Roma Tre University
Ph.D. in Computer Science and Engineering

Dynamic Visualization of Service Performance and Routing on the Internet

Claudio Squarcella

Dynamic Visualization of Service Performance and Routing on the Internet

A thesis presented by
Claudio Squarcella
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Engineering

May 2014

Committee:
Prof. Giuseppe Di Battista

Reviewers:
Prof. Ulrik Brandes
Prof. Antonios Symvonis

Les trois quarts du mal des gens intelligents
viennent de leur intelligence.

Marcel Proust

Acknowledgments

I would like to thank my advisor, Giuseppe Di Battista, for his continued support during these years. It all started back in 2009, when he proposed me to complete my Master's thesis work in Amsterdam. Since then he inspired me with his brilliance, discipline, patience, clarity, and humility.

I extend my gratitude to Maurizio Patrignani, Maurizio Pizzonia, and all the past and current members of the Compunet Research Group at Roma Tre University that I had the pleasure to meet over the past three years: Patrizio Angelini, Massimo Candela, Marco Chiesa, Luca Cittadini, Giordano Da Lozzo, Marco Di Bartolomeo, Valentino Di Donato, Fabrizio Frati, Gabriele Lospoto, Bernardo Palazzi, Marco Passariello, Massimo Rimondini, Vincenzo Roselli, Giorgio Sadolfo, and Stefano Vissicchio.

I am grateful to folks at the RIPE NCC in Amsterdam, the Netherlands, and at the Microsoft Skype Division in Tallinn, Estonia, for opening their doors and exposing me to challenging and *addictive* real-world problems that deeply inspired a big portion of my research activity.

I thank all my co-authors from different universities and organizations for sharing their ideas and supporting mine: Emile Aben, Lukas Barth, Till Bruckdorfer, Kimberly C. Claffy, Alberto Dainotti, Sara Irina Fabrikant, Michael Kaufmann, Stephen Kobourov, Anna Lubiw, Tamara Mchedlidze, Wolfgang Nagele, Martin Nöllenburg, Yoshio Okamoto, Antonio Pescapé, Sergey Pupyrev, Michele Russo, Torsten Ueckerdt, and Alexander Wolff.

I thank Ulrik Brandes and Antonios Symvonis for reviewing this thesis.

Finally, and most importantly, I thank my loved ones. Thank you for all you did in good and bad times.

Contents

Chapter 1

Introduction

The Internet has become a crucial tool in many activities of everyday life. These range from social and recreational efforts to critical tasks dealing with health, science, money, and politics. All such activities are supported by *Internet services* that grow at a fast pace to serve the needs of users from all over the world. The size of networks and customer bases is indeed exploding with the help of new technologies, e.g. mobile data connections. Given the distributed and decentralized nature of the Internet, all services are exposed to a number of potential technical issues: router disconnections, changes in routing policies between Internet service providers, network attacks, and other unexpected events. The goal of guaranteeing an acceptable quality of service for global users therefore implies the adoption of prompt countermeasures in case of failing portions of the network, as well as long-term planning for the progressive improvement of the service.

It is relatively easy to collect huge quantities of data that contain key indicators on the performance and reliability of a distributed system. However, the real challenge consists in interpreting data and using it for actual decisions. The *visualization of the Internet* is therefore becoming a crucial research topic for many stakeholders. A dynamic, interactive interface can make the difference and unveil the potential of so called "big data". We identify three main actors that help us shape the main requirements to address: *network engineers*, *managers*, and *researchers*.

First of all, network engineers need powerful tools to support the maintenance and monitoring of complex and heterogeneous networks. In particular, it is crucial to promptly react to unexpected failures or technical problems, es-

pecially when these may result in the disruption of critical, real-time Internet services. It is therefore essential to build graphical interfaces that can fill the gap for more efficient network administration and troubleshooting.

Further, the analysis of long-term trends on the Internet is equally important at the management level. Managers and policy makers are eager to use dashboards that simplify the analysis of big quantities of data collected over extended periods of time. These interfaces should allow to build hypotheses for major decisions, including architectural redesign and service improvement. Note that the focus is shifted from near real-time and targeted monitoring to a multifaceted and intuitive exploration of aggregate, historical data.

Finally, the observation of the Internet phenomenon is relevant from a purely scientific point of view. New research activity can find inspiration from the usage of interactive tools that reveal meaningful patterns or unexpected behavior. Visualization becomes the bridge for "serendipitous" discoveries that pave the way for innovation in crucial aspects of the Internet.

The first aspect to take into account when designing network visualization tools consists in guaranteeing that the expectations of prospective users are met. The requirements should be collected at the beginning and verified throughout the design process. The metaphors used for the interface and the available interactions should be carefully tested with prospective users, ideally in real-world scenarios. In case the output of the design process consists of more than one alternative, a good option is to let the users compare different alternatives weighting their strengths and weaknesses.

The drawing algorithms that constitute the main building blocks of the proposed solutions are also a crucial topic to address. Several techniques for the efficient representation of evolving graphs and networks are available in literature, as well as results detailing the limited tractability of specific problems. A mature network visualization framework should strive for a perfect balance between the beauty of theoretical algorithmic solutions and the constraints imposed by user requirements. For example, a real-time graph visualization system should rely on optimized layout algorithms, whereas an interactive interface should be responsive and therefore the computation time needed to update the visualization should be negligible.

Last but not least, technical issues are equally important when designing any kind of software. This holds in particular for data-driven visualizations with a focus on user interaction. The Web is a perfect target platform for the development of powerful graphical tools. However, it is filled with different standards, frameworks, and design patterns that evolve at a very fast pace together with the underlying technologies. Each visualization system should

be designed keeping in mind the different alternatives, in order to meet as many non-functional requirements as possible. Efficiency is often the most important of such requirements, especially when dealing with large datasets or highly responsive interfaces. Cross-browser compatibility is also a recurring issue that every Web developer needs to take into account. Reliability and testability are requirements proper of standard software development that are becoming more and more crucial for Web applications. Finally, it is preferable to use frameworks that allow to build scalable and reusable components, in particular when the reference scenario is narrow and well established.

In this thesis we present several approaches to the problem of visualizing Internet service *performance* and *routing* data. We first consider them as two independent topics, proposing interactive tools for both. In a second phase, we introduce the possibility to correlate datasets of both types in order to obtain a combined visualization. All the proposed approaches are preceded by an informative description of the context in which the visualization is needed and the tasks that prospective users would need to perform when using the tool. Design decisions are then illustrated and motivated based on user requirements. The algorithms used for the visualization and animation of graphs and networks are described thoroughly, in particular when they clearly improve on alternatives found in the literature. Technological aspects are always mentioned and highlighted when they represent a key quality of the work. Where available, user studies and feedback collected during the development are reported and discussed. Finally, given the interactive and dynamic nature of our work, we provide the reader with support material (animations, videos, images) accessible through the Web.

Part I introduces the reader to key definitions and concepts that are essential to the understanding of the following chapters.

Chapter 2 is focused on the general field of graph drawing and information visualization applied to networks. We start with preliminary notions on graphs and related data structures. We follow up with an introduction to the problem of visualizing graphs under specific constraints and aesthetic criteria. Further, we briefly present standard algorithmic approaches for the computation of graph layouts that are relevant to our work.

Chapter 3 contains a brief introduction to the real-world scenario under investigation, i.e. the Internet. First, the general notion of "computer network" is explored, with the main entities composing it and the protocols that regulate the transmission of information between endpoints. A few network diagnostic tools that are crucial to our work are briefly mentioned. Finally, the focus is shifted towards a small number of organizations that have a crucial role in the

analysis of the Internet and publish their datasets on the Web.

Part II deals with visualization metaphors conceived to monitor the performance of Internet services. We tackle the question from two different perspectives. First, we observe performance metrics observed in the communication between pairs of hosts in a dense geographical network. Secondly, we focus on the traffic load experienced by the servers of a distributed service and its evolution over time.

In Chapter 4 we tackle the problem of exploring the traffic flow in dense geographical networks where links between nodes are characterized by multiple, time-labeled quantitative metrics. We present FLOWCLIQR, a prototype framework that allows to interactively visualize arbitrarily huge datasets by means of a dual visualization based on a matrix and a geographical map. Users are presented with a concise visualization in which all the information is aggregated at the coarsest level (i.e., macro-regions of the world). By means of intuitive interaction steps, it is possible to explore the flow between specific geographical regions, allowing to distinguish patterns and build initial hypotheses on the underlying data. We evaluate our work with a qualitative study conducted with prospective users and give details on the algorithmic and technical aspects of the framework.

Chapter 5 presents VISUAL-K, a near real-time monitoring system for the K-root name server, one of 13 root name servers in the world. Given their crucial importance these name servers are "anycast", i.e. their service is offered through many identical replicas (or instances) distributed geographically to cover the needs of millions of clients. VISUAL-K helps network operators monitor how the traffic load is distributed amongst all available instances, focusing on unexpected migrations of clients between instances. We propose two different visualization metaphors that are well-suited for the scope, both based on the visualization of a graph in the form of a geographical map. We let prospective users compare them highlighting strengths and weaknesses of both. We also insist on the algorithms needed to dynamically update the two types of visualization. Finally, we report the challenges of dealing with data coming from a distributed cluster of servers in near real-time.

Part III is focused on the exploration of Internet routing and its effect on network topology. In particular, this can be achieved by either probing the Internet with active measurements or directly accessing the control information available at one or more routers. We show examples of visualization based on both approaches.

Chapter 6 is dedicated to BGPLAY.JS, a Web-based framework designed for the visualization of evolving routing graphs. It improves over its acclaimed pre-

decessor, BGPlay, used by network operators to visualize inter-domain routing updates concerning the Internet reachability of specific IP prefixes over extended periods of time. BGPlay.js brings a fresh and renovated implementation that is based on modern Web technologies. The interface is enhanced with advanced controls that help users pinpoint individual routing events. Further, our framework can be easily reused to build new tools for the visualization of graphs and network that evolve over time. We detail all the new requirements collected with the help of experienced users of BGPlay, and explain how our new design addresses them. We also prove the effectiveness of our tool borrowing a use case from a recent research activity on country-wide Internet outages caused by censorship. Finally, given the nature of the framework, we explain in detail the technical and algorithmic aspects of our work.

Chapter 7 deals with the visualization of router paths originated with the `traceroute` command, i.e. a network utility that allows to discover how a computer connected to the Internet reaches a certain target. We present TPlay, a tool for the dynamic visualization of traceroutes at multiple abstraction levels. It gives the user a high-level, hierarchical overview of how a portion of the Internet is connected to a specified target, and allows to interactively expand specific subnetworks to reveal greater details on how the traffic is routed inside them. TPlay is based on BGPlay.js, proving the great degree of abstraction and reusability of the framework in Chapter 6. We describe in great detail the algorithms that allow to efficiently compute the initial drawing as well as the updated views, in response to user interaction. The technical side is also explored, with a focus on the implementation of the layout algorithms.

Part IV represents a meeting point between Parts II and III. The focus is on the combined visualization of the correlation between active measurements (e.g. the output of the `ping` network diagnostic tool) and routing information.

In Chapter 8 we present a methodology to automatically assess the impact of inter-domain routing changes on the round-trip delay measured to reach specific targets on the Internet. Such effort is motivated by the growing number of Internet services that are characterized by strict performance requirements. Therefore it becomes crucial to constantly monitor the performance of the network, as well as to analyze recurring routing behaviors for long-term improvements (e.g. changes in the agreements with Internet service providers). We detail a matching methodology that helps us identify correlations between routing updates and variations in the measured performance. We also present experiments to validate our methodology and post-hoc analysis on the quantitative impact of routing changes on round-trip delay.

Chapter 9 details a visualization framework, called Hydra, in which we

exploit the concepts and results presented in Chapter 8. Our goal consists in graphically exploring the correlation between active Internet measurements and inter-domain routing information, driven by the same requirements of near real-time monitoring and long-term strategic planning. We detail the specific user needs and the corresponding interface design, based on a mixed geographical and abstract metaphor. Further, we introduce the reader to the algorithmic challenges that we faced and the adopted solutions. Finally, technical aspects of the implemented prototype are briefly explored.

Part V contains further research work and a list of articles, papers and technical reports that were completed during the Doctorate Program.

Chapter 10 focuses on additional research activity that has little or no overlap with the network visualization topics presented in Parts II, III and IV, but still gave rise to interesting results and publications.

Chapter 11 concludes our work with a list of journal articles, conference papers, and technical reports that were published (or accepted for publication) during the past three years. Each publication is the tangible result of a research topic explored in one of the previous chapters of the thesis.

Part I

Preliminaries

Chapter 2

Visualization of Graphs and Networks

We start by giving an overview on basic definitions and entities in graph theory, followed by an introduction to graph drawing and network visualization. The reader can refer to more advanced literature (see, e.g., [DETT98]) for further information.

2.1 Graph Theory: Basic Definitions

A *graph* $G = (V, E)$ is a structure composed of a set V of *vertices* (or *nodes*) and a multiset E of unordered pairs of vertices called *edges* (or *arcs*). A graph is said to be *directed* if the pairs of vertices in E are ordered. Given an edge $e = (u, v) \in E$, we say that u and v are *incident* to e and that e is incident to u and v. Two vertices are *adjacent* if they are incident to a common edge. Two edges are *adjacent* if they are incident to the same vertex.

Given a graph $G = (V, E)$, a *self loop* is an edge $(u, u) \in E$, while a set of *multiple* edges is a set of edges incident to the same pair of vertices $u, v \in V$. A graph without self loops or multiple edges is called *simple*. Unless otherwise specified, in the following we assume all given graphs are simple.

A graph $G = (V, E)$ is *complete* if there is an edge $(u, v) \in E$ for each pair of vertices $u, v \in V$. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The *degree* of a vertex is the number of edges incident to it. The degree of a graph is the maximum among the degrees of its vertices.

A *path* in a graph is a sequence of edges connecting a sequence of vertices. A path with no repeated vertices is called *simple*; in the following we assume all paths are simple. In an undirected graph $G = (V, E)$, two vertices $u, v \in V$ are *connected* if there is a path from u to v. G is said to be connected if every pair of vertices in V is connected. A *connected component* of a graph G is a maximal connected subgraph of G.

Some special classes of graphs are heavily used in this work. A *cycle* is a connected graph such that each vertex has degree exactly two. A *tree* is a connected acyclic graph. A *leaf* of a tree is a vertex with degree one, while a leaf edge is an edge incident to a leaf. A *star* is a tree such that removing all the leaves and leaf edges yields an isolated vertex. A *rooted tree* is a tree with one distinguished vertex, called *root*. In a rooted tree, the *depth* of a vertex v is the length of the unique path between v and the root. The depth of a rooted tree is the maximum depth among all the vertices.

2.2 Graph Drawing and Network Visualization

Graph drawing is an area of mathematics and computer science that combines methods from geometric graph theory and information visualization, with the goal of deriving useful and informative representations of graphs. Applications of such discipline traditionally included cartography, circuit design, and biology. Further, new topics are gaining momentum with the rise of new technologies: these include, for example, the analysis of social network and the visualization of computer networks. In this context, the term *Network visualization* can be used as a more general definition encompassing different techniques, not necessarily limited to formal and geometrical approaches.

A *drawing* of a graph $G = (V, E)$ is a mapping of each vertex in V to a distinct point of the two-dimensional plane and of each edge $(u, v) \in E$ to a simple open Jordan curve between the points assigned to u and v. A drawing is *planar* if no two edges intersect except, possibly, at common endpoints. A *planar graph* is a graph admitting a planar drawing. A planar drawing of a graph determines a rotation scheme for each vertex, i.e. the circular ordering of the edges incident to that vertex. Two drawings are equivalent if they determine the same rotation scheme. A *combinatorial embedding* (or simply *embedding*) is an equivalence class of planar drawings.

An embedding partitions the plane into topologically connected regions called *faces*. Vertices and edges are *incident* to a face if they belong to the cycle that delimits it. All the faces are bounded by a cycle except for one face

called *external* face (or *outer* face). The other faces are called *internal* faces.

A *drawing convention* for graphs is a set of rules that the drawing must satisfy. These rules usually depend on the context and the type of usage associated with the drawing. For example, UML diagrams in software engineering represent vertices as boxes and edges as polygonal chains consisting of alternating horizontal and vertical segments. Some typical drawing conventions follow. In *polyline drawings* each edge is drawn as a polygonal chain. *Straight-line drawings* and *orthogonal drawings* are special cases of polyline drawings, with edges represented respectively as straight line segments and polygonal chains of alternating horizontal and vertical segments. For acyclic directed graphs, *upward drawings* represent edges as curves monotonically non-decreasing in the vertical direction. In *contact representations* of planar graphs, each vertex is pictured as a bounded shape and each edge is implicitly represented by the contact between pairs of vertex shapes.

The *aesthetics* in graph drawings are usually measured in terms of specific properties that we would like to preserve and highlight in the final depiction, to achieve better readability. The *minimization of edge crossings* is a standard example, based on the fact that planar drawings of graphs are easier to follow and understand. The *area* of the drawing is also a crucial property: standard techniques generally allow to minimize the total area occupied by the drawing, or to assign prescribed areas to elements in the drawing (e.g. faces, shapes representing vertices, etc). *Edge length* can give a visual clue on specific properties of the graph (e.g. the shortest distance between pairs of vertices), therefore the minimization of the variance of the lengths of edges is often taken into account. Many more criteria are available, depending on specific needs.

Like every optimization problem, the computation of a drawing for a graph usually involves a set of *constraints* related to specific subgraphs or subdrawings. For example, we may require a specific vertex to be placed at the center of the drawing, or on the outer boundary. Vertices may also be required to be drawn close together to form a *cluster*. Specific subgraphs may be required to fit a given "shape". Further constraints apply based on the context.

There are many algorithms and formal approaches to produce drawings of graphs under different drawing conventions, aesthetic criteria, and constraints. We focus on two main techniques that are particularly relevant in our work.

Hierarchical approaches are a common choice for drawings of directed graphs that represent dependency relationships between entities. Examples include PERT diagrams and organizational charts. The goal in such drawings is to highlight the ordered structure of the underlying graph, so that viewers can immediately appreciate dependencies and relationships between vertices. As-

suming for simplicity that the input graph $G = (V, E)$ is directed and acyclic, hierarchical drawings are usually computed in four steps: 1. vertices in V are assigned to layers $L_1, \ldots, L_n$ such that for each edge $(u, v) \in E$ with $u \in L_i$ and $v \in L_j$ we have $i > j$; 2. the graph is transformed into a proper layered graph by inserting dummy vertices along the edges that span more than two layers; 3. the order of vertices on each layer is determined with a procedure that minimizes the number of crossings between adjacent layers; 4. the actual coordinates are computed for each vertex. We detail examples of such techniques in Chapters 7 and 9.

Force-directed algorithms are among the most intuitive methods to create straight-line drawings of graphs. In its simplest form, a force-directed layout evolves from an initial configuration in a simulated environment where a system of forces influences the positions of vertices. The typical example consists of assigning a positive charge to each vertex and an elastic force to each edge, then running a fixed number of iterations in which the position of each vertex is updated based on the forces that interact with it. In Chapters 5 and 6 we show two examples of algorithms that adapt force-directed layout techniques to specific and partially constrained scenarios.

Finally, a more general technique not strictly related to graph drawing is that of *matrix visualization*. It simply consists of representing a graph in tabular format, thus using a number of graphical elements that is quadratic with respect to the number of vertices. Each element can be enriched with colors and patterns to reveal special properties of the underlying graph. For this reason, matrix visualization is particularly effective for (almost) complete graphs with properties assigned to each edge. In Chapter 4 we show how to combine such technique with standard geographical visualization for very dense networks.

Chapter 3

Our Reference Scenario: Computer Networks

This chapter gives a very high level introduction to the basic concepts of Internet communication and routing. We focus on key topics that are later explored throughout our work. The reader can refer to one of the many available surveys and textbooks (see, e.g., [Tan02]) for a more detailed background.

3.1 Networks, Protocols, and Tools

The Internet is a collection of computers that can communicate using different physical channels, from optical fiber cables to WiFi and mobile connections. Computers are organized in simple hierarchical networks, and the interaction between any pair of computers is regulated by a number of routing protocols that operate at different levels of abstraction. Networks are grouped into independent domains called *Autonomous Systems* (ASes), i.e. portions of the Internet that belong to specific real-world organizations. Typical examples of AS owners are *Internet Service Providers* (ISPs), private companies, or public organizations (e.g. universities). Neighboring ASes interact each other by means of a defined set of rules for the transmission of information.

Each computer in a network is identified with an *IP address* (or simply *IP*). IPs can be unique locally (i.e. in the network hosting the computer) or globally (i.e. in the entire Internet). In its simplest form, a computer transmits information by splitting it into data packets labeled with the IP address of the target. Some computers are highly specialized for the transmission of informa-

tion between end hosts. In particular, *routers* are responsible for forwarding data packets between computer networks. The transmission of a data packet hence follows a router path starting at the source computer and ending at the target computer.

Data packets can theoretically come in any format, size, and encoding, as long as the source and target computers agree beforehand on the rules to follow. In practice, however, most computer applications transmit information using standard Internet protocols. For the purposes of our work, some protocols deserve to be mentioned. The *Internet Control Message Protocol* (ICMP) is a protocol typically used for diagnostic or control purposes, or to generate packets in response to errors in IP operations. It underlies important network diagnostic tools like `ping` and `traceroute`, both used extensively in our work. The *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP) are the two protocols used in the vast majority of Internet-based applications. They allow concurrent streams between pairs of computers by enriching their IPs with additional numeric identifiers called *port*. TCP is a rather structured protocol that provides reliable, ordered, and error-checked delivery of packets. Example usages include connecting to Web pages, delivering email, and transferring files between computers. UDP, on the other hand, is a "barebone" container that does not provide mechanisms to guarantee the delivery or the ordering of packets. It is preferred for time-sensitive communications (e.g. real-time video streaming) where losing packets is preferable to waiting for them. Both protocols are founding blocks of the Internet and thus they have a role in nearly all the chapters of our work.

The policy by which a router decides what is the next hop (i.e. the router topologically closer to the target to which packets should be sent) depends on the adopted *routing protocol*. Pairs of routers in a network instantiate sessions to exchange control messages that allow to understand the topology of the network and to make the best choice in terms of transmission speed and reliability. Routers therefore operate at two different levels: the *control plane* is the stage where control information coming from different routers is evaluated to compute the best route to reach any possible destination; the *forwarding plane* is the stage where packets are simply forwarded to the appropriate router, based on previous computation.

Border Gateway Protocol (BGP) is the protocol that regulates the routing of packets traversing networks that belong to different Autonomous Systems. The routers that "speak" BGP are usually placed at the edge of their network and are therefore called *border gateways*. BGP is based on TCP sessions instantiated between pairs of BGP-capable routers, called *peers*. It allows net-

work operators to define expressive sets of policies to influence the routing, based on commercial agreements between ASes and traffic optimization rules. For example, the owner of an AS that is connected to the rest of the Internet through two different ISPs can selectively configure either of the two as the default route provider depending on the target to reach. We take into account several research topics related to BGP in Chapters 6, 8 and 9.

The traffic inside each AS is regulated with simpler routing protocols called *Interior Gateway Protocols* (IGPs), usually based on the computation of the shortest path between any pair of destinations within the network. In Chapter 7 we show a framework for the visualization of router paths that are the combined result of IGP and BGP protocols.

One last important mechanism of the Internet is the *Domain Name System* (DNS), a hierarchical naming system for computers on a network. Domain names (e.g. `www.uniroma3.it`) represent an easier way to identify computers as opposed to IPs and therefore are heavily used by common users on the Web. Computers send DNS queries to *name servers* every time a domain name has to be mapped to the corresponding IP address. All ISPs provide their customers with one or more name servers in order to answer their requests. Upon receiving a query, each name server executes a process called resolution, during which an answer is computed for the query by iteratively querying other name servers. This often implies querying special name servers called *root name servers* (or simply *root servers*). Currently there are 13 root servers, each identified by a letter from A to M and operated by different organizations: e.g. A by VeriSign, B by USC, and C by Cogent. We explain how to monitor the performance of a root name server in Chapter 5.

Many standard networking tools are available for researchers, network operators, and common users to get insights on different aspects of the Internet. The `ping` command is a simple utility that is used to test the reachability of a host on a network and to measure the *round-trip time* (RTT) of messages sent from the source computer to the selected host. It works by sending ICMP "Echo Request" packets to the selected destination and waiting for the ICMP response. We employ `ping` for our research on the correlation between different network data sources in Chapters 8 and 9. A more complex diagnostic tool, `traceroute`, is used to display the path and measure round-trip times of packets across a network. It works by sending ICMP packets with gradually increasing *time-to-live* (TTL) value, that determines the number of allowed intermediate hops before the packet is discarded by the router receiving it. For example, the first ICMP packet has TTL equal to 1: the first router receives the packet, decrements the TTL value to zero, drops the packet because it ex-

pired, and sends an ICMP "Time Exceeded" message back to the source. The process stops when the final target sends a response. Traceroute outputs are the main data source used in Chapter 7, and they also have a role in Chapter 8.

3.2 Organizations and Datasets

The task of designing compelling visualizations for Internet traffic and topology would not make sense without a wide array of datasets and organizations curating them. Some of these organizations regularly publish their data for a wide audience of network operators and researchers. This section presents the main examples that recur often in our work.

The Réseaux IP Européens Network Coordination Centre (RIPE NCC) is an independent, not-for-profit membership organisation that supports the infrastructure of the Internet in Europe, the Middle East, and parts of central Asia. The most prominent activity of the RIPE NCC is to act as the Regional Internet Registry providing global Internet resources and related services (IPv4, IPv6 and AS Number resources) to members in the RIPE NCC service region. Further, the RIPE NCC provides databases and monitoring tools that support stable, reliable and secure Internet operations.

RIPE Atlas [RIP10] is an Internet measurement network. It consists of globally distributed probes that measure Internet connectivity and reachability with standard networking tools like `ping` and `traceroute`. Researchers and network operators can schedule measurements towards custom Internet targets to satisfy different use cases, e.g. verifying the reachability of an Internet service or discovering the different ISPs that route traffic towards a certain destination. Atlas probes also run default "static" measurements towards interesting Internet services including all the root name servers. Most measurements are publicly available for statistical analysis and research. We detail the usage of RIPE Atlas data in Chapters 7 and 8.

The Routing Information Service [RIP01] (RIS) collects and stores BGP routing data from several locations around the globe. It is based on Remote Route Collectors, i.e. software routers typically placed inside Internet Exchange Points to collect routing information from other routers. In particular each route collector instantiates BGP sessions with *collector peers*, i.e. border gateways belonging to real world ISPs and organizations, and collect all their BGP updates. The University of Oregon maintains a similar project worth to mention, called Route Views [Uni97]. The datasets coming from both projects are often used in conjunction to perform various studies on the AS-level Inter-

net topology. We cite numerous examples of previous studies and detail our own findings in Chapters 6, 8 and 9.

Finally, it is worth to mention RIPEstat [RIP11], a Web toolbox that makes it easier to access the various datasets maintained by the RIPE NCC. For example, these include geolocation and registration records that allow to build a mapping between IP addresses and ASes owning them. We mention RIPEstat in Chapters 6, 7 and 8.

MaxMind [Max02] is a provider of geolocation and online fraud detection tools. They regularly update and publish databases containing information related to IP addresses publicly announced on the Internet. Examples of such information include the geolocation of IPs and the mapping to the Autonomous Systems that announce them. MaxMind databases are used by researchers in many different projects. In Chapter 4 we describe how to use their geolocation database to aggregate flow data based on the geography of source and target hosts.

Other organizations collect Internet measurements from various sources and share their results on the Web. The Cooperative Association for Internet Data Analysis (CAIDA) investigates practical and theoretical aspects of the Internet, providing insights into its infrastructure, behavior, usage, and evolution. One of their main projects, Archipelago [CAI07], is an active measurement infrastructure composed of geographically distributed monitors that can perform coordinated Internet probing. CAIDA is also very active in the topic of Internet visualization; we will detail some examples in Chapter 6. SamKnows [Sam08] is a company specialized in providing an accurate picture of global broadband performance. Their goal is accomplished with "whiteboxes" installed in houses and offices around the world to get an indication of the quality perceived at the user end. The collected data and related statistics are published online.

Part II

Visualizing Service Performance

Chapter 4

Exploring Flow Metrics in Dense Geographical Networks

The visualization framework described in this chapter was designed and developed during a three-month internship at the Microsoft Skype Division in Tallinn, Estonia. Microsoft Corporation owns the original intellectual property rights. The datasets used during the development were de-identified before producing material for this publication.

4.1 Introduction

In recent years there has been a considerable growth in the availability of massive datasets describing bidirectional flow of information between end hosts in dense networks. This poses new challenges to researchers and network operators that are eager to use such information for business intelligence, decision making, and service optimization. Skype, the popular VoIP service with more than 300 million active users, makes no exception: the cloud infrastructure supporting its core services is a potential source of crucial information, but it cannot turn useful without proper ways to make sense of it. What is needed is a general purpose tool that allows users to build initial hypotheses by looking at raw data, discovering the extent of knowledge hidden behind the surface.

The main difficulty lies in the density of the networks under examination. The elevated number of interconnections makes it practically impossible to adopt any standard technique for the rendering and interactive exploration of graphs. Simple tabular visualizations of matrices do not work either, as soon

as the size of the input grows beyond few dozens of vertices. On top of that, the input data becomes more interesting when put in a historical perspective and enriched with many different facets and key indicators. The combination of all these elements can be tackled reasonably well for highly specific studies and analyses. However, it is very challenging to build general purpose tools aimed at exploring raw data with the naked eye.

In this chapter we present a framework, FLOWCLIQR, designed for the interactive exploration of time-labeled multivariate flow datasets. FLOWCLIQR offers a dual visualization that allows for both quick lookups and general pattern recognition. The input data is automatically aggregated and the user can navigate forth and back between different levels of detail, while keeping an eye on the general picture. The design of our framework privileges the execution of a number of simple tasks: assessing the volume and features of the flow between pairs of locations, enumerating destinations with poor performance, sorting flow streams based on their volume, and so on. These tasks can be easily combined based on user needs, leading to useful insights about the underlying data. In a broader sense, FLOWCLIQR can visualize arbitrarily dense directed graphs with quantitative attributes on the edges between pairs of vertices. Our work, however, is primarily focused on the visualization of quantitative network metrics between pairs of geographical locations and their evolution over time.

Section 4.2 presents the latest techniques for the visualization and exploration of flows and dense graphs or matrices. Section 4.3 introduces the reader to the simple data structures expected as input and some preliminary operations that we perform on the data. In Section 4.4 we describe the interface of our framework and explain how users can interact with it. Section 4.5 contains the details of the qualitative study we performed to validate our user interface. Algorithmic and technical details are presented in Section 4.6. Finally, conclusions and ideas for future improvements are in Section 4.7.

