T}$, allowing direct comparison.

The *controller panel* is located in the upper right corner and is used to control the animation. It is modeled following the metaphor of a video recorder, and presents buttons that activate the typical functions play, pause, step-backward, and step-forward. When the play button is pressed, the user is presented with a sequence of animation steps, each regarding a single path change and with a short pause between two changes. At each step the cursor in the timeline panel and the visualization in the graph panel are updated accordingly. The duration of an animation between two time instants $t_1$ and $t_2$ is proportional to the logarithm of the elapsed time between the two instants, which gives an approximate perception of elapsed time while limiting the overhead on the total animation time.

Finally, the *info panel* is in the upper part of the window. When the cursor hovers on an AS, a node, or a path, the info panel shows any available information about that element.

Our choice to make a topological visualization, discarding geographical visualizations of traceroute data, comes from the conclusions drawn in Section 3.1. Our use cases are better supported by a graph visualization, which gets easily cluttered when geographical positions for nodes are applied. Also, understanding the structure of the network at the AS level is more interesting for understanding routing policies, and this would be disrupted by a geographical visualization since ASes are usually distributed. Finally, *anycast* addresses are assigned to more than one physical device and could not be mapped to a single location. The node-link metaphor was chosen for representing the graph because, as explained in Section 3.2, it is intuitive to networking users, since it looks like the real network on which traceroutes were performed. The graph is visualized with a radial layered layout because it is sparse (see Section 5), and this style of drawing is notably effective for visualizing sparse hierarchical graphs (see, e.g., [52]). Also, in our visualization the focus is put on the target, avoiding to give too much importance to specific probes due to a privileged geometric position. Finally, having all traceroutes flowing in a same direction helps comparing their lengths. The choice of animations, instead, was less immediate. Although users generally like them, Section 3.2 outlined how this technique is limited in that users need to rely on their visual memory in order to track graph updates. On the other hand, the main alternative consisting of small multiples suffers from a too low scalability to support our use cases. Indeed, in a time span of a few hours, hundreds of route changes can happen depending on the size of the network, which would produce too many small multiples to fit in the screen.

For this reason we adopted animations, which support very long time spans. We applied several techniques to mitigate the limits of animations. First, we enforce the preservation of the mental map (see Section 3.2 for a discussion). Vertices maintain their positions at different time instants. Updating $\mathcal{E}$ may change the position of some vertices, but all vertices preserve their relative orderings along and across layers, which is similar to the preservation of horizontal and vertical orderings of [31]. Section 7.1 describes in detail the algorithms for obtaining such property. The mental map is further preserved by visualizing nodes that are not traversed by a traceroute at the current time instant (but that are traversed at some different instants), which also furnishes to the user hints on the fact that a given part of the network is subject to dynamics at some instant. Our choice limits graph updates to paths only, which reduces the number of elements that the user needs to track. This is also a natural representation in this domain, since users tend to think of routing changes as changes of traceroute paths. Finally, outlining with thicker lines traceroute paths that change during an animation is a form of transition encoding. The visualization of ASes as boxes that enclose vertices satisfies the user expectation that ASes represent a partition of the network. Also, this organization offers an AS-level view of the network, which is an abstraction useful in all use cases described in Section 2.2. Finally, we exploited this representation to let the user collapse an AS and hide its content. This allows to focus the attention on the detailed dynamic of only few ASes of interest, while maintaining a high level overview of the rest. The results is a reduction of the cognitive load for understanding graphs that, how showed in Section 5, can be relatively large. The possibility to arbitrarily collapsing and expanding ASes poses specific algorithmic challenges for the preservation of the user mental map. Namely, consider two different sequences of expansions and contractions and let $\mathcal{E}'$ and $\mathcal{E}''$ be the resulting sets of expanded clusters. If $\mathcal{E}' = \mathcal{E}''$ the user expects to see the same drawing, independently on the specific sequence of performed actions. This is an algorithmic requirement that is not satisfied by all graph drawing algorithms (e.g. by spring embedders) and is faced and solved in Section 7.1.

Fig. 3 contains various details on how the interaction with the visualization works. A graph with static paths and no expanded clusters is presented in Fig. 3(a). It is related to a target $\tau$, a set of probes $\mathcal{S}$, and a small time interval $\mathcal{T}'$. Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. From this figure we can see what ASes provide connectivity to reach the target, namely 1200 and 20965. A graph for $\tau$, $\mathcal{S}$ and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$) is presented in Fig. 3(b). Some dynamic paths are visible. The same graph is presented in Fig. 3(c) with one expanded cluster. Note how the ordering of clusters and vertices on the radial layers is preserved. In this figure the length and structure of the paths from each of the three probes 619, 602, and 265 is clearly visible. Fig. 3(d) shows the same expanded graph at a different time instant. The intermediate vertices of two paths are different, that is, we can see how the route changes affected the paths of probes 619 and 602.

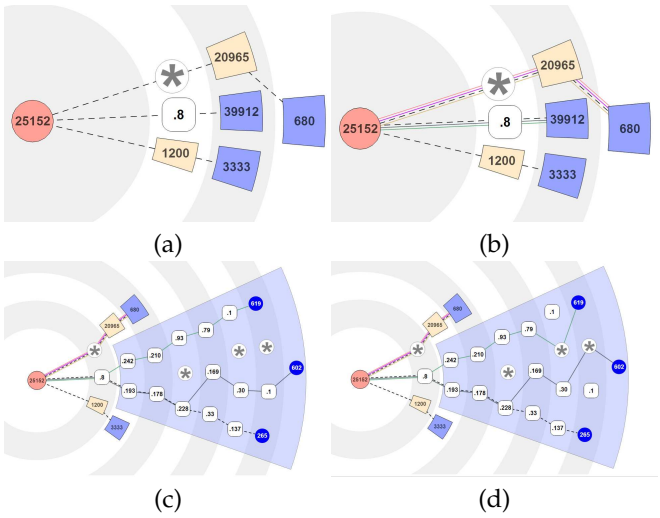


Fig. 3. Details of the interactive features of our visualization. (a) A graph $G_{\mathcal{T}'}$ relative to a target $\tau$, a set of probes $\mathcal{S}$, and a time interval $\mathcal{T}'$. All paths in $G_{\mathcal{T}'}$ are static and all clusters contracted. (b) A graph $G_{\mathcal{T}''}$ relative to $\tau$, $\mathcal{S}$, and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$). Some paths are dynamic and all clusters are contracted. (c) $G_{\mathcal{T}''}$ with an expanded cluster. (d) $G_{\mathcal{T}''}$ at a different time instant.

