Radian: Visual Exploration of Traceroutes

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

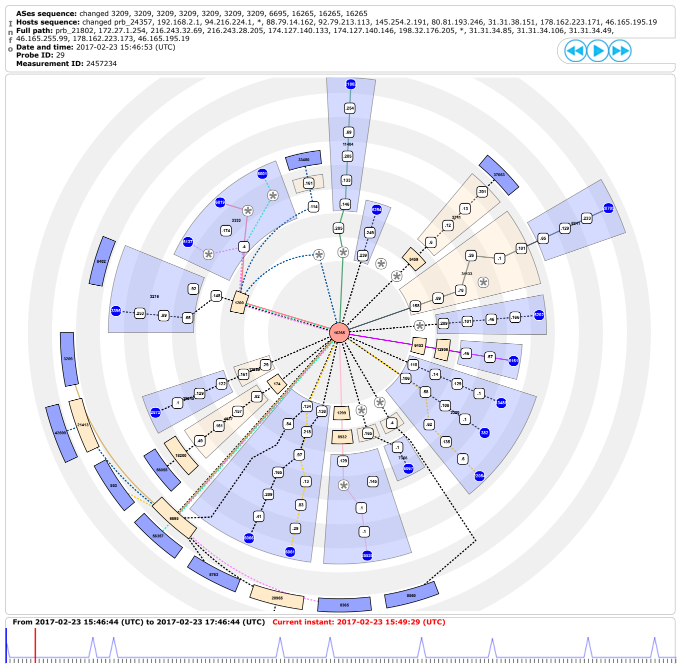
Abstract—Several projects deploy probes in the Internet. Probes are systems that continuously perform traceroutes and other networking measurements (e.g. ping) towards selected targets. Measurements can be stored and analyzed to gain knowledge on several aspects of the Internet, but making sense of such data requires suitable methods and tools for exploration and visualization. We present Radian, a tool that allows to visualize traceroute paths at different levels of detail and to animate their evolution during a selected time interval. We also describe extensive tests of the tool using traceroutes performed by RIPE Atlas Internet probes.

Index Terms—Traceroute, Graph Drawing, Network visualization, Network probes.

1 Introduction

Several organizations distribute *probes* in the Internet with the goal of monitoring the status of the network and measuring its performance. A probe is an autonomous device that periodically executes popular network commands like ping, traceroute, etc. towards selected targets. A few examples of such organizations follow. SamKnows [2] distributes tens of thousands of probes worldwide to get broadband performance data for consumers, governments, and Internet Service Providers (*ISPs*). BISmark [3] uses probes to measure the performance of ISPs. RIPE Atlas [4] is an open project of RIPE NCC with almost 10 thousand probes that can be used by anyone willing to host a probe in order to conduct customized measurements. MisuraInternet [5] is a 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]. Such organizations store measurement results into repositories of data, which are hard to analyze without some visualization facility. This is especially evident for traceroutes, which combine topological and performance information. *Traceroute* is one of the simplest and most popular computer network diagnostic tools that is used to discover the Internet topology. When executed, it outputs the path of routers (*traceroute path*) traversed to reach an IP destination.

We present Radian, a tool for the visualization of traceroute data collected by network probes. We follow requirements gathered from ISPs within the Leone FP7 EC Project. Radian works as follows: the user selects a set P of probes, a target IP address d, and a time interval T, and obtains a visualization of how the traceroutes issued by the probes in P reach d during T. A snapshot of Radian is in Fig. 1 and a demo is available at [8]. Our tool contributes to the



ASes sequence: changed 3209, 3209, 3209, 3209, 3209, 3209, 3209, 6695, 16265, 16265, 16265
Hosts sequence: changed prb_24357, 192.168.2.1, 94.216.224.1, *, 88.79.14.162, 92.79.213.113, 145.254.2.191, 80.81.193.246, 31.31.38.151, 178.162.223.171, 46.165.195.19
Full path: prb_21802, 172.27.1.254, 216.243.32.69, 216.243.28.205, 174.127.140.133, 174.127.140.146, 198.32.176.205, *, 31.31.34.85, 31.31.34.106, 31.31.34.49, 46.165.255.99, 178.162.223.173, 46.165.195.19
Date and time: 2017-02-23 15:46:53 (UTC)
Probe ID: 29
Measurement ID: 2457234

From 2017-02-23 15:46:44 (UTC) to 2017-02-23 17:46:44 (UTC) **Current instant: 2017-02-23 15:49:29 (UTC)**

Fig. 1. The main interface of Radian.

state of the art with: 1) A metaphor for visualizing the evolution of Internet routing as captured by traceroutes. 2) User interaction techniques for increasing or decreasing the amount of details in the visualization, by which the user can either analyze individual network devices or collapse them into the ISP they belong to. 3) New algorithmic techniques tailored for the visualization of traceroute data.

The paper is organized as follows. Section 2 describes the scenario that originated Radian. In Section 3 we compare our contribution with the state of the art. Section 4 introduces basic terminology. In Section 5 we present a preliminary analysis of the data of interest that influenced the design of our tool. Section 6 presents the tool design, along with the adopted visual metaphor and the user interface. Section 7 illustrates the algorithms devised for Radian. Section 8 describes a user study performed with domain experts. Conclusions are in Section 9.

• *M. Candela is with RIPE NCC, Amsterdam, Netherlands.*
E-mail: mcandela@ripe.net
• *M. Di Bartolomeo and G. Di Battista are with the Department of Engineering, Roma Tre University, Italy.*
E-mail: gdb@ing.uniroma3.it, dibartolomeo@ing.uniroma3.it
• *C. Squarcella is with Sysdig, San Francisco, CA, USA.*
E-mail: claudio@sysdig.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 [1]

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 System*s (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 an AS Number (ASN). *Traceroute* is a standard networking tool that reports the sequence of routers used to reach a given destination from a given source. The reported sequence of routers, each labeled 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. Although the packets sent with traceroute are often called *probes*, in this paper we use a convention adopted by systems of probes [2], [4] and use the term only to denote the device that performs the measurement. Several traceroute paths collected at the same time t can be merged to form a *traceroute graph* which is a partial router-level view of the topology of traversed networks at time t. Also, given that each IP address belongs to an AS, a traceroute path can be converted into a sequence of ASes. The result is a partial AS-level view of the same network. Since routing continuously changes, performing such merge at different times allows to capture different states of the network. Traceroute data are affected by several limits: 1) a router could reply to traceroutes from different ports, an effect known as *aliasing*; 2) MPLS tunnels are a layer-2 technology, therefore invisible to traceroutes; 3) *spurious links* could be detected because of load balancers; and 4) the discovered topology depends on the positions of probes and targets in the network, since only traversed nodes are reported. Heuristics exist that mitigate these limits. We chose however not to adopt them because they do not impact the methodology of this work and hence can be externally applied to data by users, at measurement time or with post-processing. As a consequence 1) nodes in a traceroute graph represent network interfaces rather than routers, 2) the graph only represents a partial topology, and 3) some links involving load balancers may be spurious. Note that we mitigate the third problem in our experiments by using Paris Traceroute [9]. Although this work does not deal with routing protocols or their data, in the rest of the paper we will use the term *routing* to refer to the packet forwarding that is the result of such protocols, since the term is commonly used in the trace-“route” domain.

2.2 Users and Use Cases

Several types of users can benefit from monitoring the evolution of routing with a system of probes. ISPs need to check the proper operation of their networks; government authorities need to verify the quality of the connectivity offered to citizens; cyber-security agencies look for abuses of the Internet that may implicate cyber attacks towards or from specific countries; finally, researchers study the Internet because it is a complex and only partially understood ecosystem. We focus our attention on ISPs, having collected clear user requirements from some partner ISPs within the Leone FP7 EC Project. We specifically identify three use cases concerned with the status of routing: *troubleshooting*, *upgrade verification*, and *inter-domain consistency check*.

