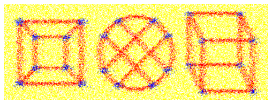

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 16, no. 3, pp. 675–699 (2012)
DOI: 10.7155/jgaa.00267

How to Visualize the K-root Name Server

Giuseppe Di Battista[1] *Claudio Squarcella*[1] *Wolfgang Nagele*[2]

[1]Dipartimento di Informatica ed Automazione,
Roma Tre University
[2]RIPE NCC, Amsterdam, The Netherlands

Abstract

We present a novel paradigm to visualize the evolution of the service provided by one of the most popular root name servers, called K-root, operated by the RIPE Network Coordination Centre (RIPE NCC) and distributed in several locations (instances) worldwide. Our approach can be used to either monitor what happened during a prescribed time interval or observe the status of the service in near real-time. We visualize how and when the clients of K-root migrate from one instance to another, how the workload associated with each instance changes over time, and what are the instances that contribute to offer the service to a selected Internet Service Provider. In addition, the visualization aims at distinguishing usual from unusual operational patterns. This helps not only to improve the quality of the service but also to spot security-related issues and to investigate unexpected routing changes.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
December 2011	February 2012	March 2012	May 2012	May 2012

Published:
September 2012

Article type:
Regular paper

Communicated by:
M. van Kreveld and B. Speckmann

Partially supported by the ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations" and by the MIUR of Italy, under project AlgoDEEP, prot. 2008TFBWL4.

E-mail addresses: gdb@dia.uniroma3.it (Giuseppe Di Battista) squarcel@dia.uniroma3.it (Claudio Squarcella) wnagele@ripe.net (Wolfgang Nagele)

1 Introduction

A computer sends a query to a *name server* every time it needs to know the IP address corresponding to a *domain name*. Hence, all Internet Server Providers (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*. Such process computes an answer for the query by iteratively querying other name servers. This often implies sending a query to special name servers called *root name servers*, or simply *root servers*. For this reason root servers are a critical part of the Internet. They receive hundreds of thousands of queries per second and must answer immediately. 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. Name servers select their favorite root servers according to various query optimization policies.

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 in turn on the current status of the Internet routing to determine the instance which is most appropriate, i.e. topologically nearest. Hence it is possible that consecutive queries issued by a name server to the same root server, say K-root, are received by different instances, depending on the current status of the routing. This has consequences both from the point of view of the name server (i.e. the client of the root 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.

The purpose of this work is to visualize the evolution of the service provided by one of the most popular root servers, called K-root. It is operated by the RIPE Network Coordination Centre (RIPE NCC), one of five Regional Internet Registries in the world providing Internet resource allocations, registration services and coordination activities that support the operation of the Internet globally. Our approach can be used to either monitor what happened during a prescribed time interval or to observe 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 a certain ISP.

This helps not only to improve the quality of the service but also to spot security-related issues and to investigate unexpected routing changes.

The paper is organized as follows. In Section 2 we explain the requirements and discuss the adopted visualization metaphor. In Section 3 we present the layout algorithm used in our approach. In Section 4 we describe the design of a prototype tool based on our framework and the technical challenges faced during implementation. Section 5 presents the feedback collected from the users. In Section 6 we compare our approach with the existing literature. Concluding remarks and future work are in Section 7.

2 Selecting a Metaphor for User Requirements

Like many distributed information services, K-root challenges its operators with many important questions. *Performance*: how to optimize load balancing between instances? *Reliability*: what happens if some of the instances are faulty? *Design*: what is the best topology for existing instances, and where should new instances be deployed? *Routing*: how does the system react to changes and fluctuations in the (inter-domain) routing? *Security*: how does the system react to external attacks?

A crucial requirement, that is related to all the above questions, consists in understanding how and when clients *migrate* from one instance to another. The concept of migration can be defined as follows. Let u, v be a pair of instances. We say that a client *migrates* from u to v during an interval of time t', t'' $(t' < t'')$ if its last request of service before time t' is asked to u and its last request of service before time t'' is asked to v.

If many clients migrate from an instance to another one, the system could suffer from unbalancing, stressing over its capacity the instance that accepts the migrating clients. Also, in case of a routing change, a massive migration can happen reflecting the evolution of the topology. Further, some of the attacks that can be performed against root servers (like distributed DoS) can be mitigated by suitably forcing the migration of part of the clients to specific servers. To give an idea of the migration phenomenon, in 24 hours of normal operation about 50 thousand clients issue a service request to more than one instance.

