# Visualization of the Growth and Topology of the NLANR Caching Hierarchy*

Bradley Huffaker, Jaeyeon Jung, Duane Wessels, k claffy

**Abstract**

As the NLANR Caching Hierarchy, has increased in size and complexity, researchers and users have had a more difficult time depicting a comprehensive view of the overall topology. Planet Cache was one of the first attempts at visualizing the hierarchy, but was not sufficiently flexible to customize views of subsets of the topology by specific attributes. CAIDA[1] has expanded Planet Cache's visualization to Plankton including both topological and geographical depictions of the hierarchy. We have also invested in converting from VRML to a Java implementation to facilitate more interactive visualization. Our Java implementation allows the user to tailor the complexity, focus, and topological layout to match their specific visualization objectives. This ability to customize the visualization, combined with the universal availability that the web and Java provide, allows a much larger set of users to access data on the global caching infrastructure and gain insight into their own position in the hierarchy. While our goal is a universally accessible visualization tool, problems with the platform independence of Java and privacy sensitivities with respect to transcribing even logical topology information have resulted in our rendering some of the more useful aspects of the visualization only in standalone ('non-applet') mode.

Despite these drawbacks, the user-friendly interface and the visually informative and dynamic nature of both the standalone and the applet version are vast improvements over currently available tools, and provide a good basis for future analysis.

**Keywords:** Visualization; Caching hierarchy; Java

## 1 Introduction

Web caching is a technology for migrating copies of documents from a server across the network toward points closer to the end users requesting those documents. Caching can reduce response time and network bandwidth consumption by reducing the number of repeated transmissions of the same documents across the wide area Internet. Consequently, web caching is now being widely deployed to mitigate problems imposed by explosive growth of Internet traffic.

By extending this concept across a cooperative mesh, one can form cache meshes, or hierarchies, to distribute load and leverage content among caches, further increasing overall performance. For these reasons, rapid growth in the demand for web caching has led to many research, development, and deployment projects in the last few years. One well-known caching project in the United States, the

*This work is supported by the National Science Foundation under the CAIDA grant (NCR-9711092) and the NLANR Cache grant (NCR-976082).

[1] `http://www.ciada.org`

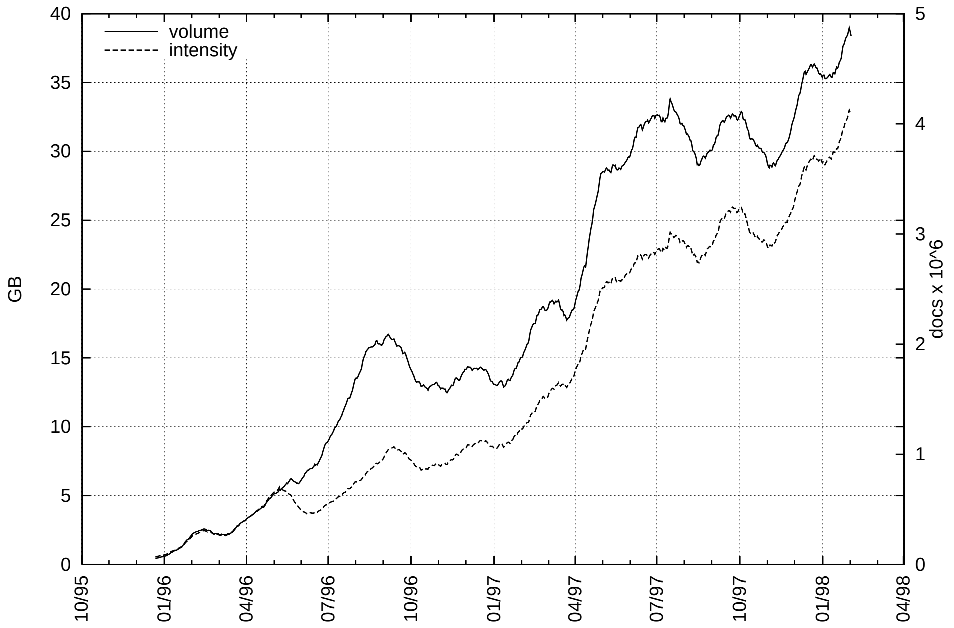


Figure 1: Daily traffic through NLANR root caches since the project started.