Troubleshooting copes with any unexpected event that disrupts the normal operation of the network. For example, routers could stop forwarding packets due to overload or damage, causing disconnection or slowness of some network services; also, wrong configuration caused by human mistakes can lead to the overload of a node because too many paths traverse it. Troubleshooting can be generally performed by analyzing traceroute paths link after link, much like low-level debugging in software engineering. It is often useful to compare routing changes with the variation of some metric of interest (e.g. round-trip time, or path length): for example, a sudden increment in latency of a link could explain why many paths abandoned that link almost at the same time. We also stress that traceroute is one of the very few analysis tools that can be used when a problem originates in an external network: ISPs can indeed set up automated alerts on their routers, but cannot control infrastructure that belongs to other ISPs.

An *upgrade verification* is performed when the ISP modifies the network to improve existing services or add new ones. New routers and links are added to connect more sources and targets, or to improve redundancy as a countermeasure to failures and high loads. Upgrades can involve specific devices: for example, a router can distribute incoming traffic with configurable load-balancing criteria (e.g. per-source, per-destination). After an upgrade, the ISP can verify that the desired effect is indeed achieved with an analysis of the dynamics of routing. By inspecting the paths followed by packets, it is possible to check if traceroutes issued by selected probes traverse the expected routers. Also, metrics associated with the paths help assess the outcome of the upgrade: for example, a sudden drop of latency after a path transition shows that the change is beneficial.

Inter-domain consistency check consists of verifying the relationship between the actual routing reported by traceroutes and the intended routing computed with *BGP*, i.e. the protocol used by ISPs to route packets through different ASes. ISPs need to verify the consistency between BGP and IP routing to check that traffic is exchanged with other partner ISPs according to commercial agreements, or that a backup link with a partner AS is operating correctly.

Our use cases are meaningful both in an *online* and *offline* setting, i.e. where the user respectively receives a stream of traceroutes as soon as they are measured in the network, or the whole set of traceroutes at the end of a campaign of measurements. The online setting allows to detect events early, with immediate benefit for each of our use cases; however, it needs specific support from the measurement infrastructure that is not always guaranteed. For example, Atlas [4] only recently added support for streaming measurements, while CAIDA Ark [6] does not support streaming. With this caveat in mind, we developed our tool focusing on the offline setting in order to support a larger set of data sources.

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 to reconstruct and display portions of 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 [10] displays traceroute paths on an interactive rotating globe as series of yellow lines between sites, shown as small spheres of different colors. GTrace [11] and VisualRoute [12] 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 make 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 for a period of time. Also, drawing geolocalized graphs is particularly challenging [13]. 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.

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 [14], 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. As with geographical visualizations, relating the edge length to the values of a metric can produce visual clutter. In the visualization provided by ThousandEyes [15], 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 see the traceroute graph obtained at a given instant. The user can collapse parts of the topology that are not of interest, or highlight elements that experienced a degradation in performance under a metric. Finally, Zenmap [16] gives a radial view of the graph, with one focal node (the source of traceroutes towards several destinations) 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. As with ThousandEyes, 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 that support 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 compare the traceroute graph with metrics. Some useful features, however, have been overlooked in current tools. Routing is a dynamic entity and traceroute graphs change over time, but tools mostly show static graphs. 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 routing at a higher level of abstraction. Finally, 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 model is a key distinguishing feature of 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 [17]. Following that work, methods for visualizing dynamic graphs can be classified by: 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 [18]. 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. As explained in Section 2.2, we focus on the offline setting.

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 (e.g., [19], [20]). Animations are intuitive to the user, and are a compact encoding since the amount of used screen space does not depend on the number of graph updates. On the other hand, it is hard for users to trace the history of an object across a long sequence of frames, because they can only rely on their visual memory [21]. 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 (e.g., [22] [23]). This approach allows the user to visually compare frames. On the other hand, discovering differences between two frames can be hard [17]. 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 [20]. It is still unclear which approach is the best for representing dynamic graphs. Past studies gave mixed results, depending on the experimental setting and the specific tasks (e.g., [24] [25]). We adopt a technique inspired by [20], mixing animations with a timeline for providing the user with an overview of the time interval.

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 property of a visualization of a dynamic graph [26]. The underlying intuition 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 (e.g., [27] [28]), others report insignificant differences or even negative performance (e.g., [29], [30]). 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, possibly at the expense of a suboptimal space utilization within each frame. At the other extreme, a *local optimization* strategy optimizes the layout of single frames possibly prejudicing the mental map. We adopt the global optimization strategy explained in Section 6.2.

Modeling of Transitions Users can be helped in comparing different time instants by explicitly encoding transitions, i.e. by emphasizing the differences between two or 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 (e.g., [31], [19]). However, even when staged, many changes at the same time can be hard to follow [20]. The approach we use on transition encoding will be described in Section 6.2.

3.3 Layout Algorithms for Dynamic Clustered Graphs

As introduced in Section 2.1, the Internet is clustered in ASes. In our use cases this aspect is relevant (Section 2.2), so 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 [32]. 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, but on sparse graphs may produce drawings with fewer edge crossings. Also, hierarchical relationships between nodes are explicitly represented in the layering. Since our graphs are dynamic, we have an additional requirement of stability across changes for nodes, edges, and clusters [17]. For flat clustered graphs, [33] introduces a force-directed algorithm that exploits virtual nodes and cohesive forces so to keep clusters compact and well separated. In [34], the stability of clusters over time is obtained by merging the cluster hierarchies over all instants. This aggregated information is used to produce a superlayout of the graph through a force-directed algorithm, which works as a template for drawing single instants. In [35] a similar approach is used applying an algorithm for layered layouts. Also, during an animation, clusters that remain unchanged are collapsed.

When choosing an algorithmic style for drawing graphs produced from traceroutes, trade-offs must be considered. Force-directed algorithms pose no constraints on the input and have good scalability, but they do not guarantee the drawing quality. Also, the direction of edges is usually ignored. Algorithms for layered layouts have lower scalability and only work on acyclic graphs, but they use edge directions for producing a clear representation of node hierarchies, which can support user tasks. 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, with the results of experiments that we performed on our typical data.

4 Terminology

This section introduces notation used in the paper.

Consider a time interval T and a set of probes P. During T each probe periodically issues a traceroute towards a target IP address d. A traceroute from probe $p \in P$ outputs a directed path on the Internet from p to d, called *traceroute path* or simply *traceroute*. The set of all traceroutes performed by the probes in P during T is denoted by R. If a traceroute is available in the Internet at time $t \in T$, then it is *valid* at time t. We consider a traceroute as a path of edges directed from the target to the source, following a convention that is commonly adopted by algorithms for hierarchical drawings of directed graphs (e.g., [36]). Given a directed edge (v', v''), we say that v' is a *parent* of v'' and v'' is a *child* of v'. A path with no repeated vertices is *simple*. Each vertex of a traceroute in R represents an IP address of a traversed network node. Vertices are identified as follows: (1) p has an identifier assigned by the system of probes; (2) vertices with a *public* IP address [37] are identified by it; (3) vertices with a *private* IP address [37] are assigned an artificial identifier; (4) the remaining vertices are labeled with a "*" (i.e. an unknown IP address). For simplicity, consecutive vertices labeled with "*" are merged into one vertex.

A digraph G^t is defined at each instant $t \in T$ as the union of all the traceroute paths valid at t produced by the traceroutes issued by the probes of P. A digraph Γ is defined as the union of all graphs G^t with $t \in T$. Each vertex of Γ 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 can be extracted from public databases like RIPEstat [38], and for some public IP addresses may be missing. (3) For each vertex v that is not assigned to a cluster after the previous steps, let k' and k'' be the clusters of the nearest predecessor and the nearest successor of v that have an assigned cluster. If $k' = k''$ then v is assigned to k'. (4) Each remaining vertex is assigned to a new *fictitious* cluster. Given a graph, V_k is the set of its vertices assigned to cluster k. The set of all clusters assigned in Γ is denoted by χ. We define $G^t[k]$ as the subgraph of G^t induced by V_k. Analogously, we define $\Gamma[k]$ as the subgraph of Γ induced by V_k.