4.2 Related Work

The visualization of flow in graphs and geographical networks is a recurrent topic in the field of graph drawing. Edge bundling [Hol06] is a well-known technique for the visualization of compounds graphs. It applies to any graph and any mapping of its vertices to arbitrary positions on the plane. The technique consists of bending adjacent edges to reduce visual clutter and highlight the implicit structure of the connectivity between vertices. We discarded this option in our framework because it does not solve clutter in very dense networks

and drawings of bended edges can be misleading. Buchin et al. [BSV11] build flow maps using spiral trees to induce a clustering on the targets and smoothly bundle lines. In a more recent work [NB13] similar maps are obtained with a new edge bundling technique that avoids ambiguous connections between pairs of vertices. Both techniques are visually compelling when describing the flow from a single source to many targets, but are not adequate for dense graphs.

A number of solutions can also be found in the field of information visualization. Andrienko et al. [AAD+08] present a taxonomy of the possible approaches available for the geovisualization of dynamics, movement, and change. They identify three alternatives: 1. direct depiction of data, which can easily lead to clutter and slow rendering; 2. use of summaries like aggregation, generalization and sampling; 3. use of statistical methods to extract patterns before visualizing them. The authors claim that such visualizations can generally be evaluated by small numbers of experts, and that usual evaluation tools like error rate and task completion time are not sufficient. Guo [Guo09] proposes an interface to render large spacial interaction data, consisting of multiple views: a geographical map with arrows representing flow between regions, a self-organizing map, and a parallel coordinate plot. The tool is based on a precomputed hierarchical regionalization based on the volume of flow between pairs of regions. Although reasonable, such precondition is too strict for our purposes. Boyandin et al. [BBBL11] present Flowstrates, a visualization approach in which the origins and the destinations of the flows are displayed in two separate maps, and the changes of flow magnitudes over time are represented in a separate heatmap view in the middle. Wood et al. [WDS10] divide the geographical space with a grid and draw in each cell a replica of the original map that shows inbound flow from all the other regions. Their idea is further expanded in [WSD11], where replicas show approximate flow patterns by means of time series plots. All the above three approaches, however, can lead to cluttered views when the input dataset grows in size. Elmqvist et al. [EDG+08] present a matrix visualization that features fast reordering of rows and columns, data aggregation with explicit representation, and GPU acceleration to optimize the rendering on screen. Their approach inspired part of our work while constructing a dynamic matrix optimized for pattern recognition.

4.3 Data Abstraction

Our work would not make sense without the availability of great quantities of flow data. This section presents a formal description of the type of input and

Time	Region		Country		Network		Packet loss
	source	*target*	*source*	*target*	*source*	*target*	
17:42	Asia	Europe	China	Spain	WiFi	3G	0.37%
18:35	Europe	Europe	Spain	Italy	WiFi	Cable	1.12%
19:01	Europe	Asia	Italy	Japan	3G	3G	0.69%
22:30	Asia	Asia	Japan	China	Cable	3G	4.45%

Table 4.1: Example input records for our visualization framework. Each record features three dimensions (region, country, and network) and one quantitative metric (packet loss).

the simple precomputation required to appreciate the power of our framework.

We call *record* each building block of data in our reference scenario. Each record is labeled with a timestamp and contains attributes extracted from a single unit of flow between a *source* host and a *target* host (e.g. a stream of packets between two hosts, an ICMP echo request, etc). A *metric* is a feature in the dataset, proper of each record and typically belonging either to a continuous or discrete domain. In the field of computer networks, for example, the first group includes percentage of packet loss and round-trip delay, while the second includes the network protocol used for the connection (e.g. UDP or TCP). In our work we focus on the analysis and representation of quantitative metrics with a continuous domain. A *dimension* is an attribute proper of both source and target hosts in each record. It is typically characterized by a discrete domain. Common examples in a networking scenario include the country and the type of Internet access (e.g. cable, WiFi or 3G). An example list of records is presented in Table 4.1.

Dimensions are crucial to make sense of data by means of *aggregation*. Our visualization metaphor makes heavy use of grouping, allowing the user to interactively explore dimensions on request. More formally, we define a *dimension path* to be the rule by which individual records are recursively grouped into larger sets, from the finest to the coarsest level of aggregation. Figure 4.1 presents an example dimension path based on the records in Table 4.1. Given

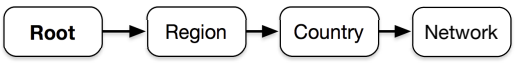


Figure 4.1: Example dimension path for the records in Table 4.1.

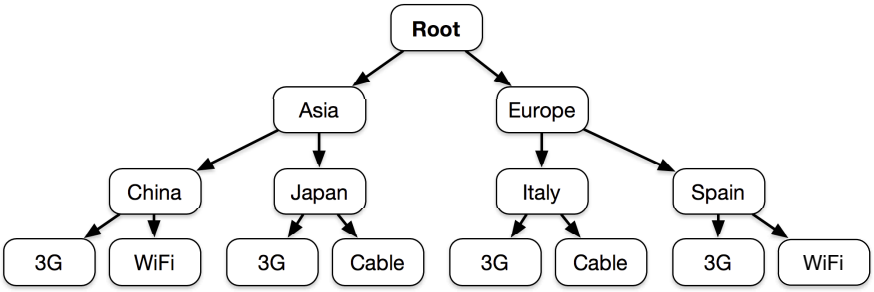


Figure 4.2: Example navigation hierarchy built with the records in Table 4.1 and the dimension path in Figure 4.1.

a set of records and a compatible dimension path, we define the *navigation hierarchy* as the tree that is constructed with the actual values of dimensions in each record, following the structure imposed by the dimension path. Figure 4.2 shows the navigation hierarchy obtained using the records in Table 4.1 and the dimension path in Figure 4.1. A *navigation frontier* is a subtree obtained from a navigation hierarchy, where for each node all its children are either pruned or kept in the tree. The number of navigation frontiers available for a single navigation hierarchy is exponential. Given a navigation frontier, we call *frontier node* each of its nodes. We also refer to each leaf as *frontier leaf* for convenience. We will use the navigation frontier as an abstraction to describe the mechanism of interactive exploration that is built into our framework.

4.4 Interface and Interaction Design

The interface of FLOWCLIQR is presented in Fig. 4.3. It is split into four main views: the control panel (upper left corner), the timeline (lower right corner), the dynamic matrix (lower left corner), and the dynamic map (upper right corner). The example dataset used for this section has three dimensions that are directly translated into a navigation path, from the coarsest to the finest: region, sub-region, and country. The two available metrics are round-trip delay and packet loss.

The *control panel* contains basic information on the current state of the visualization and some controls to change the representation of metric values.

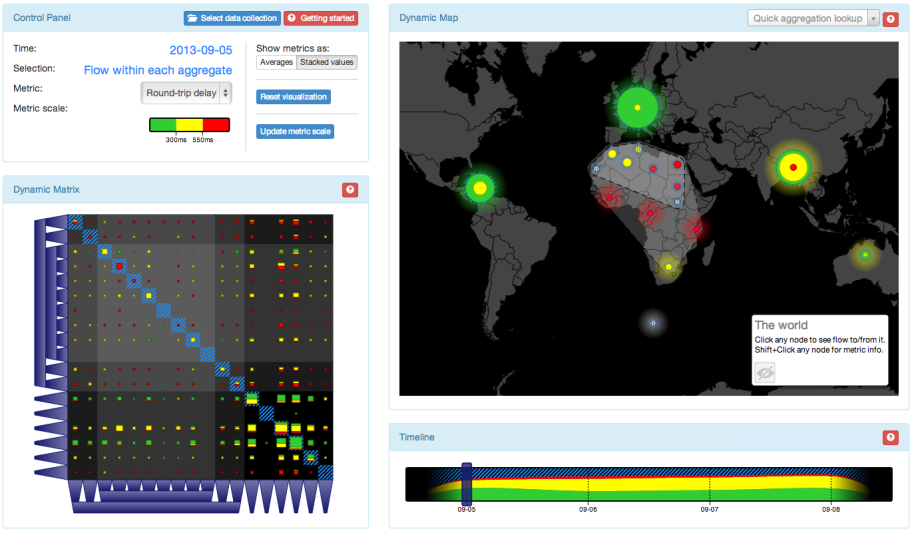


Figure 4.3: Overview of the main interface of our framework.

In Fig. 4.3 the focus is on a specific day (5 September 2013) and each view is highlighting the flow "within each aggregate", i.e. having both source and target hosts within any of the current frontier leaves (e.g. flow within Europe, within Americas, etc). The selected metric is round-trip delay. The three colors used in the other views (green, yellow, red) identify three different classes of values for each metric, reported in the metric scale that is visible just below the metric name. In the above example green means "below 300ms", yellow means "between 300ms and 550ms", and red means "above 550ms". The right side of the control panel contains additional controls. The user can choose between two different types of metric representation for each pair of frontier leaves: 1. *averages*, i.e. only one average value; or 2. *stacked values*, i.e. the weighted distribution of metric values in the three different classes, identified with corresponding colors. The two threshold values in the metric scale that determine the three corresponding classes of values can be dynamically adjusted by clicking the appropriate button. Finally, a button allows the user to bring the visualization back to the original state before any interaction.

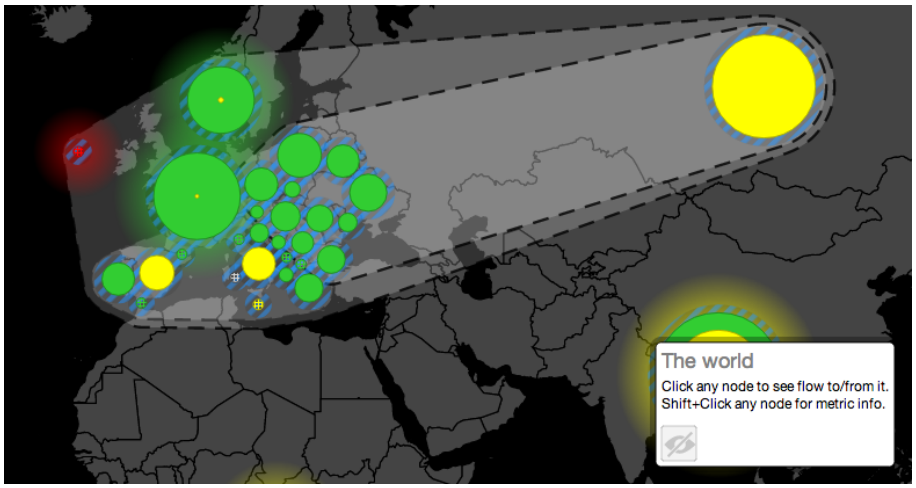
The *timeline* contains a time series with aggregate information for each date available in the data collection. More specifically, a stacked graph shows

the volume of flow over time for each of the three classes of metric values, as defined in the control panel. Detailed information for the closest date is shown when the mouse pointer is placed over the graph. The user can either drag the slider or directly click the timeline to pick a different date. Upon interaction, all the views are updated accordingly to show the corresponding data.

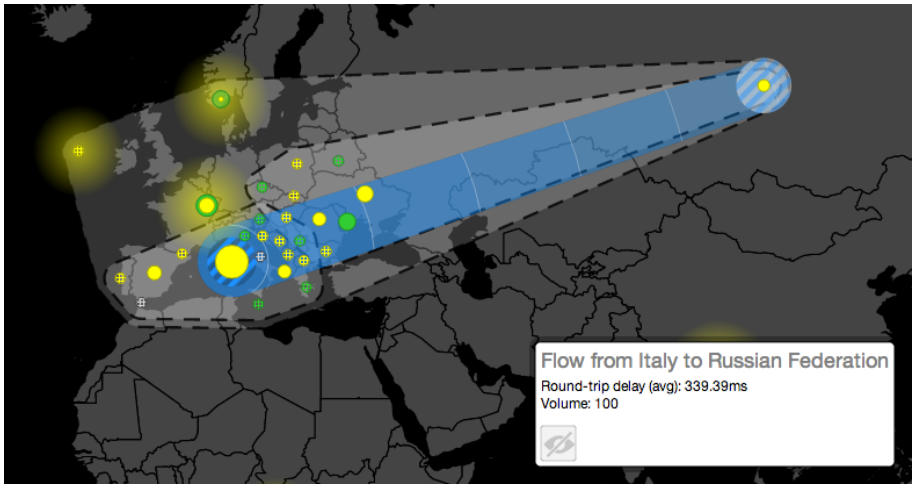
The *dynamic matrix* shows metric values for all possible pairs of frontier leaves on the selected date. It is a square matrix, where rows and columns respectively represent source and target frontier leaves. Each square in the matrix represents a pair of frontier leaves, with size logarithmically proportional to the total volume of flow and colors reflecting the metric values in the current representation. The trapezoids on the left and bottom sides of the matrix represent the full navigation frontier. The hierarchy is explicitly represented by means of side contact between parents and children. All matrix elements are left intentionally unlabelled, so that the user can focus on discovering patterns by looking at the colors of the squares. Hovering any element with the mouse reveals aggregate information for the corresponding entity (either a frontier node or a pair of frontier leaves).

The *dynamic map* shows a circle for each frontier leaf in the current navigation frontier, positioned at a meaningful location (e.g. for regions of the world, circles are placed at the centroid of corresponding regions). At any time, each of such circles shows the same volume and metric values as one of the squares in the matrix. The dynamic map, therefore, only shows a portion of the data contained in the dynamic matrix: such portion can change based on user interaction. In the initial state, circles in the map are in correspondence with squares on the diagonal of the matrix, i.e. each of them represents the flow with both source and target hosts within the corresponding frontier leaf. The regions with dashed borders that enclose groups of circles represent the same hierarchy of frontier nodes that is pictured with trapezoids in the dynamic matrix. The information panel at the bottom right corner shows information depending on interaction. The user can also hover individual circles and dashed regions to get related metric and volume information.

We designed our framework with a focus on interactivity and responsiveness. The two main operations that the user can perform are the following: 1. *data selection*, i.e. narrowing the analysis to the flow originating from a frontier node, targeted at a frontier node, or between two frontier nodes; 2. *frontier expansion*, i.e. updating the navigation frontier, by either exploring the children of frontier leaves or collapsing sibling frontier leaves into their parent frontier node. Both operations can be achieved by interacting indifferently with the dynamic matrix, the dynamic map, or a combination of both. Upon



(a)



(b)

Figure 4.4: Sequence of user interactions needed to study the flow from Italy to Russia. (a) The dynamic map is zoomed on Europe, where two sub-regions (Southern and Eastern Europe) are expanded to reveal their respective countries. The size of each circle is proportional to the volume of flow within the corresponding country. (b) The dynamic map is focused on the flow from Italy to Russia.

user interaction, all the views in the interface are automatically updated to reflect the new state of the visualization. The following subsections give some examples on how to use FLOWCLIQR to solve specific user needs.

Finding the total volume of flow and the average round-trip delay from Italy to Russia on 8 September 2013

First of all, we select the correct date on the timeline and we choose to represent metrics with average values through the control panel. We then locate the circle that represents Europe in the dynamic map and double-click it. The interaction has the effect of updating the navigation frontier by adding all the sub-regions within Europe to the current representation: therefore both the dynamic map and the dynamic matrix are updated accordingly. We repeat the same step with the two circles representing Southern Europe and Eastern Europe, with the effect of enriching the navigation frontier with all associated countries, including Italy and Russia. The current state of the map is in Fig. 4.4(a). Note that the circles representing countries lack the semi-transparent "glow" effect, meaning that they cannot be further expanded to reveal more detailed information (i.e. we reached leaves in the navigation hierarchy). We click the circle representing Italy, triggering the following updates: 1. the row representing the flow from Italy in the dynamic matrix is highlighted; 2. the size and color of each circle in the dynamic map represents the flow from Italy to the corresponding frontier node, and the flow itself is pictured with animated concentric circles centered at the clicked circle; 3. the stacked graph in the timeline shows aggregate data for the flow from Italy to all other destinations. We can achieve the same result by looking up Italy with the search box positioned at the upper right corner of the dynamic map (see Fig. 4.3). Finally, we hold the Shift key and click the circle representing Russia. New updates are triggered: 1. the square in the dynamic matrix representing the flow from Italy to Russia is highlighted; 2. the flow in the dynamic map is represented with "waves" from Italy to Russia, as shown in Fig. 4.4(b); 3. the stacked graph in the timeline shows aggregate data for the flow from Italy to Russia. The requested information can be found in the info panel on the dynamic map.

Counting how many country pairs in Europe have average packet loss greater than 10% on 6 September 2013

First of all we focus on the control panel, choosing the right metric and updating the range of values such that flows with packet loss greater than 10% are

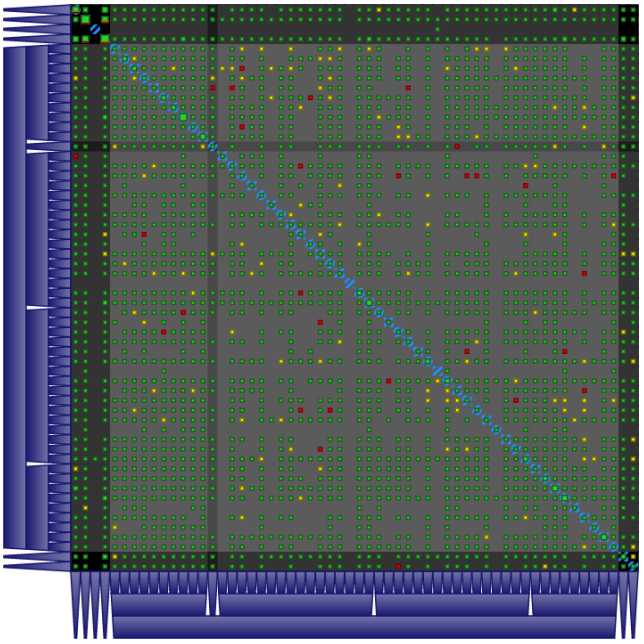
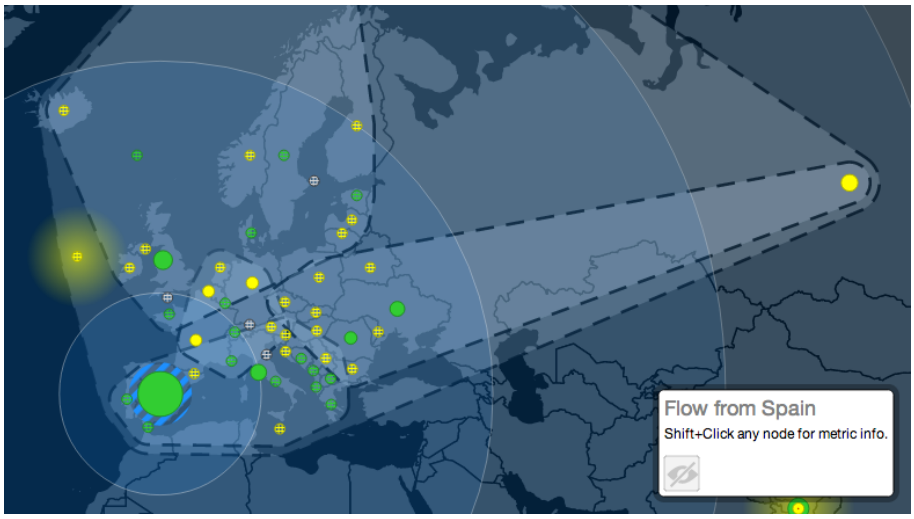


Figure 4.5: Dynamic matrix showing country pairs with average packet loss greater than 10% in red.

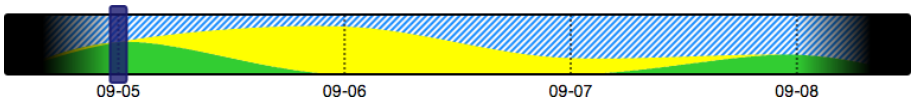
identified with the red color. We select the right date and metric representation. We then proceed to interact with the dynamic map, first double-clicking the circle representing Europe and then all the circles representing sub-regions in Europe, until we reach the country level. We can finally focus on the dynamic matrix and simply count all the occurrences of red squares that fall within the portion of matrix related to European countries, as visible in Fig. 4.5.

Finding out what European country receives the highest volume of flow from Spain on 5 September 2013 and which day sees the highest volume of flow between the same pair of countries

We select the right date on the timeline. Since we focus on the volume, the selected metric is irrelevant. We interact again with the dynamic map to show circles for all European countries and click the circle representing Spain. Apart



(a)



(b)

Figure 4.6: Views showing details for the third example task. (a) The flow from Spain is pictured with concentric blue circles. The size of each circle is proportional to the volume of flow from Spain to the corresponding country. (b) The stacked graph in the timeline reaches its peak on 6 September 2013.

from Spain itself, the biggest circle in Europe is the one representing United Kingdom, as visible in Fig. 4.6(a). To answer the second question we hold the Shift key, click the second circle, and focus on the timeline (see Fig. 4.6(b)): the day with the highest volume of flow from Spain to United Kingdom is 6 September 2013.

4.5 Evaluation

FLOWCLIQR was born with the goal of creating a general purpose framework, targeted at users previously exposed to the underlying data, allowing them to

get quick insights and build initial hypotheses before starting deeper investigations. Since any interaction with the tool can be decomposed into recurring tasks, it becomes crucial to verify that these can be correctly and quickly accomplished by prospective users. This section presents the results of the evaluation study we conducted after implementing the initial prototype.

We initially thought of conducting a comparative study, where participants would need to solve a list of tasks both with our framework and with standard tools (e.g. database queries). However, we quickly discarded this option because even expert users did not have experience with a standard, unified set of tools for the purpose of accessing and analyzing the same dataset. Therefore any comparison would have suffered from potential bias, depending on the relative experience of the participant. We opted instead for a qualitative study, where participants were given a set of tasks and feedback was collected at the end of each task.

In preparation for the study we fed our prototype with a precomputed dataset, structured like the one used for the figures in Section 4.4 and featuring four days of flow data from 5 September 2013 to 8 September 2013. The study was conducted with ten participants (nine male, one female) between 25 and 35 years old. They are all domain experts with a background in computer science, statistics, telecommunications, or electronics. At the time of the study, they were already familiar with the data collection from which we derived the dataset used as test input. More than half of the participants had worked with the same data collection before, accessing its content by means of database queries or simple time series plots.

Each participant was initially tested for color blindness. A thorough introduction to the framework followed, with a focus on each of the views and all the available user interactions. A couple of example tasks were illustrated step by step. After that, each participant was asked to solve the 15 tasks listed in Table 4.2. The first four were treated as training tasks, i.e. the participant had the possibility to ask for help. For each task the examiner recorded the completion time with a stopwatch, gathered feedback afterwards, and showed a quicker way to achieve the same result in case the strategy adopted by the participant was clearly suboptimal. General feedback was asked from each participant as a final step after the last task. The average time required by each participant to complete the study was 50 minutes.

All the participants successfully completed the proposed tasks, adopting different strategies. The statistics on task completion times are reported in Table 4.2. It is evident that users quickly learned from mistakes done in previous tasks. For example, 60% of participants had an instinctive preference

#	Task	Time (s)		
		avg	*med*	*stdev*
1	Find volume of flow from Africa to Europe on 7 September 2013	17.2	15.5	9.13
2	Enumerate regions receiving flow from Africa with average round-trip delay greater than 550ms on 6 September 2013	42.9	41.5	17.06
3	Enumerate pairs of regions that have more than 50% of flow with round-trip delay greater than 700ms on 6 September 2013	90.3	96.5	37.29
4	Enumerate continents that receive flow from Italy on 5 September 2013 with average packet loss smaller than 1.2%	82.2	72.5	27.52
5	Find volume of flow from Italy to Spain on 5 September 2013	34.2	30	13.25
6	Find average round-trip delay from Italy to Spain on 7 September 2013	39	33.5	20.66
7	Find day with highest volume of flow from Italy to Spain	27	21	15.23
8	Find which region receiving flow from Americas has highest percentage of flow with packet loss higher than 2% on 5 September 2013	54	52	12.21
9	Find pairs of regions with average round-trip delay greater than 700ms on 5 September 2013	35.9	34	13.31
10	Find pairs of regions with more than 50% of flow with round-trip delay greater than 700ms on 5 September 2013	26	25.5	7.94
11	Find days in which the average round-trip delay within Italy is greater than 300ms	49.7	45	14.07
12	Find days in which the average round-trip delay from Italy to Russia is between 320ms and 330ms	58.6	52.5	16.63
13	Find the European country receiving the highest volume of flow from Italy on 6 September 2013	124.5	112	45.06
14	Find the European country receiving the highest volume of flow from Northern Africa on 5 September 2013	46.7	50	8.54
15	Find how many European country pairs have average round-trip time greater than 1s on 5 September 2013	59.6	57	14.13

Table 4.2: List of tasks and results of our qualitative study. For each task the average (*avg*), median (*med*) and standard deviation (*stdev*) values for completion times are listed.

for the dynamic matrix when solving Task #13 (i.e. they updated the navigation frontier and compared the size of different squares without using the dynamic map). After being shown a faster solution with the dynamic map, they quickly changed their strategy and performed much better with Task #14 and Task #15. This is confirmed by the relatively small standard deviation for

the completion time of both tasks, which suggests that users knew precisely what steps where needed to complete them. Note also how the median completion time is smaller than the average time for most of the tasks, which suggests that the outliers can be interpreted as occasional difficulties or distractions experienced by individual users.

The feedback was overall very positive and enthusiastic. All participants were particularly impressed by the possibility to finally "see" the data they had only been able to access with database queries and simple two-dimensional charts. They also appreciated the power of exploring data both on the dynamic map and the dynamic matrix at the same time, depending on the specific use case. Many important suggestions for improvement were collected during the study. 80% of participants found the timeline to be not enough intuitive to compare the volume of flow on different days. 50% would have appreciated the possibility to expand frontier nodes straight to the finest level of aggregation, without intermediate levels (e.g. sub-regions in the specific example). 50% had trouble to come up with the right sequence of interactions to highlight the flow within a specific frontier node on the dynamic map (i.e. click followed by shift-click on the same circle). 50% overlooked smaller squares in the dynamic matrix at first sight and 20% suggested to add a "full-screen" capability to each view as a solution. 50% spent a non-negligible amount of time wondering where to find the actual answer for some tasks, after completing all the right interactions. 40% suggested to add a smarter search box to programmatically specify a query in the form "flow from A to B". 40% complained that the size of circles in the dynamic map is not a sufficient clue to estimate the volume. Further minor observations were related to the specific dataset (e.g. 40% were not sure whether Russia was to be found under Europe or Asia) and to the lack of experience with the interface (e.g. 70% of users needed some time before appreciating the distinction between average and stacked metric values).

4.6 Algorithms and Technical Details

The algorithmic background of FlowCliqr is pretty straightforward. The drawing of circles in the dynamic map is achieved with a force-directed graph layout algorithm (see Section 2.2 for a general introduction) including collision detection. Each circle is represented with two vertices connected by one edge: the first vertex is fixed at the ideal position for the center of the circle, while the second is subject to forces and represents the actual position of the visualized circle. The areas with dashed borders that enclose circles in the dynamic map

are obtained with a state-of-the-art algorithm [Rap92] for the computation of convex hulls of circles. We use the same algorithm to compute a clipping path for the representation of flow "waves" between any pair of frontier nodes.

FLOWCLIQR was implemented as a pure JavaScript Web application, using the popular D3.js framework [BOH11] for the development of highly interactive data-driven visualizations. All the views are coordinated following the Publish-subscribe pattern, so that any user interaction is transformed into an event that triggers appropriate updates in each view. A special effort was made to improve the performance of the tool, limiting animations and redraws where possible. That is crucial especially in the dynamic matrix, where the number of graphical elements grows quadratically with respect to the size of the navigation frontier.

4.7 Conclusions and Future Work

We presented a framework for the interactive exploration of the flow between pairs of hosts in a dense network. It allows researchers, network engineers and managers to quickly assess the performance over time of a network at various levels of detail, while keeping an eye on the general picture.

In the future we will extend the set of features of our prototype, overcoming its current limitations. First of all we will follow the suggestions that came out of the qualitative study presented in Section 4.5. Further, we will extend the representation of metrics, including the display of non-quantitative metrics, the explicit rendering of the distribution of values for each metric, and the possibility to filter specific value ranges for a cleaner visualization. The user will have the possibility to pick pairs of dates on the timeline, in order to compare related metric values looking for potential drops or improvements in performance. We will also run experiments with different dimension paths in order to verify that the dynamic map metaphor is effective even with non-geographical data. Finally, we will introduce the possibility to specify a *dimension hierarchy* rather than a simple dimension path, allowing the user to choose between two or more dimensions when expanding individual frontier leaves.

Chapter 5

Monitoring the Load of an Anycast Root Name Server

The visualization framework presented in this chapter was designed in collaboration with the RIPE NCC. The implementation started at Roma Tre University and was completed during a six-month internship at the RIPE NCC. The latest version of the prototype is currently maintained by the RIPE NCC and available online at `http://k.root-servers.net/visualk`. The two scientific publications based on our work are listed in Chapter 11. The reader can refer to [DBSN12] for additional material based on our work.

5.1 Introduction

Root name servers are a critical part of the Internet. They receive hundreds of thousands of queries per second from name servers (as explained in Section 3.1) and must answer immediately. Each root server is implemented with a number of computers spread across several locations worldwide, in order to improve resiliency and efficiency. Each of such locations is called *instance*. Currently each root server comprises at most 70 instances: e.g. A, F and K respectively have 6, 49 and 18 instances.

While a name server can freely select a root server for each of its queries, it cannot select the specific instance that will answer it. A widely adopted mechanism called *anycast* is instead responsible for the decision. It relies on the current status of the Internet routing to determine the instance which is most appropriate, i.e. topologically nearest. Therefore it is possible that consecutive

queries issued by a "client" name server to the same root server are received by different instances, depending on the current status of the routing. This has consequences both from the point of view of the client name server and the root server itself: the first can experience fluctuations in the elapsed service time, while the latter can suffer from changes in the distribution of workload among its instances.

We designed and implemented VISUAL-K, an interface to visualize the status of the service provided by one of the most popular root servers, called *K-root*, operated by the RIPE NCC (see Section 3.2). Our approach can be used to either study what happened during a prescribed time interval or to monitor the status of the service in near real-time. We visualize: 1. how and when client name servers *migrate* from one instance to another; 2. how usual migrations patterns differ from unusual migration patterns; 3. how the workload associated with each instance changes over time; and 4. what is the status of the service offered to specific ISPs. Our framework helps network operators improve the quality of the service offered by K-root. Further, it can be used to spot security issues and to investigate unexpected routing changes.