### 6.2 User Tasks

This section describes the tasks that a user can execute with the interface of Radian. The use cases described in Section 2.2 are decomposed in user tasks, that are classified in two groups: *topology tasks*, devoted to understanding the structure of the routing and of the network at a given time instant, and *dynamics tasks* devoted to understanding how the routing evolved over time in a given interval. Topology tasks are executed on the routing as captured at a single time instant, while dynamics tasks require the user to compare different time instants and make use of animations. For brevity, we refer to use cases Troubleshooting, Upgrade Verification, and Inter-domain Consistency Check as UCT, UCV, UCC, respectively.

The list of the topology tasks follows. Consider a set of probes $\mathcal{S}$ and a target $\tau$.

**Find what nodes are traversed by $\sigma \in \mathcal{S}$.** This task is fundamental for all use cases, since it implies discovering the route followed by a $\sigma$ at a given instant to reach $\tau$, which is the final product of the routing. The task is accomplished by following the colored curve across the graph, from the $\sigma$ to $\tau$, and see what nodes are intersected.

**Find the topological distance to $\tau$.** This tasks implies to discover the number of nodes to traverse to reach $\tau$. Often, shorter paths provide better performance. This is useful in UCT and UCV, either to find out the reason for experienced bad performance, or to spot in advance long paths that could be future causes of issues.

**Test if $\tau$ is reachable.** This task supports UCT and UCV. $\tau$ is reachable from $\sigma \in \mathcal{S}$ if the curve from $\sigma$ intersects it. An unreachability can have different meanings. Depending on how far from $\tau$ the path is interrupted, it could either mean that the routing was wrong or that $\tau$ was not working properly. Note that an unreachability does not necessarily imply a fault, since routers can be configured to not respond to traceroutes, for security reasons. However, an ISP usually knows, in its own AS, which routers respond.

**Find what probes in $\mathcal{S}$ traverse a node.** This task is useful to understand what parts of the network depend on a node for reaching the target. In UCV this implies to check if the node designated for connecting a subset of the network is doing it correctly. In UCT, a node shared by several probes that experienced reduced efficiency is possibly the cause of the problem, and a start point for a deeper analysis. The task is executed by following all curves that traverse the node, from that node back to the probes.

**Find single points of failure.** This task is somewhat in the middle between UCT and UCV. The objective is to find in advance nodes whose fault or overload would impact a large part of the network. The task is executed by finding nodes with many traversing paths.

**Find the ASes that make $\tau$ reachable.** The task consists of finding what ASes are traversed to reach $\tau$ from a probe $\sigma \in \mathcal{S}$. The user contracts all clusters to simplify the visualization, and looks for a path of traversed clusters. Such a path can be compared with the BGP announcements to check if the actual routing is consistent with BGP. Understanding what ASes are traversed to reach $\tau$ means reasoning about the routing and the network at a high abstraction level, and it is the fundamental task of UCC.

**Test if $\sigma_1, \sigma_2 \in \mathcal{S}$ are treated equally in the network.** In UCT, an ISP may receive mixed feedbacks from its customers regarding the use of a remote service. While those near $\sigma_1$ experienced low performance or even an outage, those near $\sigma_2$ did not. By comparing the paths towards $\tau$, the ISP may discover that intermediate nodes forwarded data through different paths depending on the source. Also in UCC, if an external AS is suspected to offer different levels of service, the ISP expands the relative cluster in the visualization and keeps the other collapsed, to focus the study on the internal routing of that AS.

The list of the dynamics tasks follows. Consider a set of probes $\mathcal{S}$ and a target $\tau$ within a time interval $\mathcal{T}$.

**Find probes of $\mathcal{S}$ subject to path changes.** As described in Section 6.1, routes that are not subject to dynamics in the selected time interval are depicted with dashed lines. That is, from a single frame, the interface of Radian tells whether a probe changed its path at some time instant. This is a general feature supporting all use cases in which it is necessary to understand the routing evolution.

**Find intense routing activity.** In UTC, the user needs to restrict $\mathcal{T}$ and locate routing changes related to some event of interest, with only a rough temporal indication. For example, a customer ticket could complain about degraded performance or lack of service starting from a given hour of the day. In UCV, the same analysis is done to check if the routing changed in correspondence of a configuration change of the devices. The timeline panel gives an overview of the distribution of path changes during $\mathcal{T}$. The user, for example, may find out that nothing happened until a given instant, and therefore focus on what happened after.

**Find failing nodes.** The paths of several probes in $\mathcal{S}$ could abandon a specific node at the same time. This is clearly outlined in the graph panel, which shows with an animation that the corresponding paths stop traversing that node and morph to another path. In UCT, this could be the sign of a hardware failure on that node. In UCV, the node could have been turned off for maintenance, and the probes redistributed to different paths.

**Find repetitive phenomena.** Repetitive routing dynamics are shown in the timeline panel as spikes of activity happened with a certain regularity. In UCT, the user could start the analysis from a given time instant were suspect dynamics happened, and then discover that more activity is present in the hours before or after that instant with regularity. For example, the network of an ISP could be not able to respond with acceptable performance to a traffic load in the early night hours, caused by many customers connecting after work from their homes. This would be the cause of many path changes happening around that time for several days, as the result of intense load balancing.