For any $t \in T$, graph G^t can be visualized at different abstraction levels. Namely, the user can select a set $C \subseteq \chi$ of clusters that are fully visualized, or *expanded*, and each cluster that is in the complement $\bar{C}$ of C is *contracted* into one vertex. Such graph contains edges from all vertices in the fully displayed set of clusters to one another. Also, edges with an end point not in the displayed set of clusters terminate in single vertices that represent contracted clusters. More formally, given the pair (G^t, C), the *visualized graph* G^t_C is defined as follows. The set of its vertices is the union of the V_k for all clusters $k \in C$, plus one vertex for each cluster in $\bar{C}$. The set of its edges is defined as follows. Consider edge (u, v) of G^t and clusters k' and k'', with $u \in k'$ and $v \in k''$. If $k' \neq k''$, $k' \in C$, and $k'' \in C$, then add edge (u, v). If both k' and k'' are in $\bar{C}$ then add edge (k', k''). If $k' \in C$ ($k' \in \bar{C}$) and $k'' \in \bar{C}$ ($k'' \in C$) then add edge (u, k'') ((k', v)). We define Γ_C as the union of the G^t_C for each $t \in T$. Note that $\Gamma_\emptyset$ denotes the graph in which all clusters are contracted, while Γ_χ denotes the graph in which all clusters are expanded and it is equivalent to Γ. We also denote by Δ_C a *drawing* of Γ_C, that is, an assignment of geometrical coordinates to the nodes and the edges of Γ_C. We assume that Γ_C is an acyclic graph for any $C \subseteq \chi$. 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 between one every minute and one every 30 minutes. Then, we processed the data and generated $12,500$ random visualization scenarios for Radian. Each scenario is identified by a tuple $\langle d, P, t, h\rangle$, which identifies a set of traceroute paths collected by 50 probes P, towards target d, starting from instant t and for a period of h hours. Note that, in this context, the time interval T when the traceroutes were collected starts at t and has a duration of h. To construct $12,500$ visualization scenarios we first selected uniformly at random 100 pairs (P, t) in the available data. These were then multiplied by the 5 targets d, and by integer values of h from 0 to 24.

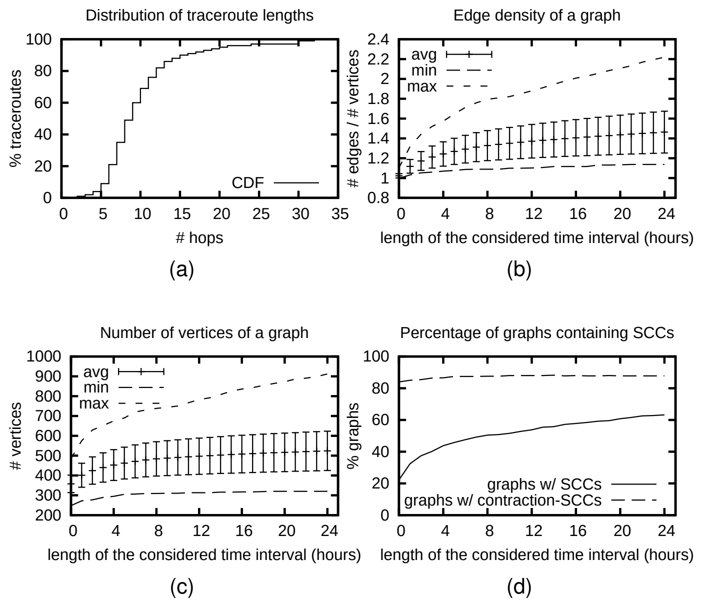


Fig. 2. Statistics on traceroute data. Error bars report the standard deviation. (a) Cumulative distribution function (CDF) of the length of traceroute paths in our dataset. (b) Edge density of a graph in our dataset. (c) Number of vertices of a graph in our dataset. (d) Percentage of graphs that contain SCCs, and that contain contraction-SCCs.

For each scenario we also computed graphs Γ_χ and $\Gamma_\emptyset$ (see Section 4). In this section we will refer to Γ_χ simply as the *graph*, and to $\Gamma_\emptyset$ 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.

In Fig. 2a we plot a cumulative distribution function of path lengths in the dataset. From the figure we see that about 80% of traceroutes have at most 15 vertices, and traceroutes with more than 20 vertices are rare. We confirmed this result by analyzing two additional datasets, collected in Nov. 2016. The first one is [39], where, for a week, a small set of probes (~ 40) performed traceroutes towards every network announced on the Internet ($\sim 650k$). The second dataset is Atlas *anchor measurements* [4], where, for 24h, a large number of probes ($\sim 9k$) performed traceroutes towards Atlas anchors (~ 250). These two datasets are large and complementary on the number of probes and of targets, which makes them a representative sample of path lengths. An important insight of Fig. 2a is that a traceroute path can reasonably be visualized on the screen in its full extent. Figs. 2b-d are related to graphs Γ_χ and $\Gamma_\emptyset$. On the x axis there is the time duration h ranging from 0 to 24 hours, while the y axis shows the value of a metric averaged over all graphs with duration h. 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 ~ 1). Fig. 2c shows the number of vertices in a graph. This number depends 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 Fig. 2c we also draw conclusions on the impact that dynamics has on the size of a graph. We can see that the size 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 would produce an incorrect routing. We performed experiments to test this assumption. Fig. 2d counts the *strongly connected components* (SCC) of our graphs, which imply the existence of directed cycles and can be efficiently detected. A SCC is a subset of the vertices such that there is a directed path between any pair of them. One surprising insight from these charts is that, in graphs produced from traceroutes, cycles exist with non negligible probability. Namely, Fig. 2d shows that, in our dataset, 20% of graphs with $h = 0$ (i.e. almost static routing graphs) contain cycles. The percentage grows to 60% for graphs with $h = 24$. It is easy to see that a SCC that spans more than one AS in the graph 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, and they correspond to AS-level cycles in the data. Fig. 2d shows that contraction SCCs exist in the 90% of graphs in our dataset.

The cycles discussed in this section have different characteristics and causes. First, a single traceroute can contain a cycle because performing a traceroute takes time in the order of seconds, and it can capture transient states of routing. Second, a cycle can be formed in a graph due to the union of paths measured at different instants, which hence might not exist at the same time, and it is a unavoidable effect of the metaphor Γ is based on. This factor is common for graphs with $k > 0$, but it is also possible for static graphs, since the probes that collected the measurements are not necessarily synchronized Finally, BGP-level cycles induced by contraction-SCCs are technically allowed by the BGP protocol, but unlikely to exist in the context of commercial agreements between providers. More likely, a BGP-level cycle is due to an incorrect IP-to-AS mapping, or to several ASes that actually belong to a same organization. By recalling that our analysis was performed on Γ, observe that cycles due to the union over different instants (which are artifacts of our data processing) do not appear in G^t, hence in the visualization.

We draw several conclusions from this analysis. 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 our use cases (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 [40], [41] applied to our graphs are quite effective. However, these algorithms do not guarantee the absence of edge crossings in the drawing. The density of our graphs is so low that very often they are *planar*, i.e. can be drawn without crossings, or quasi-planar, and it would be disappointing for the user to see easily avoidable crossings. For this reason we devised new algorithmic techniques, based on planarity, suitable for our application (Section 7.1). Since our algorithm works on acyclic graphs, and cycles exist in our data with high probability, we describe in Section 7.2 special algorithms for dealing with cycles.

6 Tool Design

This section presents the main elements of Radian, along with user tasks and motivation behind the design.

6.1 Input Data

An *instance* is the input unit of Radian, and it is composed of: 1) A set of traceroute paths towards d, each being a sequence of IP addresses; and 2) a partial mapping from IP addresses to ASes. Traceroute paths are performed in a time interval T and the same path can be repeated. The probe issuing a traceroute is a regular node of the path. The AS mapping, where given, is used by Radian to cluster IP addresses.

6.2 User 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*. Given an instance, the user can select a time instant $t \in T$ and a set of expanded clusters C. The interface presents at any moment the state of the data at t.