NSF-sponsored NLANR root cache system has been developing and deploying a prototype global web caching hierarchy since 1995 [1].

With root caches located at each node of the NSF's very high performance Backbone Network Service (vBNS), the NLANR caches have experienced a steady increase in use. This growth can be seen in figure 1 [2].

Currently over a thousand caches in over 45 countries participate in the NLANR caching hierarchy, with around four million requests per day reaching the NLANR root caches. The growth rate has clarified the need for a more organized structure that supports a scalable global Internet information distribution system.

Web caching can serve at least three purposes: content control; increased performance in terms of reduced external bandwidth required; and faster response times (lower latency). Web caching activities have grown much faster outside the United States than domestically, primarily due to the relatively higher cost of bandwidth – typically 10-100 times the cost in the US. A few countries (i.e. China and Singapore) even mandate the government-sponsored use of caching to allow content control by blocking web pages from particular sites.

In the next section we describe the first prototype visualization of the underlying structure of the NLANR caching hierarchy, Planet Cache, undertaken in 1996. Section 3 covers the design and implementation of Plankton, the follow-on visualization tool to Planet Cache. Section 4 describes the Plankton user interface and options. We conclude with a discussion of the results of this visualization and possible future directions.

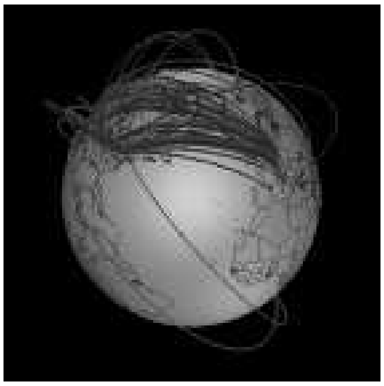

Figure 2: Planet cache; A 3-dimensional visualization of caching hierarchy drawn as arcs on a globe. It is limited to the caches directly connected to NLANR root caches.

## 2 Previous Study

Planet Cache was NLANR's first attempt to visualize the structure of the caching hierarchy, as seen from the perspective of the U.S. NLANR 'root' servers [1]. The original implementation used VRML to depict the geographical locations of the caches on a 3-dimensional globe. One compelling result of the visualization was its illustration of an abundance of unnecessary simultaneous connections between many non-U.S. caches and multiple NLANR root caches.

Using data collected at the NLANR caches by their web caching software (Squid) [3], Munzner created a 3D model of the hierarchy with nodes on the globe representing caches and lines reflecting peering relationships between them. The color of the lines indicated the number of *HTTP* requests received and the node positions reflected the latitude and longitude of the participating caches. NLANR stored VRML images reflecting the configuration for each day, posted them on the web for analysis, and created a video animating the change in configured hierarchy over the course of the project.

The visualization in figure 2, taken from the original Planet Cache project shows several caches in Europe and Asia peering with between 4 and 6 NLANR caches, so that every request to the NLANR system results in transmission of a separate query to each cache. Although this behavior may maximize the client's chance of getting a HIT response quickly, it is not in the best interests of either the user or the system as a whole. What really matters is how long it takes to retrieve the entire document, not just the single packet acknowledging the HIT.

Since the NLANR caches are connected via the vBNS, they effectively behave as a single distributed cache with six access points. Therefore, even if a document is only in a west coast NLANR cache, a cache client in Europe will get better response time by retrieving it via an east coast NLANR cache, which can leverage the relatively uncongested vBNS rather than having to use the often overloaded commodity Internet.

Insight gained from this initial visualization resulted in NLANR's cache access restriction policies, designed to force a more hierarchical global cache topology. Limiting access to the closest (throughput-wise) of the NLANR caches significantly reduced the number of duplicated transoceanic requests to the NLANR cache system, and at the same time decreased response time delay for requests for documents already stored in any NLANR cache.

While this investigation proved useful at the time, it depicted only a general sense of topology, limited to geographical illustration of the architecture, rather than allowing display of the hierarchical complexity of the virtual global cache mesh. Furthermore, the original tool could not zoom in to narrow the field of focus.