The chapter is organized as follows. In Section 5.2 we explain the user requirements. In Section 5.3 we describe our visualization metaphor. Section 5.4 presents the feedback collected from the users. In Section 5.5 we present the layout algorithm used in our approach. In Section 5.6 we describe the design of a prototype tool based on our framework and the technical challenges faced during implementation. In Section 5.7 we compare our approach with the existing literature. Concluding remarks and ideas for future improvement of our framework are in Section 5.8.

5.2 User Requirements

Like many distributed information services, K-root challenges its operators with several questions. From the point of view of *performance*, the main goal is to optimize load balancing between instances. In terms of *reliability*, it is important to understand what happens if some of the instances are faulty. It is also crucial to know what is the best topology for existing instances, and where should new instances be deployed from a *design* perspective. Finally, operators are interested in knowing how the system reacts to external attacks (for *security* reasons) and to changes in the inter-domain *routing*.

A crucial need to tackle all the above questions consists in understanding how and when client name servers *migrate* from one instance to another. The

concept of migration can be defined as follows. Given two instances u, v, we say that a client *migrates* from u to v during an interval of time t', t'' ($t' < t''$) if its last query before time t' is sent to u and its last query before time t'' is sent to v. To give an idea of the migration phenomenon, in 24 hours of normal operation about $50,000$ clients issue service requests to more than one instance. The length of the time interval t', t'' plays an important role: short intervals lead to a granularity which is too small, while longer intervals can leave behind short migrations happening between t' and t''. We discussed the issue with the RIPE NCC and agreed upon the fact that a range of few minutes, with an upper bound of 10 or 15 minutes, is a good empirical choice for two reasons: 1. routing events and dynamics have compatible times and 2. migrations lasting only a handful of minutes can be generally ignored by operators.

Furthermore, since migrations are not all the same, a second requirement consists in clearly separating them into classes. The migration between specific pairs u, v of instances is considered *usual* by the operators. For example, that is the case when u and v are placed in network locations that have high connectivity between them, or if it is known that the Internet routing frequently oscillates moving clients form u to v or vice versa. Other migrations are instead considered *unusual*, for example when they involve pairs of instances deployed in places with very poor connectivity between them. Given a pair of instances u, v, deciding if u, v is subject to usual or unusual migration is an evaluation made by K-root operators, based on extensive knowledge of the underlying network. Such information is of course dynamic and subject to change over time, but the evolution rate is much lower than the frequency of observed migrations and traffic patterns. From the point of view of operators, both kinds of migrations are important. Unusual migrations can put in evidence suspicious activities, misconfigurations, or large-scale faults. Timely detection is crucial in order to take appropriate actions and countermeasures, e.g. repairing wrong configurations or limiting the damage. On the other hand, showing usual migration patterns on a regular basis can help for long-term decisions, e.g. to understand where the routing is more unstable, what instances exchange clients more frequently between each other, and where the next instance of K-root should be deployed.

A third important requirement is to monitor the workload of each instance. This can be expressed in many different ways: we identified two main metrics, i.e. the number of clients served by each instance and the number of queries received by each instance. Both metrics are influenced by migration patterns and therefore subject to change over time. Operators need to get an immediate perception of the workload of each instance and the effect of migrations on

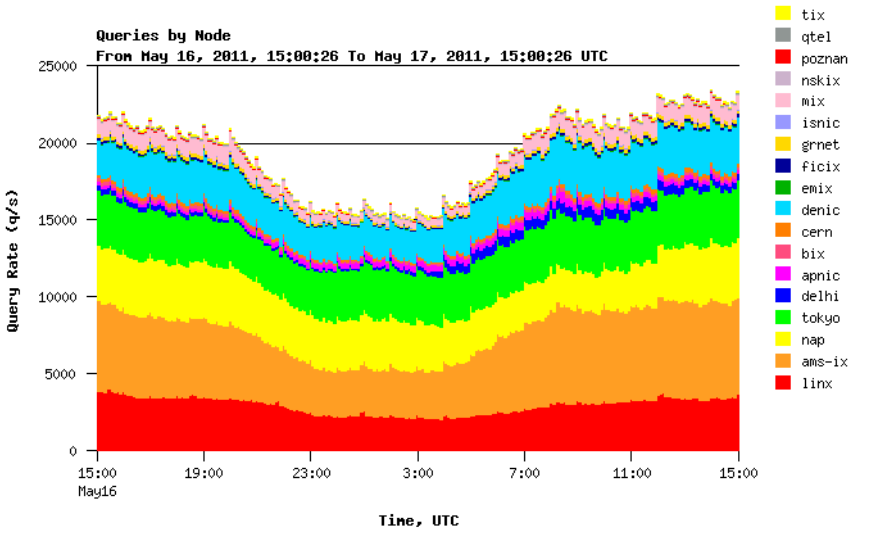


Figure 5.1: Chart showing the query distribution among all the instances of K-root.

it, i.e. how clients and queries are redistributed between instances and what patterns arise. For simplicity, throughout the chapter we will use the number of clients served by each instance as the reference metric to assess the workload.

Finally, as a last requirement it is sometimes important to focus on the dynamics of a specific subset of clients of K-root. In particular, monitoring the evolution of all the queries issued by a specific ISP becomes interesting for a larger group of stakeholders: not only K-root operators, but also people directly working for the ISP in question. This is mainly motivated by concerns on security and performance.

All the above requirements are strongly influenced by events over time, implying the design of a dynamic monitoring tool. More specifically we should allow for both offline analysis within a prescribed time interval in the past and near real-time monitoring. We will consider this as an implicit requirement influencing the design and architecture of the system as a whole.

Fig. 5.1 contains an example chart traditionally used at the RIPE NCC to visualize the distribution of queries received by the instances of K-root. The

chart simply shows the cumulative number of queries per second received by each instance, stacking all the values on top of each other to give an idea of the total workload of the system. Note that it does not give any indication about usual or unusual migrations, nor it allows to visualize the relationship between different instances. As an example, consider the sudden small peak which is visible towards the end of the chart. It is not possible to understand whether it simply implies an increase in the amount of queries on one or more instance, or rather a migration of clients between different instances.

5.3 Defining a Suitable Metaphor

Our study started with a formalization step, aimed at designing a data structure to contain the information collected in our interaction with the RIPE NCC. In particular we tackled the distinction between usual and unusual migrations and agreed upon a graph, called *migration graph*. Each vertex represents one instance, and each edge connects a pair of instances that are subject to *usual* migrations. Consequently we consider *unusual* any migration between pairs of instances that are not adjacent in the migration graph.

We then designed an interface that supports the requirements listed in Section 5.2, after an intensive discussion with K-root operators. We decided to adopt a *geographic map metaphor*. The service offered by K-root is represented as a map. Each instance is a bounded region and its size is proportional to the number of clients that it currently serves. Two regions are adjacent if the corresponding instances are adjacent in the migration graph. The map changes over time as follows: 1. regions change their size according to the fluctuations in the number of served clients, 2. usual migration flows are pictured as bubbles traversing the boundaries of adjacent regions, and 3. unusual migration flows are highlighted with impact graphics as bridges across the regions.

We opted for the above metaphor for a number of reasons. First of all, geography is generally appropriate to describe abstract entities, quantitative informations and relationships between elements (see Section 5.7 for a more detailed discussion). Further, the multiplicity of potential stakeholders must be addressed with a simple and unified model, where all the relevant information is ideally visible at a glance without the need for an extensive technical background. Finally, given the traditional meaning of the concept of "migration", we found the map metaphor to be quite natural and well suited.

We investigated and screened out alternative metaphors for different reasons. As an example, we could visualize the service on a real geographical map,

since the actual coordinates of each instance are known. We discard this choice for several reasons: 1. the dynamics by which a client chooses a specific instance depend on routing policies defined by ISPs, that usually overlap geographically and span over more than one country or continent; 2. migration patterns are also influenced by the status of the routing and largely independent on the geography; 3. combining the geographical data with migration patterns can easily lead to information cluttering; finally, 4. network operators are used to the concept of a *logical view* as opposed to a *geographical view*, as long as they can rely on a new mental map that does not change significantly over time. Section 5.7 documents a number of additional approaches that are related to our choice, together with the reasons why we did not follow any of them.

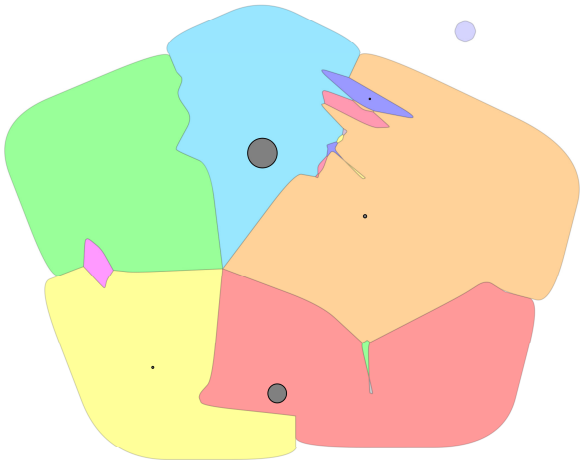
Our metaphor can be implemented in many different ways. We propose two approaches: the *country map* and the *octopus map*.

Country Map

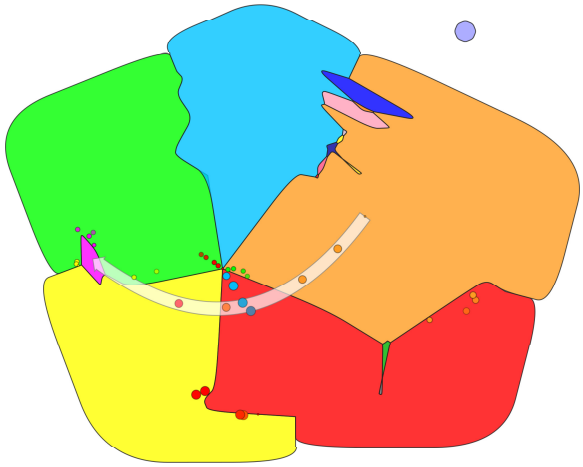
A *country map* is a contact representation (see Section 2.2) of a migration graph, where each instance has an identifying color and a shape resembling a country on a real world map. The adjacency between two instances is implicitly represented with the shared boundary between the two corresponding countries. Non adjacent instances are instead represented as countries separated by oceans, lakes, or other countries. Usual migration flows between two instances can traverse any shared segment of the boundary between the corresponding countries. They are realized with bubbles that grow on the boundary with the color of the instance losing clients, and then move into the receiving instance assuming its color. Unusual migrations are simply represented with overlaid bridges temporarily connecting non adjacent countries. They are visually implemented as arrows pointing to the instance that receives the flow, traversed by bubbles with size proportional to the amount of flow.

One might object that only *planar* graphs can be represented if 1. vertices are planar regions with disjoint interiors, 2. vertices are adjacent in the graph only if they share a point in the map, and 3. *no four regions meet at a point*. However, if we ignore the third condition we can represent a much wider class of migration graphs. These are called *planar map graphs* in [CGP98] and can contain up to $27n$ maximal cliques, where n is the number of vertices.

Fig. 5.2 shows two snapshots from a prototype implementation of a country map. Fig. 5.2(a) shows the map at a certain instant. The instances are colored based on the chart in Fig. 5.1. The green, light blue, orange, red, and yellow countries from a clique and therefore they share a point. The name servers of a

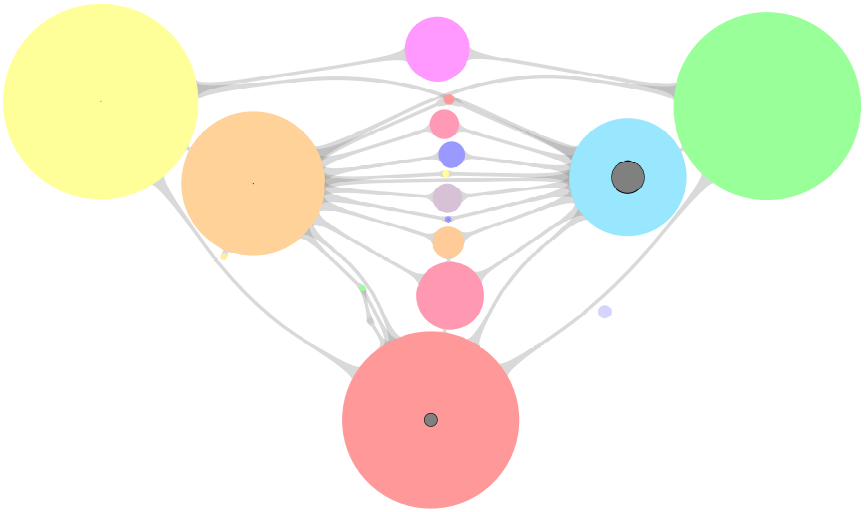


(a)

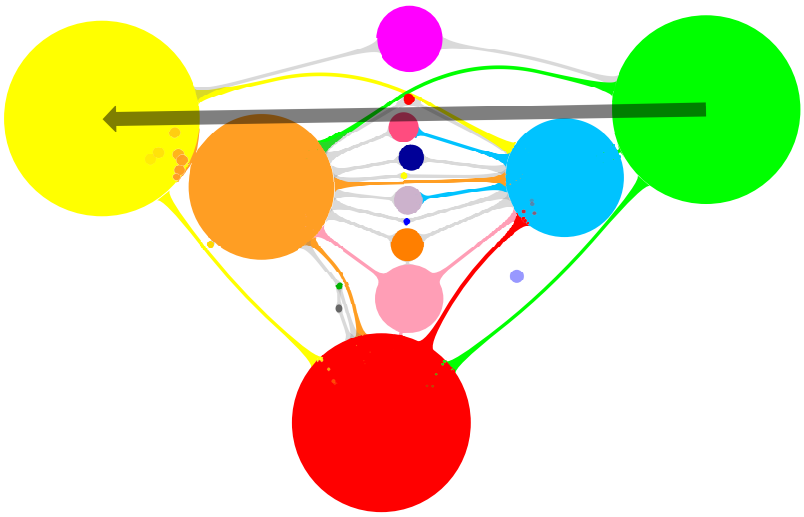


(b)

Figure 5.2: Country map implementation. Each instance is represented as a country, with an area proportional to its relative weight. The adjacency between two instances in the migration graph is implicitly represented with the shared boundary between the two corresponding countries. The ocean separates the instance at the top right corner from all the others. (a) The map at a certain instant. Note how for each country the portion of clients belonging to a specific ISP (if any) is represented with grey circles. (b) A snapshot of an animation of the same map, where usual and unusual client migrations are visible.



(a)



(b)

Figure 5.3: Octopus map implementation. Each instance is represented as a circle. The adjacency between two instances in the migration graph is represented as a "tentacle" connecting the corresponding circles. (a) The map at a certain instant. For each circle, the portion of clients belonging to a certain ISP (if any) is represented as a concentric grey circle. (b) A snapshot of an animation of the same octopus, where usual and unusual client migrations are visible.

well known ISP are also shown as circles of appropriate size, distributed among different instances. Fig. 5.2(b) shows a step of the animation. Usual flow is rendered with bubbles traversing the borders of adjacent countries. Unusual flow is instead represented with an arrow connecting non-adjacent countries.

Octopus Map

An *octopus map* is an abstract visualization where each instance is represented as a circle with an identifying color, while each adjacency between two instances is represented as a "tentacle" of neutral color connecting the corresponding circles. Usual migration flows between adjacent countries are represented with two visual effects: 1. the tentacle traversed by the flow changes color and size, reflecting respectively the instance that releases clients and the amount of clients flowing to the receiving instance; 2. the flow itself is represented with bubbles pouring into the receiving instance from the tentacle, starting with the color of the instance losing clients and progressively assuming the color of the receiving instance. Unusual migrations are represented again with temporarily overlaid bridges between non adjacent circles, realized as arrows of appropriate size pointing to the receiving instance.

Fig. 5.3 shows two snapshots from a prototype implementation of an octopus map. Fig. 5.3(a) shows the map at a certain instant. The name servers of a well known ISP are also shown as circles of appropriate size, distributed among different instances. Note that we used the same colors in Fig. 5.2, while the underlying migration graph is different and specifically not planar: an edge crossing can be identified right under the topmost instance. Fig. 5.3(b) shows a step of the animation with usual and unusual client flows, respectively represented with bubbles pouring into instance circles and arrows pointing to the receiving instance.

5.4 User Feedback

This section describes the impact of our visualization framework from the perspective of different types of potential users. We also give a comparison between the two visualization approaches presented in Section 5.3, based on the feedback collected during the design and implementation phases.

We identify two types of potential users: 1. all the RIPE NCC employees directly or indirectly involved with the management and improvement of K-root, and 2. the vast audience of ISP operators that rely on K-root as one of the foundations for their DNS services. About users of type 2 we observe that

the possibility to visualize data specific of a certain ISP is currently inhibited by the privacy policy adopted by the RIPE NCC. That is motivated by the strict confidentiality of all the query logs collected by K-root instances. The policy can be openly discussed and expanded (e.g. introducing anonymization rules) once the large community of ISPs and network operators shows interest for the service. The remainder of this section focuses on users of type 1.

The cooperation with the RIPE NCC played a major role throughout the creation of our visualization framework. Fifteen staff members were periodically involved in testing and evaluation. Four of them are personally responsible for the maintenance and development of K-root and also participated in important discussions on requirements and qualities of the visualization. With their help we managed to focus on the requirements while looking for a suitable metaphor. As an example, during our initial interactions the users gave a negative evaluation of a first version of the migration graph where both usual and unusual migration patterns were represented using country adjacencies. This allowed to devise the current version of the graph.

We collected feedback during plenary meetings, presentations, and informal discussions. All the users were presented with snapshots and example videos of the two implemented approaches and asked to compare them with respect to the requirements discussed in Section 5.3. In particular, once a stable prototype of the two visualization metaphors was ready, the users were asked to provide at least one preference for each requirement, together with a brief motivation. The results of the comparison are presented and explained in Table 5.1. Note that the second requirement in the table (*Topology of the migration graph*) is a minor concern, because the topology is a needed input of the framework, rather than an expected output.

The comparison highlights a slight preference towards the octopus map implementation. The motivations expressed by different users to justify their preference, together with comments gathered during the early evaluation phases, are valuable because they allow to determine advantages and disadvantages of the two implementations. We present a brief list of contributions. Both visualizations are appreciated for the new insight they provide on existing data and in particular for the detection of usual and unusual migrations, which did not emerge from previous visualizations (see e.g. the one presented in Section 5.3). That gives a better representation of the dynamics of the system, and can help both as a validation tool for load distribution and as a system for anomaly detection. The circular shapes of instances in an octopus map are generally more readable than the complex shapes in a country map, and the corresponding area can be estimated more precisely. The octopus map conveys information

	Country map	Octopus map
Usual migration patterns		✓
Topology of the migration graph	✓	
Unusual migration patterns	✓	✓
Workload of each instance		✓
Status of the service offered to an ISP	✓	✓

Table 5.1: Results of the comparison between the two available implementations for our visualization framework. For each requirement, users were requested to choose at least one between the two alternatives. In the above table we marked every requirement-visualization pair with a total preference equal or greater than $\frac{2}{3}$, i.e. at least 10 votes out of 15.

on usual flows of clients in a more explicit and static way, by means of tentacles with appropriate color and width. On the other hand the usual flow of country maps, although generally considered more appealing, only relies on the size of flowing bubbles to give an indication on the amount of flow. The representation of instance adjacency in the country map is preferred because it displays input information in a simple way, without using overwhelming graphical elements. The second is instead less preferred because tentacles may intersect each or present different lengths, resulting sometimes confusing to the users. Unusual migrations are basically identical in the two approaches, so no preference applies. The adoption of impact graphics (i.e. arrows overlaid onto the map) is particularly appreciated because it helps operators to spot anomalies at a glance, distinguishing them from usual migration patterns, even when they are not paying attention to the overall animation.

Furthermore, the two approaches present differences that are relevant from a more theoretical point of view. First of all, the total area needed by a country map is usually much less than the one needed for an octopus map, which positively affects the readability of the visualization. On the other hand, octopus maps can represent any kind of graph, although dense non-planar graphs can result in poor and confusing visualizations.

The latest version of our prototype runs on a dedicated display in the Global Information Infrastructure department at the RIPE NCC. The animation con-

stantly runs as a background monitoring tool for the operators of K-root. Unusual patterns catch the attention of the operators more easily, because of the way they are visualized, and therefore motivate deeper analysis and appropriate countermeasures when needed. All the other features (usual migrations, size of instances, etc) are instead always visible, so that operators can appreciate an overview of the system at any time and get inspiration for long-term improvements.

On a related note, operators find it particularly interesting to use the system in conjunction with BGPlay [CDM+05], an inter-domain routing diagnostic tool that we describe extensively in Chapter 6. Once an unusual migration is spotted, BGPlay can be used to check if there is a correlation with some routing change or if the event needs a more thorough analysis.

5.5 Algorithms

The algorithms that support our two visualization approaches take as input a migration graph $G = (V, E)$ and a sequence of time instants $t_1, \ldots, t_k$, where t_1, t_k is the time interval of interest and $t_2, \ldots, t_{k-1}$ depend on the adopted sampling unit. We assume that G is connected. If not, the algorithm is repeated independently on each connected component and the results are combined at the end. Both algorithms construct an animation describing the behavior of the clients in the sequence of time instants $t_1, \ldots, t_k$. We denote by $c_t(v)$ the number of clients whose last request of service before time t is received by the instance v. Given a time interval t', t'', the number of *migrants* associated with u, v at t', t'', denoted $m_{t',t''}(u, v)$, is the number of distinct clients that migrate from u to v during t', t''. We denote the *flow* between u and v as $f_{t',t''}(u, v) = \max(0, m_{t',t''}(u, v) - m_{t',t''}(v, u))$.

Given the features of our two implementations, the respective algorithms are quite different. However they share a subdivision in two main phases: the *preprocessing*, which is computed when the system starts, and the *animation*, that is repeated for each t_i. In the following subsections we explain the two algorithms in detail.

Country Map Algorithm

The preprocessing for country maps is composed of three steps:

1. Check if G is a map graph. If that is the case construct its *backbone*, i.e. a planar graph obtained from G by substituting some of its cliques

with stars. Otherwise, remove edges until G is a map graph. Compute a planar topology for the backbone.

2. Find a straight-line drawing of the backbone preserving its planar topology, such that each vertex v has a surrounding "free area" roughly proportional to the average of the clients that it serves in any $t \in t_1, \ldots, t_k$.

3. Construct the *skeleton*, i.e. a constrained Delaunay triangulation of the drawing found in the previous step. The skeleton will be used as the underlying graph during the entire animation.

The animation is performed for each interval t_i, t_{i+1} and is composed of two steps:

4. Draw the skeleton: construct a planar straight-line drawing of the skeleton preserving its topology, such that for each vertex v its incident faces can be split to determine an area surrounding v roughly proportional to $c_{t_{i+1}}(v)$.

5. Draw the map: construct a drawing of the country map at time t_{i+1}, based on the drawing of the skeleton, and compute the animation from t_i to t_{i+1}.

In Step 1 we check if G is a map graph. If that is the case, we construct a planar embedded backbone. The backbone is obtained from G by removing the edges of a suitable set of cliques and substituting the edges of each of such cliques with a star connecting a new vertex to the vertices of the clique. More formally, let $v_1, \ldots, v_k \in V$ be the vertices of a selected clique. We replace the edges in the clique with a new vertex c and edges $(v_1, c), \ldots, (v_k, c)$. An example is presented in Fig. 5.4(a) and Fig. 5.4(b), using the map graph of Fig. 5.2(b). In [Tho98] it is shown that testing if a graph is a map graph can be done in polynomial time. However, in [CGP06] it is argued that the exponent of the polynomial bounding the running time from above is about 120. Therefore we use a much simpler heuristic that works as follows. We first check if G is planar. If yes, we are done. Otherwise, we look for a maximal clique in G with the algorithm in [BK73], that is known to be efficient in practice. Then, we replace the clique with a star and perform again the planarity testing. This is repeated until either the obtained graph is planar or until no clique is found. If we are not able to find a backbone for G, then we remove the edge (u, v) with the smallest number of migrations in the given time interval, i.e. such that $\sum_{i=1}^{k-1} f_{t_i,t_{i+1}}(u, v) + f_{t_i,t_{i+1}}(v, u)$ is minimized, and repeat the process.

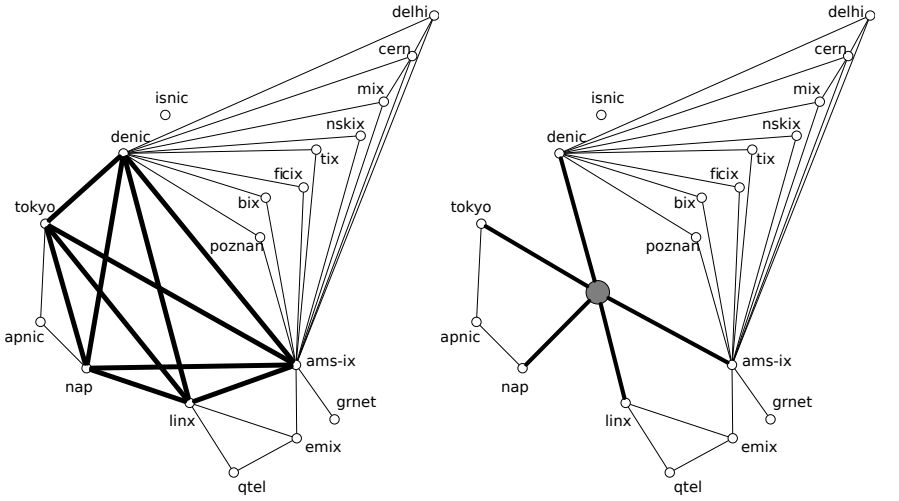


Figure 5.4: (a) Migration graph for K-root. A clique of size 5 is highlighted with thick edges. (b) Backbone used in Fig. 5.2(b), obtained replacing the clique with a star centered at the grey vertex.

The removed edges correspond to migration patterns that we can consider less interesting. It is also possible to involve RIPE NCC experts in this process, identifying and discarding less interesting migration patterns with their help.

Step 2 is devoted to find a straight-line drawing of the backbone, such that each vertex has a surrounding free area that is roughly proportional to the average area it will have during the animation. To perform this step we use a spring embedder (see Section 2.2) that preserves the given planar topology (see, e.g., [DLR11]). Charges and spring lengths are assigned so that each vertex is ideally surrounded by the desired free area. In particular, each vertex v has a positive charge $w(v)$ equal to the average number of clients in $t_1, \ldots, t_k$ and each edge (u, v) is a spring with preferred length equal to the sum of the radii of two circles whose areas are respectively equal to $w(u)$ and $w(v)$.

Step 3 adds an additional set of edges E' to the drawing of the backbone, transforming it into a maximally triangulated planar drawing. E' is needed to easily morph the geographical map later in Step 5. All edges in the subset $A =$

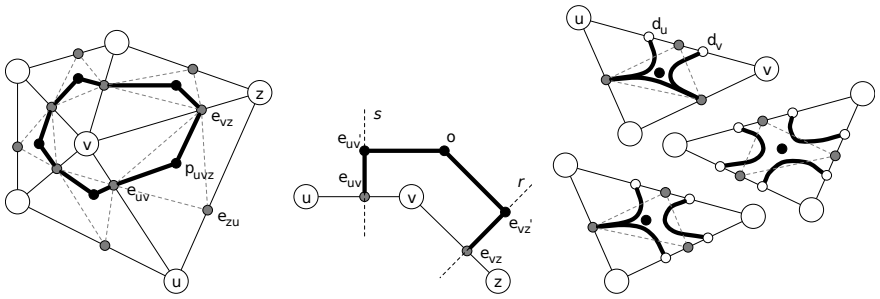
$E' \setminus E$ are marked as *additional*. We use a constrained Delaunay triangulation, in order to maximize the angles between adjacent edges in the resulting graph. This is useful to give more degrees of freedom to the spring embedder used in Step 4.

In Step 4 the layout of the skeleton is modified to make it suitable for the construction of the map at any instant $t \in t_1, \ldots, t_k$. We use a spring embedder in which charges and preferred spring lengths change over time (see, e.g., [EHK+04]). The initial setting is similar to the one explained for Step 2: each vertex v has a positive charge $w(v)$ that is equal to $c_t(v)$, while each edge (u, v) is a spring with preferred length equal to $\frac{\sqrt{w(u)}+\sqrt{w(v)}}{\sqrt{\pi}}$. The layout evolves with an additional constraint on the convexity of the external face. Consider the angle $\widehat{uvz}$ that is spanned in the external face by each triplet of vertices u, v, z that are consecutive on the convex hull. The condition $\widehat{uvz} > \pi$ is ensured. Moreover, positive charges (vertices) and springs lengths (edges) are constantly updated to increase the precision of the map. Each triangle $\Delta(v_1, v_2, v_3)$ with area denoted by $A(\Delta(v_1, v_2, v_3))$ is split such that each of its vertices v_i is assigned an area denoted by $A(\Delta(v_1, v_2, v_3), v_i) = A(\Delta(v_1, v_2, v_3))\frac{c_t(v_i)}{c_t(v_1)+c_t(v_2)+c_t(v_3)}$. Hence, given the set of triangles F_v with a common vertex v, the positive charge of v is regularly updated as $w(v)' = \frac{\alpha}{2\pi}\frac{c_t(v)^2}{\sum_{i\in F_v} A(i,v)}$, where α is the angle spanned by F_v (which is smaller than 2π only for the vertices of the external face). Spring lengths are updated accordingly with $\frac{\sqrt{w(u)'}+\sqrt{w(v)'}}{\sqrt{\pi}}$. This can be seen as a simple implementation of a control system that is periodically updated based on feedback in order to minimize the error on the output, i.e. the area surrounding each vertex.

In Step 5 the map is computed, based on the skeleton. Each edge (u, v) is split at a point e_{uv} such that $\overline{ue_{uv}}/c_t(u) = \overline{e_{uv}v}/c_t(v)$. Then, for each triangle $\Delta(u, v, z)$ a point p_{uvz} is found such that the polygons $(u, e_{uv}, p_{uvz}, e_{zu})$, $(v, e_{vz}, p_{uvz}, e_{uv})$ and $(z, e_{zu}, p_{uvz}, e_{vz})$ have areas respectively proportional to $c_t(u)$, $c_t(v)$ and $c_t(z)$. It is easy to prove that p_{uvz} always lies inside the triangle $\Delta(e_{uv}, e_{vz}, e_{zu})$.