The *graph panel* displays the graph G^t_C with a radial drawing centered in d. All vertices and clusters that appear in at least one traceroute in T are shown. Probes in P 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 C encloses the nodes that are assigned to it, while clusters not in C are empty and have a fixed size. Clusters containing probes in P are light blue, the cluster containing d is light red, and the remaining clusters are light yellow. Fictitious clusters are not displayed. Each traceroute path from a probe $p \in P$ to d is represented as a colored curve from p to d passing through intermediate vertices. For each probe the latest traceroute available before t is shown, and it corresponds to a path in the graph Γ. 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 [42], and draw them using different colors. Paths that change in T are represented with solid lines. For paths that do not change and thus represent static routes, we borrow a technique from [43] and show them with aggregated, dashed lines. 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 highlighted when the pointer hovers on it, and, similarly, hovering on a vertex highlights all paths passing through it. Finally, the user can arbitrarily change the set C of expanded clusters. Double-clicking an expanded cluster collapses it and removes if from C, and vice versa.

The *timeline panel* is in the lower part of the window and shows a chart of the number of path changes over time. A red cursor points at the current time instant t. The timeline panel and the graph panel are linked views, and the visualized graph corresponds to the current time instant. 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 clicking that probe in the graph panel. All timelines have the same width and represent the same time interval 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. With the play button, the user sees a sequence of animation steps, each regarding a single path change. At each step the cursor in the timeline panel and the visualization of the graph are updated accordingly. The duration of an animation between two time instants is proportional to the logarithm of the elapsed time between the two instants, which gives an approximate perception of elapsed time while limiting 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.

We implemented a topological visualization, discarding geographical visualizations of traceroute data, for what discussed in Section 3.1. Our use cases are better supported by a graph visualization that shows an abstract topology. The node-link metaphor was chosen for 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. Geography was not considered because it produces visual clutter, and because it compromises the visualization of the AS structure useful for understanding routing policies, 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 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 (e.g., [44]). Also, in this visualization the focus is on the target, avoiding giving unnecessary 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. As noted in Section 3.2, users tend to like animations but are also forced to rely on their visual memory. On the other hand, the scalability of small multiples is too low for 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 snapshots 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 relative positions across different time instants. Updating C 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 [26]. 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 gives hints on parts of the network that are subject to dynamics at some instant. Our choice limits graph updates to paths, 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, we encode transitions by highlighting with thicker lines traceroute paths that change during an animation. 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 the user 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, as shown in Section 5, can be relatively large. Arbitrarily collapsing and expanding ASes poses algorithmic challenges to the preservation of the user mental map. Namely, consider two different sequences of expansions and contractions and let C' and C'' be the resulting sets of expanded clusters. If $C' = C''$ the user expects to see the same drawing, independently on the specific sequence of performed actions. In Section 7.1 we explain how we enforced this property.

Fig. 3 contains 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 d, a set of probes P, and a small time interval T'. Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. This figure shows which ASes are traversed to reach the target. The same graph is presented in Fig. 3(b) with some expanded clusters. The ordering of clusters and vertices on the layers is preserved. This figure shows which network interfaces are traversed inside each cluster. A graph for d, P and a larger interval T'' ($T' \subset T''$) is presented in Fig. 3(c). A dynamic path is visible, meaning that T'' contains additional traceroutes. Fig. 3(d) shows the graph at a different time instant. The

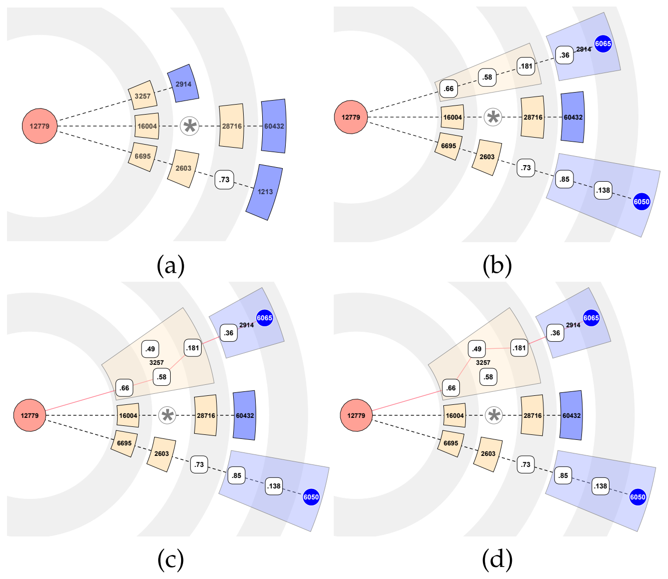


Fig. 3. Details of the interactive features of our visualization. (a) A graph $G^{t'}_{\emptyset}$ relative to a target d, a set of probes P, and at time $t' \in T'$. All paths are static and all clusters contracted. (b) $G^{t'}_{C}$ with expanded clusters C. (c) $G^{t'}_{C}$ at time $t' \in T''$, $T' \subset T''$. A dynamic path is present. (d) $G^{t''}_{C}$ at time $t'' \in T''$, $t'' > t'$. The dynamic path has changed.

dynamic path traverses a different set of vertices.

6.3 User Tasks

This section describes the tasks that a user can execute with Radian. They are classified in two groups: *topology tasks*, devoted to understanding the structure of routing and of the network at a given time instant, and *dynamics tasks* devoted to understanding how routing evolved over time. Topology tasks are executed on the routing captured at a single time instant, while dynamics tasks require the user to compare different time instants and make use of animations. The use cases in Section 2.2 can be resolved by executing sequences of user tasks. For brevity, we refer to the use cases as *Troubleshooting*, *Verification*, and *Consistency*.

The list of the topology tasks follows.

Find which nodes are traversed by $p \in P$. This task is fundamental for all use cases, since it discovers the route followed by p to reach d at a given instant. It is accomplished by following the colored curve across the graph, from p to d, and seeing which nodes are intersected.

Find the topological distance to d. This tasks implies discovering the number of nodes to be traversed to reach d. Often, shorter paths provide better performance. This is useful in Troubleshooting and Verification, 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 d is reachable. This task supports Troubleshooting and Verification. d is reachable from $p \in P$ if the curve from p intersects it.

Find which probes in P traverse a node. This task is useful to understand which parts of the network depend on a node for reaching the target. The task is executed by following all curves that traverse the node, from that node back to the probes. In Verification this implies checking if the node designated to connect a subset of the network is working properly. In Troubleshooting, 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.

Find single points of failure. This task is in the middle between Troubleshooting and Verification. The objective is finding 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 d reachable. The task consists of finding which ASes are traversed to reach d from a probe. The user contracts all clusters to simplify the visualization, and looks for paths of ASes to the target. Such paths can be then compared with BGP announcements to check if the actual routing is consistent, fundamental in Consistency.

Test if $p', p'' \in P$ are treated equally in the network. In Troubleshooting, an ISP may receive mixed feedbacks from its customers regarding the use of a remote service. While those near p' experienced low performance or even an outage, those near p'' did not. By comparing the paths towards d, the ISP may discover that intermediate nodes forwarded data through different paths depending on the source. In Consistency the ISP could compare external ASes, if suspected of offering different levels of service to the ISP.

The list of the dynamics tasks follows.

Find probes of P subject to path changes. As described in Section 6.2, 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. The timeline panel gives an overview of the distribution of path changes during T, and the user restricts T so to include some event of interest, with only a rough temporal indication. In Troubleshooting, a customer ticket could complain about degraded performance or lack of service starting from a given hour of the day. In Verification, the same analysis is done to check if routing changed after a device configuration change.

Find failing nodes. The paths of several probes could abandon a specific node at the same time. This is clearly highlighted in the graph panel, which shows with an animation that the corresponding paths stop traversing that node and morph to another path. In Troubleshooting, this could be the sign of a hardware failure on that node. In Verification, 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 periodicity. In Troubleshooting, the user could start the analysis from a given instant and then discover that more activity is present in the hours before or after that instant with regularity.