**Test if a BGP backup link is operational.** An ISP can be connected to the Internet through several BGP links for redundancy, either with a same partner ISP or with several ISPs. In case of a fault on a link, a backup link becomes operational due to BGP policies. In UCC, the user can follow the evolution of paths traversing a failing BGP link and check that, in correspondence of the failure, all of them start traversing the backup link.

**Test if a path change improved the performance.** After the user has spotted a path change of interest, this could be not enough to decide if it was positive or not. In all use cases, the user is interested to check the value of metrics in correspondence of the change, to see if there was an improvement. This is done by selecting one or more probes in the graph panel, which shows for each probe an additional line chart with the metric trend over time. Line charts are displayed one on top of another, which allows to compare them instant by instant.

## 7 ALGORITHMS

This section describes the algorithm used by Radian for drawing acyclic traceroute graphs, and the algorithm for removing cycles from traceroute graphs.

### 7.1 Layout Algorithm

Following the conclusions of the analysis in Section 5, the layout algorithm of Radian produces layered layouts and is planarity oriented. It works by creating a static layout of a super-graph obtained by merging all time instant, which is then used to visualize single instants. This is supported by the fact that our visualization problem is offline (Section 3.2), and that node positions are fixed to preserve the user mental map (Section 6). For our purposes an interesting reference is [53], which constructs radial drawings adapting techniques of the Sugiyama Framework. Unfortunately, it does not support clusters. The algorithm in [54], which extends the one described in [55], inspired part of our work. However, it proposes a clustered planarity testing algorithm, while we rather need an algorithm for clustered graph planarization, and [54] is not easily extensible for this purpose (neither is the algorithm in [56] that is not suitable for hierarchical drawings). In [57] an algorithm is proposed for the expansion/contraction of clusters of hierarchical drawings, building on [58]. Unfortunately it uses a local layering scheme, while global layering [48], [49] produces

more compact drawings and hence is more suitable for the average length of traceroute paths (Section 5). For these reasons, we devised a new algorithm to produce clustered hierarchical drawings, as a planarization-oriented variation of [54], and which uses a global layering scheme.

At a high level, our algorithm works as follows. We pre-compute a hierarchical drawing $\Gamma_0$ of $G_{\mathcal{T}}$. In that drawing all clusters are expanded. The layout is computed in such a way to have few crossings that involve connections between clusters. The quality of the layout inside the clusters is considered with lower priority. Moreover, the quality of the drawing of edges that are part of many traceroutes in $\mathcal{T}$ is privileged among the edges of $G_{\mathcal{T}}$. The drawing computed for each cluster is stored and reused in any drawing where that cluster is expanded. The obtained drawing is mapped to a radial drawing with a coordinate transformation.

What follows gives more details on our algorithmic framework. In a preprocessing step several information are computed on $G_{\mathcal{T}}$ that will be used for actual drawings. Given any $G_{\mu,\mathcal{T}}$, a vertex is a *source* (*sink*) of $G_{\mu,\mathcal{T}}$ if it is the first (last) vertex of $G_{\mu,\mathcal{T}}$ encountered in some traceroute path. Each graph $G_{\mu,\mathcal{T}}$ is augmented with extra vertices and edges so that all the longest paths from a source to a sink have the same length. The added vertices are called *fictitious vertices* of $\mu$ and ensure that, given an edge $(u, v) \in G_{\mathcal{T}}$, $u \in \mu$, $v \in \nu$, $\mu \neq \nu$, clusters $\mu$ and $\nu$ do not share a layer in any drawing of $G_{t,\mathcal{E}}$ for any choice of $\mathcal{E}$. Moreover, they force edges that leave a cluster by spanning several layers to be routed inside that cluster. A *$\mu$-drawing* is pre-computed for each $G_{\mu,\mathcal{T}}$. It consists of 1) assigning vertices to layers so that all edges are between consecutive layers and 2) computing a total order for the vertices of each layer. A partial order $\prec$ is computed for clusters, such that for any two clusters $\mu$ and $\nu$ with $\mu \prec \nu$, the vertices of $\mu$ appear to the left of the vertices of $\nu$ for any drawing $\Gamma$ where $\mu$ and $\nu$ share one or more layers. This helps preserve the mental map during expansions/contractions. The preprocessing step obtains a $\mu$-drawing for each cluster and a partial order $\prec$ of clusters from a drawing $\Gamma_0$ of $G_{\mathcal{T}}$ with all clusters expanded. The algorithm to compute $\Gamma_0$ is similar to that in [54], where a PQ-tree [59] is used to order vertices along the layers of the drawing. Our PQ-tree is initialized with a spanning tree of $G_{\mathcal{T}}$ and incrementally updated with the remaining edges that induce ordering constraints. An edge is added only if it does not produce a crossing (i.e. the PQ-tree does not return the null tree). A rejected edge will produce crossings in $\Gamma_0$. Edges are added with priority given by their aesthetic importance: namely, they are weighted by the number of traceroutes that traverse them in $\mathcal{T}$. As an implementation detail, we actually compute a total order for clusters to represent a partial order $\prec$. Such order is produced by a DFS visit of the embedded spanning tree of $G_{\mathcal{T}}$. The tree has an embedding induced by the layer orders produced by the PQ-tree algorithm, and the children of a vertex are visited in clockwise order. Intuitively, we preserve the geometric left-to-right order for clusters from $\Gamma_0$, and reuse it to produce a drawing of any $G_{t,\mathcal{E}}$.