Our design goal for Plankton was to fill in this gap in functionality, to provide the ability to see the complete hierarchy either geographically or topologically, to focus in on a specific part, to dynamically alter the graph layout, and to explore the changes in the network over time. Such visualization will facilitate our ability to understand and improve on current load balancing and resource distribution as well as to predict future growth.

# 3 Design and Implementation

In this section, we examine several aspects of the design and implementation of Plankton.

## 3.1 Data

The raw data we visualize is collected only at the NLANR root caches. Because an HTTP request only arrives at one of the root caches after it has missed at all the caches along the way, the number of HTTP requests seen in the data is not representative of the number of requests throughout the system. The root caches see only misses from client caches directed at them. Cache log files from client caches would be necessary for a more complete picture of the total number of HTTP requests.

One restriction of the "Planet Cache" visualization was that we could only show our direct neighbor caches (i.e. the other caches which connect to ours), since we were limited to data generated from only our daily access logs. One goal of Plankton was to allow viewing of more than our direct neighbors.

We modified Squid to log the HTTP/1.1 *Via* header from HTTP requests, which allows us to retain knowledge of the identity of our client's clients, and so on. RFC 2068 [4] encourages, but does not require, proxy applications to append their hostname (and other information) to the request *Via* header. Also note that the *Via* header never includes information about the originating client, but only the intermediate proxies.

To create a complete view of the hierarchy we combine data from all seven NLANR caches, with resulting file sizes that range from 2 to 4 Mbytes, large enough to render download times problematic. These data sets also contain fully qualified domain names (hostnames) of caches within the hierarchy. We realize that many people use Web proxies to achieve some level of anonymity. Thus, to protect their privacy and identities, we only make smaller, filtered versions of Plankton data available for public consumption.

## 3.2 Tools

In choosing the programming tools for implementing Plankton, we considered both our requirements for viewing the data and the desire to make the system easily accessible via the World Wide Web. Our goal of widespread accessibility limits us to either static HTML documents and images, CGI-driven HTML forms, or Java. Static HTML documents and images would have been the easiest since they allow the greatest degree of control over user interaction. But flexible user interaction is another fundamental goal, because we do not know in advance what visualization perspective might be most valuable to the user. By making the tool as dynamic and flexible as possible, we support the user in efficiently finding what is interesting to them.

CGI scripts provide another level of user control over the data set, but without sufficient user interactivity, particularly in the ability to dynamically modify the caching topology layout. Achieving this level of interactivity using a standard CGI-script interface would have been difficult for the programmer and for the user, the latter of whom would experience a large wait time between user action and feedback while data was in transit across the net.

Choosing Java had many other advantages as well. First, Java allows us to implement both an applet and a standalone application. An applet is a complete set of code designed to be downloaded across the network for safe execution by a browser on a client machine. An applet provides advantages of both faster response time to the user as well as decreased network traffic. In contrast, a standalone program is a more canonical, independent executable piece of code, local to a one's system and able to run independently from other applications.

So Java allows the same source code to provide the advantages of an applet's wide-spread ('all-you-need's-a-browser') accessibility to the data set, and the standalone program's improved performance without the requirement for Internet connectivity.

### 3.3 Placement Algorithm

To visualize the basic hierarchical structure, we chose a system of nodes and links, where nodes represent caches and links reflect peering relationships between them. The size of the data set dictates an automated placement algorithm that can leverage the hierarchical nature of the data. Our algorithm uses the following placement heuristics:

1. the more children a node has, the further away its children should be so that they will not overlap each other;
2. children that themselves have lots of children should be even further away so that the grandchildren do not overlap.

The algorithm begins by determining whether there are any root caches in the data set. The default roots are the NLANR caches, but the user can specify any particular nodes as roots while the program is running. If Plankton does not find the root caches, it selects as root the node with the most children. If there is only one root node, Plankton places it in the center of the display window; otherwise Plankton places the root nodes evenly around the circumference of a circle centered in the middle of the display window. The radius of each node from the center is proportional to its number of children.

As Plankton places each root node in the image, it also puts the node's data structure into a queue that it can pass to a function to process each node. This function pops the node from the queue and puts all of its as yet unplaced children into an array. Plankton then places each node in this array across a base angle that defaults to 180 degrees unless there is only a single root node, in which case it uses 360 degrees.