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 Consistency, 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.

```
function PREPROCESS($\Gamma$)
    $\Gamma'$ := augment_clusters($\Gamma$)
    $L$ := assign_layers($\Gamma'$)
    $\Delta_0$ := pqtree_layout($\Gamma', L$)
    $L'$ := get_ordered_layers($\Delta_0$)
    $\prec_\chi$ := cluster_partial_order($\Gamma', L'$)
    for all $k \in \chi$ do
        $\prec_V^k$ := vertex_partial_order($\Gamma', L', k$)
    end for
    return $(\prec_\chi, \prec_V)$
end function
```

Fig. 4. Preprocessing phase of the layout algorithm.

```
function DRAW($\Gamma, C, \prec_\chi, \prec_V$)
    $\Gamma_C$ := contract_clusters($\Gamma, C$)
    $\Gamma_C{}'$ := augment_clusters($\Gamma_C$)
    $L$ := assign_layers($\Gamma_C{}'$)
    $y$ := compute_vertical_coordinates($\Gamma_C{}', L$)
    $L'$ := enforce_orders($\Gamma_C{}', L, \prec_\chi, \prec_V$)
    $x$ := compute_horizontal_coordinates($\Gamma_C{}', L'$)
    return $(x, y)$
end function
```

Fig. 5. Drawing phase of the layout algorithm.

Test if a path change improved the performance. In all use cases, the user could be interested to see the value of metrics in correspondence of a path change, to check the correlation. 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 comparison.

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 Section 5, the layout algorithm of Radian produces layered layouts and is oriented to planarity. Relevant techniques in literature do not support clusters (e.g., [45]), are planarity tests that do not work with non-planar graphs (e.g., [36], [46]), do not produce hierarchical drawings (e.g., [47]), or use a local layering scheme, while global layering produces more compact drawings ([40], [41]) which is suitable for the average length of traceroute paths (Section 5). For these reasons, we devised a new algorithm as a planarization-oriented variation of [46], and that uses a global layering scheme.

The algorithm works in two phases. At a high level, the *preprocessing phase* (Fig. 4) computes relative positions for the vertices of Γ. Then, every time the user selects an instant t and expanded clusters C, the *drawing phase* (Fig. 5) computes a drawing of G_C^t enforcing the relative positions computed for Γ. The approach builds on the offline setting (Section 2.2), enabling the pre-computation of relative positions for vertices appearing in the graph at any t. This preserves the user mental map (Section 6.2).

In the preprocessing phase, function *augment_clusters* produces Γ' by adding vertices and edges to Γ, so that every path traversing a cluster k has the same number of vertices in k. These new vertices and edges are fictitious and are not shown in the final visualization. Given any cluster k and remembering that we consider traceroutes as directed path originating from d, we call a vertex a *source* (*sink*) of $\Gamma[k]$ if it is the first (last) vertex of $\Gamma[k]$ encountered in some traceroute path. Function *augment_clusters* also merges the sources of each cluster in one vertex, forcing every path that traverses the cluster to contain this vertex. Function *assign_layers* computes a layering L for the vertices of Γ'. It is an assignment of each vertex to an integer such that edges are all directed towards increasing layers, and such that the difference between the layer of a vertex and that of the root d is equal to the longest path between them. The layering is also proper, meaning that an edge that spans more than two consecutive layers (*long edge*) is replaced with a path of fictitious vertices (*bends*). Function *augment_clusters* ensures that two clusters do not share layers if they are connected by an edge, and that all sources of a cluster are in the same layer. Function *pqtree_layout* computes a drawing Δ_0 of Γ', where vertices in the same layer are assigned the same y coordinate, and x coordinates are computed in a similar way as in [46]. Namely, we use a PQ-tree [48] to order vertices along the layers so that 1) vertices on a layer and belonging to the same cluster are consecutive; 2) the number of edge crossings is reduced. Our PQ-tree is initialized with a spanning tree of Γ' and is incrementally updated with the remaining edges, which 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 Δ^0. Edges traversed by many traceroutes are prioritized, since they are visually prominent. The PQ-tree could produce unnecessary edge crossings due to the source-merging step performed by function *augment_clusters*. To mitigate this effect, we start from the sink vertices of Δ_0 and perform a barycentric reordering sweep, an approach inspired by Sugiyama algorithms [32]. Function *get_ordered_layers* orders the layers in the same way as they are in Δ_0, which implicitly encodes an embedding of the graph. From ordered layers L', function *cluster_partial_ordering* extracts a partial order $\prec_\chi$ of clusters such that $k_1 \prec_\chi k_2$ if $v \in k_1$ appears before $u \in k_2$ in some ordered layer. In a similar way, function *vertex_partial_order* computes a partial order $\prec_V$ for the vertices of a cluster that belong the the same layer. As an implementation detail, $\prec_\chi$ is actually computed as a total order, by performing a DFS visit of a spanning tree of Γ' embedded following L', and visiting the children of a vertex in counter-clockwise order. This visit returns a left-to-right scan of the clusters of Δ_0.

In the drawing phase, the user selects a graph G_C^t to visualize. Function *draw* returns a drawing of Γ_C by using the information computed during the preprocessing, then the vertex coordinates of this drawing are reused for visualizing G_C^t. Note that the interface shows all vertices of Γ_C but only the edges of G_C^t (Section 6.2). In detail, *draw* works as follows. First, function *contract_clusters* creates Γ_C by contracting in Γ all clusters not in C. Function *augment_clusters* performs on Γ_C the same augmentation of the preprocessing, considering only expanded clusters. A proper layering L is computed by function *assign_layers*. If any bend is produced, it is assigned to a fictitious cluster and inserted in $\prec_{\chi}$ at an intermediate position between the endpoints of its long edge. Function *compute_vertical_coordinates* produces for each vertex a y coordinate that is proportional to its layer, assuming that layers are visualized as equally spaced horizontal lines. Function *enforce_orders* produces ordered layers L' by applying $\prec_{\chi}$ and $\prec_V$ to L. Note that, although two clusters may or may not share a layer depending on the particular G_C^t, enforcing $\prec_{\chi}$ fixes their relative ordering. Function *compute_horizontal_coordinates* produces x coordinates for vertices by solving a linear program, such that: 1) the order of vertices in L' is preserved; 2) let $\bar{v}$ be the horizontal distance of a vertex v from the barycenter of its parents, then $\sum_v \bar{v}$ is minimized. To obtain a radial drawing, the coordinates of vertices returned by function *draw* are geometrically 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 curved edges are drawn only in the space between two consecutive 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 $\Gamma_{C''}$ that is produced by contracting a cluster in graph $\Gamma_{C'}$ can contain a cycle that does not exist in $\Gamma_{C'}$. For this reason, given a graph Γ with clusters χ, an algorithm for removing cycles must ensure that Γ_C is acyclic for any set of expanded clusters $C \subseteq \chi$. A naive approach like checking all possible sets of expansions is infeasible, since it requires checking a number of graphs that is exponential in the number of clusters. However, Theorem 1 shows that, for Γ_C to be acyclic for any C, it is sufficient that $\Gamma_\emptyset$ and Γ_χ are acyclic.

Theorem 1. There exists a cycle in Γ_C for some $C \subseteq \chi$ if and only if there exists a cycle in $\Gamma_\emptyset$ or in Γ_χ.

Proof: The proof in one direction is trivial: $\emptyset$ and χ are sets of expanded clusters, therefore if $\Gamma_\emptyset$ or Γ_χ contain a cycle then some Γ_C does. For the other direction, assume that Γ_C, with $C \neq \emptyset$ and $C \neq \chi$, contains a cycle Ω. If all vertices of the cycle belong to a same cluster then that cycle also exists in Γ_χ and the theorem holds, so assume that Ω spans more than one cluster. Let $k \in C$ be an expanded cluster that is traversed by Ω. The cycle enters and exits the boundary of k several times, producing pairs of entering and exiting crossings on such boundary. There is a path inside k for each such pair of crossings, because the cycle is connected. Contracting k collapses such path and identifies the two crossings, preserving the connectedness of the cycle. Namely, the graph obtained by replacing k with a vertex v_k contains a cycle Ω' that traverses v_k as many times as Ω traversed k. Contracting every cluster produces a cycle in $\Gamma_\emptyset$ and the theorem holds. □

Reversing edges is a common technique for removing cycles from a directed graph. The number of reversed edges should be small to preserve the original graph as much as possible. Since finding the minimum number of edges to reverse is a NP-hard problem, heuristics are commonly used [32]. However, existing heuristics give little or no control on which edges are reversed, and this can lead to undesirable effects on a layered layout. For example, Fig. 6(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 6(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 6(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. The natural way to interprets this graph is that the path from vertex 1 to vertex 8 is the main one, while the path from 5 to 2 is "going back" and is the cause of the cycle. Figure 6(d) is a drawing that follows this intuition, reversing the subpath from 5 to 2 and producing a drawing with a more natural look.