The result of the preprocessing is used to compute a drawing $\Gamma_{\mathcal{T},\mathcal{E}}$ of $G_{\mathcal{T},\mathcal{E}}$, as detailed in the following. Note that, once $\Gamma_{\mathcal{T},\mathcal{E}}$ is computed, we display, for any $t \in \mathcal{T}$ all the vertices of $G_{\mathcal{T},\mathcal{E}}$ but only the edges of $G_{t,\mathcal{E}}$. First, a layering of $G_{\mathcal{T},\mathcal{E}}$ is computed such that for each vertex the distance from $\tau$ is minimized. Also, dummy vertices, called *fictitious vertices* of $G_{\mathcal{T},\mathcal{E}}$, are added so that each edge spans two consecutive layers. Vertices are horizontally ordered on each layer such that: 1) $\prec$ is enforced; 2) for each cluster $\mu$ of $\mathcal{E}$, the orders on the layers of its $\mu$-drawing are enforced; and 3) the fictitious vertices of $G_{\mathcal{T},\mathcal{E}}$ are placed in such a way to have few crossings. In particular, they must not be interleaved with the vertices of any cluster, that is, the vertices of each cluster must be consecutive on every layer. For this reason, each fictitious vertex is assigned to a new fictitious cluster, which is inserted in the partial order $\prec$ in an intermediate position between the endpoints of the edge it belongs to. The described technique can create edge crossings even if the instance admits a planar drawing. The reason is that, for a vertex resulting from merging the sources of a cluster, the PQ-tree does not preserve the consistency between the orders of its incoming and outgoing edges. When the vertex is split again in the embedded graph to reconstruct the original sources, crossings can result in the layer that precedes or follows the one of the sources. We mitigate this effect by post-processing the embedding produced by the PQ-tree. Namely, the position in each layer of the vertices that represent probes is preserved, while the position of any other vertex is computed by a single bottom-up sweep of the layers that uses a barycentric positioning. The technique is inspired by those commonly used in algorithms for Sugiyama-like layouts [42].

Finally, the ordered layers are used to assign geometric coordinates to vertices. The vertical coordinate of a vertex is proportional to the layer number it belongs to. Horizontal coordinates are computed by solving a linear program that minimizes the sum of the horizontal distances between a parent node and its children, while preserving the vertex orderings of layers and a minimum horizontal parent-to-child distance. To obtain a radial drawing, the coordinates of vertices are transformed in such a way that each layer is mapped to a circumference, and these circumferences are nested. An edge $(u, v)$ is drawn either as a straight segment or a curved arc, depending on the angle it must sweep to connect vertices $u$ and $v$. Note that each edge connects only vertices in two consecutive layers, hence a curved edge can be drawn only in the space between these layers.

### 7.2 Handling Cyclic Graphs

As pointed out in Section 5, the contraction of clusters in a graph can create cycles. Namely, the graph $G_{\mathcal{T},\mathcal{E}_2}$ that is produced by contracting a cluster in graph $G_{\mathcal{T},\mathcal{E}_1}$ can contain a cycle that does not exist in $G_{\mathcal{T},\mathcal{E}_1}$. For this reason, given a graph $G_{\mathcal{T}}$ with clusters $\mathcal{C}$, an algorithm for removing cycles must ensure that $G_{\mathcal{T},\mathcal{E}}$ is acyclic for any set of expanded clusters $\mathcal{E} \subseteq \mathcal{C}$. A naive approach like checking all possible sets of expansions is unsuitable, since it requires to check a number of graphs that is exponential in the number of clusters. However, Theorem 1 states that it is sufficient to check a limited number of graphs.

***Theorem 1.*** There exists a cycle in $G_{\mathcal{T},X}$ for some $X \subseteq \mathcal{C}$ if and only if there exists a cycle either in $G_{\mathcal{T},\emptyset}$ or $G_{\mathcal{T},\mathcal{C}}$.

*Proof:* The proof in one direction is trivial: $\emptyset$ and $\mathcal{C}$ are sets of expanded clusters, therefore if $G_{\mathcal{T},\emptyset}$ or $G_{\mathcal{T},\mathcal{C}}$ contains

a cycle then some $G_{\mathcal{T},X}$ does. For the other direction, assume that $G_{\mathcal{T},X}$, with $X \neq \emptyset$ and $X \neq \mathcal{C}$, contains a cycle $C$. If all vertices of the cycle belong to a same cluster then that cycle also exists in $G_{\mathcal{T},\mathcal{C}}$ and the theorem holds, so assume that $C$ spans more than one cluster. Be $\mu \in X$ an expanded cluster that is traversed by $C$. The cycle enters and exits the boundary of $\mu$ several times, producing pairs of entering and exiting crossings on such boundary. There is a path inside $\mu$ for each such pair of crossings, because the cycle is connected. Contracting $\mu$ collapses such path and identifies the two crossings, preserving the connectedness of the cycle. Namely, the graph obtained by replacing $\mu$ with a vertex $v_\mu$ contains a cycle $C'$ that traverses $v_\mu$ as many times $C$ traversed $\mu$. Contracting every cluster produces a cycle in $G_{\mathcal{T},\emptyset}$ and the theorem holds. □

The result of Theorem 1 is that, to make a graph $G_{\mathcal{T}}$ acyclic, it is sufficient to make acyclic $G_{\mathcal{T},\emptyset}$ and $G_{\mathcal{T},\mathcal{C}}$.

Reversing edges is a common technique for removing cycles from a directed graph. In applications it is usually desirable to reverse a small number of edges to preserve the original graph as much as possible, but finding the minimum number of edges to reverse is a NP-hard problem and heuristics are used [42]. However, existing heuristics

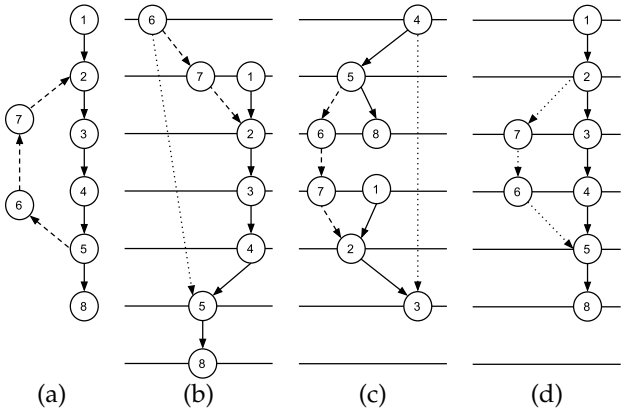


Fig. 4. Layouts induced by different choices of edges reversed for removing cycles. The edges that close the cycles are dashed, the reversed edges are dotted. (a) A cyclic graph. (b) Layered layout induced by reversing edge $(5, 6)$. (c) Layered layout induced by reversing edge $(3, 4)$. (d) Layered layout induced by reversing the path from vertex $5$ to vertex $2$.