In this context the length of the time interval t', t'' plays an important role in observing migrations: 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. That is true because:

1. Routing events and dynamics have compatible times and
2. Migrations lasting only a handful of minutes are generally uninteresting for operators.

Furthermore, since migrations are not all the same, a second requirement consists in clearly separating them into classes. There are pairs u, v of instances

such that migrating from u to v or vice versa is considered *usual* by the operators. This is true for example 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. There are other migrations that are instead considered *unusual*, for example those involving 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; however 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: detecting them on time 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: Where is the routing more unstable? What instances exchange clients more frequently between each other? Where should the next instance of K-root be deployed?

A third important requirement is to understand the workload of each instance. This can be expressed in many different ways: we identified two main measures, i.e. the number of clients served by each instance and the total number of queries received by each instance. Both measures are clearly related to migration patterns and are subject to change over time. Operators need to get an immediate perception of the workload of each instance and the effect of migrations on it, i.e. how clients and queries redistribute on different instances and what patterns arise. For simplicity, throughout the paper we will assume that the indicator for the workload of instances is represented with the number of clients, although other measures can be easily adopted.

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 Internet Service Provider (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 happening over time, implying the design of a dynamic monitoring tool. More specifically, such a tool should allow for both offline analysis, i.e. within a prescribed time interval in the past, and near real-time monitoring. We can consider this as a last, implicit requirement influencing the design and architecture of the system as a whole.

Fig. 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

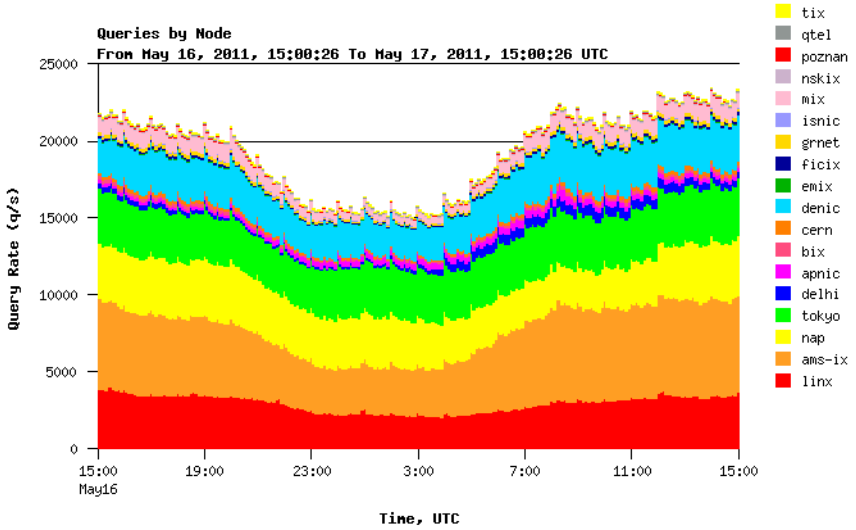


Figure 1: Chart showing the query distribution among all the instances of K-root.

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. Our study started with a formalization step, where we aimed at finding a clear structure for the information collected during our interactions with the RIPE NCC. In particular we tackled the distinction between usual and unusual migrations, agreeing upon a graph structure to represent the domain which we call *migration graph*. Its vertices are the instances, while there is an edge (u, v) if migrating from u to v or vice versa is considered usual. Hence, given a pair u, v of instances subject to a migration, if (u, v) is an edge the migration is considered *usual*, otherwise it is considered *unusual*.

In order to help answering all these questions, we designed an interface that supports the work of the employees of the RIPE NCC. The choice of the visualization metaphor and the interaction features is the result of an intensive discussion with such employees. 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 usually exchanging clients, i.e. 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, which we summarize as follows. 1. 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. 2. As the cognitive study presented in Section 6 points out, geography can be regarded as a good way to describe abstract entities, quantitative informations and relationships between elements. 3. Given the traditional meaning of the concept of *migration*, we found the map metaphor to be quite natural and well suited.