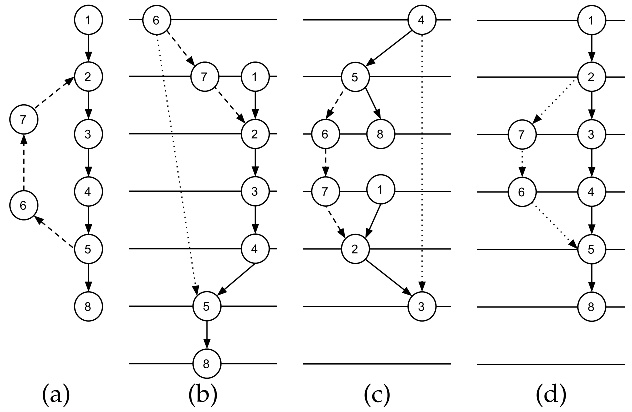


Fig. 6. Layered layouts induced by different choices of edges reversed to remove cycles. The edges that close the cycles are dashed, the reversed edges are dotted. (a) A cyclic graph. (b) Layout induced by reversing edge $(5, 6)$. (c) Layout induced by reversing edge $(3, 4)$. (d) Layout induced by reversing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

```
function REMOVE_CYCLES($\Gamma_C, R$)
    $R'$ := maximal_acyclic_traceroutes($\Gamma_C, R$)
    $\Gamma_C'$ := merge_traceroutes($R'$)
    $L$ := assign_layers($\Gamma_C'$)
    for all $r \in R \setminus R'$ do
        ($\Gamma_C', L$) := reinsert_path($\Gamma_C', L, r$)
    end for
    return $\Gamma_C'$
end function
```

Fig. 7. Algorithm for removing cycles from a traceroute graph.

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 $\Gamma_\emptyset$, then the same procedure is applied to Γ_χ in such a way that the edge directions of the two graphs are consistent.

The algorithm is described in Fig. 7. Function *remove_cycles* is applied to $\Gamma_\emptyset$ and to the set R of traceroutes that induce it. Here, each traceroute in R is considered contracted, i.e., it is a sequence of clusters. Function *maximal_acyclic_traceroutes* finds a maximal set $R' \subseteq R$ that induce an acyclic graph. This is done by iterating on each SCC of Γ_C, and by iteratively removing edges from it until it is acyclic. Edges traversed by few traceroutes are removed first. All traceroutes traversing a removed edge is finally ignored. Function *merge_traceroutes* merges the remaining traceroutes and produces an acyclic graph $\Gamma_C{}' \subseteq \Gamma_C$. Function *assign_layers* produces a layering L of $\Gamma_C{}'$, which is possibly non-proper. Then, function *reinsert_path* iteratively adds to $\Gamma_C{}'$ each discarded traceroute. Such a path is first split into maximal subpaths such that the two extreme vertices of a subpath have a layer assigned. Then, each subpath is added to Γ'_C, reversing its direction if it violates the current layering. The layering is updated with the new vertices at every addition. If the traceroute to add is not simple, maximal simple 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 $\Gamma'_\emptyset$ which is an acyclic version of $\Gamma_\emptyset$. Function *remove_cycles* is then applied to Γ_χ, with the following constraint: inter-cluster edges preserve the direction established in $\Gamma'_\emptyset$. Namely, let $e = (v', v'')$ be an edge of Γ_χ such that $v' \in k'$ and $v'' \in k''$. The direction of e is (v', v'') if $(k', k'') \in \Gamma'_\emptyset$, otherwise it is (v'', v'). The constraint keeps $\Gamma'_\emptyset$ and Γ'_χ consistent. Finally, Γ'_χ is the acyclic graph to be processed by the layout algorithm (Section 7.1).

8 User Study

We evaluated the effectiveness of our tool by performing an informal user study in two phases, interviewing expert users in network analysis to collect comments on Radian.

8.1 Phase 1

The phase 1 of the user study was preliminary, and aimed at understanding how useful for their work users perceived Radian. Our null hypothesis was that users considered analyzing traceroute data with Radian ineffective for the use cases described in Section 2.2. We interviewed 6 technicians of a large Italian ISP, in the context of the Leone FP7 EC research project. 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 users were presented an incomplete prototype, lacking the visualization of metrics.

The interview had three parts. 1) A presentation of the tool. 2) A supervised session of use of the tool, in which users were asked to discuss about the routing dynamics of a proposed real-world scenario. The focus was not collecting correct answers, but to let the users experience the interface of the tool. 3) A questionnaire that the users were asked to anonymously fill out with their opinions. We decomposed the evaluation of the tool in the evaluation of its main features. The questionnaire contained several statements, each affirming 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 on the reasons. Table 1(a) presents the rankings given to the following statements.

TABLE 1
Rankings collected in the user study questionnaires. For each entry, the minimum, average, and maximum user ranks are shown. Ranks are between 1 and 5, higher ranks are favorable to Radian.

User study phase 1

Statem.	min	avg	max
S1	4	4.83	5
S2a	3	3.33	4
S2b	3	3.83	4
S3a	2	3.17	5
S3b	3	4.2	5
S4a	3	3.6	4
S4b	3	3.83	5
S5a	3	4.5	5
S5b	3	4.33	5

(a)

User study phase 2

Task	min	avg	max
T1	2	4.60	5
T2	2	4.33	5
T3	1	3.60	5
T4	1	4.27	5

(b)

S1 Traceroute data produced by a probe system, because of the magnitude, are hard to exploit without a visualization tool. **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 at a given instant. **S2b** Radian represents the topology of a network in a comprehensible way. **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. **S3b** Radian represents routing changes in a comprehensible way. **S4a** There are 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. **S4b** Radian supports focusing the attention on a specific AS, maintaining at the same time an overview of the neighbor ASes. **S5a** Understanding the topology of a network and the dynamics of routing supports typical activities of network administration and monitoring, including the study of complex scenarios otherwise difficult to analyze. **S5b** Analyzing traceroute data with Radian supports the study of complex scenarios.

Statement S1 asked the users whether visualization is useful in this domain. It received very high ranks. In the comments they confirmed that understanding large amounts of traceroutes, like those produced by a system of probes, is challenging, and that visualization is a fundamental tool for this kind of task. Some keywords that frequently appeared in the comments as desirable in 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. Aggregation is pursued by merging traceroute paths, and by AS clustering. 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 at a given instant. The opinions were mixed. The users were not completely convinced of S2a, because of some known limitations of traceroute data (see Section 2.1). To mitigate the problem, several users suggested to integrate traceroute data with information from routing protocols, which is available to an ISP regarding its own network. On the other hand, all users admitted that these limits of traceroutes are hardly avoidable, and that when measuring a network belonging to someone else, traceroute is one of the very few, if not the only, tools available. Surprisingly, Statement S2b received higher ranks than S2a, meaning that, besides limits of data sources, the graph metaphor of Radian was considered effective.

Statements S3a and S3b regard the effectiveness of periodically performed traceroutes in sampling the dynamics of routing, and the effectiveness of Radian animations in representing such dynamics. Statement S3a received a particularly low average rank and was subject to criticisms similar to S2a: traceroutes were considered too poor in information to let the user fully understand routing dynamics. Comments were fundamental to understand the reasons behind the answers. The users complained that simply seeing the routing changes is not enough to understand the reasons behind them, which requires additional data sources. Two of them were very specific, saying that traceroutes do not allow a "root-cause analysis" of routing events, and that they do not help "understand why". We believe that Statement S3a was too strong, and that this caused the low ranks. Radian is not a tool for root-cause analysis, but it rather allows to spot path changes with special behavior, which enables a deeper and more focused analysis to understand the causes of that behavior. As with S2b, the very high ranking of Statement S3b confirmed that Radian is effective for this task. In comments, users appreciated animations, and considered them an effective an intuitive way to represent changes in a traceroute path. Users commented also on the lack of a comparison between routing changes and other metrics, like the round-trip time, which would help understand the meaning of path changes. This motivated us to implement a feature of this kind (Section 6.2).

