How to Visualize the K-Root Name Server (Demo)*

Giuseppe Di Battista[1], Claudio Squarcella[1], and Wolfgang Nagele[2]

[1] Dipartimento di Informatica e Automazione, Università Roma Tre, Italy
{gdb,squarcel}@dia.uniroma3.it
[2] RIPE NCC, Amsterdam, The Netherlands
wnagele@ripe.net

Abstract. We present a system that visualizes 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. The system can be used either to monitor what happened during a prescribed time interval or to observe the status of the service in near real-time. The system visualizes how and when the clients of K-root migrate from one instance to another, how the number of clients 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.

1 Introduction

A computer that needs to know the IP address which corresponds to a *domain name* sends a query to a *name server*. Hence, all Internet Server Providers (ISPs) make one or more name servers available to their customers in order to answer their requests.

A name server that receives a query executes a *resolution* process. The resolution computes an answer to the query by iteratively querying other name servers and quite often requires to send 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, 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. A name server selects its favorite root servers according to its query optimization policies.

For resiliency and efficiency reasons, each root server is implemented with computers spread across several locations distributed worldwide. Each location is an *instance*. Currently each root server has from 1 to 70 instances, e.g., A has 6 instances, F has 49, and K has 18. 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. The instance is selected by a widely adopted mechanism called *anycast*, which leaves the responsibility of choosing the topologically nearest instance to the current status of the Internet routing. Hence, a

* 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.

M. van Kreveld and B. Speckmann (Eds.): GD 2011, LNCS 7034, pp. 191–202, 2012.

name server of a certain provider that sends a sequence of queries to a root server, say K-root, can have that each of the queries is answered by a different instance, according to the current status of the routing. This has consequences both from the point of view of the name server and from the point of view of the root server. The first can experience fluctuations of the elapsed service time, while the latter can suffer changes in the distribution of the workload among the 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) that is one of five Regional Internet Registries providing Internet resource allocations, registration services and coordination activities that support the operation of the Internet globally. The visualization can be activated either to monitor what happened during a prescribed time interval or to check the status of the service in near real-time. We visualize how and when name servers (that are the clients of the root server) *migrate* from one instance to another, how the number of clients associated with each instance changes over time, and which is the status of the service offered to a certain ISP. 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.

The paper is organized as follows. In Section 2 we discuss the adopted visualization metaphor. In Section 3 we present the layout algorithm used in our system. In Section 4 we give technical details and briefly address user feedbacks. In Section 5 we compare our system with the existing literature. Concluding remarks are in Section 6.

2 Selecting a Metaphor

The choice of the visualization metaphor and of the interaction features of our system comes from an intensive discussion with the RIPE NCC, aimed at collecting the most important visualization requirements for the K-root service. It comes out that the crucial need is to visualize 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 interval 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. To give an idea of the migration phenomenon, in 24 hours of normal operation about $50,000$ clients issue a service request to more than one instance.

Further, migrations are not all the same. There are pairs u, v of instances such that migrating from u to v or vice versa is considered *usual* by the operators. For example, u and v are placed in network locations that have high connectivity between them, or the Internet routing frequently oscillates moving clients form u to v or vice versa. There are other migrations that are considered *unusual*, like for example those involving pairs of instances in places with very poor connectivity between them. Unusual migrations can put in evidence suspicious activities, misconfigurations, or large-scale faults. Given a pair of instances u, v, deciding if u, v is subject to usual or unusual migration is knowledge that comes from the RIPE NCC experts and is subject to change over time, with a frequency that is much lower than the one of the migrations.

An example of chart currently used by RIPE NCC to visualize the distribution of queries to the instances of the K-root service is in Fig. 1.a. Note that it does not provide

any migration information, while the operators need to perceive unusual migration patterns keeping in the background usual migration behaviors. For this reason we define a *migration graph*. Its vertices are the instances and 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, then the migration is considered *usual*, otherwise it is considered *unusual*.

A second requirement is to visualize the relative weight of instances, giving an immediate perception of the number of clients they serve. Also, it is required to visualize how this weight is related to the migrations. More generally, there is the need to visualize the evolution of the service over time, both within a prescribed time interval, for ex-post analysis, and in near real-time. This requirement is extremely challenging, not only from the visualization perspective, but also because of the very high volumes of involved data: K-root receives about $20,000$ queries per second.

Finally, there is the need of understanding what happens to the queries issued by a specific Internet Service Provider (ISP). As an example, consider the main name servers that provide the name resolution service to the customers of an ISP. It is interesting to observe which K-root instances answer their queries and how this evolves over time.