For each vertex v that is not on the convex hull, consider the related set of triangles $F_v = \Delta(u_1, v, u_2), \Delta(u_2, v, u_3), \ldots, \Delta(u_{last}, v, u_1)$ surrounding v in clockwise order. The country border for v is the closed polygon $(e_{u_1v}, p_{u_1vu_2}, e_{u_2v}, p_{u_2vu_3}, \ldots, e_{u_{last}v}, p_{u_{last}vu_1})$. See Fig. 5.5(a) for details.

Vertices on the convex hull are handled in a different way. Note that for graphs with at least three vertices, each of such vertices v has two neighbors u and z on the convex hull. We denote the set of triangles surrounding v as $F_v =$



(a) (b) (c)

Figure 5.5: (a) Construction of the country border for a vertex that is not on the convex hull. Each white circle represents a vertex of the skeleton. For each edge (u, v), a small grey circle represents the point e_{uv}. For each triangle $\Delta(u, v, z)$, a small black circle represents the point p_{uvz}. (b) Construction of the country border for a vertex on the convex hull. (c) Three possible cases of construction of the country border with additional edges. For each additional edge (u, v), two small white circles represent the points d_u and d_v.

$\Delta(z, v, u_1), \Delta(u_1, v, u_2), \ldots, \Delta(u_{last}, v, u)$. The angle $\widehat{uvz}$ that is spanned in the external face is always greater than π, as explained in Step 4. As a consequence, v can get an arbitrary area on the external face that is only bounded by the line s orthogonal to (u, v) passing through e_{uv} and the line r orthogonal to (v, z) passing through e_{vz}. Given the area value $R = c_t(v) - \sum_{i \in F_v} A(i, v)$, we build the polygon $(v, e_{uv}, e'_{uv}, o, e'_{vz}, e_{vz})$ whose area is R, where e'_{uv} lies on line s, e'_{vz} lies on line r and o lies on the external face. Hence, the country border for v is the closed polygon $(e_{vz}, p_{zvu_1}, e_{u_1v}, p_{u_1vu_2}, \ldots, e_{u_{last}v}, p_{u_{last}vu}, e_{uv}, e'_{uv}, o, e'_{vz})$. See Fig. 5.5(b) for an illustration. Finally, connected graphs with less than 3 vertices are easily converted into maps assigning circle-like country borders to each vertex.

Once all the country borders have been computed, the animation is performed. The country map evolves from its previous state with a linear morphing preserving adjacencies at any time. Usual migrations between countries are represented as bubbles traversing the border at randomly chosen points. Unusual migrations are represented as bridges connecting two countries, with bubbles traversing them. The size of bubbles and bridges reflects the amount of clients flowing from one country to another.

Apart from the main algorithm described above, a number of expedients are implemented to obtain a map that looks better and fully represents the underlying data. First, country borders are represented with Bézier curves where possible. This helps to give a natural look to the map. Furthermore, at the end of Step 3, each vertex v in the skeleton that represents an instance and has degree $\delta(v)$ greater than a threshold T_δ is replaced with a path of $m = \lceil \frac{\delta(v)}{T_\delta} \rceil$ consecutive vertices. Each of them is assigned $\frac{c_t(v)}{m}$ clients and retains a fraction of the original adjacencies, with degree lower than T_δ. This helps find better layouts for the skeleton graph in Step 4. The country border for m is computed as the symmetric difference between the borders of its vertices. Finally, edges added in Step 3 and marked as *additional* are later handled in a different way. In particular, the spring embedder used in Step 4 assigns a fixed additional length D to springs representing additional edges. During the construction of the map (Step 5), two points d_u and d_v are found on each additional edge (u, v) together with e_{uv}, such that $\overline{ud_u}/c_t(u) = \overline{d_v v}/c_t(v)$ and $\overline{ud_u} + \overline{d_v v} + D = \overline{uv}$. Then the construction of the border is slightly different with respect to the one explained in Step 5. For each edge (u, v) marked as additional, the two vertices u and v respectively choose d_u and d_v as boundary points, instead of e_{uv}. In this way countries that are not adjacent in the graph do not share boundary points in the map. Note that for each triangle $\Delta(u, v, z)$ the point p_{uvz} is still shared by country borders for vertices u, v and z. This inconsistency is removed in practice using Bézier curves. See Fig. 5.5(c) for an illustration.

Octopus Map Algorithm

The algorithm that generates octopus maps is significantly simpler than the one for country maps. The preprocessing only requires one step:

1. Compute a topology for G such that minimizes crossings between its edges. Find a straight-line drawing for G respecting such topology.

The animation is performed for each interval t_i, t_{i+1} and is composed of three steps:

2. Compute the minimum scaling factor for the drawing of G that allows for an initial drawing of the octopus map, composed of circles and tentacles. Avoid unnecessary intersections between shapes. Draw the *octopus*.

3. Find a new drawing for the octopus that minimizes the total amount of needed area without introducing additional intersections between shapes.

4. Draw the map: construct a drawing of the octopus map at time t_{i+1} and compute the animation from t_i to t_{i+1}.

In Step 1 a straight-line drawing is computed for the migration graph G, such that the number of crossings between its edges is minimized. We impose such condition to improve the readability of the octopus maps derived from the drawing. The crossing minimization problem is known to be NP-hard [GJ83, MNKF90]; however, heuristics exist to compute an approximate result in polynomial time (see [GM03] for reference). Moreover, given that Step 1 is only computed once for every migration graph, we can even assume to use an exact algorithm if the size of the graph is limited.

Step 2 is devoted to compute the minimum scaling factor for the drawing of G, such that all the graphical elements of the corresponding octopus map can be drawn without unnecessary overlap. More formally, the operation is performed such that 1. for each vertex v it is possible to draw a circle C_v centered at v with area equal to $c_{t_{i+1}}(v)$ without intersecting any other circle, and 2. for each edge (u, v) it is possible to draw a rectangle surrounding (u, v) with width equal to $2 * \sqrt{\frac{\max(f_{t_i,t_{i+1}}(u,v), f_{t_i,t_{i+1}}(v,u))}{\pi}}$ that only intersects C_u, C_v and all the rectangles surrounding edges that cross (u, v) in the topology, if any. The algorithm to calculate the scaling factor is fairly easy and consists in computing an intersection test on each pair of shapes not supposed to intersect each other, increasing the scaling factor of the migration graph until there is no overlap. In its simplest form, Step 2 has a quadratical complexity with respect to the number of shapes. However, optimization is possible to some extent, e.g. testing each shape for intersection with its sole neighboring shapes. After this operation the initial drawing for the octopus map can be computed: 1. each vertex is replaced by a circle with area equal to the number of clients that it serves, and 2. each edge is replaced by a link with width proportional to the amount of flow between its two adjacent vertices.

In Step 3 the layout of the octopus is modified in order to reduce the total area needed to draw it. The goal is to fit the final version of the octopus inside a predefined rectangle of fixed size, which represents the screen where the animation is projected. This is achieved with a constrained spring embedder that not only preserves the given planar topology (see, e.g., the already cited [DLR11]), but also does not introduce additional intersections between shapes of the octopus. In order to shrink the input octopus as desired, each edge is considered as a spring with preferred length equal to the sum of the radii of C_u and C_v, while vertices are not assigned any charge. Instead, during each iteration of the spring embedder all the coordinates in the current drawing

are subject to a scaling aimed at fitting the whole drawing into a rectangle of fixed aspect ratio. The additional constraint for the spring embedder regarding intersection avoidance between shapes is ensured at the end of each iteration. The implementation follows an approach similar to the one explained in Step 2, where the coordinate transformation of each shape is limited until it does not introduce any new intersection with other shapes.

In Step 4 the new octopus map is finally drawn and the animation is performed. The octopus map evolves from its previous state with a linear morphing preserving adjacencies at any time. Instance circles change radius and position based on computed coordinates. Tentacles between circles vary in position, width and color based on the new coordinates and on the actual flow traversing them. Usual migrations between instances are represented as bubbles pouring from the tentacles into the circles. Unusual migrations are represented with arrows connecting pairs of circles. The size of bubbles and arrows reflects the amount of clients flowing from one circle to another.

A number of additional details are implemented also for octopus maps, to ensure their clarity and simplicity. First of all, Step 3 does not guarantee that the final drawing will fit the predefined screen size. Therefore, when this is not the case, the whole drawing is scaled down accordingly. The new scaling factor is always available together with the new octopus map, so that the user can be informed of the necessary change: e.g. visualizing a legend in the bottom right corner with a circle whose size changes accordingly as a reference. On a related note some of the tentacles in the octopus map are not straight, but rather present bend points realized as Bézier curve. This is aimed at allowing more compact and flexible drawings of the map. The refinement is easily implemented adding dummy vertices to the drawing, represented as circles whose diameter fits the width of the corresponding tentacle.

5.6 Technical Details

The implementation of Visual-K predictably leads to a number of questions and challenges, given the technical features of the system under analysis. First of all K-root is reached by high volumes of queries, in the range between ten and twenty thousands per second. Original queries are recorded at each location and regularly sent to a central repository at the RIPE NCC, where they are permanently stored. This process was recently improved to make use of a Hadoop [The05] cluster deployed at the RIPE NCC for distributed storage and analysis of scientific data and network measurements. To give an idea

of the volume of data, 5 minutes of query logs correspond to approximately 300 megabytes, with a compression factor oscillating between two and three. The remainder of this section presents our implementation, together with the solutions we devised to approach a complex system like K-root.

We implemented our visualization framework as a Web application. The prototype is currently maintained by the RIPE NCC and available at the address `http://k.root-servers.org/visualk/`, featuring most of the requirements described in Section 5.3. More specifically, users can monitor the status and evolution of K-root in near real-time, while the possibility to perform offline analysis or to focus on a subset of clients of K-root (e.g. those of a specific ISP) is not yet available. The workload of each instance is measured in terms of queries per second. The only metaphor available for the online prototype is the octopus map, given the preference expressed by K-root operators and detailed in Section 5.4. Some example videos of both metaphors are also available online [DBSN12].

The front-end of Visual-K is written in JavaScript. It has the main responsibility of visualizing the map of K-root and animating it from time to time with the new data sent by the server. Apart from the map itself, the Web application also shows some control information (e.g. the timestamp in which the map was last updated). The actual visualization has been realized with a cross-browser JavaScript library for vector graphics called Raphaël, based on the Scalable Vector Graphics format. This implies that images and snapshots can be zoomed and exported without loss of quality; see e.g. Figg. 5.2, 5.3.

The Java server contains the core of the application logic. An associative map is kept in memory to store the current state of each client, including the instance that answered its last query. The real-time implementation is divided into four main steps. 1. At regular time intervals MapReduce jobs are sent to the Hadoop cluster. For each client of K-root we retrieve the total number of queries and the instance that received its last query. 2. Before updating the associative map with the retrieved data, we perform a comparison to detect usual and unusual migrations. 3. For each instance the total number of queries received since the last MapReduce job is computed, along with migrations. 4. The layout algorithm produces a new map with the computed data and sends it via a messaging service, allowing Web clients to receive it asynchronously. Note that only one layout is computed for each step of the animation, independently on the size and aspect ratio of the screen of any connected client. This avoids the need for expensive computation on every client. The layout itself is sent as a high-level, vectorial description of the drawing. Web clients can independently adapt it to the screen and, most importantly,

render it with any library or graphical tool.

The communication between server and client is implemented with asynchronous messages, in order to guarantee low coupling between them. We used Apache ActiveMQ as a message broker, which easily allows the client to register as a listener and asynchronously receive updates as soon as they are published by the server.

The performance of the system has been constantly tested and improved during the implementation. The computation of the geographic map of course plays a crucial role. We ran stress tests on a laptop with a 2.4 GHz Intel Core 2 Duo processor and 4 GB of RAM. We noticed that in our most advanced implementation octopus maps are generally computed in about one second per iteration, which makes them suitable for a near real-time tool. On the other hand, earlier code for country maps takes more time for the computation of each map (between 10 and 15 seconds). We consider also the latter to be an acceptable result, given that our framework needs a time interval of comparable length to perform a smooth animation to morph a map into the next one. Both results are of course subject to improvement on more powerful hardware.

An additional note is required on data storing and access. As explained above, all the query logs collected by K-root instances are downloaded and stored on the Hadoop cluster. The average delay between the instant when a query is received by an instance and the time when the corresponding data is actually available on the cluster varies largely. It depends on many factors, e.g. the latency in the communication between instances and central repository or the delay in the storage time depending on the load of the clusters. Although query logs are usually available about five minutes after being generated at various locations, every now and then the system experiences huge delays imposed by distant nodes (e.g. Tokyo and Delhi). Therefore our prototype implementation applies a safe delay interval of one hour, to wait for the data from all instances to be correctly sent and stored. Note that this delay is completely independent from our framework, and can be adjusted at any time in case the storage time is significantly improved.

The default time period between two layout updates in the prototype implementation is set to five minutes. The average time needed to run each MapReduce job is heavily influenced by the current load on the Hadoop cluster. Under normal circumstances it usually takes less than one minute to process all the query logs spanning a time interval of five minutes. That means that theoretically the system could be improved allowing for a smaller refresh rate. However, as explained in Section 5.3, the current rate is already acceptable for common operational needs.

5.7 Related Work

The problem of using geographical maps to visualize non-geographical information has been extensively studied. In this section we provide a brief overview of the literature, focusing on similarities and differences with our approach.

A methodological reference is provided by the cognitive study in [FS05]. It identifies four semantic primitives to be used when representing data with a geographical metaphor. *Boundaries* are discontinuities in the information space that can be represented with borders. *Aggregates* are homogeneous zones that preferably represent homogeneous entity types. We use aggregates and boundaries to group clients using the same instance and to separate such groups, respectively. *Loci* are information items that have a meaningful location in the information space. We link together or put side-by-side instances that are expected to share clients. Finally, *trajectories* are semantic relationships between information entities at different locations that can be shown with paths or routes. We exploit different types of trajectories to represent migrations.

There are at least two systems whose features are similar to the ones in our framework: GMap and BGPlay Island. GMap [MKH11] visualizes clustered graphs by means of geographical maps. After determining the layout of the graph with a force directed approach, clusters of nodes are detected according to their relative distance. A cluster is represented with one or more geographical regions. GMap produces maps that look very similar to our maps. However, its target is quite different from ours: 1. if two vertices are connected by an edge it is not guaranteed that they have a common boundary, 2. if two vertices have a common boundary is not guaranteed that they are connected by an edge, and 3. GMap is not meant to visualize maps whose borders evolve over time. Using the terminology of [FS05] we can say that GMap privileges the *aggregate* primitive. BGPlay Island [CDM$^+$06] extends the widely used BGPlay routing visualization system [CDM$^+$05] and uses a topographic metaphor to show hierarchies of ISPs. However BGPlay Island uses the metaphor of a terrain map rather a political map, and the most stressed primitive is the *locus*.

Other related literature is the one on *cartograms*. Area cartograms are drawings derived from standard geographical maps, where each country is deformed so that its area is proportional to a variable specific of that country, e.g. its population. The deformation process should preserve the original shape as much as possible. The idea behind cartograms is very close to our map metaphor, which in fact can be seen as an area cartogram derived from an imaginary world. Many algorithms for computing area cartograms are available in the literature (see, for example, [GN04, IS06, OR00]). However, their

attempt to preserve the original shape is irrelevant in our setting, since our countries do not have a prescribed shape. Also, they have high computational costs, which make them unsuited for a real-time monitoring tool. In [OR00] the latter issue is tackled with an algorithm that can be parallelized, but unfortunately results are exposed to inaccuracy (e.g. overlap between countries). Recent approaches [vKS07, dBMS06, KN07, RMN09, AJSS11, BV10, BRV11] for the computation of area cartograms tend to keep the countries in their original locations but give them a regular shape, like a rectangle or a "T" or and "L". However, the more regular the shapes are, the less graphs can be represented. Further, none of the above results takes into account scenarios that include planar map graphs. Finally, the computed layouts are sometimes hard to read and therefore not suitable for an intuitive visualization.

Voronoi diagrams represent an option for partitioning information spaces into separate regions. In [RT07] the authors introduce an adaptive version of the multiplicatively weighted Voronoi diagram [OBSC00], where each vertex in a graph is assigned a closed region with prescribed area. Similarly to Voronoi diagrams, however, region adjacencies depend on geometric proximity. Therefore the solution is not compatible with the notion of adjacency graph.

In [GHKK10] it is shown that planar graphs can be represented with adjacent convex hexagons. Such shapes could be a valid alternative for our scope. Although it may be possible to modify the proposed algorithm to represent also planar map graphs (e.g. using polygons with more sides and loosing the convexity), the problem of assigning prescribed areas to the shapes remains difficult to address.

A previous attempt at visualizing the activity of Internet services, including K-root, is described in [HFc08]. The authors present a visualization called Influence Map, which renders a compressed representation of geo-spatially distributed Internet data. Sets of clients sending requests to the same instance are located on a real geographical map and a coordinate centroid is computed. Then a circle is displayed, placed at the centroid and composed of wedges that represent the amount, distribution and latency of clients. Their work differs from our approach, in that it is meant to visualize static snapshots of the service, without focusing on the migrations of clients.

5.8 Conclusions and Future Work

We have presented a framework for the visualization of K-root, one of the 13 root name servers in the world. It relies on a map metaphor that uses

animations to show the migration of clients among the instances that compose the root name server.

While in [RFF+08] the authors prove that animations are not generally suitable to convey information on trends in data visualization, they also argue that animated drawings are quite useful to create a visualization that is appealing to the user. At the same time, a real-time monitoring tool necessarily deals with the evolution of the underlying data. In our framework we find a reasonable balance between the two needs, using graphical elements that are independent on the animation. A static snapshot of each step of the animation contains all the information we want to visualize, as Figg. 5.2 and 5.3 clearly show. The animation is only needed to gracefully link two consecutive steps, helping the user to focus on the context.

There are several future research directions that can be undertaken. One would be to deploy our system to other root servers. This is relatively easy from a technical perspective; however, there are drawbacks from the organizational point of view, since logs of queries are strictly confidential and dealing with them requires an adequate agreement. Another interesting possibility would be to apply the same techniques to other Internet services based on anycast. One possible example, mostly interesting nowadays, is the IPv6 6to4 Relay Routing Service, devised to facilitate the transition between IPv4 and IPv6.

Part III

Visualizing Internet Routing

Chapter 6

Designing a Web-based Framework for the Visualization of Inter-domain Routing Events

The project described in this chapter was partially supported by the RIPE NCC. Part of the material in this chapter is based on [Can12]. The latest version of the tool we developed is currently maintained by the RIPE NCC and available online at `http://stat.ripe.net`. Further material is available at [CDBS13].

6.1 Introduction

The Internet is often described as a "network of networks". The task of retrieving, analyzing and visualizing its complex structure has always been a hard challenge for researchers and network operators. However, the availability of datasets containing routing information can give important indications on how the Internet topology works and how it evolves over time. Examples of such data sources include the Routing Information Service and the Route Views Project, as explained in Section 3.2.

At the same time, the rise of sophisticated frameworks for the development of rich Web applications opens a new array of possibilities for the exploration of routing data. Network operators and researchers demand tools to access network datasets directly through Web pages, with the goal of exploring and analyzing specific events and features based on their needs. Cross-browser

compatibility and performance are therefore primary concerns that have to be addressed when designing new tools for network data analysis.

In this chapter we present BGPlay.js, a framework originally conceived for the visualization of BGP routing events. It allows the user to select a specific IP prefix and a time interval of interest, in order to explore the evolution of the routing policies related to that prefix. The interface is based on a routing graph that shows how different ASes reach the target prefix. It also features a timeline that allows the user to focus on specific routing events, while keeping an eye on the entire routing history.

BGPlay.js builds on an already existing tool called BGPlay [CDM+05]. The design and implementation, however, were renovated from the ground up to meet new crucial requirements. In particular, since the early design phases, we opted for cutting edge Web technologies that would allow to build a modern, scalable and reusable framework. We also redesigned part of the interface, adding new intuitive tools while keeping all the power of the original design. The result is a framework that improves on its predecessor and facilitates the development of more advanced network visualization tools.

The remainder of the chapter is organized as follows. Section 6.2 presents various references to the state of the art, including the previously existing version of BGPlay. Section 6.3 presents our new contribution, with a focus on the interface and user interaction. Technical and algorithmic details are discussed in Section 6.5. In Section 6.4 we present an interesting use case for BGPlay.js, related to massive Internet outages caused by censorship over the past few years. Section 6.6 presents our conclusions and ideas for future development.

6.2 Related Work

The main reference project for our work is BGPlay [CDM+05], as already discussed in Section 6.1. BGPlay is a network diagnostic tool that provides a graphical representation of a portion of the AS topology and its evolution over time. More specifically, it combines BGP updates that carry information about a specific IP prefix and shows the AS-paths used from different vantage points to reach the prefix over time.

The interface of BGPlay is presented in Fig. 6.1. The core of the visualization is the animated graph. Each vertex represents an AS and is labeled with the AS number. Red vertices identify ASes that originate the selected IP prefix: there may be more than one (i.e. if the prefix is allocated to different

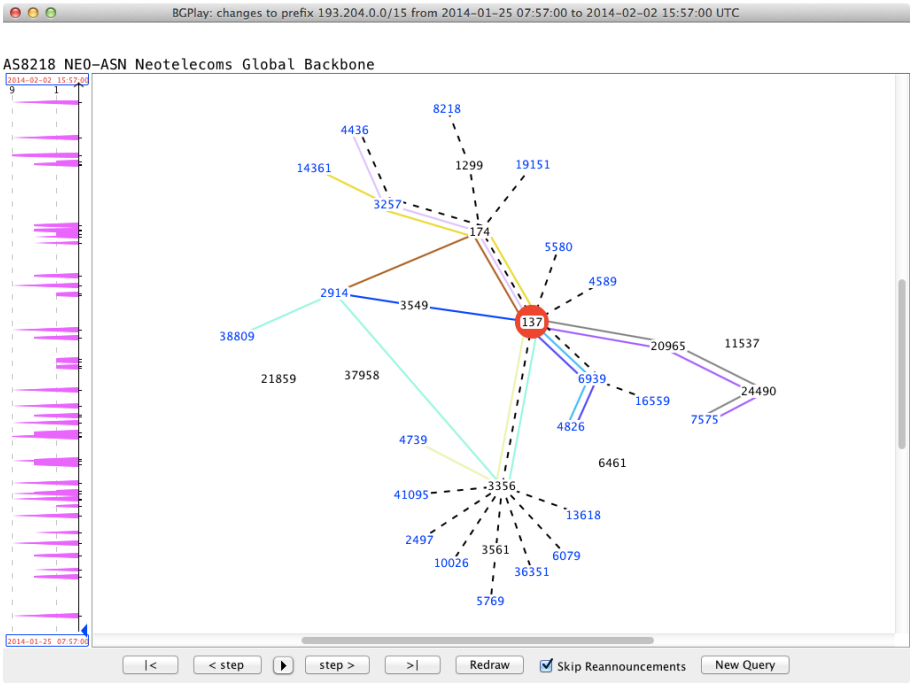


Figure 6.1: BGPlay applet as found on `http://bgplay.routeviews.org/`.

ASes over time, or in case of BGP hijacking). Blue labels identify ASes that contain at least one collector peer, i.e. a source of BGP routing information (see Section 3.2 for details). Each collector peer is assigned a distinctive color. At any time the graph shows all the currently active AS-paths reported by each of the collector peers. AS-paths that do not change during the reference time window are grouped together to form trees, pictured with dashed lines that connect pairs of ASes. All remaining AS-paths are pictured with solid lines using the distinctive colors of the corresponding collector peers. The timeline on the left shows a time series with the aggregate number of BGP events observed within the time interval of interest. High peaks indicate periods of intense activity (e.g. disconnections, routing policy changes, etc). The available user interaction resembles that of a media player. The user can animate the routing graph to see how the reachability changes over a prescribed interval of time.

The reader can refer to [CDM+05] for a detailed description of the interface and the way to use it.

An instance of BGPlay is currently available as a Java applet at the address `http://bgplay.routeviews.org/`. Note that, although heavily used until few years ago, such technology is now outdated and even considered potentially harmful in modern Web browsers. BGPlay was also offered as a service by the RIPE NCC for many years, using data from the Routing Information Service (see Section 3.2). The tool gained attention year after year and became very popular amongst network operators and researchers.

Over the past few years a number of variations of BGPlay were developed to deal with specific use cases. iBGPlay [Piz07] helps network operators monitor the reachability of specific target prefixes, based on BGP updates collected by routers in their ISPs. In other words, the original metaphor of BGPlay is "inverted" to show how a small group of routers inside an ISP is connected to an arbitrary portion of the Internet. BGPlay Island [CDM+06] enriches the visualization metaphor of BGPlay by enclosing nodes within concentric regions that represent the relative importance of the corresponding Autonomous System. In this way the user can immediately perceive how the traffic to reach a specific prefix traverses the hierarchy of ISPs on the Internet. Historical BGPlay [Squ10] is optimized for time intervals spanning entire years. It features a new technique to filter transient BGP updates, allowing the user to focus on long-term routing states. It is therefore a privileged tool for the historical analysis of AS-level connectivity.

The visualization of routing events and topology changes has been explored by other researchers. Cyclops [OCLZ08] is a tool that allows ISPs to verify how their services are perceived from hundreds of vantage points around the world. NetViews [oMCSDNRL08] shows the AS topology on a geographical map. CAIDA (see Section 3.2) regularly updates visualizations of the IPv4 and IPv6 Internet topology [CAI00] where ASes are drawn on a circle with polar coordinates that represent their relative importance. The same research group also worked on Walrus [CAI01], a framework for the visualization of large graphs in three dimensions that can be used to visualize the AS-level topology.

6.3 User Requirements and Interface Design

We started our research activity collecting all the main requirements that emerged over the past few years following the first release of BGPlay. In this phase we also interacted with RIPE NCC representatives that showed interest

in feeding a renewed version of BGPlay with historical data coming from the Routing Information Service. Once the requirements were clear, we focused on the design and implementation of BGPlay.js.

Gathering New Requirements

The new functional requirements for BGPlay.js mostly deal with usability and are influenced by the rise of Web-based technologies. First of all, BGPlay only features an aggregate representation of routing events in the timeline. This is generally sufficient to appreciate trends over time, but very limiting when the attention is shifted to specific events. The sequence of routing events collected within the time interval of interest should therefore be accessible at any time in a more intuitive way. In terms of usability, users should also be able to embed the interactive visualization of specific IP prefixes in arbitrary Web pages, e.g. websites reporting news on network events. Any visualization should be in correspondence with a unique URL, so that users could quickly use it for sharing and indexing purposes. Further, the layout of the animated routing graph should be customizable and the tool should allow to export custom graph layouts for easy reuse.

Non-functional requirements are also heavily influenced by the new technological challenges. First of all, as already mentioned in Section 6.2, the current implementation of BGPlay relies on Java, which represents a limitation in modern Web browsers. Our renovated framework should therefore be a Web application purely based on JavaScript, without the need for third-party plugins. It should be compatible with all major browsers and platforms, including smartphones and tablets. As a consequence, the visualization should adapt seamlessly to different screen resolutions and computing capabilities.

As a last important requirement, the framework should support data abstraction and future development based on different data sources (e.g. traceroute data, as explained in Chapter 7). Fig. 6.2 presents the abstract domain model that we designed when collecting requirements, mostly inspired to graph concepts. A *node* is an entity in the network. Each node may contain zero or more *sources* and/or *targets*. An *event* is originated by a specific source and has the effect of updating the *path* that starts at the node containing the source and reaches the node containing one of the targets in the network. In the case of inter-domain routing nodes are ASes, sources are collector peers, targets are IP prefixes, events are BGP updates, and paths are AS-paths announced in the updates over time.

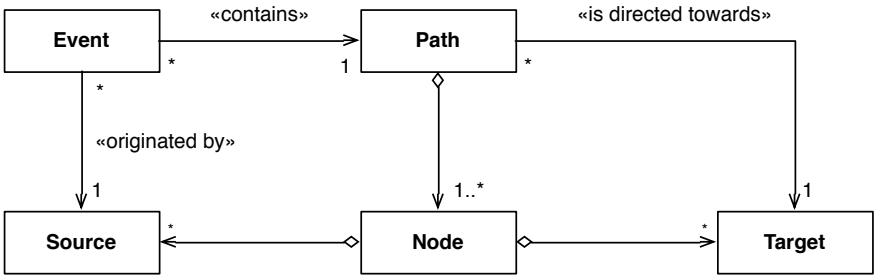


Figure 6.2: Abstract domain model describing the main entities of BGPLAY.JS and their mutual relationships.

Interface and User Interaction

The main interface is presented in Fig. 6.3. It is composed of four main views: the controller, the graph panel, the info panel, and the timeline panel. The interaction with any of the four views causes appropriate updates in the entire visualization. We detail the main functionalities below.

The *controller* is a sliding panel located in the upper right corner. It allows the user to input queries composed of a target and a time interval $\mathcal{T}$. Once the visualization is ready, the controller can be used to animate the graph with the AS-paths available during $\mathcal{T}$. The play, repeat-last, step-back, and step-forward buttons allow for a fine-grained management of the graph animation, resembling a standard media player.

The *graph panel* displays the interactive graph, initially centered and fitted to the window. Given the success of the standard interface of BGPlay, the graph representation of entities in the domain model is substantially equivalent to that explained in Section 6.2. The user can pan and zoom the graph with the mouse. The animation of the graph consists of a sequence of morphing steps. Each step transform the graph by applying the effects of an event involving one or more AS-paths. Differently from BGPlay, the animation of events happening at the same time is simultaneous, instead of following an arbitrary ordering.