The distance from the parent to a given child node is a weighted sum of the number of children of the parent plus the number of children of the node itself:

*(node's radius) = (# of node's parent's children) + (# node's children)*

After having drawn the position of the node on the display, Plankton puts the nodes at the end of the queue, takes the next node off the top of the queue, and starts the cycle again. The size of child nodes is one less than the size of their parents, which renders node size an indicator of its distance from a root node. Because the placement algorithm uses the size of the node when it calculates distance from the parent, decreasing size allows more nodes to fit on the screen without overlapping. Fig 3 illustrates some simple examples.

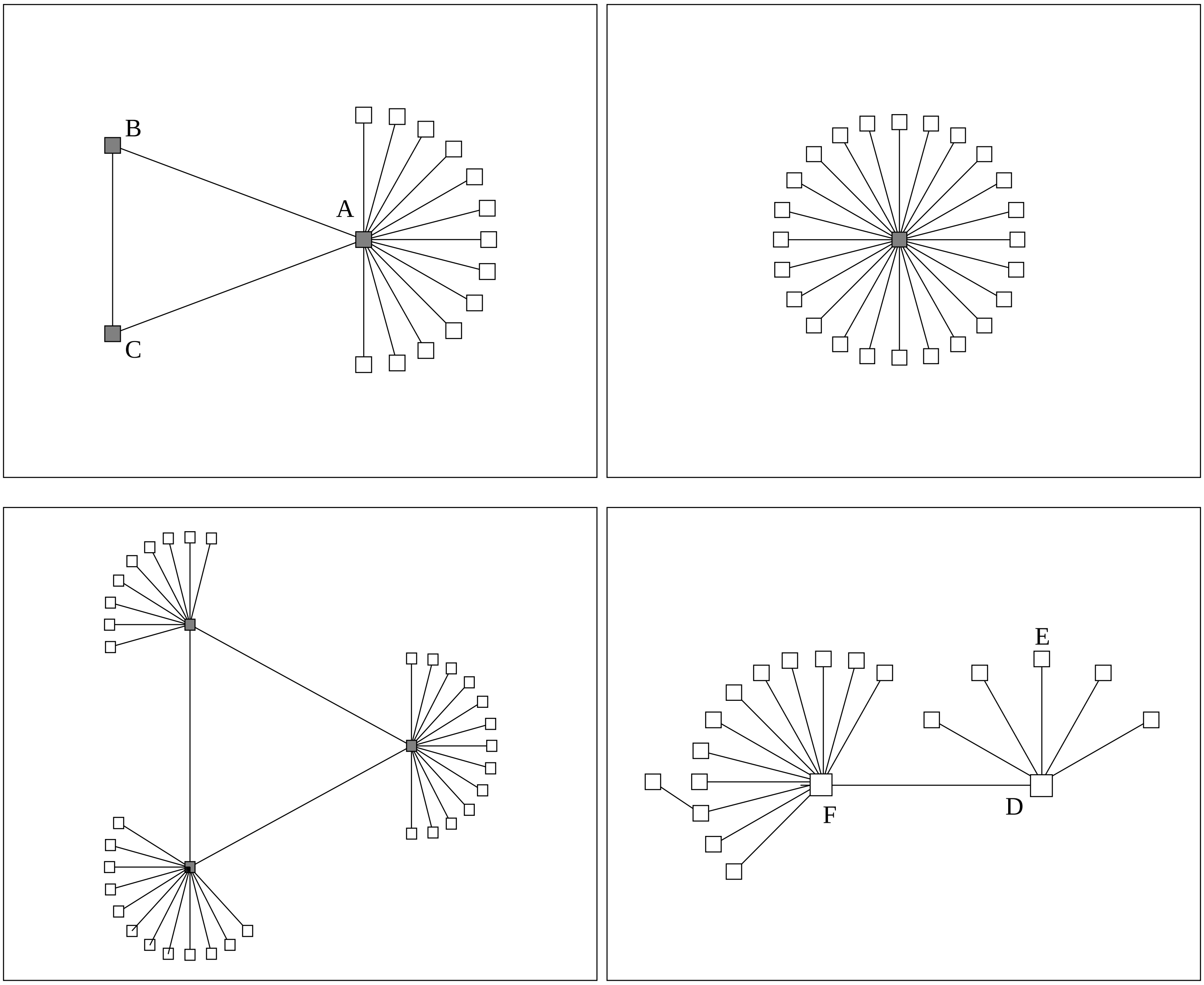


Figure 3: A is further from the center because it has children and both B and C do not. Upper right illustrates the default case when a single or no root node is present. If there are more than two root nodes, each node is placed across a base angle whose maximum is 180 degree. F is further away from D than E because F has more children than E does. Likewise, F's children are further from F than D's children are from D because F has more children than D.