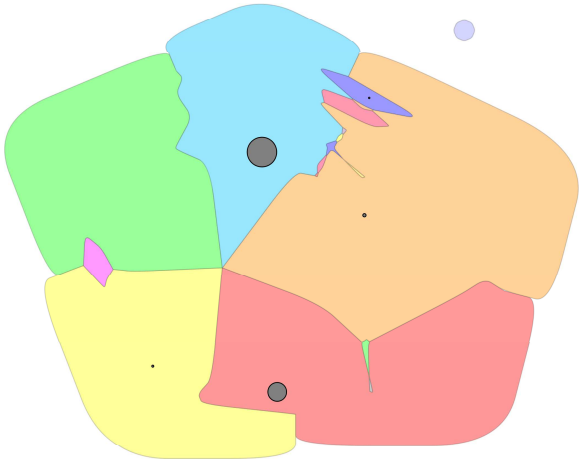
The geographic map metaphor described above can of course be implemented in many different ways. In this paper we propose two approaches: the *country map* and the *octopus map*.

A *country map* is a contact representation 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.

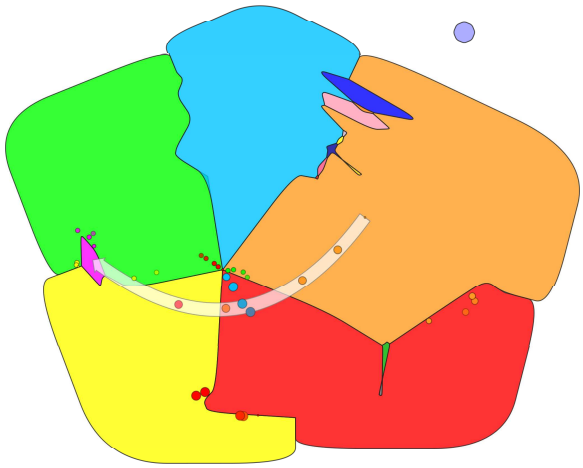
One might object that only a few migration graphs can be represented as country maps. In fact, if: 1. vertices are represented as planar regions with disjoint interiors, 2. vertices are adjacent in the graph iff they share a point in the map, and 3. *no four regions meet at a point*, then only planar graphs can be represented. However, following [8], we remove the emphasized condition from our implementation. Hence, we can represent a much wider class of graphs as migration graphs. Such graphs are called *planar map graphs* in [8], and they can contain up to $27n$ maximal cliques in a graph with n vertices.

Fig. 2 shows two snapshots from a prototype implementation of a *country map*. Fig. 2.a shows the map at a certain instant. The instances are colored based on the chart in Fig. 1. The green, light blue, orange, red, and yellow countries (corresponding to the main instances) share a point, and hence they are a clique. The name servers of a well known ISP are also shown as circles of appropriate size, distributed among different instances. Fig. 2.b shows a step of the animation. Usual flow can be recognized as a set of bubbles traversing the borders of adjacent countries. Unusual flow is instead represented with an arrow connecting non-adjacent countries.

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 corre-

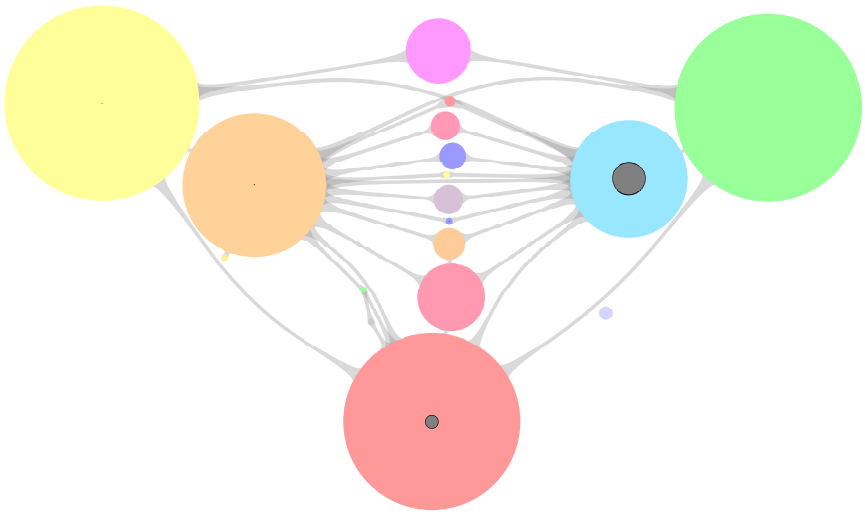


(a)