The *info panel* is in the upper part of the window. It shows all the available information about any selected network component represented in the graph. In the case of BGP data, the panel shows the owner of any AS in the graph, the sequence of ASes in any AS-path, the IP of any collector peer. It also displays

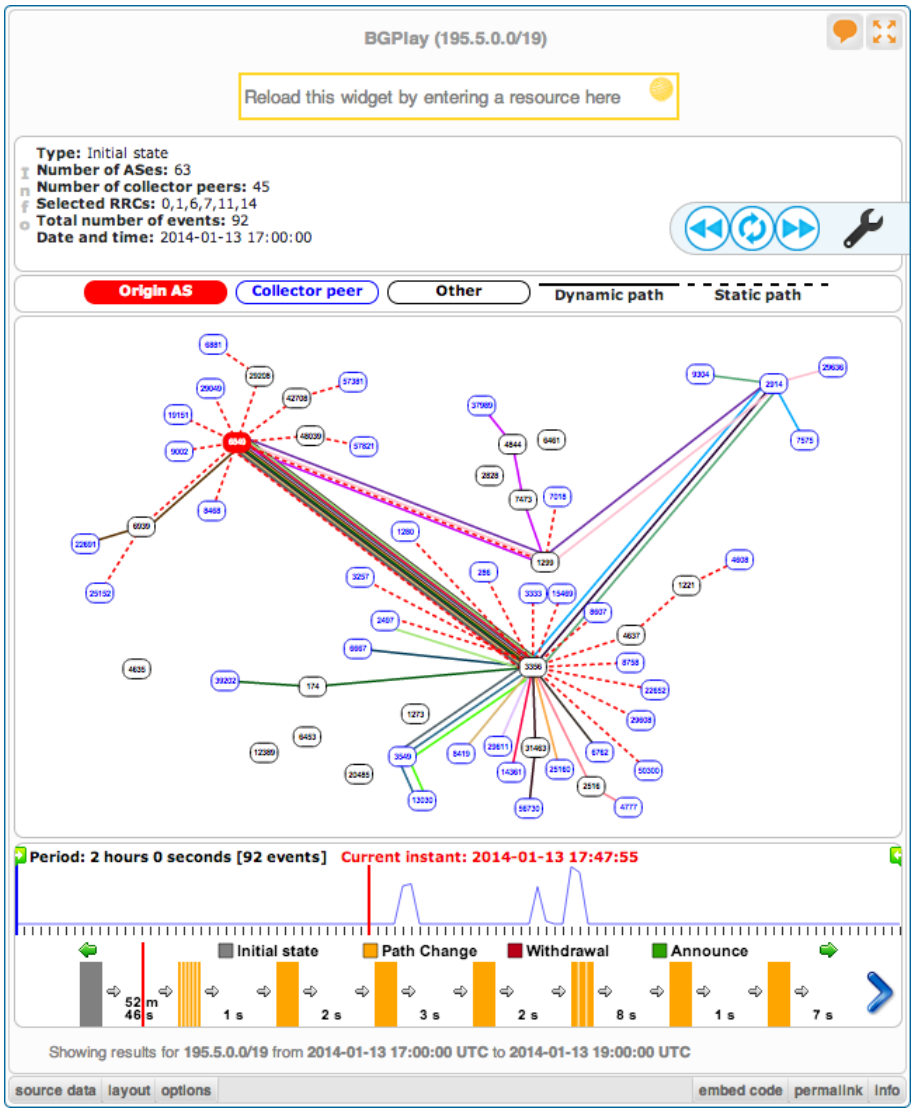


Figure 6.3: Interface of BGPLAY.JS as found at `http://stat.ripe.net`.

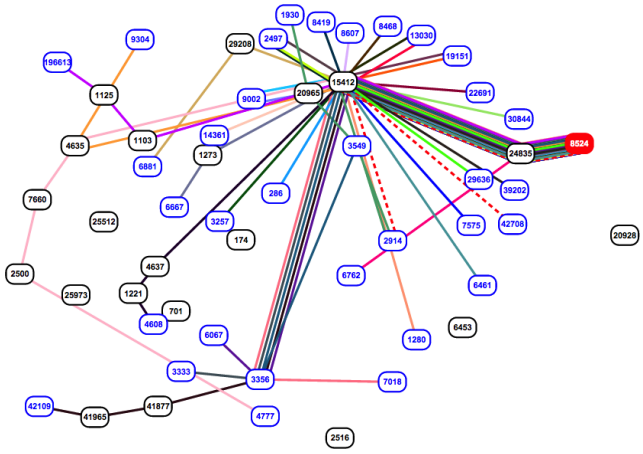
a textual description of the routing event currently visualized.

The *timeline panel* is in the lower part of the window and contains two timelines that allow to accurately navigate the routing information in $\mathcal{T}$. The first timeline, called *control timeline*, is equivalent to the one already implemented in BGPlay and provides a fast overview of the trend in the number of events over time. The second, called *selection timeline*, is instead devoted to the visualization of individual events ordered in time and is designed for fine-grained analysis. Each block in the selection timeline contains a sequence of events happening at the same time, represented with colored rectangles. Different colors are used for different types of routing events: for example the announcement of a new path from a source to a target is green, while the update of an already existing AS-path is yellow. The elapsed time between any two consecutive blocks is reported in the area between them. Both timelines feature a red cursor that points at the current time instant and is continuously updated during the animation. The user can drag the cursors, changing the current instant and updating the graph accordingly. The selection timeline can only show a limited number of events due to its constrained area. In case there are more events, the animation of the graph also affects the visualized portion of the selection timeline, causing the smooth horizontal translation of involved events. The user can scroll horizontally to reveal hidden events. Further, the user can limit the animation to a particularly interesting subinterval within $\mathcal{T}$ by dragging the two green sliders at the top of the control timeline. The sliders on the selection timeline are updated accordingly.

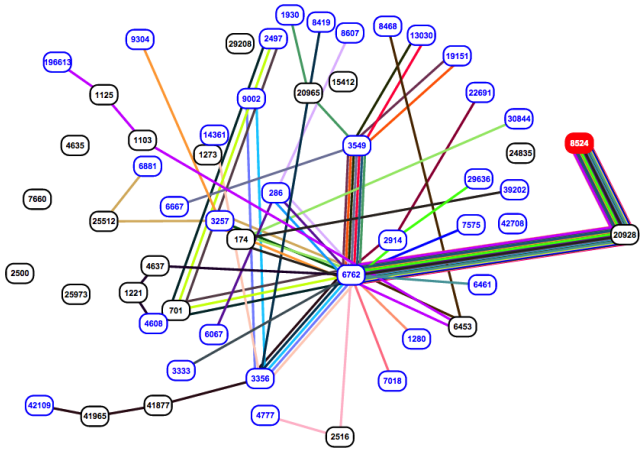
6.4 Visualizing Internet Outages Caused by Censorship

BGPlay.js can be used in a number of different scenarios involving the evolution of BGP routing policies over time. This section presents an interesting use case that we encountered in a recent research project focused on the analysis of country-wide Internet outages.

In 2011 a wave of protests took place in Northern Africa and the Middle East, giving rise to the so called "Arab Spring". In particular, Egypt and Libya were among the first countries to experience violent demonstrations over extended periods of time. In both countries there were reports of partial or complete disconnection from the Internet during the most delicate phases of the protests. This gave us inspiration to start a research project dealing with the dynamics and causes of the disconnection. The reader can refer to Chapter 10 for additional details on the project.



(a)



(b)

Figure 6.4: Two different phases of the BGP reachability of the Egyptian IP prefix 213.181.237.0/24 between 27 and 28 January 2011. (a) The prefix is reachable through AS 24835 (RAYA Telecom Egypt). (b) After the disconnection, the prefix becomes reachable again through AS 20928 (Noor Advanced Technologies).

We made use of different data sources, both related to the control plane and data plane. On the BGP side, we used data from the already mentioned Routing Information Service and Route Views projects. We selected the full set of prefixes announced by Egyptian and Libyan ISPs and downloaded the corresponding routing data for the days in which there were reports of outages. Our analysis lead to discover the exact timing of the disconnection, which put in evidence the fact that the outage was conducted on purpose.

In this context, BGPlay.js helps us visualize the exact dynamics of the disconnection operated at the BGP level for certain targets. Fig. 6.4 presents the evolution of the reachability of the IP prefix 213.181.237.0/24, announced by AS 8524 (American University in Cairo). In Fig. 6.4(a) we see the status before 27 January 2011, with the prefix reachable through AS 24835 (RAYA Telecom Egypt). Fig. 6.4(b) shows how the connectivity was restored on 28 January 2011, using a backup link through AS 6762 (Telecom Italia) and AS 20928 (Noor Advanced Technologies). Further, the exact timing of individual BGP updates is shown, revealing the dynamics of the outage.

6.5 Technical and Algorithmic Details

We made a great effort to design a generic visualization framework, in line with the most appropriate patterns and standards for Web development. All our decisions were the result of critical analysis and intense interaction with the RIPE NCC.

The main architectural style of BGPlay.js is a mixture of two well-known styles, respectively known as *Client-Server* and *Model-View-Controller* (MVC). The first is based on the distribution of tasks between a provider of resources, called server, and one or more service requesters, called clients. The second consists of splitting the application into three interconnected parts, with the goal of clearly separating the internal data structures from the actual representation of the information for the end user. The combination of the two styles has become the de-facto standard for many Web applications. Initially the development was focused on the idea of a "thin client", with most of the business logic confined to the server. However, in recent years the increased power and capabilities of user devices have supported a number of intermediate solutions. In modern Web applications the server only provides raw data structures on demand, while the client is responsible for adding the logic and converting the data into a meaningful and interactive representation.

In BGPlay.js the *model* is the piece of code responsible for retrieving,

indexing, and processing the routing data available on the server. More specifically, it builds an abstract representation of the sequence of routing events that can be efficiently queried at runtime for both random and sequential access. It also keeps track of the current routing status selected by the user and the corresponding timestamp. The latter information is of course dynamic and depends on user interaction: for example, when the user clicks a specific rectangle representing an event on the timeline, the model changes the internal representation to reflect the new routing status to visualize.

Each of the four main components of the interface of BGPlay.js described in Section 6.3 corresponds to one or more *views*. Each of them is responsible for representing at least part of the information managed by the model. For example the graph panel corresponds to a view that is responsible for displaying the routing graph, and it does so by initializing in turn one "node view" for each vertex in the graph and one "path view" for each source-target pair. In our design strategy views can also be considered as *controllers*, because they are responsible for parsing the input of the user and converting it into notifications for other views or for the model.

The communication between different modules in BGPlay.js is handled with a widespread messaging pattern called *Publish-Subscribe*. In this setting *publishers* can send updates to a channel (i.e. a queue identified by a unique label), while *subscribers* can listen to any number of channels and receive updates when they are available. In BGPlay.js this paradigm is used for the communication between objects in the model, or from objects in the model to objects in one or more views. For example, when the timestamp of the event currently visualized changes, the model publishes a message that is received by all the views that need to be updated accordingly. The communication between different views is instead achieved with the *Event Aggregator* pattern, where a dedicated object is responsible for subscribing to all the channels used by different views, receiving their messages, and then triggering updates accordingly in appropriate views.

The implementation of BGPlay.js was completed using two widespread JavaScript frameworks, called Backbone.js and Raphael. The first is a core library with a stable implementation of the MVC pattern, used to give structure to Web applications. The second allows to develop data-driven visualizations without worrying about the underlying implementation of vector graphics in different browsers.

BGPlay.js was extensively tested to verify its functionalities on different browsers and platforms. Its model is accompanied by standard Unit tests to check the correctness of algorithms and business logic. The overall performance

of the framework is acceptable even on mobile platforms with graphs containing hundreds of nodes.

The algorithms implemented in BGPlay.js mostly resemble those of BGPlay. Examples include the grouping of "static" paths (i.e. paths from source to target that never change during the selected interval) into the minimum number of trees and the actual drawing of the graph, implemented as a standard force-directed layout (see Section 2.2 for a general introduction).

As a notable exception, we decided to implement a "tree map" in JavaScript, i.e. a data structure where both the insertion and lookup of an element have logarithmic time complexity, while it is possible to scan the entire set of ordered keys in linear time. We use it in the model to accomodate all the routing events, indexing them with a combination of their timestamp and a unique identifier to distinguish multiple events happening at the same time. The structure helps improve the efficiency of specific user interactions. For example, when the user clicks on the control timeline, the coordinate of the click is converted to a time instant and the tree is queries for the event closest in time to that instant.

6.6 Conclusions and Future Work

We have presented a framework for the visualization of inter-domain routing events. It was designed and implemented with a focus on modern Web technologies. Further, it is very generic and can be reused for more Web application dealing with the visualization of evolving graph topologies.

There are many possible improvements that could further enrich our work. We plan to extend the graphical representation to other events, e.g. the exchange of BGP "state" messages that detail the establishment of sessions between routers. It would be interesting to have a real-time visualization of BGP updates, as opposed to the current historical version. Some of the features of the variants of BGPlay presented in Section 6.2 would easily find room in BGPlay.js (e.g. displaying the ranking of ASes or filtering the events on the timeline to focus on those related to a specific AS). Finally, on the algorithmic side, we plan to improve the drawing of different paths between the same pair of nodes using appropriate techniques (see, e.g., [ABKS10]).

Chapter 7

Dynamic Visualization of Traceroutes at Multiple Abstraction Levels

The research project described in the following sections is based on the framework presented in Chapter 6. In Chapter 11 we report a conference publication based on the outcomes of the project. The reader can refer to [CDBDBS13] for additional media (videos, pictures) that support the description of our work.

7.1 Introduction and State of the Art

The *traceroute* command is one of the most popular computer network diagnostic tools. As explained in Section 3.1, it can be used on computers connected to the Internet to compute the path (route) towards a given IP address, also called *traceroute path*. It is probably the simplest tool to gain some knowledge on the Internet topology. Because of its simplicity and effectiveness, it attracted the interest of several researchers that developed services for the visualization of Internet paths discovered by executing one or more traceroute commands.

Broadly speaking, there are two groups of traceroute visualization systems. The first group includes tools developed for local technical debugging purposes in networks of limited size. Tools in the second group are instead aimed at reconstructing and displaying large portions of the Internet topology. Several examples in the first group visualize a single traceroute on a map, showing the geo-location of traversed routers. A few examples follow. Xtraceroute [Aug03]

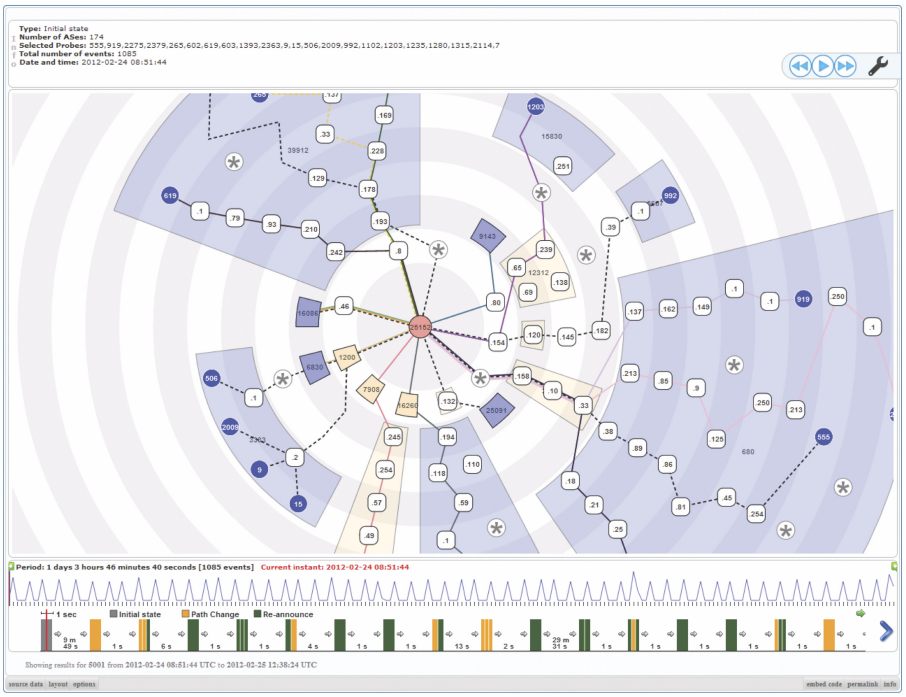


Figure 7.1: Main interface of TPlay.

is a graphical version of the traceroute program. It displays individual routes on an interactive rotating globe as a series of yellow lines between sites, shown as small spheres of different colors. GTrace [PN99] and VisualRoute [Vis97] are traceroute and network diagnostic tools that provide a two-dimensional, geographical visualization of paths. VisualRoute also explores abstract representations taking into account other information, e.g. the round-trip time between intermediate hops. In the second group of tools there are several examples (see e.g. [Hok08, Sco11]) that merge the paths generated by multiple traceroutes into directed graphs and show them in some type of drawing.

In recent years the visualization of Internet measurements has seen a growing interest. This is mainly due to the existence of several projects that deploy *probes* in the Internet. Probes are systems that perform traceroutes and other measurements (e.g. ping, HTTP queries) towards selected targets. They pro-

duce a huge amount of data that is difficult to explore, especially when dealing with the network topology. The reader can refer to Section 3.2 for a list of currently active projects based on geographically distributed probes.

In this paper we present TPlay, a system for the visualization of traceroute data. It was designed to support ISPs and owners of Autonomous Systems in the management and maintenance of their networks. The requirements were gathered interacting with several ISPs within the Leone FP7 EC Project and with the RIPE NCC. The user of TPlay selects a set $\mathcal{S}$ of probes of a certain Internet measurement project (all the experiments in this paper have been conducted using RIPE Atlas probes), a target IP address τ, and a time interval $\mathcal{T}$, and obtains a visualization of how the traceroutes issued by the probes in $\mathcal{S}$ reach τ during $\mathcal{T}$. TPlay can be used to study several properties of traceroute paths. These include assessing the reachability of τ over time, discovering the ISPs that provide connectivity to reach it, monitoring the length of traceroute paths as a performance indicator, and inferring how routing policies affect the paths of different probes in $\mathcal{S}$.

A snapshot of TPlay is in Fig. 7.1. The routing graph is presented with a radial drawing. The geometric distance between the target τ in the center and any object reflects the topological distance of that object in the network. Also, since traceroutes yield a very fine-grained and detailed understanding of network topology, the system allows to look at the network at different abstraction levels. Finally, the evolution of traceroute paths over time is presented by means of geometric animation.

The chapter is organized as follows. In Section 7.2 we detail the use cases, describe the adopted visualization metaphor, and introduce some formal terminology. In Section 7.3 we detail the algorithms used to compute the visualization comparing them to the state of the art. Section 7.4 gives some technical information on our prototype implementation. Section 7.5 contains conclusions and future directions.

7.2 Metaphor and User Interaction

There are four main tasks that motivate the design and development of TPlay. They are somewhat reminiscent of the use cases already in place for BGPlay.js (see Chapter 6), but traceroutes operate at a different level of abstraction that open different possibilities. The *security* is a primary concern. Knowing what ASes provide connectivity to reach any target over time is crucial for privacy and data protection, because some ASes may be less trusted than others. In

terms of routing *policy*, seeing how traffic is routed inside a specific AS over time helps discover load balancing issues or differences in the routing applied to different probes. On a simpler level, the network *distance* of any Internet service is an important metric. Longer paths are indeed potentially responsible for instability and inefficiency, therefore it makes sense to monitor the number of hops traversed by each probe over time. Finally, the *dynamics* of the routing are often responsible for performance and reachability issues. Users want to see how the routing changes at a specific time instant, based on external key indicators. For example, they may want to check if the routing has changed after a noticeable drop in the round-trip delay experienced when reaching the selected target from their network.

The first idea we explored for the representation of traceroute data was based on the simple representation of hops at their geographical location. However, we discarded solutions based on geographic representations for many reasons. First of all, the fact that a router belongs to a certain ISP or AS is the main piece of information for our purposes, whereas geography is only a secondary feature that further characterizes the nodes in the network. Also, the geolocation data associated with IP addresses is often wrong or incomplete, and anycast addresses (i.e., those assigned to more than one physical device) can not be mapped to a single location. Finally, the use of landmarks on geographical maps would require special care to avoid geometric cluttering. Motivated by the above, we focused on a topological representation of the data.

The visualization metaphor we adopted is presented below together with supporting motivations. Graphs are represented with *radial layered drawings*, where vertices are placed on concentric circles and targets are in the center. This style of drawing is notably effective for visualizing sparse hierarchical graphs (see, e.g., [YFDH01]). In Section 7.3 we show that our application domain meets such requirement. The probes originating the traceroutes are displayed in the periphery of the drawing, in order to effectively represent topological distances. Moreover, radial drawings have their center as the only focus point, which avoids giving probes additional importance due to a privileged geometric position. Finally, the drawing looks like an abstract geography and therefore borrows the typical user experience deriving from cartography and geographical visualization.

The need of visualizing the network at different abstraction levels is met by partitioning the set of routers into *clusters*. In our setting, clusters are in correspondence with ASes. The user can modify the representation by interacting with any cluster to either contract or expand it. A *contraction* causes all the routers in the cluster to be merged into a single object representing

the cluster, while an *expansion* does the opposite. Collapsing all clusters leads to a high-level, uncluttered view of the graph, which pretty much resembles the graphical effect of BGPlay.js (see Chapter 6). On the other hand, the user can expand the entire set of clusters to see all the routers traversed by at least one traceroute. In general, the user can arbitrarily expand any subset of clusters to examine them in detail.

Paths for reaching the target from the probes change over time. A natural way to show the evolution of traceroutes at different time instants is to present an animation of the drawing. More precisely, for each instant in a given time interval we show a different drawing, corresponding to the traceroutes that are available at that instant. We animate the change from a drawing to a successive one by means of a geometric morph.

Since the visualization is highly interactive and the graph changes over time, preserving the mental map is of paramount importance. Indeed, the user can both animate the drawing in a specific time interval and expand/contract individual clusters. We require that the same drawing is visualized for any two sequences of cluster expansions/contractions that produce the same graph. Also, the graph should be animated smoothly, even at the expense of traversing drawings that are not aesthetically optimal.

Traceroute paths cannot simply be merged and displayed in an aggregate fashion, since each of them has its own informative value and can change over time. For this reason, we represent paths adopting a metro-line metaphor (see e.g. [Rob12]) and draw them using different colors. Further, paths that never change in the selected time interval should be easily distinguished. We adopt the same method described in Chapter 6 to visually separate stable paths by grouping them into trees.

The objects to be visualized are formally defined as follows. Consider a time interval $\mathcal{T}$ and a set of probes $\mathcal{S}$. During $\mathcal{T}$ each probe periodically issues a traceroute towards a target IP address τ. A *traceroute* from a probe $\sigma \in \mathcal{S}$ produces a simple directed path on the Internet from σ to τ. If such a path is available at time $t \in \mathcal{T}$ (i.e. if the visited hops are actually reachable), then it is *valid* at time t. Each vertex of a traceroute originated from σ is either a router or a computer. Vertices are identified as follows: 1. σ has a unique identifier selected by the RIPE NCC; 2. vertices with a *public* IP address are identified with it; 3. vertices with a *private* IP address can be identified with a pair composed of their address and the identifier of σ. The reader can refer to [RMK+96] for a distinction between public and private IP addresses; 4. the remaining vertices are labeled with a "*" (i.e. an unknown IP address). For the sake of simplicity, consecutive vertices in a traceroute labeled with "*"

are merged into one. A vertex labeled with "*" is then identified with the identifiers of its neighbors in the traceroute.

A directed graph G_t is defined at each instant $t \in \mathcal{T}$ as the union of all the paths valid at t and produced by the traceroutes that are issued by the probes in $\mathcal{S}$. A directed graph $G_{\mathcal{T}}$ is defined as the union of all graphs G_t. Each vertex of $G_{\mathcal{T}}$ is assigned to a *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. Such information is extracted from the RIPEstat database (see Section 3.2 for details) and may occasionally be missing. 3. Each vertex v that is not assigned to a cluster after the previous steps is managed as follows. Consider all traceroute paths P containing v. For each traceroute $p \in P$ let μ (ν) be the cluster assigned to the nearest predecessor (successor) of v with an assigned cluster. If $\mu = \nu$ then μ is added to the set C_v of candidate clusters for v. If $|C_v| = 1$, v is assigned to the only candidate cluster. If there is more than one candidate, an inconsistency is detected and the procedure terminates prematurely. 4. Each remaining vertex is assigned to a corresponding *fictitious* cluster. We define V_μ as the set of vertices assigned to cluster μ.

For any $t \in \mathcal{T}$, G_t can be visualized at different abstraction levels. Namely, the user can select a set $\mathcal{E}$ of clusters that should be expanded to reveal the internal topology of routers, while each cluster that is in the complement $\bar{\mathcal{E}}$ of $\mathcal{E}$ is contracted into one vertex. More formally, given the pair $G_t, \mathcal{E}$ the visualized graph $G_{t,\mathcal{E}}(V, E)$ is defined as follows. V is the union of the V_μ for all clusters $\mu \in \mathcal{E}$, plus one vertex for each cluster in $\bar{\mathcal{E}}$. E contains the following edges. Consider edge (u, v) of G_t and clusters μ and ν, with $u \in \mu$ and $v \in \nu$. If $\mu \neq \nu$, $\mu \in \mathcal{E}$, and $\nu \in \mathcal{E}$, then add edge (u, v). If both μ and ν are in $\bar{\mathcal{E}}$ then the edge (μ, ν) is added to E. If $\mu \in \mathcal{E}$ ($\mu \in \bar{\mathcal{E}}$) and $\nu \in \bar{\mathcal{E}}$ ($\nu \in \mathcal{E}$) then we add edge (u, ν) ((μ, v)) to E. We define $G_{\mu,t}$ as the subgraph of G_t induced by V_μ. Analogously, we define $G_{\mu,\mathcal{T}}$ as the subgraph of $G_{\mathcal{T}}$ induced by V_μ. We define $G_{\mathcal{T},\mathcal{E}}$ as the union of the $G_{t,\mathcal{E}}$ for each $t \in \mathcal{T}$.

Fig. 7.1 shows an overview of our prototype implementation. Let $t \in \mathcal{T}$, τ and $\mathcal{S}$ be respectively the time instant, the target, and the set of probes selected by the user. Graph $G_{t,\mathcal{E}}$ is represented with a radial drawing centered at τ. All vertices and clusters that appear in at least one traceroute in $\mathcal{T}$ are in the drawing, including those that are not traversed by any traceroute at time t. Probes in $\mathcal{S}$ are represented as blue circles and labeled with their identifier. Vertices are represented as white rounded rectangles and labeled with the last byte of their IP address or with a "*", based on their unique identifier. Clusters

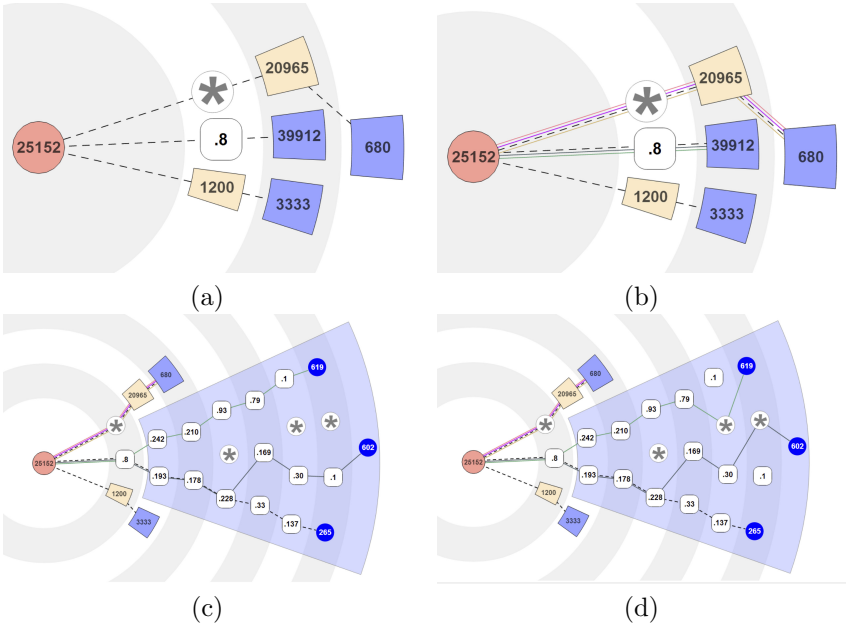


Figure 7.2: Details of the interactive features of our visualization. (a) A graph $G_{\mathcal{T}'}$ relative to a target τ, a set of probes $\mathcal{S}$, and a time interval $\mathcal{T}'$. All paths in $G_{\mathcal{T}'}$ are static and all clusters contracted. (b) A graph $G_{\mathcal{T}''}$ relative to τ, $\mathcal{S}$, and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$). Some paths are dynamic and all clusters are contracted. (c) $G_{\mathcal{T}''}$ with an expanded cluster. (d) $G_{\mathcal{T}''}$ at a different time instant.

are represented as annular sectors and labeled with their AS number. Note that vertices assigned to expanded clusters are enclosed in their sectors, while sectors of contracted clusters are empty. The light red cluster in the center contains τ. Clusters containing probes in $\mathcal{S}$ are light blue. The remaining clusters are light yellow. Fictitious clusters (i.e. those containing vertices that could not be assigned to any proper cluster) are not displayed. Each path from a probe $\sigma \in \mathcal{S}$ to τ is represented with a colored curve from σ to τ passing through all intermediate vertices. Paths are either solid or dashed, depending on whether they change or not during the time interval $\mathcal{T}$. Concentric circles in the background represent the increasing topological distance of vertices.

Fig. 7.2 showcases various details of the user interaction available in TPlay.

A graph with static paths and no expanded clusters is presented in Fig. 7.2(a). It is related to a target τ, a set of probes $\mathcal{S}$, and a small time interval $\mathcal{T}'$. Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. A graph for τ, $\mathcal{S}$ and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$) is presented in Fig. 7.2(b). Some dynamic paths are visible. The same graph is presented in Fig. 7.2(c) after the expansion of one cluster. Note how the ordering of clusters and vertices on the radial layers is preserved. Fig. 7.2(d) shows the same expanded graph at a different time instant. The intermediate vertices of two traceroute paths are different.

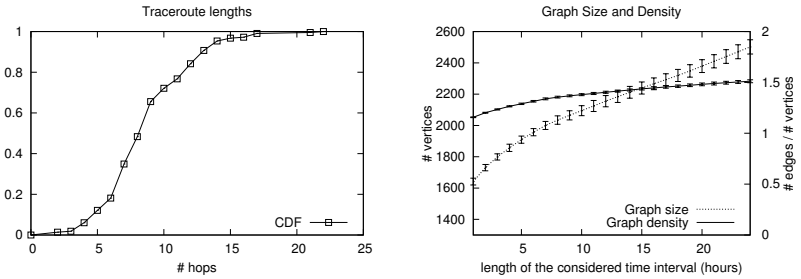
Fig. 7.2 also helps us explain how the tasks detailed at the beginning of the section can be accomplished. The task involving *security* is satisfied in Fig. 7.2(a): we can see how ASes 1200 and 20965 provide connectivity to reach the target. The tasks related to *policy* and *distance* are instead addressed in Fig. 7.2(c), where the length and structure of the paths from each of the three probes 619, 602, 265 is clearly visible. Finally, the *dynamics* of the routing are visible in Figg. 7.2(c)-(d), where users can understand how the paths for probes 619 and 602 change after a routing event.

The user interaction plays a major role in our metaphor. The reader can refer to [CDBDBS13] for an example video of the interaction with TPlay.