Motivated by the above requirements we adopt a geographic map metaphor, which is quite appropriate for describing migrations. The service offered by K-root is represented as a map. Each instance is a country and its size is roughly proportional to the current number of its clients. Two countries 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. countries change their size according to the fluctuations in the number of their clients, 2. usual migration flows are pictured as bubbles traversing the boundaries of adjacent countries, and 3. unusual migration flows are highlighted with impact graphics as bridges across the countries.

One might object that a few migration graphs can be represented with our metaphor. 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 [6], we remove from our metaphor the emphasized condition. Hence, we can represent a much wider class of graphs as migration graphs. Such graphs are called *planar map graphs* in [6], and they can contain up to $27n$ maximal cliques in a graph with n vertices.

Fig. 1 shows the system in action. Fig. 1.b shows the map at a certain instant. The green, light blue, orange, red, and yellow countries (corresponding to the main instances) share a point, and hence they are a clique. The other figures show several snapshots of an animation: Fig. 1.c shows how countries have different sizes in different instants. Fig. 1.d shows how we represent usual migration flows. Fig. 1.e shows the bridges that appear to emphasize unusual migration flows. Fig. 1.f shows how the name servers of a well known ISP are distributed among the instances. Each circle represents a group of name servers of that ISP with size proportional to their number.

Of course, the selected metaphor is not the only possible choice. Alternatives have been investigated and screened out for different reasons. As a first 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. Even if instances have

a geographical location, the choice done by the clients is largely independent on the geography, 2. The (usual and unusual) migration patterns are also largely independent on the geography, and 3. Combining the geographical data with the (usual and unusual) migration patterns leads to information cluttering. Alternatively, we could visualize the