Statements S4a and S4b regard the usefulness of clustering nodes by AS to have a more abstract view of the topology, and how effective was the interface of Radian. Despite the good rankings, these were the only Statements for which no significant comments were provided. We believe that S4a was too abstract and was not understood completely, therefore the users may have given ranks similar to those of the previous statements. However, during their use of Radian, we noticed that everyone kept ASes not involved in any dynamics contracted. We expected this, since in the instance some ASes with only static routing were very large and caused clutter on the screen, while the interesting part was the very particular inter-AS routing dynamics. This observation agrees with our expectation on S4a and S4b.

Finally, Statement S5a generally evaluates the soundness and the effectiveness of Radian. The average ranking is quite high for both, which is a strong confirmation of the quality of our work. In the comments of S5a, 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". However, for S5b they noted that the comparison of routing changes with metrics was a fundamental missing feature, and that this influenced their general opinion on the tool.

8.2 Phase 2

The phase 2 aimed at comparing Radian with existing tools for traceroute visualization. Our null hypothesis was that Radian was not better than existing tools for its typical use cases. We focused on open source tools, among which only Zenmap [16] aggregated paths in a graph for comparing several of them, so we selected it for the study. The user group of phase 1 was not longer available, so we interviewed 11 engineers of RIPE NCC and 4 academics, PhD and Master's students in computer networks. The interview plot was similar to that of phase 1. The users were asked to solve a real-world troubleshooting problem with data coming from Atlas [4], regarding a target that at some point became unreachable to a subset of probes. In particular, they were asked to solve four tasks. **T1**: Find a narrow time window when the outage was measured [potential existence of an outage]. **T2**: Decide if the probes were affected roughly at the same time, or distant in time [potential existence of a common cause]. **T3**: Find the probes that were more affected by the outage [potentially near to the victims]. **T4**: Find nodes or ASes (other than the probes) that are impacted by the outage [potentially near to the cause]. For each task, the questionnaire asked to give a solution, to rank between 1 (strong preference for Zenmap) and 5 (strong preference for Radian) their preference on the tools, and to write a short comment on the reasons. To avoid that the experience gained with a tool impacted the other, we prepared two different instances (I1, I2) of the problem with similar structure, and the users analyzed a different instance with each tool. Also, to remove any bias of difficulty of the two instances, the users were separated in two groups: one solved I1 with Radian and I2 with Zenmap, the other did the inverse. The instances included traceroutes performed every 15 minutes in an interval of few hours, while the produced graphs had about 15 probes, 100 vertices, and 25 ASes. Zenmap does not support time evolution (nor do other open source tools), so to allow a comparison we preprocessed the instances by creating static snapshots at regular intervals. The frequency was chosen to produce 10 snapshots, an acceptable first move from a user who does not know what he needs to search in the data and yet wants to analyze a reduced number of frames. In the preprocessing we also inverted all paths since Zenmap by default shows one source and several targets, and labeled nodes with their IP address and ASN. The users were informed of these caveats. Table 1(b) presents the rankings of the four tasks.

T1 required users to navigate time and spot paths interrupted before reaching the target. Then, for narrowing the time window, they needed to find two instants where paths started being interrupted and where the connectivity was restored. The answers were all similar, with a strong preference for Radian. The users all praised the stability of the layout across time, while with Zenmap "it's even hard to find the same nodes in different screenshots". The timeline of Radian was considered a "good abstraction of time", and highlighting changing paths "convenient". One user had a strong preference for Zenmap. He considered the visualization of Radian too cluttered and distracting, and generally liked the aesthetics of Zenmap for its simplicity, which helped him stay focused on the paths. However, he did not give solutions to the task, nor commented on time navigation. He also noted that in Radian ASes far from the target were unnecessarily larger, a side effect of our geometrical morph (Section 7.1) that could be reconsidered.

T2 required deeper exploration of time to understand the temporal distribution of disconnections. There was a significant preference for Radian. Users agreed that the finer time granularity of Radian made them more confident on the distribution of path changes. One commented that with Zenmap "you can only imagine what happened between two different screenshots". Having to switch between snapshots was considered inconvenient. Also, some noted that inconsistent node positions across snapshots made it hard to follow the evolution of paths. On the other hand, some found that locating interrupted paths with Zenmap was more natural because they were represented with dashed lines, while Radian used them for a different purpose.

T3 required users to find the probes related to disconnected paths. The answers were controversial, without a clear preference. Several users considered the tools equivalent, because the task could be solved with both tools by following interrupted paths up to their probes. Users in favor of Radian praised its interactive features (e.g., highlighting paths of interest), while others preferred the lack of interaction of Zenmap for its simplicity. The solutions were less consistent and correct than for other tasks, meaning that the users had a false confidence of its triviality. However, users in favor of Radian, or who considered the tools equivalent, gave solutions at least partially correct, while the others gave wrong solutions or did not give one.

T4 could be solved by both looking at the node-level and at the AS-level graph. All users provided a list of ASes as a solution, preferring a more abstract view of the graph. This preference was confirmed in the questionnaire. There was a preference for Radian, and most users appreciated the aggregation of its explicit AS graph, e.g., "Radian makes it trivial, Zenmap shows too many things". One user commented that the AS graph was easy to find in Zenmap too, since the induced graph of an AS was a tree, which made the higher visual complexity of Radian unnecessary. This is correct since our instances did not have ASes with a disconnected graph, a factor that we overlooked when preparing the experiment. Another user complained for the interface of Radian being too cluttered, with a bad use of the radial space especially when all ASes were expanded. The correctness of the solutions was variable. With Radian, 8 users gave correct solutions, 2 gave wrong solutions, and 5 gave no solution. With Zenmap the values were 3, 3, and 9. We believe that the higher number of correct solutions with Radian was due to the fact that users did not feel confident enough at providing an answer with Zenmap.

8.3 Conclusions on the Study

Radian was considered a useful and effective tool by expert users of a large ISP (Table 1(a)). The analysis of traceroute data collected by probes over time was considered challenging, eased by the various forms of visual aggregation and by the interactions of Radian. Some users had concerns for intrinsic limits of traceroute data, which would need additional sources of information to be mitigated; however, they admitted that in some cases these limits are unavoidable, and that Radian supports the analysis of the available data.

When compared with an existing tool on a typical use case, Radian was generally preferred (Table 1(b)). Some supporting, general user comments from the questionnaire follow. *"If I want to discover what is going on on my network I would use something that gives me a complete control of the data over time like Radian"*. *"I think on real and complex cases, Zenmap would be a mess"*. *"Overall, Radian is a more captivating, intuitive and easy to use tool"*. *"I would use it everyday!"*. *"The BGP view generated with traceroute data is seriously practical and interesting"*. The preference was stronger for tasks that involved time, with the timeline control preferred over an example of small multiples. The stability of the layout was also considered fundamental for comparing data at different instants. While the aggregation at AS level was praised by most users, some considered it a cause of visual clutter, preferring a less aggregated but simpler visualization. These users also preferred a static view over the interactivity of Radian. However, in the general comments of the questionnaire, the majority of users commented that aggregation and interaction are necessary for the analysis of larger real-world cases, and desired more interactions for comparing topological elements with time. These users considered the effort necessary for preprocessing data and analyzing them with a static visualization excessive. Users who strongly preferred Radian also produced solutions to the tasks more frequently and with higher accuracy, but the causality of such observation is not clear from this study. The visual clutter noted by some users, e.g., the excessive proximity of graph elements, could be reduced by revising the layout algorithm (Section 7.1) to produce a circular layout starting from the preprocessing phase (Figure 4), rather than applying a radial transformation to a linear layout.