7.3 Algorithms

We started our analysis by computing several statistics on the RIPE Atlas dataset that we used to test our system. It consists of traceroutes executed in one month (July 2012) by 200 probes distributed all over the world. Fig. 7.3 presents the main results of our analysis. In Fig. 7.3(a) we plot a cumulative distribution function of the length of traceroute paths. That gives us a rough indication on the maximum distance between a probe in $\mathcal{S}$ and τ. The plot shows that traceroutes with more than 15 vertices are rare, which makes the radial metaphor particularly suitable for the scenario. In Fig. 7.3(b) we plot the number of vertices and the density ($|E|/|V|$) of $G_{\mathcal{T}}$ as a function of $\mathcal{T}$. It turns out that $G_{\mathcal{T}}$ is quite sparse for time intervals that are compatible with the application domain. In particular, the density ranges between 1.2 and 1.5 for time intervals within 24 hours. The number of vertices is in the range of 2000. That further motivates the adoption of interactive clusters to limit the clutter in our visualization.

As a second step, we looked for a satisfying representation of the input data. We performed preliminary experiments using spring embedders, hierar-



(a) (b)

Figure 7.3: Statistics on the data set. (a) Cumulative distribution function (CDF) of the length of traceroute paths at July, 1st 2012 at 00:00. CDFs at different instants exhibit similar features. (b) Plot showing the number of vertices and the density of $G_{\mathcal{T}}$ as a function of $\mathcal{T}$. For each day in the month we set an initial time at 00:00 and grow $\mathcal{T}$ from 1 to 24 hours. For each value of $\mathcal{T}$ we plot the average density and number of vertices. We report the standard deviation with error bars.

chical drawing, and upward drawing algorithms (see Section 2.2 for a general introduction to different graph drawing techniques). Layouts produced with spring embedders are unsuitable for our metaphor, because the topological distance between vertices is not always represented and because the resulting drawings are characterized by limited regularity. Also, spring embedders tend to introduce crossings that are avoidable, given the expected density of the data set. For hierarchical drawing, we experimented both basic algorithms and variations that allow to represent clustered graphs [San96, San99]. The experiments put in evidence that crossing-reduction heuristics like those in [San96, San99] are quite effective. However, in our case most graphs are planar or quasi-planar and therefore planarity-based methods are more attractive. Finally, we discarded upward planar drawings. The main reason is that they tend to use vertical space to resolve crossings, which may result in large geometric distances between vertices that are topologically close.

A very high level and informal description of our algorithmic framework is detailed as follows. We precompute a hierarchical drawing Γ_0 of $G_{\mathcal{T}}$ that integrates all the traceroutes in $\mathcal{T}$. All clusters in Γ_0 are expanded. The layout is computed limiting the number of crossings that involve connections

between clusters. The quality of the layout inside the clusters is considered with lower priority. Moreover, the quality of the drawing of edges that are part of many traceroutes in $\mathcal{T}$ is privileged among the edges of $G_{\mathcal{T}}$. The drawing computed for each cluster is stored and reused in any drawing where that cluster is expanded. The hierarchical drawing is mapped to a radial drawing with a suitable coordinate transformation. Changes in the drawing due to an expansion or contraction of a cluster or a change in traceroutes are visualized with an animation. At any instant $t \in \mathcal{T}$ only the traceroutes that are valid in t are displayed.

We discarded several algorithms found in literature. An interesting reference for our purposes is [Bac07], in which the author shows how to construct radial drawings adapting techniques of the Sugiyama Framework. Unfortunately, his work does not deal with clusters. The algorithm in [FB04], which extends the one described in [DBN88], inspired part of our work. However, it proposes a clustered planarity testing algorithm, while we rather need an algorithm for clustered graph planarization. Further, [FB04] is not easily extensible for this purpose, neither is the algorithm in [BDM02] that is not suitable for hierarchical drawings. For these reasons we devised a new algorithm to produce clustered hierarchical drawings, as a planarization-oriented variation of [FB04]. In [Rai05] the authors propose an algorithm for the expansion/contraction of clusters of hierarchical drawings, building on [SM91]. Unfortunately their technique uses local layering for vertices, while global layering [San96, San99] is more suitable for our needs because it produces more compact drawings. Indeed a very common use case of TPlay is to expand all clusters along one or more traceroutes (for example when assessing the network *distance* of a specific probe). Local layering would visualize far from τ also vertices in unrelated paths because of the increased need for vertical space of their layers. For this reason we devised a new algorithm for expanding/contracting clusters that is based on global layering. Differently from [Rai05] it is not a local update scheme, i.e. it computes a new drawing for the whole graph at each interaction. The lower time efficiency is negligible because the graphs commonly handled by TPlay are relatively small, as explained early in this section. Finally, the preservation of the mental map during the expansion/contraction of clusters is addressed with a geometric morph, implemented as an animation of objects from their initial position to their final position.

What follows gives more details on our the algorithmic framework. In a preprocessing step we compute several information based on $G_{\mathcal{T}}$ that will be used for the actual drawings. Given any $G_{\mu,\mathcal{T}}$, a vertex is a *source* (*sink*) of $G_{\mu,\mathcal{T}}$ if it is the last (first) vertex of $G_{\mu,\mathcal{T}}$ encountered in at least one traceroute

path. Each graph $G_{\mu,\mathcal{T}}$ is augmented with extra vertices and edges so that all the longest paths from a source to a sink have the same length. The added vertices are called *fictitious vertices* of μ and ensure that, given an edge $(u, v) \in G_{\mathcal{T}}$, $u \in \mu$, $v \in \nu$, $\mu \neq \nu$, clusters μ and ν do not share a layer in any drawing of $G_{t,\mathcal{E}}$. Moreover, in this way the edges that leave a cluster by spanning several layers are necessarily routed inside that cluster. A *μ-drawing* is precomputed for each $G_{\mu,\mathcal{T}}$ in two steps: 1. we assign vertices to layers so that all edges are between consecutive layers, and 2. we compute a total order for the vertices of each layer. A partial order $\prec$ is computed for clusters, such that for any two clusters μ and ν with $\mu \prec \nu$, the vertices of μ appear to the left of the vertices of ν for any drawing Γ where μ and ν share one or more layers. This helps preserve the mental map during expansions/contractions. The preprocessing step requires to compute a drawing Γ_0 of $G_{\mathcal{T}}$ with all clusters expanded. Γ_0 gives the information needed to compute a μ-drawing for each cluster and a partial order $\prec$ for clusters. The algorithm to compute Γ_0 is similar to that in [FB04], where a PQ-tree [BL76] is used to order vertices along the layers of the drawing. Our PQ-tree is initialized with a spanning tree of $G_{\mathcal{T}}$ and incrementally updated with the remaining edges that induce ordering constraints. An edge is added only if it does not produce a crossing, i.e. the PQ-tree does not return the "null" tree. A rejected edge will produce crossings in Γ_0. Edges are added with priority given by their aesthetic importance: namely, they are weighted based on the number of traceroutes that traverse them in $\mathcal{T}$. As an implementation detail, we actually compute a total order for clusters to represent a partial order $\prec$. The order is produced with a depth-first traversal of the spanning tree of $G_{\mathcal{T}}$. The tree has an embedding induced by the layer orders produced by the PQ-tree algorithm. Children of any vertex are visited in clockwise order. Intuitively, we preserve the geometric left-to-right order for clusters from Γ_0, and reuse it to produce drawings for any $G_{t,\mathcal{E}}$.

The computation of any drawing $\Gamma_{\mathcal{T},\mathcal{E}}$ for $G_{\mathcal{T},\mathcal{E}}$ is detailed below. Before that, note that once $\Gamma_{\mathcal{T},\mathcal{E}}$ is computed, for any $t \in \mathcal{T}$ we display all the vertices in $G_{\mathcal{T},\mathcal{E}}$ but only the edges in $G_{t,\mathcal{E}}$. Our choice is motivated by the need to preserve the mental map of the user, using $\Gamma_{\mathcal{T},\mathcal{E}}$ as a "frame" where we realize the drawings for each time instant t. First, a layering of $G_{\mathcal{T},\mathcal{E}}$ is computed such that for each vertex the distance from τ is minimized. Also, dummy vertices (called *fictitious vertices* of $G_{\mathcal{T},\mathcal{E}}$) are added so that each edge spans two consecutive layers. Vertices are horizontally ordered on each layer such that 1. $\prec$ is enforced, 2. for each cluster $\mu \in \mathcal{E}$ the orders on the layers of its μ-drawing are enforced, and 3. the fictitious vertices of $G_{\mathcal{T},\mathcal{E}}$ are placed on the layers in such a way to have few crossings. In particular, fictitious vertices

must not be interleaved with the vertices of any cluster, so that vertices of each cluster are consecutive on every layer. For this reason, each fictitious vertex is assigned to a new fictitious cluster, which is inserted in the partial order $\prec$ in an intermediate position between the endpoints of the edge it belongs to. Finally, the ordered layers are used to assign geometric coordinates to vertices. The width of each cluster μ is computed as follows. Consider the layer containing the largest number of vertices assigned to μ. The cluster is assigned a width proportional to this number. Vertices of μ are assigned horizontal coordinates such that they can be enclosed by a rectangle with height proportional to the number of layers assigned to the vertices of μ and width equal to the width of μ. We avoid intersection between enclosing rectangles by means of an auxiliary directed acyclic graph where vertices are clusters of $G_{\mathcal{T},\mathcal{E}}$ and edges are selected from $\prec$ depending on which pairs of clusters share a layer in the current layering of $G_{\mathcal{T},\mathcal{E}}$. Edges are weighted based on the widths of the clusters they are incident to. The total width of the drawing is given by the longest path in this graph. The above is applied recursively to compute the horizontal spacing among all clusters. The vertical coordinate of a vertex is equal to the one assigned to its layer, which is proportional to the index of that layer in the total order of layers.

Going back to the state of the art, we consider the restrictions that a planar clustered hierarchical drawing should obey, as expressed in [FB04]. $R1$ says that all the vertices on each layer that belong to the same cluster should be consecutive. $R2$ says that clusters should not cross each other.$R3$ says that edges should not cross clusters. With respect to these three rules, drawings produced by our algorithm satisfy $R1$ and $R2$. We deliberately ignored $R3$ because we consider it too restrictive for our application. $R1$ is satisfied in the preprocessing step by merging all sources of each cluster into one new vertex. The PQ-tree is initialized with a spanning tree that contains all these new vertices, which has the effect of keeping the vertices of each cluster consecutive on each layer. $R2$ is automatically satisfied for the initial drawing Γ_0, and consequently in any drawing of $G_{t,\mathcal{E}}$ by exploiting the partial order $\prec$.

To obtain a radial drawing, the geometric coordinates computed in the previous steps for each vertex are transformed as follows. Each vertex is placed on the perimeter of a circle centered at an arbitrary fixed point and having radius equal to the vertical coordinate of the vertex. Then the horizontal coordinate of the vertex is mapped to a circular coordinate on the perimeter of that circle. The perimeters of clusters are mapped with a similar radial transformation. 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 in our setting each edge connects only vertices in two consecutive layers, hence a curved edge can be drawn only in the space between these layers.

7.4 Technical Details

The implementation of TPlay is split into three main blocks: a visualization front-end, a layout engine, and a data back-end.

The *visualization front-end* is a Web application. It allows the user to specify input parameters and to visualize and animate interactive graphs. The implementation of the front-end required to focus on some algorithmic details. The arrangement of paths in a metro-line fashion is implemented as follows. First of all, an arbitrary total ordering is computed on the set of visualized paths. For each edge without bends in the graph, the paths that traverse it are drawn as parallel segments connecting the two endpoints of the edge. The order of such segments reflects the total order of paths, in order to promote consistency between edges. In case the edge contains bends, the drawing is computed in two steps. First, we split the bended edge in a sequence of intermediate edges $e^1, \ldots, e^n$ and compute the path segments for each of them. Second, for each pair of consecutive intermediate edges (e^i, e^{i+1}) and for each path that traverses it, we call (u, v) and (w, z) the two segments computed respectively for e^i and e^{i+1}. If there is an intersection point p between (u, v) and (w, z), we rewrite the two segments as (u, p) and (p, z). Otherwise, we add a connection (v, w) between (u, v) and (w, z). Path colors are computed with the algorithm described in [Kis12] to ensure that they are distinguishable from each other. The front-end is written in JavaScript and HTML. As already mentioned, it is based on the BGPlay.js framework described in Chapter 6.

The visualization always starts with an overview of the traceroutes. The *layout engine* is invoked at the beginning to produce a drawing of $G_{\mathcal{T},\emptyset}$. When the user expands/contracts a cluster (i.e. a cluster is added or removed from $\mathcal{E}$) the layout engine is invoked again on $G_{\mathcal{T},\mathcal{E}}$. In the implementation of the radial drawing we artificially increase the radius of each layer by an additional offset, such that vertices on dense layers are not overlapped. For the sake of simplicity, curved segments are uniformly sampled and drawn as polylines. The layout engine is implemented in Java. We initially designed it to be implemented as part of the visualization front-end, but later moved to a back-end implementation in order to make use of already existing libraries. In particular, we adopted a PQ-tree implementation [Har02] and Apache Commons Graph [Apa12] for general graph models and algorithms. We optimized the output of the layout

engine after the initial layout is computed, so that only graph elements with new drawing coordinates are included.

Finally, the *data back-end* is mainly responsible for retrieving and preprocessing traceroute data. The results are then used by the front-end to animate traceroute events and by the layout engine to compute the drawings.

7.5 Conclusions and Future Work

We presented a metaphor for the visualization of traceroute measurements issued towards specific targets on the Internet. It is based on a radial drawing of a clustered graph, where vertices are routers or computers and clusters are administrative authorities (ASes) that control them. Our metaphor allows the user to interact with the visualization, both exploring the content of individual clusters and animating the graph to see how traceroute paths change over a time interval of interest.

In the future we will take into account the *DNS resolution* of selected targets in the visualization. That means that some targets may be represented by more than one vertex, giving rise to an anycast behavior of the target, depending on the policies implemented at the DNS level. We will also explore the possibility to process streams of incoming data, adding or removing elements in the visualization incrementally. Finally, from a technical perspective, we plan to move a significant part of the layout engine to the front-end, including a brand new implementation of PQ-trees in JavaScript.

Part IV

Mixing Up

Chapter 8

Automating the Analysis of the Impact of Routing Changes on Round-trip Delay

This chapter mostly deals with network data analysis, without a specific focus on visualization. However, our results are at the basis of the visualization framework reported in Chapter 9. The reader can refer to Chapter 11 for the details of two scientific publications based on the work in this chapter.

8.1 Introduction

Strict performance requirements characterize an ever-growing base of Internet services. Keeping certain performance levels is not only critical for the satisfaction of Service Level Agreements (SLAs), but also important to ensure that the quality perceived by end users is always high. Lots of applications, including streaming, VoIP conferencing, gaming, and financial transactions, rely on steady performance levels. However, it is a matter of fact that performance fluctuations may occur, depending on several factors like bandwidth, congestion, and routing changes.

In this chapter we focus on understanding the relationship between variations of network performance, measured in terms of round-trip times (RTTs), and inter-domain routing changes, computed by the Border Gateway Protocol (BGP). We concentrate on RTTs because they are the most commonly available measurement, ICMP requests are unlikely to be filtered out, and latency

is nowadays regarded as an important performance indicator. Similarly, we consider BGP routing changes because their impact is significant, as stated in [PZMH07]. The reader can refer to Section 3.1 for additional details on network performance and routing.

As a result of our research, we bring three main contributions. First of all, we describe a matching methodology to determine whether a routing change caused a significant variation of the RTT, with two key novel aspects: it exploits state-of-the-art statistical methods and it can compute the matching automatically. Second, we perform an experimental verification of the effectiveness of our matching methodology in the wild, using publicly available datasets, i.e. BGP updates from the RIPE Routing Information Service and RTT measurements from the RIPE Atlas project (see Section 3.2 for details). Finally, we present a set of a-posteriori analyses based on the results of our matching methodology, which lead to interesting findings for several practical applications and motivate the visualization framework presented in Chapter 9.

The first added value brought about by our methodology is clearly the augmented awareness of performance fluctuations. However, we envision several other applications. For example, learning that BGP routing changes recorded by certain vantage points affect more significantly the RTT towards a certain destination can motivate usage of those vantage points to predict the impact of future routing changes and drive traffic engineering decisions. In principle, placement of a delay-sensitive service can also benefit from knowing that routing changes observed at certain network locations are more likely to affect its reachability. Discovering that certain routing changes affect RTTs towards apparently unrelated destinations can help network administrators in troubleshooting tasks.

The rest of the chapter is organized as follows. Section 8.2 contains a brief introduction to the state of the art. In Section 8.3 we describe our methodology to match BGP routing changes and significant RTT variations. In Section 8.4 we apply the methodology to search for BGP-RTT correlations in the wild, using data from RIPE RIS and RIPE Atlas. In Section 8.5 we introduce various analyses based on the results of our matching methodology. Conclusions and future work are discussed in Section 8.6.

8.2 Related Work

The correlation between between routing and performance has been already explored as a challenging research topic. However, although many contributions

bring interesting aspects to take into account, none of them is targeted at a reproducible, automated methodology. We detail the main examples below.

In a pioneering contribution [PZMH07], the impact of routing changes on RTTs has been confirmed and shown to be non-negligible. Interesting arguments in that paper strongly motivate our study. Most delay variations are indeed caused by routing changes rather than congestion. Further, inter-domain routing changes are those that impact most on the average delay variation. However, the problem of automatically associating RTT variations with BGP path changes is not encompassed.

In [CBD02, WMW+06, ZMW07] the authors study transient network performance degradations due to routing convergence periods, while we concentrate on RTT values during stable routing states. Other contributions exploit statistical tools for the analysis of trends in network data. In [MSG+10, MGW+11] the authors detect the impact of network upgrades by using statistical rule mining and network configuration information to identify meaningful patterns in performance changes. A recent work [TG13] applies change detection algorithms to compute network coordinates, i.e., metrics that help predict the network delay between pairs of hosts, in a realistic environment.

8.3 Methodology

In the rest of the chapter we assume that the inputs to our methodology are collected in the following scenario. We consider an AS A that is connected to several other ASes by means of BGP border routers. As explained in Section 3.2 a subset of these routers, called *collector peers* (CPs), forwards all the computed inter-domain routes to a central BGP collector that in turn stores them. AS A also comprises *probes* that periodically run standard tools (e.g., `ping`, `traceroute`) to measure RTTs (and, possibly, IP routing paths) towards a fixed set of *targets* that are external to AS A. The results of these *measurements* are stored as well. When leaving AS A, traffic from a probe to a target traverses a border router that may or may not be a CP. If it is, a correlation necessarily exists between the measurements performed by the probe and the BGP updates received by the border router for a prefix comprising the target. If it is not, a correlation may still exist because the BGP updates recorded by other CPs for the same prefix may also influence the behavior of the traversed border router. Our goal is to find such correlations, using CPs and probes available in the AS under consideration as vantage points.

The main steps of our matching methodology are in Fig. 8.1. The method-

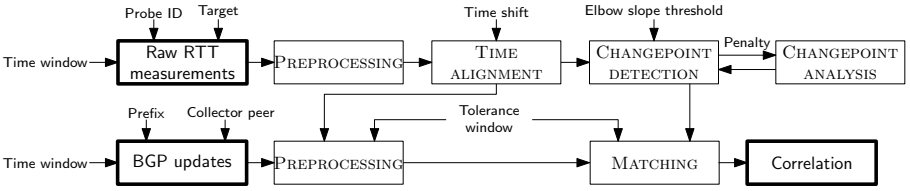


Figure 8.1: Main steps in our matching methodology. Thick-border boxes are inputs and outputs. Thin-border boxes are operations. Arrows indicate data flows. Tunable parameters are represented without a box.

ology takes as input RTT measurements from a set of probes and BGP updates from a set of CPs. We assume that all the inputs to our methodology are collected within a Time window. The two datasets taken into account are very different in nature: RTTs are recorded on a periodical basis and are subject to lots of fluctuations, whereas BGP updates may arrive in bursts and usually involve a limited number of different routing paths. To fill this gap, and to clean up the inputs, we apply several preprocessing steps. We then account for a possible time difference between the two data sets and use a state-of-the-art statistical method to detect significant variations in RTT values. The output of the methodology is an estimate of how much BGP routing changes have an observable impact on RTT values. No additional information (e.g., network topology) is required to apply our methodology. Although we are unable to observe routing changes happening on the reverse path from the target to the probe and RTT measurements may be affected by some biases (probe clock synchronization, presence of load balancers, etc.), our methodology is still able to produce significant results.

Processing of RTT Measurements and BGP Updates

The first input is a sequence of timestamped RTT measurements performed by a probe towards a destination IP address Target. The probe is identified by a Probe ID. Being usually performed with standard tools like `ping`, we assume each measurement records a fixed number of RTT values (3 in the case of RIPE Atlas probes) as well as the IP address that was actually reached. In the Preprocessing step we discard measurements that recorded fewer RTT values than expected or reached an unintended IP address. That is motivated by the existence of spurious records in many measurement networks, caused by

many factors (e.g. misconfigured probes, bugs in the firmware, etc). Further, we only consider the minimum RTT value in each measurement to better isolate the effect of propagation and transmission delays, which exhibit low variability and depend mostly on the length of routing paths and on the physical distance of devices (see, e.g., [HM07]). RTT timestamps are then shifted by a fixed Time shift in the TIME ALIGNMENT step, in order to compensate offsets between the clocks of the probes and those of the CPs, and to consider possible delays in the propagation of BGP routing changes (depending, e.g., on the relative position of probes and CPs or on the MRAI timer).

The second input is a sequence of timestamped BGP routing updates observed by a Collector peer for a specific Prefix. Each update describes how, according to BGP, traffic should be routed from the CP to the range of IP addresses falling within Prefix. For this reason, an update carries at least an AS-level path (possibly empty in the case of a withdrawal). During the PREPROCESSING step we retain only BGP routing changes that are eligible for further analysis, based on the outcome of the TIME ALIGNMENT step. In particular, of all the BGP updates happening between two consecutive RTT measurements we only retain the most recent one. If the two measurements are separated by a time lapse longer than a Tolerance window, all the BGP updates in between are discarded. Note that the Tolerance window should always be longer than the period of RTT measurements. In this way we get rid of routing changes that can not be "seen" in RTT measurements, preventing any improper deductions on them.

Detection of Significant Delay Variations

A remarkable challenge in our methodology is that RTT values are highly variable. In the not-so-extreme case when every value is representative of an RTT variation, any BGP routing change could in principle be matched with an RTT measurement that is close in time. Such result would of course have very little scientific value.

To avoid this, in the CHANGEPOINT DETECTION step we seek for time instants at which the mean values of RTT measurements change persistently. We exploit a technique called Pruned Exact Linear Time (PELT) [KFE12], one of the most recent contributions in the field of *changepoint analysis* statistical methods (for a survey, see [BN93]). PELT uses an efficient algorithm to detect mean and variance shifts in time series data. The precision of the analysis can be tuned by an input parameter called *penalty*. Using low values considers volatile shifts as valid changes, whereas using high values only de-

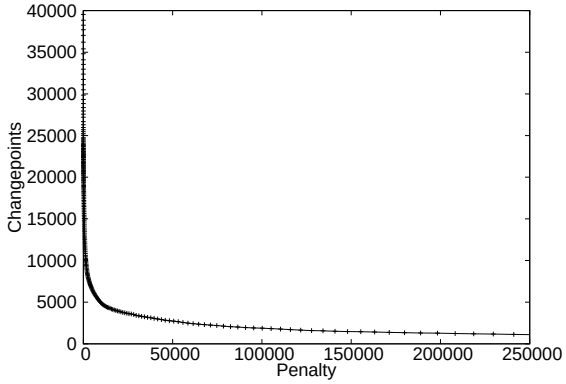


Figure 8.2: Distribution of changepoints as a function of the penalty.

tects shifts that affect a considerable portion of the input. Simpler methods like moving average would fail in equally detecting long-lasting small changes as well as short-lasting significant ones with as high precision as PELT did in our experiments. We processed RTT measurements with PELT and verified that increasing the penalty results in a hyperbolic-like decay in the number of detected changepoints. Figure 8.2 shows an example distribution of changepoints as a function of the penalty, measured with data coming from RIPE Atlas probes.

Choosing the "right" penalty value is not easy and depends on the nature of the input data. Too high penalties may result in a coarse detection of changepoints, while too low penalties may result in interpreting noise as legitimate variations. We tackle the challenge by adopting a rule called *elbow method* (see, e.g. [KS96]), traditionally used in the field of statistics. Starting from a base value p_0, we run the PELT algorithm for increasing penalties p_i, $i > 0$, and stop when the ratio $-\frac{chpt_i - chpt_{i-1}}{p_i - p_{i-1}}$ between the decrease in the count of detected changepoints $chpt_i$ and the increase of the penalty falls below an Elbow slope threshold. The highest penalty value reached at this point is selected as optimal. A further increase of the value for the penalty would discard too many potentially relevant changepoints. On a side note, PELT operates on values only and does not consider timestamps. To transform input RTT values into a step-wise function we preliminarily associate each changepoint with the

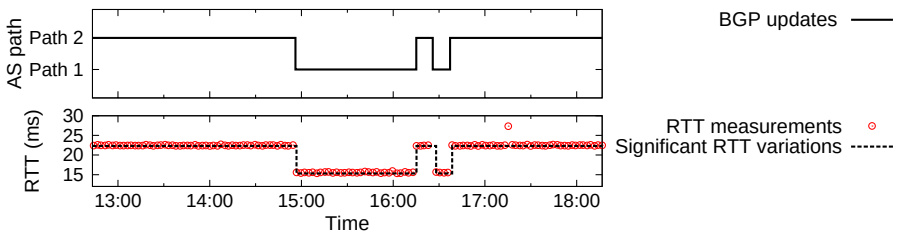


Figure 8.3: BGP path changes (upper plot) and associated significant RTT variations detected by PELT in RTT values (respectively, dashed line and dots in the lower plot).

timestamp of the RTT value that caused it, in order to retrieve the original timestamps once PELT is applied. A sample result of the application of PELT is in Fig. 8.3 (lower plot).

Matching and Correlation

The actual correlation between the two inputs takes place at the end of the process. In the MATCHING step we look for a correspondence between routing changes and RTT variations. For each BGP update with timestamp t, we consider a time window starting at t and as wide as the Tolerance window parameter. We associate RTT changepoints falling within this window with the current BGP update. Note that in order to mitigate the imprecisions of PELT we further discard changepoints corresponding to negligible RTT variations (less than 1 ms). A BGP update is marked as "correlated" if there is at least one RTT changepoint associated with it. Fig. 8.3 shows a clear example of correlation, where route flaps in the upper plot are matched with RTT changes in the lower one. To produce an overall Correlation estimate, we define the *correlation factor* as the fraction of preprocessed BGP updates marked as correlated.

8.4 Experimental Results

In order to apply the methodology in Section 8.3 to real-world data sets, we considered BGP data collected by hundreds of worldwide spread CPs managed by the RIPE Routing Information Service (RIS) and RTT data collected by

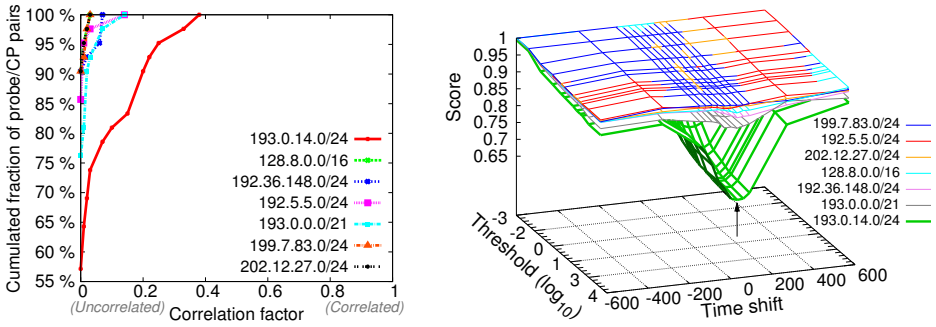
ID	Target IP address	BGP prefix	α	β
1001	193.0.14.129	193.0.14.0/24 (Anycast)	87.5%	99.5%
1003	193.0.0.193	193.0.0.0/21 (Unicast)	87.2%	97.3%
1004	192.5.5.241	192.5.5.0/24 (Anycast)	57.8%	100%
1005	192.36.148.17	192.36.148.0/24 (Anycast)	55.5%	99.1%

Table 8.1: Selected measurement targets. α is the percentage of AS paths of length ≤ 5 (global average length) towards the given BGP prefix. β is the percentage of probes for which the average RTT measured towards the target IP is $\leq$ 300ms (value recommended by ITU for VoIP).

thousands of probes deployed within the RIPE Atlas project. As explained in Section 3.2, several other projects collect similar data sets. We selected the RIPE projects because, being run by the same organization, they are likely to gather data from ASes where both probes and CPs are available, in accordance with the scenario described in Section 8.3. As of January 2013 there were 55 such ASes, hosting 126 CPs and 200 probes.

We fixed the Time window to a 2-year period ranging from January 2011 to December 2012. We kept the window intentionally large to show the potential of our methodology in finding interesting correlations, even when dealing with massive amounts of data. For all the 23 targets available in this window, we downloaded: 1. BGP updates and table dumps collected by all available CPs; 2. RTT measurements performed every 4 minutes and traceroute measurements performed every 20 minutes, collected by all available Atlas probes. We used traceroutes in further analyses (see Section 8.5 for details). Considering that for many targets the amount of measurement information was too small to identify a significant set of RTT changepoints, we restricted the application of our methodology to the targets in Table 8.1. Since three of these targets are name servers advertised as anycast BGP prefixes, they may exhibit less RTT fluctuations than other more localized targets. Although this made interesting correlations harder to find, experimental results show that our methodology was able to effectively cope with this additional challenge.

We started by executing few test runs of the methodology explained in Section 8.3, feeding it with the downloaded data. As a second step, we searched for an assignment of the tunable parameters in Fig. 8.1 that could maximize the distinction between poorly correlated and well-correlated information. Although we supervised many steps of this search, the obtained parameter values fit all the Targets and Prefixes we considered, eliminating the need to repeat it.



(a) (b)

Figure 8.4: (a) CDFs of the correlation factor for fixed Target ID 1001 and varying Prefixes, relative to the total number of probe/CP pairs. The Elbow slope threshold was fixed to 0.001, the Time shift to 0, and the Tolerance window to 5 minutes. (b) Correlation score for Target 1001 for varying Elbow slope threshold and Time shift. Each surface is relative to a Prefix. The arrow indicates a choice of parameters that maximizes the difference between "good" (low) and "bad" (high) correlation scores.