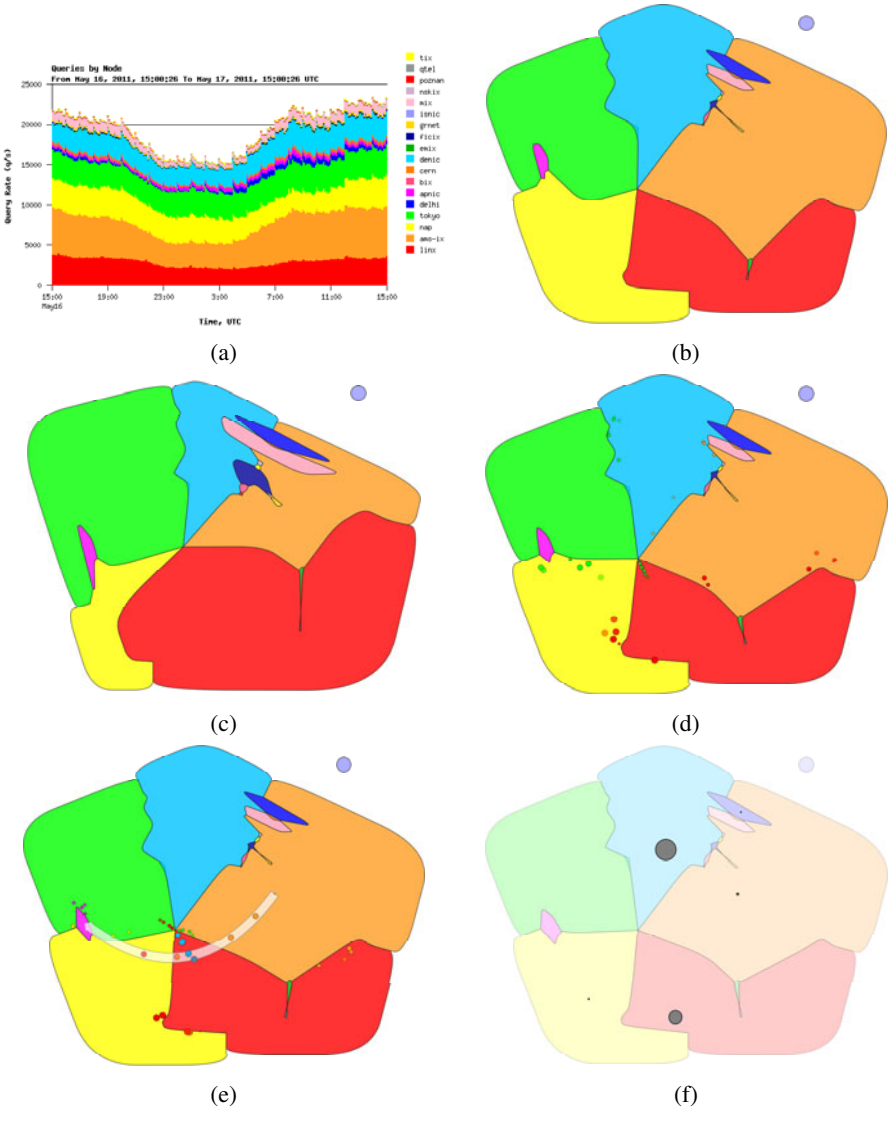

Fig. 1. (a) Distribution of the queries among the K-root instances. (b) The map at a certain instant. The animation: (c) Countries have different sizes in different instants. (d) Usual migration flows. (e) Unusual migration flows. (f) The name servers of a well known ISP.

migration graph with vertices and edges. Although it would be easy to give to each vertex an area that is proportional to its importance (see a historical example in [5]), it would be difficult to give different emphasis to usual and unusual migrations. Further, the reason behind edges having different lengths would result obscure and misleading for the user.

3 The Algorithm

The algorithm that supports our visualization framework takes 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, each connected component is considered separately. The algorithm constructs 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 asked to 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 $f_{t',t''}(u, v) = \max(0, m_{t',t''}(u, v) - m_{t',t''}(v, u))$.

The algorithm is composed by two phases: the Preprocessing and the Animation, that is repeated for each t_i. The Preprocessing is composed of three steps:

1. Check if G is a map graph. If yes, then construct its *backbone*, i.e. a planar graph obtained from G by substituting some of its cliques with stars. If not, edges are removed 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 $c_t(v)$ in $t_1, \ldots, t_k$.
3. Construct a constrained Delaunay triangulation, called *skeleton*, 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: i.e. 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 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 yes, then 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 where the map graph is the one of Fig. 1.b is presented in Fig. 2.a and Fig. 2.b. In [25] it is shown that testing if a graph is a map

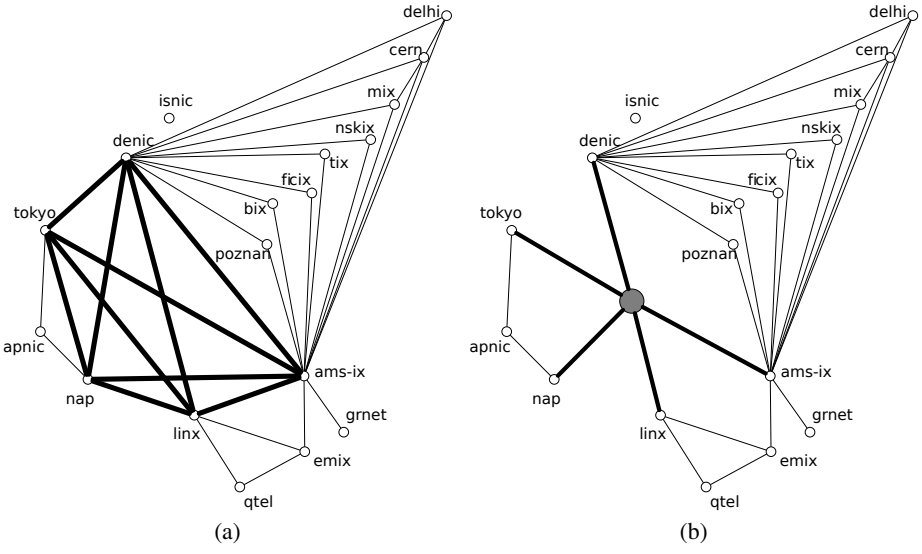

Fig. 2. (a) Migration graph for K-root. A clique of size 5 is highlighted with thick edges. (b) Backbone obtained substituting the clique with a star, centered at the grey vertex.

graph can be done in polynomial time. However, in [7] 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 [25]. 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 in G for a maximal clique with the algorithm in [4], 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 [26] that preserves the given planar topology (see, e.g., [11]). Each vertex v has a positive charge $w(v)$ equal to $\frac{\sum_{i=1}^{k} c_{t_i}(v)}{k}$ and each edge (u, v) is a spring with preferred length equal to $\frac{\sqrt{w(u)}+\sqrt{w(v)}}{\sqrt{\pi}}$, that is the sum of the radii of two circles of area $w(u)$ 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 Delaunay triangulation, in order