give little or no control on which edges are reversed, and this can lead to undesirable effects on the layouts produced by Radian, which are based on layering. For example, Fig. 4(a) shows a graph with a cycle. Vertex $1$ is the source, vertex $8$ is the target, and the edges that close the cycle are dashed. Figure 4(b) shows a layered layouts resulting from reversing edge $(5, 6)$. The drawing is slightly distorted, and vertex $6$ is on a higher level than the source, which does not correctly represent their natural hierarchy. Figure 4(c) shows a different layout, where edge $(3, 4)$ is reversed. The layout is very distorted, and the source is even on a lower layer than the target. Intuition suggests that the correct path in the graph goes from vertex $1$ to vertex $8$, while the entire path that goes from $5$ to $2$ is "going back" and is the cause of the cycle. Figure 4(d) shows a drawing that follows this intuition, by reversing the entire subpath from $5$ to $2$.

In the following we describe an algorithm to remove cycles from our graphs that builds on the fact that they are produced from traceroute paths. The algorithm first computes an acyclic version of $G_{\mathcal{T},\emptyset}$, then the same procedure is applied to $G_{\mathcal{T},\mathcal{C}}$ in such a way that the edge directions of the two graphs are consistent.

Given a graph $G_{\mathcal{T}}$ with clusters $\mathcal{C}$, let $T$ be the set of traceroutes that induce it. Also, assume that, when referred to $G_{\mathcal{T},\emptyset}$, traceroute paths are contracted, that is, each path is a sequence of clusters. Each graph edge is weighted with the number of traceroutes that traverse it, where duplicate traceroutes produced by repeated measurements over time are considered as distinguished. Then, for each SCC, its edges are iteratively removed from the graph in increasing order of weight, until the vertices of the SCC induce an acyclic subgraph. Each time an edge is removed, the traceroutes that traverse it are put in a set $T_1$. At the end of the process the obtained graph is acyclic. Then the traceroutes in set $T_2 = T \setminus T_1$ are merged to obtain an acyclic graph $G'_{\mathcal{T},\emptyset} \subseteq G_{\mathcal{T},\emptyset}$. The traceroutes in $T_1$ are added to $G'_{\mathcal{T},\emptyset}$ as follows. Traceroute paths that do not contain cycles are processed first. A path is split in maximal subpaths such that the two extremal vertices of a subpath either have a layer assigned or represent a probe. Each subpath is added to $G'_{\mathcal{T},\emptyset}$, reversing its direction if it violates the layer ordering of its extremal vertices. The layering of the graph is updated every time a subpath is added. Then cyclic traceroutes in $T_1$ are processed. For each, maximal acyclic subpaths are extracted from it and added to the graph with the aforementioned rules, until all edges of the traceroute have been added.

The procedure described so far computes a graph $G'_{\mathcal{T},\emptyset}$ which is an acyclic version of $G_{\mathcal{T},\emptyset}$. The same procedure is applied also to $G_{\mathcal{T},\mathcal{C}}$, with the following constraint: inter-cluster edges must have the same direction established in $G'_{\mathcal{T},\emptyset}$. Namely, let $e = (v_1, v_2)$ be an edge of $G_{\mathcal{T},\mathcal{C}}$ such that $v_1 \in C_1$ and $v_2 \in C_2$. The direction of $e$ is $(v_1, v_2)$ if $(C_1, C_2) \in G'_{\mathcal{T},\emptyset}$, otherwise it is $(v_2, v_1)$. The constraint keeps $G'_{\mathcal{T},\emptyset}$ and $G'_{\mathcal{T},\mathcal{C}}$ consistent. Finally, $G'_{\mathcal{T},\mathcal{C}}$ is the acyclic graph to be processed by the layout algorithm.

## 8 User Study

We conducted an informal user study at the end of the development of our tool, in order to gather expert opinions on its soundness and effectiveness. We interviewed 6 selected employees of the R&D division of a prominent Italian ISP. Their areas of expertise covered IP edge innovation, cyber security threat evolution, security solutions analysis, and video & multimedia platforms. Such heterogeneous set of expertise covers a wide spectrum of the actual needs and challenges of the ISP. The interviews were held in the context of the Leone FP7 EC research project. We had only one chance to do a user study with them before the end of the research project. The general objective of the study was to receive feedbacks about the motivations of our work, and to assess whether the functionalities offered by Radian supported the typical needs of an ISP. Namely, we wanted expert opinions on how useful the tasks described in Section 6.2 were for solving the use cases described in Section 2.2, and how effective Radian was in supporting

those tasks. The users were informed on the supported tasks and their classification into two classes: tasks aimed at understanding the topology of the network as seen by the routing, and tasks aimed at understanding the routing evolution over time. We also wanted opinions on the perceived utility of simplifying the visualization by contracting ASes. A the time the study was conducted, Radian did not support the visualization of metrics.

The interview had three parts. 1) A presentation of the tool, to explain the motivations, the input data, and the functionalities. 2) A supervised session of usage of the tool, in which we proposed a real-life scenario and they were asked to argue on the dynamics of the routing. The focus was not to actually give complete explanations, but to get an initial sense of the logic and interactions of the system. 3) A questionnaire that the users were asked to anonymously fill out with their opinions. The questionnaire contained several statements, each representing a hypothesis we made that a given feature of Radian was useful or effective. The users were requested to rank each statement between 1 (completely disagree) and 5 (completely agree), and to write a short comment with the motivation for each given rank. In the following we list the statements of the questionnaire, together with the minimum, the average, and the maximum ranking given by the users. The statements after the first can be considered in pairs: each statement ending with an "a" is about a motivational matter, and has a corresponding statement ending with a "b" which is about how good Radian was considered with respect with that motivation.