At first we arbitrarily fixed the Time shift, the Elbow slope threshold, and the Tolerance window, picked a single Target and Prefix, and applied the matching methodology to compute one correlation factor for all the data collected by each pair consisting of a Collector peer that recorded BGP updates affecting the Prefix and a Probe in the same AS that measured RTTs towards the Target. To compare "good" correlations with "bad" ones, we kept the Target fixed and recomputed the correlation factors for a sample of 7 randomly chosen Prefixes, including one that comprises the Target. Our goal was to prove that the methodology in Section 8.3 is able to discriminate between arbitrary and legitimate associations of input datasets. For each Prefix we plotted the Cumulative Distribution Function (CDF) of the values of the correlation factor, relative to the total number of probe/CP pairs. Fig. 8.4(a) shows such CDFs for Target 193.0.14.129. To compare factors for different Prefixes, in the figure we only considered CPs that recorded BGP updates for all the 7 Prefixes. As expected, correlation factors with Prefix 193.0.14.0/24, which comprises the Target, are higher (the CDF is shifted to the right), whereas the trends of CDF curves for other correlations are much steeper and similar to each other.

Based on this observation, which recurred in all our experiments with the four Prefixes in Table 8.1, we introduced a more aggregate measure, called *correlation score*. Such measure characterizes in a simpler way the relationship between a set of RTT measurements towards a Target and a set of BGP routing changes for a Prefix, regardless of the specific probe and CP. Considering the CDF of correlation factors for all the probe/CP pairs corresponding to the Target and Prefix of interest, the correlation score is computed as the area of the portion of the CDF plot that is under the curve. Note that a lower correlation score indicates a better correlation: the smaller the area under the CDF, the higher the correlation factors. For example, the correlation score for Prefix 193.0.14.0/24 in Fig. 8.4(a) is 0.95, whereas correlation scores for other prefixes are all higher than 0.99 and therefore imply the almost complete absence of correlation.

After introducing the correlation score, we could assess and quantify the impact of the Elbow slope threshold and of the Time shift on the correlation values. We computed the correlation score for each combination of a Target from Table 8.1 and a Prefix in the above sample of 7, varying the Elbow slope threshold and Time shift in a representative set of values. A sample result for Target 1001 is in Fig. 8.4(b), where each three-dimensional surface refers to a different Prefix. As it can be seen, higher Elbow slope thresholds result in better distinction between "good" and "bad" correlation scores. In fact for higher thresholds the lowest surface, corresponding to the Prefix that comprises the Target, is more separated from the other surfaces. Indeed, higher thresholds cause the selection of lower Penalty values in the CHANGEPOINT DETECTION step, which in turn causes more RTT changepoints to be detected and possibly matched with BGP updates, thus improving the score. For extremely high thresholds this phenomenon would equally affect all the Prefixes, degrading the distinction between "good" and "bad" correlation scores. For this reason, we chose a maximum threshold of 10000. Moreover, the correlation score improves significantly for specific values of the Time shift. The latter finding is a hint on the time offset between RTT measurements and BGP updates.

From the results conveyed by plots like the one in Fig. 8.4(b) we could determine the optimal values for the tunable parameters. In particular, we found out that picking an Elbow slope threshold equal to 10000 and a Time shift equal to 60 seconds results in an optimal separation between "good" and "bad" correlation scores for all the Targets and Prefixes we considered. Finally, considering the rate of RTT measurements (one every 4 minutes), as well as occasional irregularities in the measurement period, we fixed the Tolerance window at 5 minutes.

#	Feature under Analysis	1001	1003	1004	1005
1	consistent sign of ΔRTT	87.5%	78.6%	72.5%	86.4%
2	$\Delta RTT_{P_1 \to P_2} * \Delta RTT_{P_2 \to P_1} < 0$	64.8%	52.1%	43.3%	68.8%
3	$\Delta pathlen * \Delta RTT > 0$	76.4%	57.4%	64%	80.6%
4	$\sigma_{\Delta RTT}/\overline{\Delta RTT} < 0.25$	73.6%	75.5%	95.5%	93.1%

Table 8.2: Results of our statistical analyses for the Targets listed in Table 8.1

8.5 Analyses

We now discuss a few analyses based on the outcome of our matching methodology. They unveil interesting aspects of the input data sets and support potential applications of our study, one of which is detailed in Chapter 9.

Statistical Analyses

We performed various statistical analyses on the matchings between BGP changes and RTT variations. Some of our results are reported in Table 8.2.

Let a *path-change* be any occurrence of a transition from an AS path P_1 to an AS path P_2 recorded by a Collector peer and matched with an RTT variation seen by a Probe in the same AS. We first checked whether, in all its occurrences for a given probe/CP pair, a single path-change was always consistently matched with an increase, or decrease, of the RTT (analysis #1 in Table 8.2). For the majority ($\geq$72.5%) of the path-changes we collected, we found this to be true for all probe/CP pairs. At least half of the other path-changes were consistently matched with an RTT change in at least 70% of their occurrences. We then considered *path-change-pairs*, consisting of a path-change from P_1 to P_2 and the reversed path-change from P_2 to P_1, both seen by the same probe/CP pair. For a good fraction ($\geq$ 43.3%) of path-change-pairs, a path-change and its reversed counterpart corresponded, in all their occurrences, to opposite variations of the RTT (analysis #2 in the table). We then found that at least 57.4% of the path-changes that increased (decreased) the AS path length were consistently matched, in all their occurrences, with an RTT increase (decrease). See analysis #3 in the above table for details.

Next, we switched to more quantitative analyses of RTT variations. We inspected the predictability of the effect of a path-change by computing the average ($\overline{\Delta RTT}$) and standard deviation ($\sigma_{\Delta RTT}$) of the RTT variations associated with all the occurrences of the path-change for a probe/CP pair. For

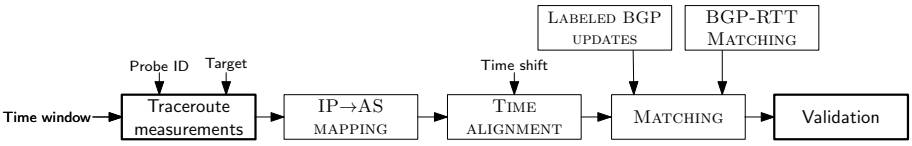


Figure 8.5: Steps of the validation with traceroute data.

most path-changes ($\geq$73.6%) the ratio $\sigma_{\Delta RTT}/\overline{\Delta RTT}$ was below 0.25, i.e., the same routing change resulted in fairly similar RTT variations (see analysis #4 in Table 8.2). We also inspected whether the position of the first modified AS in a path-change influenced the extent of RTT variations. With the exception of Target 1001, we found that changes happening close to the AS hosting the probe corresponded to larger average RTT variations, whereas changes happening far in the AS path were less impactful.

Comparison with Traceroute Data

As specified in Section 8.4, we also collected traceroute measurements from RIPE Atlas probes, which we used to perform a preliminary validation of the results of our matching methodology. The reader can refer to Fig. 8.5 for an outline of the main steps described below.

The validation process applies to a single probe/CP pair, and considers a specific Target and a specific Prefix. To perform the validation, we consider as first input a sequence of time-labeled traceroute measurements (performed with the standard `traceroute` command that we introduce in Section 3.1). Each measurement consists of a sequence of IP addresses and possibly includes "null hops", i.e., hosts that do not reply to packets sent by the probe.

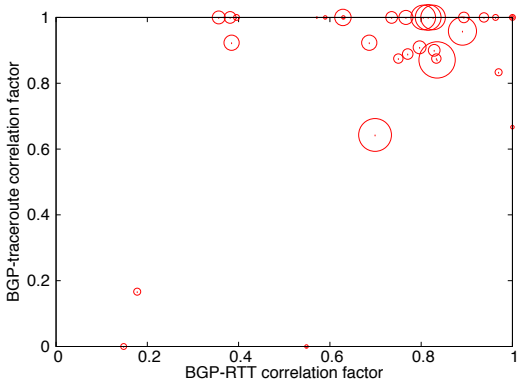
The second input is a partial result of the methodology in Section 8.3. It consists of the complete set of BGP updates collected by the selected CP for the Prefix of interest, where each BGP update is labeled as *valid* if it has been retained after the Preprocessing step, *invalid* otherwise. The last input consists of the correlation factor for the considered probe/CP pair.

The collected data is subject to an IP→AS Mapping step. The goal is to determine what ASes are traversed by each traceroute measurement, in order to match them with BGP routing changes happening at the same time. Methods for IP-to-AS mapping from the literature [MRWK03, MJR+04, ZOW+11] take into account many potential issues, e.g., the presence of IP addresses announced by Internet Exchange Points (IXPs) or peer ASes. In this paper we take a

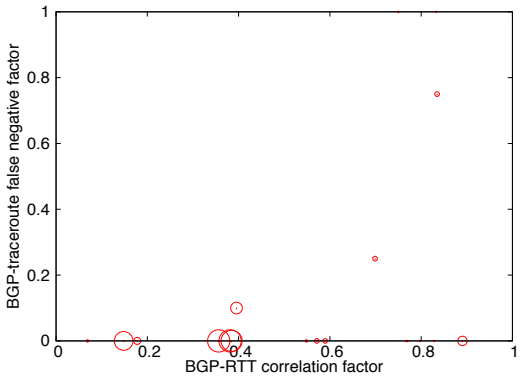
simple approach and retain the assumption supported by the literature that the IP-to-AS mapping derived from BGP routing tables is mostly correct. The main steps of our mapping step can be described as follows. First of all, each private IP address at the beginning of the traceroute path is mapped to the AS of the probe originating the measurement. Each remaining IP address is mapped to the most specific IP prefix containing it that is publicly announced on BGP and seen by RIS collectors. The first information is made available on the RIPE Atlas homepage, while the latter can be retrieved using RIPEstat (see Section 3.2 for details). For each prefix, the AS that announces it is elected as representative for all the IP addresses contained in the prefix. In case there is more than one AS announcing the same prefix, the one seen as the prefix originator by the majority of RIS collectors is elected. In case there is no IP prefix matching the IP address we map the latter to a special number representing an unknown AS. At this point, given the resulting sequence of ASes, identical consecutive AS numbers are collapsed. Finally, AS numbers corresponding to publicly known IXPs are removed from the sequence.

After performing a TIME ALIGNMENT step in the same way as for RTT measurements (see Section 8.3), we put together the traceroute measurements and the BGP routing changes in the MATCHING step. We consider the sequence of all BGP routing changes $u_1, \ldots, u_n$ with related timestamps $t_1, \ldots, t_n$. For each valid BGP routing change u_i $(1 < i < n)$ we consider two time windows $T_< = [t_{i-1}, t_i]$ and $T_> = [t_i, t_{i+1}]$ which are determined as the stability periods before and after u_i. We then obtain $M_<$ and $M_>$, i.e. the two sequences of traceroute measurements respectively falling within $T_<$ and $T_>$. We discard valid BGP updates where either $M_<$ or $M_>$ are empty. Given the last traceroute measurement in $M_<$ with the highest timestamp we call m_{i-1} the sequence of ASes obtained by mapping its IP addresses. Similarly, m_i is the sequence of ASes corresponding to the first traceroute measurement in $M_>$. We combine the above information in a quadruple $q_i = (m_{i-1}, u_{i-1}, m_i, u_i)$ and call Q the set containing all such quadruples.

Finally, the output of the correlation between RTT measurements and BGP routing changes is validated as follows. The analysis is applied to each $q_i \in Q$ computed in the previous step and makes use of the outcome of the BGP-RTT MATCHING step for BGP routing change u_i (see Section 8.3). For each $q_i \in Q$ we simply check whether $u_{i-1} \neq u_i$ and $m_{i-1} \neq m_i$, and mark u_i as "validated" if both conditions hold. Given such information, we can compute two quality measures for the Validation as follows. We first split Q into two sets Q_+ and Q_-: the first contains all the $q_i \in Q$ such that the BGP routing change u_i is correlated with an RTT measurement, while the second is defined



(a)



(b)

Figure 8.6: Validation with traceroute data for target 1005. Circles represent probe/CP pairs. (a) Plot showing the BGP-traceroute correlation factors and the BGP-RTT correlation factors for all the probe/CP pairs with $|Q_+| > 0$. (b) Plot showing the BGP-traceroute false negative factors and the BGP-RTT correlation factors for all the probe/CP pairs with $|Q_-| > 0$. The size of each circle is proportional to $|Q_+|$ and $|Q_-|$, respectively.

as $Q_- = Q \setminus Q_+$. We then define the *BGP-traceroute correlation factor* as the ratio $\frac{|q_i \in Q_+ \text{: } u_i \text{ is validated}|}{|Q_+|}$ and the *BGP-traceroute false negative factor* as the ratio $\frac{|q_i \in Q_- \text{: } u_i \text{ is validated}|}{|Q_-|}$. Intuitively, the first factor measures the precision of our BGP-RTT correlation methodology, while the second gives an indication of how many correspondences between the data plane and the control plane are not captured.

The results of the validation for Target 1005 are in Fig. 8.6. Results for other targets are substantially equivalent. Circles in each plot represent probe/CP pairs. Other features are described in the figure caption. In the left-side plot the vast majority of probe/CP pairs with a high BGP-RTT correlation factor (greater than 0.6) have a high BGP-traceroute correlation factor. That means that BGP routing changes that are well correlated with RTT changepoints are also well correlated with traceroutes. The right-side plot help us understand how many correlations cannot be detected with our methodology. As expected, we see BGP-RTT false negative factors close to zero for probe/CP pairs whose BGP-RTT correlation factor is low. Namely, poorly correlated BGP and RTT data find no evidence of correlation even at the traceroute level.

Clearly, our validation approach is currently limited by the scarce availability of traceroute data. Further, the comparison of routing states before and after BGP events could be implemented at a much finer level of detail. However, our initial positive results motivate further analyses to perform in the future.

Further Analyses and Applications

There are a few other analyses that can be performed based on the results of our methodology. We provide here a few examples, which also support various practical applications. First of all, it is possible to single out *equivalence classes* of CPs and probes that exhibit a highly correlated behavior. CPs in an equivalence class are therefore best candidates to understand, motivate, and, possibly, predict, delay variations recorded by probes in the same class, a useful piece of information for traffic engineering decisions. We make use of equivalence classes in our visualization framework detailed in Chapter 9. Moreover, quantifying the influence that routing updates for a Prefix recorded by different CPs had on the delays towards a Target falling in that Prefix could help determine the network locations where routing changes are less likely to affect the reachability of a delay-sensitive service. In addition, the presence of a correlation between routing updates for a Prefix and delays towards a Target

not comprised in this Prefix can be evidence of a network problem and aid troubleshooting.

8.6 Conclusions and Future Work

In this chapter we describe a methodology to automatically analyze the impact of BGP routing changes on network delays. We prove its effectiveness using publicly available data and propose some interesting analyses based on its outcome.

Lots of facets of our study deserve further investigation. Extending the analysis to different data sources (e.g. those reported in Section 3.2), vantage points, and targets can further validate the effectiveness of our technique. RTT variations could be analyzed with different statistical methods, in order to extract further useful information on network behavior from noise, gaps, or complex patterns in the data. Further, we want to improve our analysis by studying the impact of intra-domain routing changes even on one-way performance indicators, in a controlled scenario where routing events are triggered on purpose. Finally, we envision an extension of the analyses illustrated in Section 8.5.

Chapter 9

Visualizing the Correlation between Inter-domain Routing Changes and Active Probing

The visualization project described in this chapter is paired with the research work presented in Chapter 8. Part of the material is based on [DD13]. The two scientific publications based on the results of the project are listed in Chapter 11. The reader can refer to [DDS12] for additional material that supports the presentation of our work.

9.1 Introduction and State of the Art

Huge quantities of BGP updates and probe-originated active measurements are nowadays publicly available to researchers in time-labeled collections. Examples include respectively the RIPE Routing Information Service and RIPE Atlas, both presented in Section 3.2. As detailed in Chapter 8, the correlation between these two different types of data is an innovative topic in network data analysis that can prove useful for a number of tasks involving troubleshooting and network design.

In terms of visualization, the state of the art flourishes with systems for exploring either of the two types of information. The BGP routing information collected by collector peers is the input for a number of tools. Examples include BGPlay [CDM+05], NetViews [YMMW09], and Cyclops [COZ08]. The reader can refer to Chapter 6, where we report a more detailed analysis of the state

of the art and present in turn our recent contribution, called BGPlay.js. Visualization systems for active measurements also appear in the literature. GTrace [PN99] shows traceroutes enriched with geographical information. The IP-paths are drawn as line-joined nodes on a world map. Walrus is a tool that visualizes large graphs in 3-dimensional space and has been used to visualize round-trip delay [CAI01]. All the active measurement projects listed in Section 3.2 publish simple visualizations of their data, such as interactive plots, graphs, and geographical maps. On the other hand, there is a lack of tools capable of bringing BGP updates and active measurements together, leading to an added value which is greater than the "sum of its parts".

We identify two major features that a visualization framework should address to meet such requirement. First of all, it should provide a clear way to infer the correlation between changes in the inter-domain routing and corresponding variations in the performed active measurements. As explained in Chapter 8, this helps network operators and maintainers accomplish three main tasks: 1. understanding the impact of different AS-paths on the performance and reliability of routes towards critical Internet targets; 2. finding specific performance bottlenecks at the AS level; and 3. checking if Service Level Agreements (SLAs) offered by different upstream ISPs are actually enforced. Further, the framework should highlight the relationship between intra-domain and inter-domain routing, proposing a visual partitioning of the probes hosted by an AS with respect to the border gateways that route their outbound traffic. That is crucial for the purpose of debugging network outages, or to rapidly understand if a sudden change in performance measured by a probe should be ascribed to intra-domain or inter-domain dynamics.

The design of such a visualization tool is challenging for many reasons. First, a purely geographical representation of all the network entities is either impossible or meaningless, because the needed information can be incomplete. Some network entities are tagged with their exact geographical location, while other entities have unknown or approximate coordinates. For example, RIPE Atlas probes belong to the first group, while RIS border gateways are in the second group (see Section 3.2 for details). Further, large transit ASes [DD08] are huge bodies distributed over countries and even continents. Second, both the datasets and the status of their correlation evolve over time. A visualization system should therefore deal efficiently with the underlying dynamics, without affecting the user's mental map. Finally, due to the overwhelming amount of BGP routing information and active measurements available for exploration, the user should be presented with a mechanism to select and navigate between relevant portions of data.

We present a metaphor that combines geographical and non-geographical information and displays the interplay between BGP routing and round-trip delay. We achieve this goal with the following expedients. First of all, we visually separate entities that are geolocated from those that are not. Further, we use animation to convey the temporal evolution of the networks under inspection. Finally, we require the user to focus on a well-defined portion of the data, by selecting a specific AS that contains both border gateways and probes that originate active measurements towards an Internet target. This allows us to cope with the size of the datasets, reducing the complexity of the visualization.

We implemented our metaphor into a prototype, called Hydra. Its first two inputs are respectively the BGP routing information of a set of border gateways in the selected AS, and performance measurements originated by probes in the same AS. The third input consists of metadata on the correlation between the two datasets. More specifically, for each pair composed of a probe and a border gateway, we expect a set of time intervals in which the two are considered as "matched" (i.e. their data exhibits a clear correlation). For simplicity we assume that at any time each probe can be matched with at most one border gateway. The methodology detailed in Chapter 8 is an obvious candidate for the computation of the third input. However, our visualization metaphor can be extended to any alternative methodology that computes the same type of information.

The rest of the chapter is organized as follows. Section 9.2 provides a detailed description of the user requirements and our visualization metaphor. Section 9.3 focuses on technical and algorithmic details. Section 9.4 contains conclusions and ideas for future work.

9.2 Requirements and Interface Design

We started the design of a suitable visualization metaphor by introducing an abstraction of the main features of our data. A *source probe* is a probe that periodically performs round-trip delay measurements against a set of targets. A *source BG* is a border gateway that acts as a collector peer. A *target* is a service publicly reachable on the Internet, subject to periodic measurements originated by source probes. A *source AS* is an AS that hosts at least one source probe and at least one source BG. A *target AS* is an AS that hosts at least one target.

Main Requirements

We take a simple approach and assume that the user specifies three main parameters at the beginning of each interaction with the system. The first input is a source AS (for example an ISP, as detailed in [DD08], or the AS owned by a specific company or university, etc) that the user would like to monitor. The second input is a target that is commonly accessed by computers residing in the specified source AS. The third input is a time interval of interest.

The first requirement consists in having a clear map of both the portion of AS topology and the set of source probes that are directly related to the specified source AS and target. This implies the visualization of the AS-graph induced by the source AS, the target AS, and all the other ASes that appear in at least one of the AS-paths selected by the source BGs. As for the source probes, we focus on displaying basic information like their geolocation and IP prefix.

Once the topology is given, users are interested in studying the evolution of each of the two datasets. For BGP routing data that means showing how the AS-paths selected by each source BG change over time. For source probes, instead, that corresponds to showing fluctuations in the round-trip delay measured by each source probe to reach the selected target.

The third and most challenging requirement consists in capturing the correlation between the two datasets. In particular, the visual representation of each dataset should be augmented with further information that derives from the correlation with the other dataset. For each source BG, that means enriching the BGP routing data with a visual indication of the round-trip delay measured by all the probes that are matched with it. For source probes, instead, that translates to explicitly showing their association to a source BG and consequently to the AS-path that routes their traffic to the target.

Interface Outline

Given the inputs detailed so far, we propose the following metaphor. The source AS is a circle that contains a geographical map. Each source probe is a triangle placed on the map based on its geographic coordinates. Each source BG is a rectangle with a distinctive color that is placed on the external border of the source AS. The target AS is a blue circle. The remaining ASes are dark circles. Peerings between pairs of ASes (i.e. edges of the AS-graph) that can be inferred from at least one AS-path are pictured as dark rectangular bridges. The transition zone between the geographical and the topological

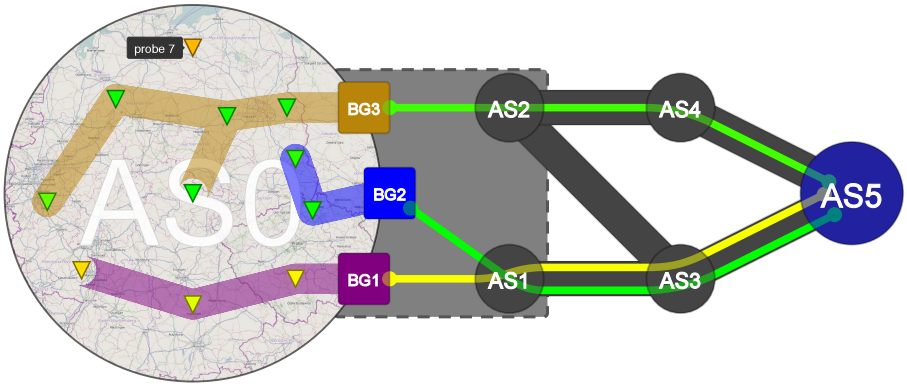
representations is a grey rectangle containing the source BGs and the ASes which have direct peerings with them.

The visualization evolves dynamically, showing the evolution of the two datasets over time. The round-trip delay measured by each source probe to reach the target is encoded in a scale of color from red to green, ranging from large to small round-trip delay values. Each source probe is assigned the appropriate color. When the round-trip delay measured by a source probe changes its color is updated accordingly. The AS-path chosen by each source BG is represented as a curve connecting the source BG and the target AS by following the corresponding route in the AS-graph displayed on screen. When a source BG announces a different AS-path, the corresponding curve is modified with a simple linear morphing procedure in order to reflect the change. Note that although animation is not generally seen [RFF+08] as an appropriate tool for trend visualization, we leverage it to represent changes in the data by means of smooth transitions.

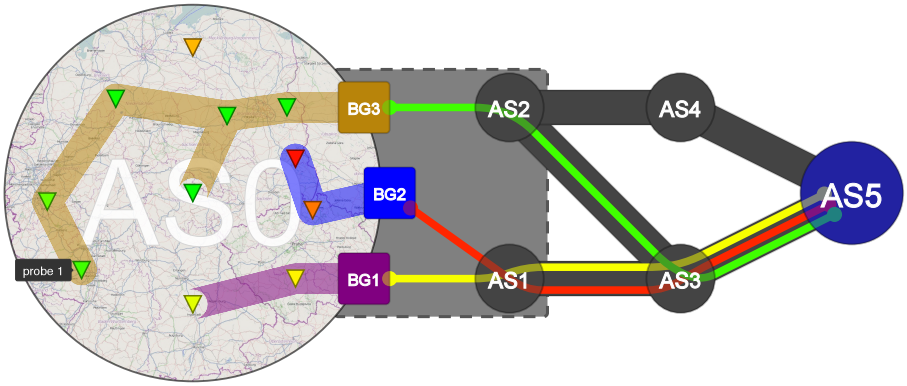
The correlation between the set of source probes and the set of source BGs in the source AS is rendered as follows. We rely on the input correlation as detailed in Section 9.1. Given a source AS containing k source BGs $BG_1, \ldots, BG_k$, the correlation defines a partitioning of the source probes in at most k disjoint clusters $\mu_1, \ldots, \mu_k$. We depict such partitioning by enclosing probes in each cluster μ_i by means of a simple closed region, which is also connected to the corresponding source BG BG_i. Whenever a source probe is no longer associated with a source BG BG_i, the boundary of the cluster μ_i is recomputed and a linear morphing procedure is applied to animate the transformation from the old to the new boundary. AS-paths are enhanced with correlation data. More precisely, each curve originating from a source BG BG_i has a color that reflects the average round-trip delay measured by all the probes in μ_i, using the same scale from red to green used for source probes.

The user can interact with the interface by hovering its main elements, triggering the visualization of a popup enclosing textual information. Note that most of such information is already conveyed by graphical features in our metaphor. Popups are needed to show exact numerical values and to put together data pertaining to the same entities. For each source probe the popup contains the geographical coordinates, the IP prefix that contains its public IP address, and the round-trip delay measured to reach the target. For each source BG the popup shows the IP address and the average round-trip delay.

Figure 9.1 presents two example snapshots of our visualization metaphor. Fig. 9.1(a) shows an example AS-graph. $AS0$ is the source AS: it hosts eleven source probes and three source BGs. Fig. 9.1(b) shows the evolution of the



(a)



(b)

Figure 9.1: Example visualization using our metaphor. Map images © OpenStreetMap contributors, CC BY-SA (`http://www.openstreetmap.org/`, `http://creativecommons.org/licenses/by-sa/2.0/`). (a) Initial state of the visualization. AS0 is the source AS and AS5 is the target AS. AS1, AS2, AS3 and AS4 complete the AS-graph. BG1, BG2 and BG3 are the source BGs of AS0. The AS-paths start from the source BGs and end at AS5. AS0 contains 11 probes, partitioned between the source BGs. Note that probe n.7 does not belong to any cluster. (b) Visualization after the animation. BG3 follows a different path to reach AS5. The AS-path chosen by BG2, the two probes in its cluster, and the probe n.1 have new colors, reflecting changes in the measured round-trip delay. The clusters have changed, with probe n.1 moving from BG1 to BG3.

graph in Fig. 9.1(a), reflecting the state of the underlying data at a different point in time. In particular one source BG follows a new AS-path, the colors representing round-trip delays are different, and the clusters representing the correlation between source probes and source BGs are also subject to updates. The reader can refer to the captions of the two figures for a more detailed explanation.

Note that our metaphor presents a mixture of geographical and topological features. In particular, we do not use real coordinates for transit ASes. Their geographical distribution can be extremely large and often groups of ASes share the same location. Therefore a geographical representation would easily lead to visual cluttering. All the source probes are instead placed at their geographical location, because the visual footprint is much more limited and it is interesting to look for patterns when the source probes are divided into clusters. Finally, we do not show the geographical information of the source BGs, even when it is known. Instead, we prefer to use them as a visual feature to bridge the relationship between the geographical and the topological representation.

The expected size of the input data was carefully taken into account while designing the interface to ensure its readability. In particular the presence of a large number of ASes and source probes may have a negative impact. With respect to the number of ASes, previous research shows that both the average AS-path length [Hus12] and the average number of different AS-paths seen by a single AS to reach a given prefix [SF02] are usually small. Also, the number of AS-paths that are visible at the same time in our interface is bounded by the number of source BGs of the source AS. However, we can safely assume that the number of border gateways is usually small. For example, about 95% of the ASes participating in the RIPE Routing Information System have less than five border gateways peering with route collectors. As for round-trip delay measurements publicly available, note that the number of source probes per AS is also generally small. For example, the statistics reported on the RIPE Atlas homepage show that less than 2% of ASes have more than ten probes. Hence, any dataset with comparable features is suitable for our metaphor. The reader can refer to Section 3.2 for a list of projects including the RIPE Routing Information Service and RIPE Atlas.

9.3 Algorithms and Technical Details

The layout of the AS-graph is obtained by applying a standard graph drawing algorithm suited for layered graphs (see, e.g., the method described by

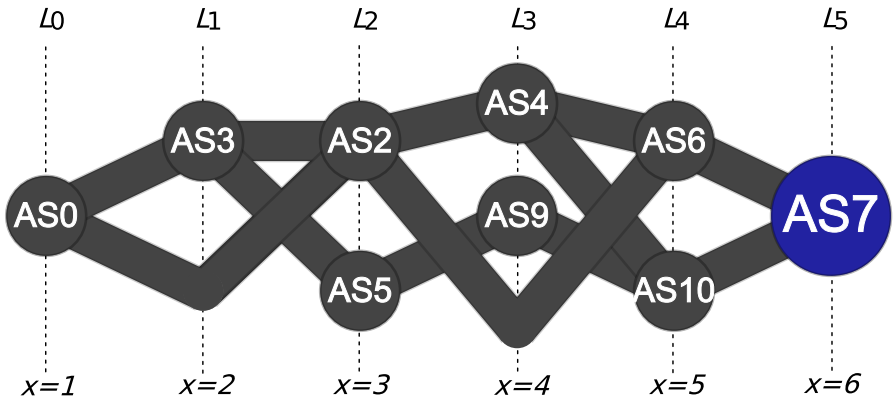
Sugiyama et al. [STT81]). Our procedure works as follows. First, vertices are assigned to vertical layers such that: 1. the source AS and the target AS are the only vertices assigned to the right-most and to the left-most layers, respectively; and 2. the other ASes are assigned internal layers according to a breadth-first visit of the AS-graph starting at the source AS. Second, a permutation of the vertices of the internal layers is performed in an attempt to reduce the number of edge crossings. The result layout meets the requirements of the visualization since it conveys the left-to-right flow of the traffic directed from the source AS to the target AS.

Figure 9.2(a) contains a layered drawing for an example AS-graph. Note that some dummy vertices have been added to make the graph proper, as explained in Section 2.2. The drawing is clearly suboptimal, because many links between pairs of ASes are overlapped (for example, the link between AS0 and AS2 and the one between AS3 and AS5). Figure 9.2(b) shows a different layout for the same AS-graph, where all crossings between pairs of links are avoided thanks to the Sugiyama method.