### 3.4 Geographical Layout

Displaying cache topology geographically requires data on latitude and longitude corresponding to each individual cache machine. The tool ip2ll [5] uses the Internic *whois* database[2] to extract city and country names associated with an IP address and resolve lat/long information from a local database. However, the tool is insufficient for geographically diverse domain names, such as *.net* sites that represent national backbones or non-US domains that resolve only to the country's capital city. Although it is possible to get location information (LOC record) from the Domain Name System [6], currently almost no one provides this information. We thus had to resort to manual methods similar to those used in previous NLANR visualizations [7]. We gathered some geographical information on caches by looking up LOC records, but often had to supplement with the Internic whois database, traceroute, searching home pages, and by actually asking cache administrators.

Plankton draws each cache on the display by converting the latitude and longitude into an (x,y) coordinate, mapped in the display area. This (x,y) coordinate pair changes as the display changes in size, such as when the user zooms in or out, causing a potentially lengthy recomputation. Plankton then draws each cache node. If multiple caches map to the same point, either because their lat/long is the same or because the display's current resolution maps them to the same (x,y) point, Plankton represents the whole group with a single node. The corresponding textual display for this node would then contain summary data for all the caches it represents. The lines drawn between the nodes represents peering between these caches.

The overall NLANR caching hierarchy, drawn in both layouts, is shown at figure 4.

## 4 Features and Interface

Once we chose the overall topology layout strategy, we added the interactive elements, parameterizing the display so that the user can interactively determine interesting views. A group of parameters control the characteristics of the display, known as *parameter focusing* [8]. These options allow the user to selectively mask and highlight different parts of the data set.

The user interface feature set includes:

- Placement
  - topological or geographical display toggle
  - zoom and pan
  - move - single node or subtree
- Object Characteristics
  - size of lines and nodes
  - color by toggle
  - display ancillary data
- Animation

[2] `http://whois.internic.net`

- step-by-step display
- time sequence animation

We describe each in more detail below.

## 4.1 Placement: Topology/Geography

The user can choose to display the hierarchy data in either a topological or geographic layout or both side by side, allowing both geographical depiction of the architecture as well as logical one that illustrates the hierarchical complexity of virtual cache mesh. Problems identified from one view can be investigated in more detail with the other.

## 4.2 Zoom

By clicking on the center of the region of interest, the user may zoom into an any particular region to highlight a local hierarchy in more detail. The user can also push less important infrastructure components away from the view point to reduce clutter.

## 4.3 Move

Plankton facilitates further control of topology placement with support for dynamic movement of either entire subtrees or individual nodes. The user can thus either amend confusing placement selections made by the automatic layout algorithm or otherwise modify the structure to be meaningful from their perspective.

## 4.4 Object Characteristics

### 4.4.1 Size

The algorithms choose reasonable defaults for the node and link size, but as the user zooms or focuses the display, these values may no longer be applicable. Plankton thus allows the user to adjust the size and thickness of both nodes and links drawn on the map for better viewing.

### 4.4.2 Color

The user can color nodes and links by the number of HTTP requests received at the NLANR caches. The coloring algorithm uses a logarithmic scale, starting from low frequencies in blue to high frequencies in red, which makes visually prominent those node and links with large traffic loads. The user can also color by top level domain, which can provide some indication of virtual geography in the topological display. By lighting up all the nodes and links with same top level domain name in both layouts, one can inspect infra-structural incongruities between topology and geography. Two options are available with this feature; *'Show objects where values are'* can help users focus on selected data sets by sending others away. *'Color objects where values are'*, on the other hand can provide a good way to highlight specific objects in an overall topology. Figure 5 compares these alternative views.