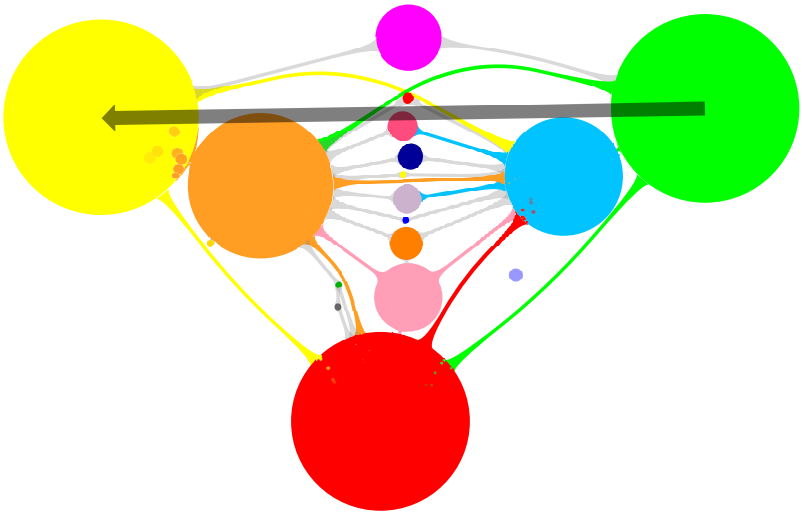


(b)

Figure 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 gray circles. (b) A snapshot of an animation of the same map, where usual and unusual client migrations are visible.



(a)



(b)

Figure 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 gray circle. (b) A snapshot of an animation of the same octopus, where usual and unusual client migrations are visible.

sponding 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. 3 shows two snapshots from a prototype implementation of an *octopus map*. Fig. 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. 2, while the underlying migration graph is different and specifically not planar: an edge crossing can be identified right under the topmost instance. Fig. 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.

Of course the selected metaphor is not the only possible choice. Alternatives have been investigated and screened out 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. The (usual and unusual) migration patterns are also influenced by the status of the routing and largely independent on the geography. 3. Combining the geographical data with the (usual and unusual) migration patterns can easily lead to information cluttering. 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 6 documents a number of additional approaches that are related to our choice, together with the reasons why we did not follow any of them.

3 The Algorithms

The algorithms that support our two visualization approaches take as input a migration graph G 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.

3.1 Country map

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" that is roughly proportional to the average of the clients that it serves in 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} and compute the animation from t_i to t_{i+1}.

In Step 1 we check if $G(V,E)$ 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 $v_1, \ldots, v_k$ be the vertices of a clique whose edges are removed. Such edges are replaced with a new vertex c and by edges $(v_1, c), \ldots, (v_k, c)$. An example is presented in Fig. 4.a and Fig. 4.b, using the map graph of Fig. 2.b. In [31] it is shown that testing if a graph is a map graph can be done in polynomial time. However, in [9] it is argued that the exponent of the polynomial bounding its running time from above is about 120. This makes it impractical to use the algorithm in [31]. Hence, we use a simple heuristic that works as follows. We first check if G is planar. If yes, we are done. Otherwise, we look for a

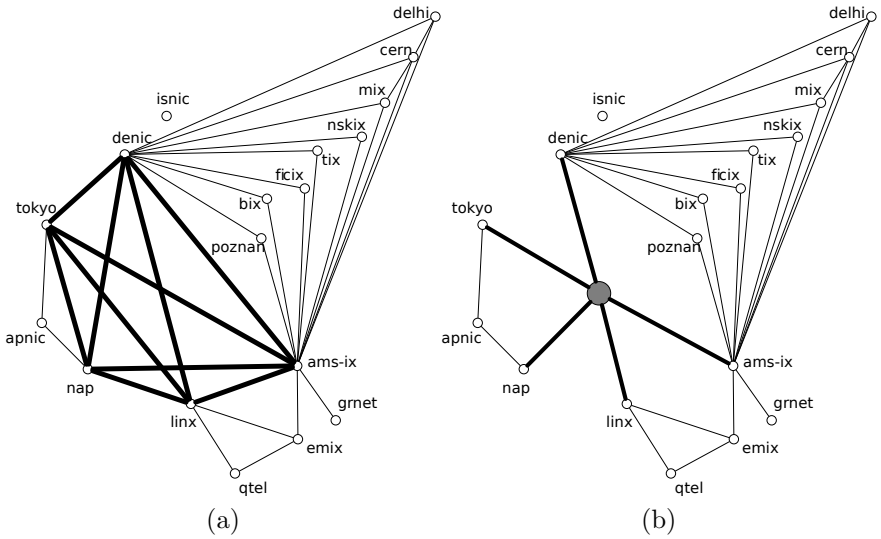


Figure 4: (a) Migration graph for K-root. A clique of size 5 is highlighted with thick edges. (b) Backbone used in Fig. 2.b, obtained replacing the clique with a star centered at the grey vertex.