**S1** Traceroute data produced by a probe system, because of the magnitude, are hard to exploit without a visualization tool (4, 4.83, 5).

**S2a** The topology of a part of a network as deduced from traceroute data, both at router and AS level, provides reasonable information to understand the state of the routing in that part of the network and in a given instant (3, 3.33, 4).

**S2b** Radian represents the topology of a network in a comprehensible way (3, 3.83, 4).

**S3a** Traceroutes performed periodically in a part of a network provide sufficient information to understand the dynamics of routing in that part of the network (2, 3.17, 5).

**S3b** Radian represents routing changes in a comprehensible way (3, 4.2, 5).

**S4a** There exist cases in which it is necessary to focus on the routing of a specific AS, keeping at the same time an overview of the routing at AS level (3, 3.6, 4).

**S4b** Radian supports focusing the attention on a specific AS, maintaining at the same time an overview of the neighbour ASes (3, 3.83, 5).

**S5a** Understanding the topology of a network and the dynamics of the routing supports typical activities of network administration and monitoring, including the study of complex scenarios otherwise difficult to analyze (3, 4.5, 5).

**S5b** Analyzing traceroute data with Radian supports the study of complex scenarios (3, 4.33, 5).

Statement S1 is about the original motivation of our work, and asked the users whether visualization is actually useful in this domain. It received very high rankings, the highest in the study. The users were already familiar with the analysis of traceroutes, and in the comments they confirmed that understanding large amount of traceroutes, like those produced by a system of probes, is challenging. Visualization was considered a fundamental tool for this kind of task. Some keywords that frequently appeared in the comments as desiderable properties of a visualization were "dynamics", "aggregation", and "overview". Radian has interface elements to support all of these. Namely, the dynamics of routing is represented with animations. Data are aggregated from two different points of view: first, several traceroutes collected over time are merged into a graph, which highly reduces the amount of data to watch. Also, the user can expand and collapse ASes, to further reduce the amount of displayed data and focus only on a portion of interest. A temporal overview of routing changes is furnished by the event timeline. Finally, the graph itself is a form of overview, since all vertices traversed by a traceroute in some time instant are always displayed.

Statements S2a and S2b regard the capability of the traceroute graph to reconstruct a reasonable and comprehensible snapshot of the routing in a given instant. The opinions were mixed. The users were not completely convinced of S2a, because of some known limitations of the traceroute itself. 1) Some routers of the Internet could be configured, for security reasons, not to respond to measurements like traceroutes. 2) Even if configured to respond, an overloaded node could decide to discard some requests. 3) A reported traceroute path could be a simplified version of the real path, because packets transit inside tunnels that are invisible to traceroutes. 4) Finally, a traceroute path could report non-existing links between some nodes, because of the presence of load balancers in the traversed network. This kind of problem is tackled by a special version of the traceroute tool (see [60]), which however could be not installed on the probes used for the measurements. For these reasons, many of the users suggested to integrate traceroute data with the decisions taken by routing protocols, which are known to an ISP for its own network. On the other hand, all users admitted that these limitations of traceroutes are hardly avoidable, and that when measuring a network belonging to someone else, the traceroute (with its limitations) is one of the very few, if not the only, tools available. The average ranking, even if not that bad (it is greater than 3), is relatively low. We believe that this was due to the strong statement we made: it would have been probably higher if we stressed more the fact that the graph reconstructed from traceroutes is only an approximation of the topology of the traversed network and of its routing. Surprisingly, Statement S2b received higher rankings: this means that, apart from the reported limitations of traceroutes, the graph metaphor implemented in Radian was considered clear and effective. This was also confirmed in the comments.

Statements S3a and S3b regard the effectiveness of periodically performed traceroutes to sample the dynamics of the routing, and the effectiveness of the animations implemented in Radian to represent such dynamics. Statement S3a received a particularly low average ranking and was subject to criticisms similar to S2a: traceroutes were considered too poor in information to let the user to fully understand the routing dynamics. The statement, in our intentions, expressed the capability of periodic traceroutes to report a sampling of the routing changes happened in the

network, so that the user becomes aware of them and has the possibility to make deductions from their comparison. However, similarly to S2a, we believe that the statement was too strong and that was interpreted by the users. Their written comments were fundamental to understand the reasons behind the answers: they complained that simply seeing the routing changes is not enough to understand the reasons behind them, which requires the usage of additional data sources like information on the routing protocols. Two of them were very specific in these terms, saying that traceroutes do not allow a "root-cause analysis" of the routing events, and that they do not help "understand why". We are obviously aware of these limitations, and never intended to present Radian as a tool for root-cause analysis, which is a challenging task that requires specific tools. All that Radian is capable of is to report the sequence of routing changes happened over time, so that the user can precisely tell what changed and when, and then draw some conclusions. The reported events are possibly starting points for a deeper analysis. And, indeed, the very high ranking of Statement S3b confirmed that Radian is effective in this task. The written comments pointed out that the users appreciated the usage of animations, and considered them an effective an intuitive way to represent changes in a traceroute path. Some very interesting comments to Statement S3a and S3b regarded the possibility of comparing routing changes to other metrics, like the round-trip time. The user considered useful, to understand the reason of a routing change, to know if some metric of interest changed at the same time. For example, a sudden improvement of the round-trip time in correspondence of a routing change may imply that some node was overloaded, and the routing protocol changed the routing to avoid that node and restore acceptable performance. A feature of this kind was indeed missing in Radian, and its conceiving is an important outcome of this user study. Hence, we labelled traceroute paths with round-trip time information. However, integrating networked data with more network metrics into one visualization is an interesting challenge to consider for future work.