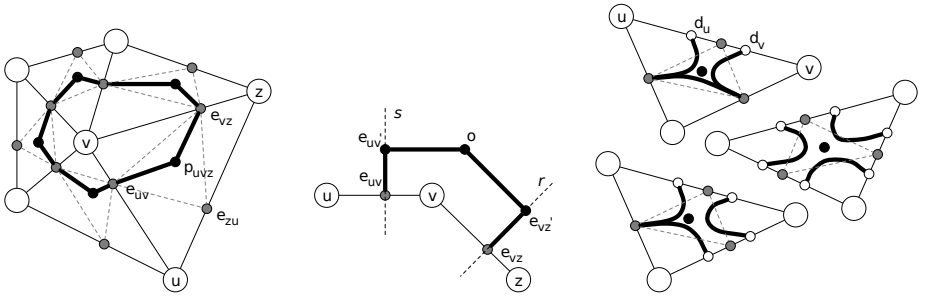

Fig. 3. (a) Costruction of the country border for a vertex that is not on the convex hull. Big white circles represent vertices of the skeleton. For each edge (u, v), a small grey circle represents the point e_{uv}. For each triangle $\Delta(u, v, z)$, a small black circle represents the point p_{uvz}. (b) Costruction 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.

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., [12]). 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 (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 different from 2π only for the vertices of the external face). Spring lengths are updated accordingly with $\frac{\sqrt{w(u)'}+\sqrt{w(v)'}}{\sqrt{\pi}}$.

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 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. 3.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 call $F_v = \Delta(z, v, u_1), \Delta(u_1, v, u_2), \ldots, \Delta(u_{last}, v, u)$ the set of triangles surrounding v. 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 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. 3.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 geographical 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. Second, 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 $\overline{ud_u}/c_t(u) = \overline{d_vv}/c_t(v)$ and $\overline{ud_u} + \overline{d_vv} + D = \overline{uv}$. 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. 3.c for an illustration.

4 Technical Aspects and User Feedback

Our visualization framework has been implemented as a Web application, composed of a Javascript front-end and a server written in Java. It relies on Google Web Toolkit,

a framework for the creation of Web applications. It makes use of 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. 1 for an example). A demo of the application is available online at `http://dia.uniroma3.it/~squarcel/visual-k/`.

An associative map is kept in memory to store the current state of each client, including the instance that answered its last query. At regular time intervals all the new queries received by K-root are analyzed to detect usual and unusual migrations. These are translated into an animation step and are later used to update the associative map.

We performed a stress test using a full trace of queries received by all the K-root instances during a 48-hour time window. Even focusing on the minimal amount of information needed, in the form of a triplet $(timestamp, client\_id, instance\_id)$, the volume of data is impressive (around 200 Gigabytes) and poses a challenge for the creation of a scalable system. We ran the stress test on a laptop with a 2.4 GHz Intel Core 2 Duo processor and 4 GB of RAM. The results show that our framework can handle an update rate between 10 and 15 seconds, including both the analysis of query data and the generation of the corresponding geographical map. The result is of course expected to improve on more powerful hardware. Hence, although such an approach is not strictly real-time, it represents an approximation that satisfies the operational needs.

The system has two types of potential users: (1) the RIPE NCC DNS Services staff and (2) the vast audience of ISPs that could benefit of knowing what instances serve their clients. The system has been designed cooperating with the users of type (1). Their participation to the design process allowed to precisely focus on the requirements. As an example, during the 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. It is particularly interesting to use the system together with BGPlay [8]: once an unusual migration is spotted, BGPlay can be used to check if there is a correlation with some routing change. About users of type (2), the possibility of putting the system at their disposal depends on the future policies of the RIPE NCC. In fact, the logs of queries are strictly confidential and an anonymization policy is currently being discussed.

5 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 [13]. 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 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 ours: GMap and BGPlay Island. GMap [19] visualizes clustered graphs with 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 [13] we can say that in [19] the aggregate primitive prevails over the others. BGPlay Island [9] extends the widely used BGPlay routing visualization system [8] 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 of [13] is the locus one.

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, [15,17,21]). 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 [21] 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 [27,10,18,22,1,3,2] 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 [23] the authors introduce an adaptive version of the multiplicatively weighted Voronoi diagram [20], 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 [14] 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 in [16]. 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.

6 Conclusions and Future Work

We have presented a system for the visualization of the behaviour of the K-root DNS name server. It relies on a map metaphor that uses an animation to show the migration of clients among the instances that compose the server.

While in [24] 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. 1 clearly shows. 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.