maximal clique in G with the algorithm in [7], 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. 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 [13] that preserves the given planar topology (see, e.g., [14]). Charges and spring lengths are assigned such that at the end 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. Such edges are needed to easily morph the geographical map in Step 5. All edges in the subset $A = E' \setminus E$ are marked as *additional*. For this purpose we use a constrained

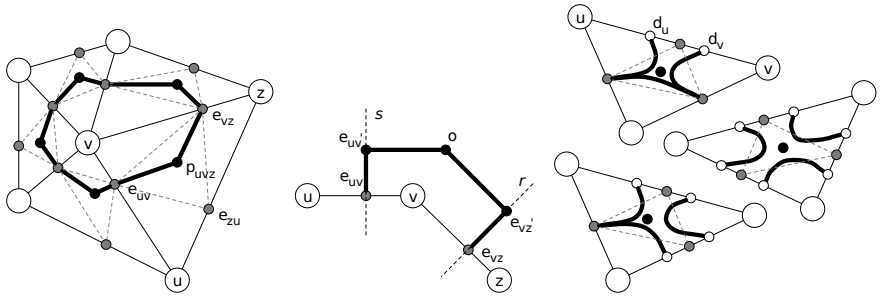


Figure 5: (a) Construction of the country border for a vertex that is not on the convex hull. Each white circle represents a vertex of the skeleton. For each edge (u, v), a small grey circle represents the point e_{uv}. For each triangle $\Delta(u, v, z)$, a small black circle represents the point p_{uvz}. (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.

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 of $t_1, \ldots, t_k$. To achieve this, a spring embedder is used where charges and preferred spring lengths change over time (see, e.g., [15]). Its 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: 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 with $w(v)' = \frac{\alpha}{2\pi}\frac{c_t(v)^2}{\sum_{i\in F_v} A(i,v)}$, 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.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 $F_v = \Delta(z, v, u_1), \Delta(u_1, v, u_2), \ldots, \Delta(u_{last}, v, u)$. 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.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. Note that borders could be smoothed in a more radical way, even leaving some empty area between countries. That could lead to more readable country borders without twists and turns, although adjacencies would become less obvious and area values would suffer from a possibly greater imprecision. We opted for simple adjacencies following the latter motivation. 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 finding better layouts for the skeleton graph in Step 4. The country border for such a path of vertices is computed as

the symmetric difference between their borders. 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.c for an illustration.

3.2 Octopus map

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 the number of crossings between its edges is minimized. Find a straight-line drawing for G respecting the computed 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 two kinds of shapes: 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. The reason behind such a condition lies in a better readability of the octopus maps derived from the drawing. The crossing minimization problem is known to be NP-hard [18, 25]. However, heuristics exist to compute an approximate result in polynomial time (see [20] 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. After this step 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, 2. each edge is replaced by a link whose width is proportional to the amount of flow between the two connected vertices. More formally, the operation is performed such that 1. for each vertex v it is possible to draw a circle C_v centered in v with area equal to $c_{t_{i+1}}(v)$ that does not intersect any other circle, 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. only checking each shape against its neighboring shapes.

In Step 3 the layout of the octopus is modified in order to reduce the total area needed to draw it. The goal is the ability to draw 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 [14]), 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 there is no positive charge assigned to any vertex. 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. Its 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 to fit the screen. The new scaling

factor is sent along 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.

4 Implementation and Technical Challenges

The implementation of the visualization framework described in the previous sections 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. Furthermore, integral query logs 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 [1] cluster deployed at the RIPE NCC for distributed storage and analysis of scientific data and network measurements. To give an idea on the volume of data, 5 minutes of query logs correspond to approximately 300 megabytes of PCAP [2] files, 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.