9 Conclusions and Future Work

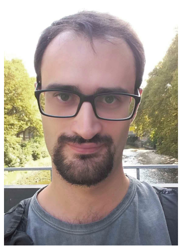
We presented Radian, a tool for the visualization of traceroute measurements towards specific targets on the Internet. It visualizes traceroute data as a radial drawing of a flat clustered graph. The user can interact with Radian to see the evolution of routing over time, and can expand and contract Autonomous Systems to select the desired amount of detail in the visualization. Radian uses new layout and pre-processing algorithms that are specific for visualizing traceroute data. The on-line demo version of Radian [8] is quite popular and attracts tens of hits every day.

This work can be expanded in several ways. 1) The layout algorithm could be improved to reduce the visual clutter and make a better use of the radial space. 2) The user study pointed out that new user interactions could support a deeper analysis of the relationships between topological elements and time, and reduce the time spent watching animations. 3) Networking metrics, currently presented with temporal charts, could use different metaphors more integrated with the graph view, easing the comparison with path changes. 4) Geographical positions of nodes, which were not considered, could be used to partially group nodes and give geographical hints to the user in specific cases. And 5) Where data are available, forward and reverse traceroutes could be compared to show forwarding asymmetries.

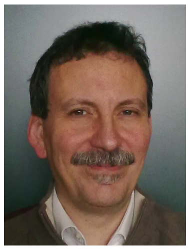
References

[1] M. Candela, M. Di Bartolomeo, G. Di Battista, and C. Squarcella, "Dynamic traceroute visualization at multiple abstraction levels," in *Proc. Graph Drawing*, 2013.
[2] "SamKnows," http://www.samknows.com/broadband.
[3] 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.
[4] RIPE NCC, "RIPE Atlas," http://atlas.ripe.net.
[5] "MisuraInternet," https://www.misurainternet.it.
[6] CAIDA, "CAIDA Ark," http://www.caida.org/projects/ark.
[7] "Measurement Lab," http://www.measurementlab.net.
[8] Roma Tre University, "Radian, traceroute visualization," http://www.dia.uniroma3.it/~compunet/projects/radian.
[9] 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. IMC*, 2006.
[10] B. Augustsson, "Xtraceroute," http://www.dtek.chalmers.se/~d3august/xt/index.html.
[11] R. Periakaruppan and E. Nemeth, "Gtrace - a graphical traceroute tool," in *Proc. USENIX Conf. on System Administration*, 1999.
[12] Visualware, "VisualRoute," http://www.visualroute.com.
[13] 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 *IVAPP*, 2015.
[14] QoSient LLC, "Argus," http://qosient.com/argus.
[15] ThousandEyes Inc., "Network monitoring software," http://www.thousandeyes.com.
[16] G. Lyon, "Zenmap," https://nmap.org/zenmap.
[17] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "The State of the Art in Visualizing Dynamic Graphs," in *Comp. Graphics Forum*, 2014.
[18] M. Ghoniem, J. D. Fekete, and P. Castagliola, "A comparison of the readability of graphs using node-link and matrix-based representations," in *Proc. INFOVIS*, 2004.
[19] U. Brandes and D. Wagner, "Visone: Analysis and visualization of social networks," in *Graph Drawing Software*, 2004, pp. 321–340.
[20] B. Bach, E. Pietriga, and J.-D. Fekete, "Graphdiaries: Animated transitions and temporal navigation for dynamic networks," *IEEE Trans. on Visualization and Computer Graphics*, vol. 20, no. 5, 2014.
[21] P. Saraiya, P. Lee, and C. North, "Visualization of graphs with associated timeseries data," in *Proc. INFOVIS 2005*.
[22] M. Pohl, F. Reitz, and P. Birke, "As time goes by: Integrated visualization and analysis of dynamic networks," in *Proc. AVI '08*.
[23] P. Federico, W. Aigner, S. Miksch, F. Windhager, and L. Zenk, "A visual analytics approach to dynamic social networks," in *Proc. i-KNOW*, 2011.
[24] D. Archambault, H. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *Trans. on Visualiz. and Comp. Graphics*, vol. 17, no. 4, 2011.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] D. Archambault and H. Purchase, "The mental map and memorability in dynamic graphs," in *Proc. PacificVis*, 2012.
[30] P. Saffrey and H. Purchase, "The *mental map* versus *static aesthetic* compromise in dynamic graphs: A user study," in *Proc. Australasian User Interface*, 2008.
[31] C. Friedrich and P. Eades, "Graph drawing in motion," *Journal of Graph Algorithms and Applications*, vol. 6, no. 3, pp. 353–370, 2002.
[32] I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia, *Graph Drawing: Algorithms for the Visualization of Graphs*. Pr. Hall, 1998.
[33] Y. Frishman and A. Tal, "Dynamic drawing of clustered graphs," in *Proc. INFOVIS*, 2004.
[34] M. Pohl and P. Birke, "Interactive exploration of large dynamic networks," in *Proc. VISUAL*, 2008.
[35] F. Reitz, M. Pohl, and S. Diehl, "Focused animation of dynamic compound graphs," in *Proc. Information Visualisation*, 2009.
[36] G. Di Battista and E. Nardelli, "Hierarchies and planarity theory," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, 1988.
[37] "RFC 1918. Address allocation for private Internets," http://www.ietf.org/rfc/rfc1918.txt.
[38] RIPE NCC, "RIPEstat," https://stat.ripe.net.
[39] CAIDA, "The CAIDA UCSD IPv4 Prefix-Probing Dataset - Nov. 2016, first week," http://www.caida.org/data/active/ipv4_prefix_probing_dataset.xml.
[40] G. Sander, "Layout of compound directed graphs," FB Informatik, Universitat Des Saarlandes, Tech. Rep., 1996.
[41] ——, "Graph layout for applications in compiler construction," *Theoretical Computer Science*, vol. 217, no. 2, pp. 175 – 214, 1999.
[42] M. J. Roberts, *Underground Maps Unravelled - Explorations in Information Design*, 2012.
[43] L. Colitti, G. Di Battista, F. Mariani, M. Patrignani, and M. Pizzonia, "Visualizing interdomain routing with BGPlay," *J. of Graph Algorithms and Applications*, vol. 9, no. 1, pp. 117–148, 2005.
[44] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst, "Animated exploration of dynamic graphs with radial layout," in *Proc. INFOVIS'01*.
[45] 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.
[46] M. Forster and C. Bachmaier, "Clustered level planarity," in *Proc. SOFSEM*, 2004.
[47] G. Battista, W. Didimo, and A. Marcandalli, "Planarization of clustered graphs," in *Proc. Graph Drawing*, 2002.
[48] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms," *J. Comp. and Syst. Sci.*, vol. 13, no. 3, pp. 335 – 379, 1976.

Massimo Candela has a Master's degree in Computer Science and Engineering from Roma Tre University, Italy, where he developed *BG-Play.js*, an open-source framework for creating network visualization tools. His main interests are network measurements and data visualization. Since 2013 he has worked as a software engineer at the RIPE Network Coordination Centre.



Marco Di Bartolomeo has a Ph.D. in Computer Science and Engineering from Roma Tre University, Italy, where he worked in the Graph Drawing and Network Visualization group. His research interests include graph drawing, network visualization, graph algorithms, and temporal data visualization. Since 2016 he has worked as a software engineer at Google, Inc.



Giuseppe Di Battista is a Professor of Computer Science at Roma Tre University, Department of Engineering, Italy, and head of the Graph Drawing and Network Visualization group. He has a Ph.D. in Computer Science from the University of Rome "La Sapienza". His current research interests include computer networks, graph drawing, and information visualization. He has published more than 200 papers in the above areas and has given several invited lectures worldwide.

Claudio Squarcella has a Ph.D. in Computer Science and Engineering from Roma Tre University, Italy, where he worked in the Graph Drawing and Network Visualization group. His research interests include network visualization, graph drawing, and network data analysis. He worked as a software engineer at ThousandEyes, Inc., and since 2016 he has been at Sysdig.