References

1. Akbari Jokar, M., Shoja Sangchooli, A.: Constructing a block layout by face area. The International Journal of Advanced Manufacturing Technology 54, 801–809 (2011)
2. Biedl, T., Ruiz Velázquez, L.: Orthogonal Cartograms with Few Corners Perface. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 98–109. Springer, Heidelberg (2011)
3. Biedl, T., Velázquez, L.E.R.: Drawing planar 3-trees with given face-areas (2010)
4. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM 16, 575–577 (1973)
5. Carpano, M.-J.: Automatic display of hierarchized graphs for computer-aided decision analysis. IEEE Transactions on Systems, Man and Cybernetics 10(11), 705–715 (1980)
6. Chen, Z.-Z., Grigni, M., Papadimitriou, C.H.: Planar map graphs. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998. ACM (1998)
7. Chen, Z.-Z., Grigni, M., Papadimitriou, C.H.: Recognizing hole-free 4-map graphs in cubic time. Algorithmica 45(2), 227–262 (2006)
8. Colitti, L., Di Battista, G., Mariani, F., Patrignani, M., Pizzonia, M.: Visualizing interdomain routing with BGPlay. Journal of Graph Algorithms and Applications, Special Issue on the 2003 Symposium on Graph Drawing, GD 2003 9(1), 117–148 (2005)

9. Cortese, P.F., Di Battista, G., Moneta, A., Patrignani, M., Pizzonia, M.: Topographic visualization of prefix propagation in the internet. IEEE Transactions on Visualization and Computer Graphics 12(5), 725–732 (2006)
10. de Berg, M., Mumford, E., Speckmann, B.: Optimal BSPs and rectilinear cartograms. In: Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems, GIS 2006, pp. 19–26. ACM, New York (2006)
11. Didimo, W., Liotta, G., Romeo, S.A.: Topology-Driven Force-Directed Algorithms. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 165–176. Springer, Heidelberg (2011)
12. Erten, C., Harding, P., Kobourov, S., Wampler, K., Yee, G.: GraphAEL: Graph Animations with Evolving Layouts. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 98–110. Springer, Heidelberg (2004)
13. Fabrikant, S.I., Skupin, A.: Cognitively plausible information visualization. Exploring Geovisualization, 667–690 (November 2005)
14. Gansner, E., Hu, Y., Kaufmann, M., Kobourov, S.: Optimal Polygonal Representation of Planar Graphs. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 417–432. Springer, Heidelberg (2010)
15. Gastner, M.T., Newman, M.E.J.: 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 (2004)
16. Huffaker, B., Fomenkov, M. Claffy, K.: Influence maps - a novel 2-d visualization of massive geographically distributed data sets. Internet Protocol Forum (October 2008)
17. Inoue, R., Shimizu, E.: 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)
18. Kawaguchi, A., Nagamochi, H.: Orthogonal Drawings for Plane Graphs with Specified Face Areas. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 584–594. Springer, Heidelberg (2007)
19. Mashima, D., Kobourov, S., Hu, Y.: Visualizing Dynamic Data with Maps. In: Proc. 4th IEEE Pacific Visualization Symposium (March 2011)
20. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial tessellations: Concepts and applications of Voronoi diagrams, 2nd edn. Probability and Statistics. Wiley, NYC (2000)
21. Ouyang, M., Revesz, P.Z.: Algorithms for cartogram animation. In: Proceedings of the 2000 International Symposium on Database Engineering & Applications, IDEAS 2000, pp. 231–235. IEEE Computer Society, Washington, DC, USA (2000)
22. Rahman, M. S., Miura, K., Nishizeki, T.: Octagonal drawings of plane graphs with prescribed face areas. Comput. Geom. Theory Appl. 42, 214–230 (2009)
23. Reitsma, R., Trubin, S.: Information space partitioning using adaptive voronoi diagrams. Information Visualization 6, 123–138 (2007)
24. Robertson, G., Fernandez, R., Fisher, D., Lee, B., Stasko, J.: Effectiveness of animation in trend visualization. IEEE Transactions on Visualization and Computer Graphics 14, 1325–1332 (2008)
25. Thorup, M.: Map graphs in polynomial time. In: Proceedings of the 39th Annual Symposium on Foundations of Computer Science, FOCS 1998 (1998)
26. Tollis, I.G., Di Battista, G., Eades, P., Tamassia, R.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall (1998)
27. van Kreveld, M., Speckmann, B.: On rectangular cartograms. Comput. Geom. Theory Appl. 37, 175–187 (2007)