Statements S4a and S4b regard the usefulness of looking at the network at different abstraction levels, exploiting the clustering of nodes into the ASes they belong to, and the effectiveness of Radian in supporting this feature. Despite the good rankings, these were the only Statements for which no significant written comments were given by the users. We believe that, more than for the previous statements, S4a was too abstract and was not understood completely. Therefore, the users may have given rankings similar to those of the previous statements. However, we observed them during the usage session of Radian, to spot interesting patterns of use, and noticed that all of them made use of the possibility of keeping collapsed the ASes that were not involved in any dynamics. This makes perfectly sense to us, since some ASes with static routing were very large and caused cluttering on the screen, while the interesting part of that instance was the very particular inter-AS routing dynamics. This observation indirectly confirms our expectation that, if available, users gladly use a feature to simplify the current visualization by abstracting those parts that are not of interest for a task.

Finally, Statements S5a regards the general functionalities offered by Radian, consisting in supporting the comprehension of the topology of a network and the dynamics of its routing, as inferred from periodic traceroute data. Statement S5b asks how good Radian is at implementing these features. The average ranking is quite high for both, which is a strong confirmation of the quality of our work. In the written comments, the users considered Radian very effective for the tasks it was designed for, and said that it could help "debugging problems" and "refine future strategies", referring to the administration of a network. Differently from before, the Statement regarding the implementation received slightly lower rankings than the motivations behind it. In the comments the users explained that the possibility of comparing routing changes to other kind of metrics is an important, missing feature of Radian, which would enable a much deeper analysis of the routing. This influenced their general opinion on the tool. See also Statements S3a and S3b.

In conclusion, the users considered Radian a very useful tool for supporting their everyday work in the administration of a network. The kind of functionalities and our implementation were well appreciated. The main complaints were for the intrinsic limits of traceroutes. On the other hand, the users admitted that these limitations are unavoidable and that traceroutes are one of the few sources of data available when analysing a network administrated by someone else, on which there is no control nor information available on the routing protocols. The possibility of visually comparing routing changes to other kind of metrics is considered a fundamental, missing feature of Radian, which motivates the changes discussed above.

## 9 Conclusions and Future Work

We presented a Radian, a tool for the visualization of traceroute measurements towards specific targets on the Internet. It visualizes traceroute data with a radial drawing of a flat clustered graph. The user can interact with the visualization using animations to see the evolution of routing over time, and contracting ASes to select the level of detail in the visualization. The tool uses new algorithms that are specific for visualizing traceroute data.

This work can be expanded in several ways. First, metrics should be integrated with the graph visualization, besides being presented with temporal charts, since visually associating metric changes to specific parts of the network can support user tasks. Second, the geographical positions of nodes, which were not considered in this work, could be used to partially group nodes and give geographical hints to the user in specific cases. Third, the user interface could provide additional functionalities to support the user in finding interesting dynamics in the data, reducing the amount of time spent watching animations. Finally, the layout algorithm could be improved to reduce the edge crossings that in our solution are handled with post-processing.

## References

[1] RIPE NCC, "RIPE Atlas," http://atlas.ripe.net/.
[2] M. Candela, M. Di Bartolomeo, G. Di Battista, and C. Squarcella, "Dynamic traceroute visualization at multiple abstraction levels," in *Graph Drawing (Proc. GD '13)*, ser. LNCS, vol. 8242, 2013.
[3] "SamKnows," http://www.samknows.com/broadband/.

[4] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband internet performance: A view from the gateway," in *Proc. SIGCOMM*, 2011.

[5] "MisuraInternet," https://www.misurainternet.it/.

[6] "CAIDA Ark," http://www.caida.org/projects/ark/.

[7] "Measurement Lab," http://www.measurementlab.net/.

[8] Roma Tre, "Radian, traceroute visualization," http://www.dia.uniroma3.it/~compunet/projects/radian.

[9] B. Augustsson, "Xtraceroute," http://www.dtek.chalmers.se/~d3august/xt/index.html.

[10] R. Periakaruppan and E. Nemeth, "Gtrace - a graphical traceroute tool," in *Proc. 13th USENIX Conference on System Administration*. USENIX Association, 1999.

[11] Visualware, "VisualRoute," http://www.visualroute.com/.

[12] G. Da Lozzo, M. Di Bartolomeo, M. Patrignani, G. Di Battista, D. Cannone, and S. Tortora, "Drawing georeferenced graphs - combining graph drawing and geographic data," in *Proceedings of the 6th International Conference on Information Visualization Theory and Applications, IVAPP 2015*, 2015, pp. 109–116.

[13] QoSient LLC, "Argus," http://qosient.com/argus.

[14] ThousandEyes Inc., "Network monitoring software," http://www.thousandeyes.com/.

[15] G. Lyon, "Zenmap," https://nmap.org/zenmap.

[16] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "The State of the Art in Visualizing Dynamic Graphs," in *EuroVis - STARs*. The Eurographics Association, 2014.

[17] M. Ghoniem, J. D. Fekete, and P. Castagliola, "A comparison of the readability of graphs using node-link and matrix-based representations," in *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, 2004, pp. 17–24.

[18] P. Eades and M. L. Huang, "Navigating clustered graphs using force-directed methods," *J. Graph Algorithms Appl.*, vol. 4, no. 3, pp. 157–181, 2000.

[19] S. Diehl and C. Görg, "Graphs, they are changing," in *Graph Drawing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, vol. 2528, pp. 23–31.

[20] M. Baur and T. Schank, "Dynamic graph drawing in visone," Fakultt fr Informatik, Universitt Karlsruhe, Tech. Rep., 2008.

[21] B. Bach, E. Pietriga, and J.-D. Fekete, "Graphdiaries: Animated transitions and temporal navigation for dynamic networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 5, pp. 740–754, May 2014.

[22] M. Bastian, S. Heymann, M. Jacomy *et al.*, "Gephi: an open source software for exploring and manipulating networks." *ICWSM*, vol. 8, pp. 361–362, 2009.

[23] P. Saraiya, P. Lee, and C. North, "Visualization of graphs with associated timeseries data," in *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, Oct 2005, pp. 225–232.

[24] D. Archambault, H. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 4, pp. 539–552, April 2011.