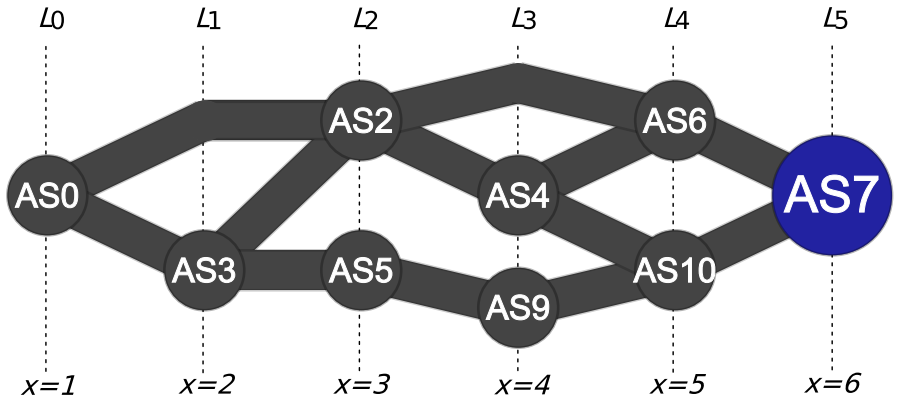
Once the layout for the AS-graph is computed, the drawing of the AS-path chosen by each source BG is routed from the source AS to the target AS inside the AS-graph's edges and vertices. Our approach is very similar to the drawing of "metro maps", which gives rise to a number of interesting optimization problems concerned with the avoidance or reduction of crossings between different lines (see, e.g., the work of Argyriou et al. [ABKS10]). In our setting, each edge of the AS-graph is traversed by parallel AS-path segments, so that no two of them cross inside the edge. Thus, we only allow AS-paths to cross inside vertices. This produces two positive visual effects. First, the resulting drawing is more readable, as each AS-path can be easily followed thanks to its color and to the lack of crossings on the edges. Second, the animation of each AS-path preserves the user's mental map, since it appears as a simple translation of the segments lying in the interior of each edge.

The animation that illustrates each AS-path change is performed as follows. Let p_1 and p_2 be the initial and final AS-path, respectively. First, dummy vertices are introduced in the polyline representation of the shortest of the two AS-paths, so that the number of vertices in p_1 and p_2 coincides. A bijection is found that binds together pairs of vertices of p_1 and p_2 at the same topological distance from the source BG. Second, a linear interpolation is computed between each of such pairs of vertices, transforming the curve for p_1 in the one for p_2. See the work of Colitti et al. [CDM$^+$05] and Bespamyatnikh [Bes02] for known polyline morphing algorithms.

The boundary of each cluster μ_i of source probes is computed as follows.



(a)



(b)

Figure 9.2: Explanation of the graph layout algorithm with an example AS-graph. (a) Each AS is assigned a layer and dummy vertices are added to the graph to make it proper. (b) The layout is modified to avoid crossings between pairs of edges.

First, an *Euclidean minimum spanning tree (EMST)* $T(\mu_i)$ is computed that connects the source probes in μ_i and the source BG BG_i. Shamos et al. [SH75] describe an $O(n \log n)$ time algorithm for computing an *EMST* of a set of n points in the plane. Second, a bottom-up traversal of $T(\mu_i)$ is performed for determining a polyline cycle $B(\mu_i)$ surrounding $T(\mu_i)$ and arbitrarily close to it. That is the boundary of the cluster.

We implemented a prototype, which we call Hydra, based on the described metaphor. The source code is open and freely accessible online, together with supplementary material that highlights the main features of the prototype [DDS12].

9.4 Conclusions and Future Work

This chapter presents a visualization metaphor and implemented a prototype to analyze the correlation between BGP routing data and round-trip delay measurements. We deal with huge datasets containing topological, geographical, and temporal information, often intertwined in complex relationships. To the best of our knowledge our prototype is the first visual tool that aims at unveiling the peculiarities of such an interesting scenario.

We plan to extend our approach to round-trip BGP routing data and Atlas traceroute data, in order to improve our methodology and obtain better and more prominent correlations. On the visualization side, we will evaluate the possibility of taking into account two or more source ASes at the same time. Such an improvement would allow the end user to compare and assess the quality of the service offered by different upstream ISPs. We will also optimize the algorithms for the visualization and the animation of data. We plan to study how to draw AS-paths reducing mutual overlaps and crossings. Finally, we will focus on different ways to represent clusters of probes, in order to achieve a smoother animation.

Part V

Publications and Bibliography

Chapter 10

Other Research Activities

This chapter presents additional research activities that took place mostly during the first two years of the doctorate program. The following sections are not part of the previous chapters of this thesis because they have little or no overlap with the topic of Internet visualization. Each research topic presented in this chapter lead to at least one scientific publication. The reader can refer to Chapter 11 for additional details.

10.1 Analysis of Country-wide Internet Outages caused by Censorship

In the first months of 2011, Internet communications were disrupted in several North African countries in response to civilian protests and threats of civil war. As already mentioned in Chapter 6, we analyzed episodes of these disruptions in Egypt and Libya relying on multiple sources of large-scale data already available to academic researchers: 1. BGP inter-domain routing control plane data; 2. unsolicited data plane traffic to unassigned address space; 3. active macroscopic traceroute measurements; 4. RIR delegation files; and 5. MaxMind's geolocation database. We used the latter two data sets to determine which IP address ranges were allocated to entities within each country. We mapped these IP addresses of interest to BGP-announced IP prefixes and origin ASes using publicly available BGP data repositories in the U.S. and Europe. We then analyzed observable activity related to these sets of prefixes and ASes throughout the censorship episodes.

Using both control plane and data plane data sets in combination allowed

us to narrow down which forms of Internet access disruption were implemented in a given region over time. Among other insights, we detected what we believe were Libya's attempts to test firewall-based blocking before they executed more aggressive BGP-based disconnection. Our methodology could be used, and automated, to detect outages or similar macroscopically disruptive events in other geographic or topological regions.

10.2 Universal Point Sets for Classes of Planar Graphs

A point set $P \subseteq \mathbb{R}^2$ is *universal* for a class of graphs G if every graph of G has a planar straight-line embedding into P. The well-known problem of finding the size of the smallest universal point set for planar graphs is still open. However, researchers have tackled the problem by finding small point sets for specific classes of planar graphs: for example, outerplanar graphs (i.e., graphs with all their vertices on the external face) can be drawn with any point set of size n.

In our research work we focused on the class of simply-nested n-vertex planar graphs. These graphs are a special kind of graphs with k-outerplanar embeddings, defined as embeddings in which removing the vertices of the outer face yields a $(k-1)$-outerplanar embedding (where 1-outerplanar is an outerplanar embedding). Simply-nested graphs are k-outerplanar graphs where all the levels are chordless cycles.

We proved that there exists a $O(n(\frac{\log n}{\log \log n})^2)$ size universal point set for simply-nested graphs. Our result is in turn based on the construction of a point set of size $8n+8$ for simply-nested graphs for which the number of vertices on each layer is known in advance. The generalization to all simply-nested graphs is obtained exploiting combinatorial properties of the relationship between the number of vertices on each layer and the total number of vertices.

10.3 Area Requirements of Euclidean Minimum Spanning Trees

A *Euclidean minimum spanning tree* (MST) of a set P of points in the plane is a tree with a vertex in each point of P and with minimum total edge length. Euclidean minimum spanning trees have several applications in computer science and hence they have been deeply investigated from a theoretical point of view.

An *MST embedding* of a tree T is a plane embedding of T such that the MST of the points where the vertices of T are drawn coincides with T. Several

results are known related to such a problem. For example, no tree having a vertex of degree at least 7 admits an MST embedding. Further, deciding whether a tree with degree 6 admits an MST embedding is computationally hard.

Monma and Suri [MS92] prove that every tree of maximum degree 5 admits an MST embedding in the plane that requires exponential area, and conjecture that their result can be improved to only use polynomial area for the construction of the tree. In our research activity we proved that there exist n-vertex trees of maximum degree 5 requiring exponential area in any MST embedding.

10.4 Semantic Word Cloud Representations

Word clouds are popular ways to visualize text. They provide an appealing way to summarize the content of a webpage, a research paper, or a political speech. Often such visualizations are used to contrast two documents; for example, word cloud visualizations of the speeches given by the candidates in the 2008 US Presidential elections were used to draw sharp contrast between them in the popular media.

While some of the more recent word cloud visualization tools aim at incorporating semantics in the layout, none provides any guarantees about the quality of the layout in terms of semantics. We propose a mathematical model of the problem, via a simple edge-weighted graph. The vertices in the graph are the words in the document. The edges in the graph correspond to semantic relatedness, with weights corresponding to the strength of the relation. Each vertex must be drawn as a box with fixed dimensions, related to the importance of the word. The goal is to "realize" as many edges as possible, by contacts between their corresponding rectangles.

In our research work we proved that the problem of realizing *all* the edges in the input graph is computationally hard, even when applied to simple classes of graphs like trees and stars. We then considered the optimization problem where each edge has a weight and the task is to maximize the sum of the weights of the realized edges. We found approximations for several classes of graphs that improve upon the best existing heuristics.

Chapter 11

Publications

Journal Publications

1. Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, Antonio Pescapé. Analysis of Country-wide Internet Outages Caused by Censorship. *IEEE/ACM Transactions on Networking*. To appear, 2014.

2. Patrizio Angelini, Till Bruckdorfer, Marco Chiesa, Fabrizio Frati, Michael Kaufmann, Claudio Squarcella. On the Area Requirements of Euclidean Minimum Spanning Trees. *Computational Geometry: Theory and Applications (Special Issue on Selected Papers from WADS '11)*, volume 47, number 2, part B, pages 200-213, 2014.

3. Giordano Da Lozzo, Giuseppe Di Battista, Claudio Squarcella. Visual Discovery of the Correlation between BGP Routing and Round-Trip Delay Active Measurements. *Computing*, volume 96, issue 1, pages 67-77, 2014.

4. Giuseppe Di Battista, Claudio Squarcella, Wolfgang Nagele. How to Visualize the K-root Name Server. *Journal of Graph Algorithms and Applications*, volume 16, number 3, pages 675-699, 2012.

Conference Publications

1. Lukas Barth, Sara Irina Fabrikant, Stephen Kobourov, Anna Lubiw, Martin Nöllenburg, Yoshio Okamoto, Sergey Pupyrev, Claudio Squarcella,

Torsten Ueckerdt, Alexander Wolff. Semantic Word Cloud Representations: Hardness and Approximation Algorithms. *In 11th Latin American Theoretical Informatics Symposium (LATIN'14), Springer-Verlag, Lecture Notes in Computer Science*. To appear, 2014.

2. Massimo Rimondini, Claudio Squarcella, Giuseppe Di Battista. Towards an Automated Investigation of the Impact of BGP Routing Changes on Network Delay Variations. *In 15th Passive and Active Measurement Conference (PAM 2014)*. To appear, 2014.

3. Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, Claudio Squarcella. Dynamic Traceroute Visualization at Multiple Abstraction Levels. *In 21st International Symposium on Graph Drawing (GD '13)*, volume 8242 of Lecture Notes in Computer Science, pages 500-511, 2013.

4. Giordano Da Lozzo, Giuseppe Di Battista, Claudio Squarcella. Visual Discovery of the Correlation between BGP Routing and Round-Trip Delay Active Measurements. *In 1st IMC Workshop on Internet Visualization (WIV 2012)*, 2012.

5. Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, Antonio Pescapé. Analysis of Country-wide Internet Outages Caused by Censorship. *In 2011 Internet Measurement Conference (IMC '11)*, ACM, pages 1-18, 2011.

6. Giuseppe Di Battista, Claudio Squarcella, Wolfgang Nagele. How to Visualize the K-Root Name Server (Demo). *In 19th International Symposium on Graph Drawing (GD '11)*, volume 7034 of Lecture Notes in Computer Science, pages 191-202, 2011.

7. Patrizio Angelini, Giuseppe Di Battista, Michael Kaufmann, Tamara Mchedlidze, Vincenzo Roselli, Claudio Squarcella. Small Point Sets for Simply-Nested Planar Graphs. *In 19th International Symposium on Graph Drawing (GD '11)*, volume 7034 of Lecture Notes in Computer Science, pages 75-85, 2011.

8. Patrizio Angelini, Till Bruckdorfer, Marco Chiesa, Fabrizio Frati, Michael Kaufmann, Claudio Squarcella. On the Area Requirements of Euclidean Minimum Spanning Trees. *In 12th Algorithms and Data Structures Symposium (WADS '11)*, volume 6844 of Lecture Notes in Computer Science, pages 25-36, 2011.

Technical Reports

1. Lukas Barth, Sara Irina Fabrikant, Stephen Kobourov, Anna Lubiw, Martin Nöllenburg, Yoshio Okamoto, Sergey Pupyrev, Claudio Squarcella, Torsten Ueckerdt, Alexander Wolff. Semantic Word Cloud Representations: Hardness and Approximation Algorithms. *Technical Report arXiv:1311.4778*, Cornell University, 2013.

2. Massimo Rimondini, Claudio Squarcella, Giuseppe Di Battista. From BGP to RTT and Beyond: Matching BGP Routing Changes and Network Delay Variations with an Eye on Traceroute Paths. *Technical Report arXiv:1309.0632*, Cornell University, 2013.

3. Patrizio Angelini, Till Bruckdorfer, Marco Chiesa, Fabrizio Frati, Michael Kaufmann, Claudio Squarcella. On the Area Requirements of Euclidean Minimum Spanning Trees. *Technical Report RT-DIA-183-2011*, Department of Computer Science and Automation, Roma Tre University, 2011.

Bibliography

[AAD+08] Gennady Andrienko, Natalia Andrienko, Jason Dykes, Sara Irina Fabrikant, and Monica Wachowicz. Geovisualization of dynamics, movement and change: Key issues and developing approaches in visualization research. *Information Visualization*, 7(3):173–180, June 2008.

[ABKS10] Evmorfia N. Argyriou, Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. On Metro-Line Crossing Minimization. *J. Graph Algorithms Appl.*, 14(1):75–96, 2010.

[AJSS11] Mohammad Akbari Jokar and Ali Shoja Sangchooli. Constructing a Block Layout by Face Area. *The International Journal of Advanced Manufacturing Technology*, 54:801–809, 2011.

[Apa12] Apache Software Foundation. Apache Commons Graph. `http://commons.apache.org`, 2012.

[Aug03] Bjorn Augustsson. Xtraceroute. `http://www.dtek.chalmers.se/{\texttildelow}d3august/xt/index.html`, 2003.

[Bac07] C. Bachmaier. A radial adaptation of the sugiyama framework for visualizing hierarchical information. *IEEE Trans. on Visualization and Computer Graphics*, 13(3):583–594, 2007.

[BBBL11] Ilya Boyandin, Enrico Bertini, Peter Bak, and Denis Lalanne. Flowstrates: An approach for visual exploration of temporal origin-destination data. In *Proceedings of the 13th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'11,

pages 971–980, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[BDM02] Giuseppe Battista, Walter Didimo, and A. Marcandalli. Planarization of clustered graphs. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *LNCS*, pages 60–74. Springer Berlin Heidelberg, 2002.

[Bes02] Sergei Bespamyatnikh. An Optimal Morphing Between Polylines. *Int. J. Comput. Geometry Appl.*, 12(3):217–228, 2002.

[BK73] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16:575–577, September 1973.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *JCSS*, 13(3):335 – 379, 1976.

[BN93] M. Basseville and I.V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., 1993.

[BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[BRV11] Therese Biedl and Lesvia Ruiz Velázquez. Orthogonal Cartograms with Few Corners Per Face. In *Algorithms and Data Structures*, volume 6844 of *Lecture Notes in Computer Science*, pages 98–109. Springer Berlin / Heidelberg, 2011.

[BSV11] Kevin Buchin, Bettina Speckmann, and Kevin Verbeek. Flow map layout via spiral trees. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2536–2544, December 2011.

[BV10] Therese Biedl and Lesvia Elena Ruiz Velázquez. Drawing Planar 3-Trees with Given Face-Areas. In David Eppstein and Emden R. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 316–322. Springer Berlin / Heidelberg, 2010.

[CAI00] CAIDA. Ipv4 and ipv6 as core. `http://www.caida.org/research/topology/as_core_network/`, 2000.

[CAI01] CAIDA. Round-Trip Time Internet Measurements from CAIDA's Macroscopic Internet Topology Monitor. `http://www.caida.org/research/performance/rtt/walrus0202`, 2001.

[CAI07] CAIDA. Archipelago. `http://www.caida.org/projects/ark/`, 2007.

[Can12] Massimo Candela. Adaptive and responsive web-oriented visualization of evolving data: The interdomain routing case. Master's thesis, Roma Tre University, 2012.

[CBD02] Chen-Nee Chuah, Supratik Bhattacharyya, and Christophe Diot. Measuring I-BGP updates and their impact on traffic. Technical Report TR02-ATL-051099, Sprint ATL, 2002.

[CDBDBS13] Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, and Claudio Squarcella. TPlay Homepage. http://www.dia.uniroma3.it/~compunet/projects/tplay, 2013.

[CDBS13] Massimo Candela, Giuseppe Di Battista, and Claudio Squarcella. BGPlay.js Homepage. `http://bgplayjs.com/`, 2013.

[CDM+05] Lorenzo Colitti, Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Visualizing Interdomain Routing with BGPlay. *Journal of Graph Algorithms and Applications, Special Issue on the 2003 Symposium on Graph Drawing, GD '03*, 9(1):117–148, 2005.

[CDM+06] Pier Francesco Cortese, Giuseppe Di Battista, Antonello Moneta, Maurizio Patrignani, and Maurizio Pizzonia. Topographic Visualization of Prefix Propagation in the Internet. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):725–732, 2006.

[CGP98] Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Planar Map Graphs. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98. ACM, 1998.

[CGP06] Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Recognizing Hole-Free 4-Map Graphs in Cubic Time. *Algorithmica*, 45(2):227–262, 2006.

[COZ08] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: the AS-level Connectivity Observatory. *SIGCOMM Comput. Commun. Rev.*, 38(5):5–16, September 2008.

[dBMS06] Mark de Berg, Elena Mumford, and Bettina Speckmann. Optimal BSPs and Rectilinear Cartograms. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, GIS '06, pages 19–26, New York, NY, USA, 2006. ACM.

[DBN88] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(6):1035–1046, 1988.

[DBSN12] Giuseppe Di Battista, Claudio Squarcella, and Wolfgang Nagele. Visual-K Homepage. http://www.dia.uniroma3.it/~squarcel/visual-k, 2012.

[DD08] Amogh Dhamdhere and Constantine Dovrolis. Ten Years in the Evolution of the Internet Ecosystem. In *Proceedings of the 8th ACM SIGCOMM IMC*, 2008.

[DD13] Valentino Di Donato. Combined visualization of bgp routing changes and round-trip delay measurements. Master's thesis, Roma Tre University, 2013.

[DDS12] Giordano Da Lozzo, Giuseppe Di Battista, and Claudio Squarcella. Visual Discovery of the Correlation between BGP Routing Changes and Round-Trip Delay Active Measurements. `http://dia.uniroma3.it/~compunet/projects/hydra`, 2012.

[DETT98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1998.

[DLR11] Walter Didimo, Giuseppe Liotta, and Salvatore A. Romeo. Topology-Driven Force-Directed Algorithms. In *Proceedings of the 18th international conference on Graph drawing*, GD'10, pages 165–176, Berlin, Heidelberg, 2011. Springer-Verlag.

[EDG+08] N. Elmqvist, Thanh-Nghi Do, H. Goodell, N. Henry, and J. Fekete. Zame: Interactive large-scale graph visualization. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pages 215–222, 2008.

[EHK+04] Cesim Erten, Philip Harding, Stephen Kobourov, Kevin Wampler, and Gary Yee. GraphAEL: Graph Animations with Evolving Layouts. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 98–110. Springer Berlin / Heidelberg, 2004.

[FB04] Michael Forster and Christian Bachmaier. Clustered level planarity. In Peter Emde Boas, Jaroslav Pokorný, Mriá Bieliková, and Július Štuller, editors, *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *LNCS*, pages 218–228. Springer Berlin Heidelberg, 2004.

[FS05] Sara Irina Fabrikant and André Skupin. Cognitively Plausible Information Visualization. *Exploring Geovisualization*, pages 667–690, 2005.

[GHKK10] E. Gansner, Y. Hu, M. Kaufmann, and S. Kobourov. Optimal Polygonal Representation of Planar Graphs. In *LATIN 2010: Theoretical Informatics*, volume 6034 of *Lecture Notes in Computer Science*, pages 417–432. Springer Berlin / Heidelberg, 2010.

[GJ83] Michael R. Garey and David S. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.

[GM03] Carsten Gutwenger and Petra Mutzel. An Experimental Study of Crossing Minimization Heuristics. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2003.

[GN04] M. T. Gastner and M. E. J. Newman. Diffusion-based Method for Producing Density-equalizing Maps. *Proceedings of the National Academy of Sciences of the United States of America*, 101(20):7499–7504, May 2004.

[Guo09] Diansheng Guo. Flow mapping and multivariate visualization of large spatial interaction data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1041–1048, November 2009.

[Har02] Jon Harris. A graphical Java implementation of PQ-Trees. `http://www.jharris.ca/portfolio/docs/pqtreereport.pdf`, 2002.

[HFc08] Bradley Huffaker, Marina Fomenkov, and kc claffy. Influence Maps - A Novel 2-D Visualization of Massive Geographically Distributed Data Sets. *Internet Protocol Forum*, October 2008.

[HM07] A. Hernandez and E. Magana. One-way delay measurement and characterization. In *Proc. ICNS*, ICNS '07, pages 114–, Washington, DC, USA, 2007. IEEE Computer Society.

[Hok08] Vidar Hokstad. Traceviz: Visualizing traceroute output with graphviz. `http://www.hokstad.com`, 2008.

[Hol06] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[Hus12] Geoff Huston. Potaroo. `www.potaroo.net`, 2012.

[IS06] Ryo Inoue and Eihan Shimizu. A New Algorithm for Continuous Area Cartogram Construction with Triangulation of Regions and Restriction on Bearing Changes of Edges. *Cartography and Geographic Information Science*, 33(2):115–125, 2006.

[KFE12] R. Killick, P. Fearnhead, and I.A. Eckley. Optimal detection of changepoints with a linear computational cost. *Jour. Amer. Stat. Assoc.*, 107(500):1590–1598, 2012.

[Kis12] Gavin Kistner. Generating visually distinct colors. http://phrogz.net/css/distinct-colors.html, 2012.

[KN07] Akifumi Kawaguchi and Hiroshi Nagamochi. Orthogonal Drawings for Plane Graphs with Specified Face Areas. In *Proceedings of the 4th international conference on Theory and applications of models of computation*, TAMC'07, pages 584–594, Berlin, Heidelberg, 2007.

[KS96] D.J. Ketchen and C.L. Shook. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic Mgmt. Journal*, 17(6):441–458, 1996.

[Max02] MaxMind. MaxMind - IP Geolocation and Online Fraud Prevention. `http://www.maxmind.com/en/home`, 2002.

[MGW+11] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert. Rapid detection of maintenance induced changes in service performance. In *Proc. CoNEXT*, CoNEXT '11, pages 13:1–13:12, New York, NY, USA, 2011. ACM.

[MJR+04] Z.M. Mao, D. Johnson, J. Rexford, J. Wang, and R. Katz. Scalable and accurate identification of AS-level forwarding paths. In *Proc. INFOCOM*, volume 3, pages 1605–1615 vol.3, 2004.

[MKH11] Daisuke Mashima, Stephen Kobourov, and Yifan Hu. Visualizing Dynamic Data with Maps. In *Proc. 4th IEEE Pacific Visualization Symposium*, March 2011.

[MNKF90] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing Minimization in Linear Embeddings of Graphs. *IEEE Transactions on Computers*, 39(1):124 –127, jan 1990.

[MRWK03] Z.M. Mao, J. Rexford, J. Wang, and R.H. Katz. Towards an accurate AS-level traceroute tool. In *Proc. SIGCOMM*, SIGCOMM '03, pages 365–378, New York, NY, USA, 2003. ACM.

[MS92] Clyde Monma and Subhash Suri. Transitions in Geometric Minimum Spanning Trees. *Discrete and Computational Geometry*, 8(1):265–293, 1992.

[MSG+10] A.A. Mahimkar, H.H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons. Detecting the performance impact of upgrades in large operational networks. In *Proc. SIGCOMM*, SIGCOMM '10, pages 303–314, New York, NY, USA, 2010. ACM.

[NB13] Arlind Nocaj and Ulrik Brandes. Stub bundling and confluent spirals for geographic networks. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 388–399. Springer International Publishing, 2013.

[OBSC00] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[OCLZ08] Ricardo Oliveira, Ying-Ju Chi, Mohit Lad, and Lixia Zhang. Cyclops: the internet as-level observatory. *NANOG43*, June 2008.

[oMCSDNRL08] University of Memphis Computer Science Department's Networking Research Lab. Netviews, 2008.

[OR00] Min Ouyang and Peter Z. Revesz. Algorithms for Cartogram Animation. In *Proceedings of the 2000 International Symposium on Database Engineering & Applications*, IDEAS '00, pages 231–235, Washington, DC, USA, 2000. IEEE Computer Society.

[Piz07] Maurizio Pizzonia. From bgplay to ibgplay: Graphical inspection of your routing data. In *55th Rseaux IP Europens Meeting (RIPE 55)*, 2007.

[PN99] Ram Periakaruppan and Evi Nemeth. Gtrace - a graphical traceroute tool. In *Proc. 13th USENIX conference on System administration*, pages 69–78. USENIX Association, 1999.

[PZMH07] H. Pucha, Y. Zhang, Z.M. Mao, and Y.C. Hu. Understanding network delay changes caused by routing events. In *Proc. SIGMETRICS*, SIGMETRICS '07, pages 73–84, New York, NY, USA, 2007. ACM.

[Rai05] Marcus Raitner. Visual navigation of compound graphs. In Jnos Pach, editor, *Graph Drawing*, volume 3383 of *LNCS*, pages 403–413. Springer Berlin Heidelberg, 2005.

[Rap92] David Rappaport. A convex hull algorithm for discs, and applications. *Computational Geometry*, 1(3):171 – 187, 1992.

[RFF+08] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. Effectiveness of Animation in Trend Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1325–1332, November 2008.

[RIP01] RIPE NCC. Routing Information Service. `http://www.ripe.net/data-tools/stats/ris/`, 2001.

[RIP10] RIPE NCC. RIPE Atlas. `http://atlas.ripe.net/`, 2010.

[RIP11] RIPE NCC. RIPEstat. `https://stat.ripe.net/`, 2011.

[RMK+96] Yakov Rekhter, Robert Moskowitz, Daniel Karrenberg, Geert Jan de Groot, and Eliot Lear. RFC 1918. address allocation for private internets. `http://www.ietf.org/rfc/rfc1918.txt`, 1996.

[RMN09] Md. Saidur Rahman, Kazuyuki Miura, and Takao Nishizeki. Octagonal Drawings of Plane Graphs with Prescribed Face Areas. *Computational Geometry: Theory and Applications*, 42:214–230, April 2009.

[Rob12] Maxwell J. Roberts. *Underground Maps Unravelled - Explorations in Information Design*. Maxwell J. Roberts, 2012.

[RT07] René Reitsma and Stanislav Trubin. Information Space Partitioning using Adaptive Voronoi Diagrams. *Information Visualization*, 6:123–138, May 2007.

[Sam08] SamKnows. SamKnows - Accurate broadband performance information for consumers, governments and ISPs. `http://www.samknows.com/broadband/`, 2008.

[San96] Georg Sander. Layout of compound directed graphs. Technical report, FB Informatik, Universitat Des Saarlandes, 1996.

[San99] G. Sander. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 217(2):175 – 214, 1999.

[Sco11] Monitor Scout. Monitor Scout Traceroute. `http://tools.monitorscout.com/traceroute/`, 2011.

[SF02] Georgos Siganos and Michalis Faloutsos. Bgp routing: A study at large time scale. In *in Proc. IEEE Global Internet*, 2002.

[SH75] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *FOCS*, pages 151–162, 1975.

[SM91] K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Trans. on Systems, Man and Cybernetics*, 21(4):876–892, 1991.

[Squ10] Claudio Squarcella. Historical bgplay, 2010.

[STT81] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical Systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.

[Tan02] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[TG13] Efthymia Tsamoura and Anastasios Gounaris. Incorporating change detection in network coordinate systems for large data transfers. In *Proc. PCI*, PCI '13, pages 55–62, New York, NY, USA, 2013. ACM.

[The05] The Apache Software Foundation. Apache Hadoop. `http://hadoop.apache.org/`, 2005.

[Tho98] Mikkel Thorup. Map Graphs In Polynomial Time. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, 1998.

[Uni97] University of Oregon. RouteViews Project. `http://www.routeviews.org`, 1997.

[Vis97] Visualware. VisualRoute. `http://www.visualroute.com/`, 1997.

[vKS07] Marc van Kreveld and Bettina Speckmann. On Rectangular Cartograms. *Computational Geometry: Theory and Applications*, 37:175–187, August 2007.

[WDS10] J. Wood, J. Dykes, and A. Slingsby. Visualisation of origins, destinations and flows with od maps. *The Cartographic Journal*, 47(2):117–129, 2010.

[WMW+06] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. *SIGCOMM Comput. Commun. Rev.*, 36(4):375–386, 2006.

[WSD11] Jo Wood, Aidan Slingsby, and Jason Dykes. Visualizing the dynamics of london's bicycle-hire scheme. *Cartographica*, 46(4):239–251, 2011.

[YFDH01] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. Animated exploration of dynamic graphs with radial layout. In *Proc. INFOVIS'01*. IEEE Computer Society, 2001.

[YMMW09] He Yan, Dan Massey, Ernest McCracken, and Lan Wang. BGPMon and NetViews: Real-Time BGP Monitoring System. IEEE INFOCOM, demo, 2009.

[ZMW07] Ying Zhang, Z.M. Mao, and J. Wang. A framework for measuring and predicting the impact of routing changes. In *Proc. INFOCOM*, pages 339–347, New York, NY, USA, August 2007. ACM.

[ZOW+11] Y. Zhang, R. Oliveira, Y. Wang, S. Su, B. Zhang, J. Bi, H. Zhang, and L. Zhang. A framework to quantify the pitfalls of using traceroute in AS-level topology measurement. *Jour. Sel. Areas Comm.*, 29(9):1822–1836, 2011.