Our visualization framework has been implemented as a Web application. It is now available at the address `http://k.root-servers.org/visualk/` as a beta prototype, featuring most of the requirements described in Section 2. 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: refer to Section 2 for a brief explanation. The only metaphor implemented so far is the octopus map, since it is slightly preferred by K-root operators (see Section 5 for details) and therefore the most advanced in terms of implementation. The remaining requirements and the second visualization metaphor are currently under development. Earlier stages of the prototype are documented online [3].

The front-end is written in HTML and JavaScript. Its main responsibility is to visualize the map of K-root and animate it from time to time with the new data sent by the server. 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 Fig. 2 and 3 for an example).

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 has been realized with asynchronous messages, in order to keep a low coupling between the two. We used Apache ActiveMQ as a message broker, which also 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 implementation octopus maps are generally computed in about one second per iteration, with occasional peaks at two seconds when the constraints of the drawing are particularly entangled. Such numbers make octopus maps suitable for a near real-time tool. On the other hand, the prototype implementation 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 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 at the moment 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. We plan to reduce this gap in the

near future, allowing for an even more up-to-date visualization.

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 in the future the system could be improved allowing a smaller refresh rate. However, as explained in Section 2, the current refresh rate is already acceptable for all common operational needs.

5 Back to the User

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 2, 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; 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, it must be observed that the possibility to visualize data specific of a certain ISP is currently inhibited by the standard privacy policies 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 proves interest for the service. For this reason, 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 at the RIPE NCC were periodically involved in testing and evaluation. Four of them were also responsible for major discussions on requirements and qualities of the visualization approach, being personally responsible for the maintenance and development of K-root. Their participation to the design process allowed to precisely 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 periodically presented with snapshots and example movies of the two implemented approaches (see [3] for some examples), and asked to compare them with respect to the understandability of the main information requirements discussed in Section 2. 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 such a comparison are presented and explained in Table 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.

	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 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. The preferred alternative is marked with a checkmark. In case of comparable preference, both alternatives are marked.

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 previous evaluation phases, are also quite valuable because they allow to determine advantages and disadvantages of the two implementations. They also confirm that both approaches are valid and preferred over the previously adopted visualization. We summarize them as follows. 1. Both visualizations are appreciated for the new insight they provide on existing data. In particular that applies to the detection of usual and unusual migrations, which did not emerge from previous visualizations (see e.g. the one presented in Section 2). 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. 2. 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. 3. The octopus map conveys information 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. 4. The country map represents every adjacency implicitly while the octopus map needs tentacles. The first representation 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 other or present different lengths, potentially causing confusion for the users. 5. Unusual migrations are basically identical in the two approaches, so no clear preference applies. The adoption of impact graph-

ics (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. 6. The visualization of the status of a specific ISP is also realized in the same way both for country maps and octopus maps, so there is no particular preference between the two.

Furthermore, the two approaches present differences that are relevant from a more theoretical point of view. 1. 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. 2. Only octopus maps can represent any kind of graph, although dense non-planar graphs can result in poor and confusing visualizations.

The actual visualization, already presented in Section 4 as a prototype tool, is available on a big screen in the Global Information Infrastructure department at the RIPE NCC. The animation constantly runs as a background monitoring tool for the operators of K-root. That means that 1. 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, while 2. all the other features (usual migrations, size of instances, etc) are always visible and operators can every now and then enjoy an overview of the system and get inspiration for long-term improvements of K-root.

On a related note, operators find it particularly interesting to use the system together with BGPlay [10], an already existing tool for the visualization of inter-domain routing and its evolution. 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.

6 State of the Art

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 [16]. It identifies four semantic primitives to be used when representing information entities with a geographical metaphor. *Boundaries*: discontinuities in the information space can be represented with borders. *Aggregate*: homogeneous zones preferably represent homogeneous entity types. We use aggregate and boundaries to group clients using the same instance and to separate such groups, respectively. *Loci*: information items preferably have a meaningful location in the information space. We link together or put side-by-side instances that are expected to share clients. *Trajectories*: semantic relationships between information entities at different locations 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 [24] 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 [16] we can say that in [24] the *Aggregate* primitive prevails over the others. BGPlay Island [11] extends the widely used BGPlay routing visualization system [10] and uses a topographic metaphor to show hierarchies of Internet Service Providers (ISPs). However, BGPlay Island uses the metaphor of a terrain map rather than the one of a political map and the most stressed primitive in it 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, [19, 22, 27]). 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 time, which makes them unsuited for a real-time monitoring tool. In [27] 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 [32, 12, 23, 28, 4, 6, 5] 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 [29] the authors introduce an adaptive version of the multiplicatively weighted Voronoi diagram [26], where each vertex in a graph is assigned a closed region with prescribed area. Similarly to Voronoi diagrams, however, adjacencies between regions depend on geometric proximity. Hence the solution is not compatible with the notion of adjacency graph.