[25] M. Pohl, F. Reitz, and P. Birke, "As time goes by: Integrated visualization and analysis of dynamic networks," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '08.

[26] P. Federico, W. Aigner, S. Miksch, F. Windhager, and L. Zenk, "A visual analytics approach to dynamic social networks," in *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, ser. i-KNOW '11, 2011, pp. 47:1–47:8.

[27] M. Farrugia, N. Hurley, and A. Quigley, "Exploring temporal ego networks using small multiples and tree-ring layouts," *Proc. 4th International Conference on Advances in Computer-Human Interactions (ACHI 2011)*, vol. 2011, pp. 23–28, 2011.

[28] T. Dwyer and P. Eades, "Visualising a fund manager flow graph with columns and worms," in *Information Visualisation, 2002. Proceedings. Sixth International Conference on*, 2002, pp. 147–152.

[29] I. Boyandin, E. Bertini, and D. Lalanne, "A qualitative study on the exploration of temporal changes in flow maps with animation and small-multiples," *Computer Graphics Forum*, vol. 31, 2012.

[30] M. Farrugia and A. Quigley, "Effective temporal graph layout: A comparative study of animation versus static display methods," *Information Visualization*, vol. 10, no. 1, pp. 47–64, 2011.

[31] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *"Journal of Visual Languages & Computing"*, vol. 6, no. 2, pp. 183 – 210, 1995.

[32] H. Purchase, E. Hoggan, and C. Görg, "How Important Is the Mental Map? - An Empirical Investigation of a Dynamic Graph Layout Algorithm," in *Graph Drawing*, ser. LNCS, 2007.

[33] S. Ghani, N. Elmqvist, and J. S. Yi, "Perception of animated node-link diagrams for dynamic graphs," *Computer Graphics Forum*, vol. 31, no. 3pt3, pp. 1205–1214, 2012.

[34] D. Archambault and H. Purchase, "Mental map preservation helps user orientation in dynamic graphs," in *Graph Drawing*, ser. Lecture Notes in Computer Science, 2013.

[35] ——, "The mental map and memorability in dynamic graphs," in *IEEE Pacific Visualization Symposium (PacificVis), 2012*, pp. 89–96.

[36] H. Purchase and A. Samra, "Extremes are better: Investigating mental map preservation in dynamic graphs," in *Diagrammatic Representation and Inference*, ser. LNCS, 2008.

[37] P. Saffrey and H. Purchase, "The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study," in *Proc. 9th Conference on Australasian User Interface*, ser. AUIC '08, 2008.

[38] D. Archambault and H. C. Purchase, ""the "map" in the mental map: Experimental results in dynamic graph drawing"," *International Journal of Human-Computer Studies*, vol. 71, no. 11, 2013.

[39] C. Friedrich and P. Eades, "The marey graph animation tool demo," in *Graph Drawing*, ser. LNCS, 2001, vol. 1984, pp. 396–406.

[40] ——, "Graph drawing in motion," *Journal of Graph Algorithms and Applications*, vol. 6, no. 3, pp. 353–370, 2002.

[41] U. Brandes and D. Wagner, "Visone - analysis and visualization of social networks," in *Graph Drawing Software*, ser. Mathematics and Visualization, 2004, pp. 321–340.

[42] I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[43] Y. Frishman and A. Tal, "Dynamic drawing of clustered graphs," in *INFOVIS 2004*, 2004, pp. 191–198.

[44] M. Pohl and P. Birke, *Interactive Exploration of Large Dynamic Networks*, ch. Proc. 10th International Conference on Web-Based Visual Information Search and Management, VISUAL 2008.

[45] F. Reitz, M. Pohl, and S. Diehl, "Focused animation of dynamic compound graphs," in *Information Visualisation, 2009 13th International Conference*, July 2009, pp. 679–684.

[46] "RFC 1918. address allocation for private internets," http://www.ietf.org/rfc/rfc1918.txt.

[47] RIPE NCC, "RIPEstat," https://stat.ripe.net/.

[48] G. Sander, "Layout of compound directed graphs," FB Informatik, Universitat Des Saarlandes, Tech. Rep., 1996.

[49] ——, "Graph layout for applications in compiler construction," *Theoretical Computer Science*, vol. 217, no. 2, pp. 175 – 214, 1999.

[50] M. J. Roberts, *Underground Maps Unravelled - Explorations in Information Design*, 2012.

[51] L. Colitti, G. Di Battista, F. Mariani, M. Patrignani, and M. Pizzonia, "Visualizing interdomain routing with BGPlay," *Journal of Graph Algorithms and Applications, Special Issue on the 2003 Symposium on Graph Drawing, GD '03*, vol. 9, no. 1, pp. 117–148, 2005.

[52] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst, "Animated exploration of dynamic graphs with radial layout," in *Proc. INFOVIS'01*.

[53] C. Bachmaier, "A radial adaptation of the sugiyama framework for visualizing hierarchical information," *IEEE Trans. on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 583–594, 2007.

[54] M. Forster and C. Bachmaier, "Clustered level planarity," in *SOFSEM 2004: Theory and Practice of Computer Science*, ser. LNCS, 2004.

[55] G. Di Battista and E. Nardelli, "Hierarchies and planarity theory," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 18, no. 6, pp. 1035–1046, 1988.

[56] G. Battista, W. Didimo, and A. Marcandalli, "Planarization of clustered graphs," in *Graph Drawing*, ser. LNCS, 2002.

[57] M. Raitner, "Visual navigation of compound graphs," in *Graph Drawing*, ser. LNCS, J. Pach, Ed., 2005, vol. 3383.

[58] K. Sugiyama and K. Misue, "Visualization of structural information: automatic drawing of compound digraphs," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 21, no. 4, pp. 876–892, 1991.

[59] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms," *JCSS*, vol. 13, no. 3, pp. 335 – 379, 1976.

[60] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with paris traceroute," in *Proc. 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06, 2006, pp. 153–158.