In a recent work [17] it is shown that planar graphs can be represented with adjacent convex hexagons. Such shapes could be a valid alternative for our scope. We think that it is possible to modify the proposed algorithm to represent also planar map graphs, using polygons with more sides and loosing the convexity. However, the problem of assigning prescribed areas to the shapes seems difficult to be addressed.

A previous attempt at visualizing the activity of Internet services, including

K-root, is described in [21]. 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, centered at the centroid and composed of wedges that represent the amount, distribution and latency of clients. Such a tool differs from our approach, in that it is meant to visualize static snapshots of the service, not focusing on the migrations of clients.

7 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 server.

While in [30] it is argued that animations are not generally suitable to convey information on trends in data visualization, it is also argued that they 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 Fig. 2 and 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. Such a step is technically easy, but it has 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.

Acknowledgements

The authors wish to thank the Global Information Infrastructure (GII) and Research and Development (RD) departments at the RIPE NCC for their continued support and dedication, which helped us improve and refine our visualization framework.

References

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

[2] TCPDUMP/LIBPCAP public repository. `http://www.tcpdump.org/`.

[3] Visual-K. `http://dia.uniroma3.it/~squarcel/visual-k/`.

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

[5] T. Biedl and L. 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.

[6] T. Biedl and L. E. R. Velázquez. Drawing Planar 3-Trees with Given Face-Areas. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 316–322. Springer Berlin / Heidelberg, 2010.

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

[8] Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Planar Map Graphs. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98. ACM, 1998.

[9] Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Recognizing Hole-Free 4-Map Graphs in Cubic Time. *Algorithmica*, 45(2):227–262, 2006.

[10] L. Colitti, G. Di Battista, F. Mariani, M. Patrignani, and M. Pizzonia. Visualizing Interdomain Routing with BGPlay. *Journal of Graph Algorithms and Applications, Special Issue on the 2003 Symposium on Graph Drawing, GD '03*, 9(1):117–148, 2005.

[11] P. F. Cortese, G. Di Battista, A. Moneta, M. Patrignani, and M. Pizzonia. Topographic Visualization of Prefix Propagation in the Internet. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):725–732, 2006.

[12] M. de Berg, E. Mumford, and B. 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.

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

[14] W. Didimo, G. Liotta, and S. 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.

[15] C. Erten, P. Harding, S. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph Animations with Evolving Layouts. In G. Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 98–110. Springer Berlin / Heidelberg, 2004.

[16] S. I. Fabrikant and A. Skupin. Cognitively Plausible Information Visualization. *Exploring Geovisualization*, (November 2004):667–690, 2005.

[17] 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.

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

[19] 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.

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

[21] B. Huffaker, M. Fomenkov, and k. claffy. Influence Maps - A Novel 2-D Visualization of Massive Geographically Distributed Data Sets. *Internet Protocol Forum*, October 2008.

[22] R. Inoue and E. 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.

[23] A. Kawaguchi and H. 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.

[24] D. Mashima, S. Kobourov, and Y. Hu. Visualizing Dynamic Data with Maps. In *Proc. 4th IEEE Pacific Visualization Symposium*, March 2011.

[25] 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.

[26] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[27] M. Ouyang and P. 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.

[28] M. S. Rahman, K. Miura, and T. Nishizeki. Octagonal Drawings of Plane Graphs with Prescribed Face Areas. *Computational Geometry: Theory and Applications*, 42:214–230, April 2009.

[29] R. Reitsma and S. Trubin. Information Space Partitioning using Adaptive Voronoi Diagrams. *Information Visualization*, 6:123–138, May 2007.

[30] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of Animation in Trend Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1325–1332, November 2008.

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

[32] M. van Kreveld and B. Speckmann. On Rectangular Cartograms. *Computational Geometry: Theory and Applications*, 37:175–187, August 2007.