### 4.5 Ancillary Data (mouse move over)

Moving a mouse over a node or link invokes the display of the corresponding information for that object, including the name(s) of cache nodes. Link information includes names of both ends of the peering relationship, along with the (server->client) direction of the peering.

### 4.6 Animation (of configuration over time)

Animation of daily updated files shows how the cache mesh infrastructure changes over time and allows the user to move back and forth through the time-series of configuration files. Animation allows one to see not only the growth of the caching hierarchy but also anomalies that do not appear readily from scanning the configuration files alone.

### 4.7 Levels

The "Step by Step" function allows the user to see how the hierarchy grows away from the roots one level at a time. This view can facilitate efficient placement of new caches.

## 5 Conclusions

One of the major successes of Plankton is that it provides an interactive method for viewing the NLANR Caching Hierarchy that is accessible to anyone with a network connection and a browser. Because Plankton allows the user to view the network both as a geographically or logically connected hierarchy of caches, it provides unique insight into how the mesh fits together. This insight helps to detect and remove configuration errors on the part of client peering relations, and allows cache administrators to see where more peering is most needed.

By studying the way the caching hierarchy changes over time, it is possible to formulate future trends, which can provide greater insight into where the network is going:

- What countries and sub-hierarchies are growing the most and how fast?
- How the number of HTTP requests are changing over time?
- Will the NLANR caches remain the hub or will some new network take it's place?

Answers to these questions will help network administrators decide where to best allocate resources to achieve an effective global caching hierarchy. Combining free Internet access, dynamic topology depictions, and the ability to view the data temporally, we are able to offer a platform through which researchers and network architects can gain new insights into web caching.

## 6 Future Directions

One of the major problems with Plankton's visualization is that it does not provide hit rates, which are a critical measure of a cache's success. They determine the overall effectiveness of the cache. Hit rate of 30%-40% are normal; we have seen numbers as high as 70% at focused sites, such as

IETF meetings. We did not include hit rates in our visualization because at development time, we were not collecting this data for the network as a whole. We would only have been able to display data about the NLANR root caches, which could easily have been misleading: requests satisfied by children never reach their parents, and since the NLANR root caches are all parents they will presumably never hear most requests. We hope to see mechanisms and policy to make this kind of data available in the future. Other data of interest include the number of bytes transferred for caches other than our own root caches, which indicates the volume of traffic served by the larger cache system and general usage trends.

The current data displayed in the public version of the tools conservatively identifies caches with labels such as *client1037.net* or *client1520.com* to avoid any possible privacy sensitivities; we hope to have future releases append some contact information, which can make it easy for cache administrators to articulate suboptimalities and fix them.

## 7 Availability

Plankton is available, with filtered data sets, from http://www.caida.org/Tools/Plankton . Running Plankton requires either a browser with Java enabled or downloading the binary version and running it with JDK locally. Note: the data sets and classes are large, so download and execution may be slow. Also, as mentioned, not all of the features we describe will work with the filtered data sets.

## 8 Acknowledgments

We are very thankful to all CAIDA staffs and Evi Nemeth for their comments on this paper. We also thanks Alex Rousskov for a nice graph.

## References

[1] "A distributed testbed for national information provisioning." `http://ircache.nlanr.net`.

[2] k claffy, A. Rousskov, and D. Wessels, "Nanog presentation," Fabruary 1998. `http://www.nlanr.net/~rousskov/research/nanog`.

[3] "Squid internet object cache." `http://squid.nlanr.net`.

[4] R. Fielding, J. Getys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," *Network Working Group RFC 2068*, January 1997. `http://ds.internic.net/rfc/rfc2068.txt`.

[5] "Ip address to latitude and longitude." http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll/.

[6] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson, "A means for expressing location information in the domain name system," *Network Working Group RFC 1876*, January 1996. `http://ds.internic.net/rfc/rfc1876.txt`.

[7] T. Munzner, E. Hoffman, K. claffy, and B. Fenner, "Visualizing the global topology of the mbone," in *Proceedings of the IEEE Symposium on Information Visualization*, 1996.

[8] R. A. Becker, S. G. Erick, and A. R. Wilks, "Visualizing network data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, March 1995.

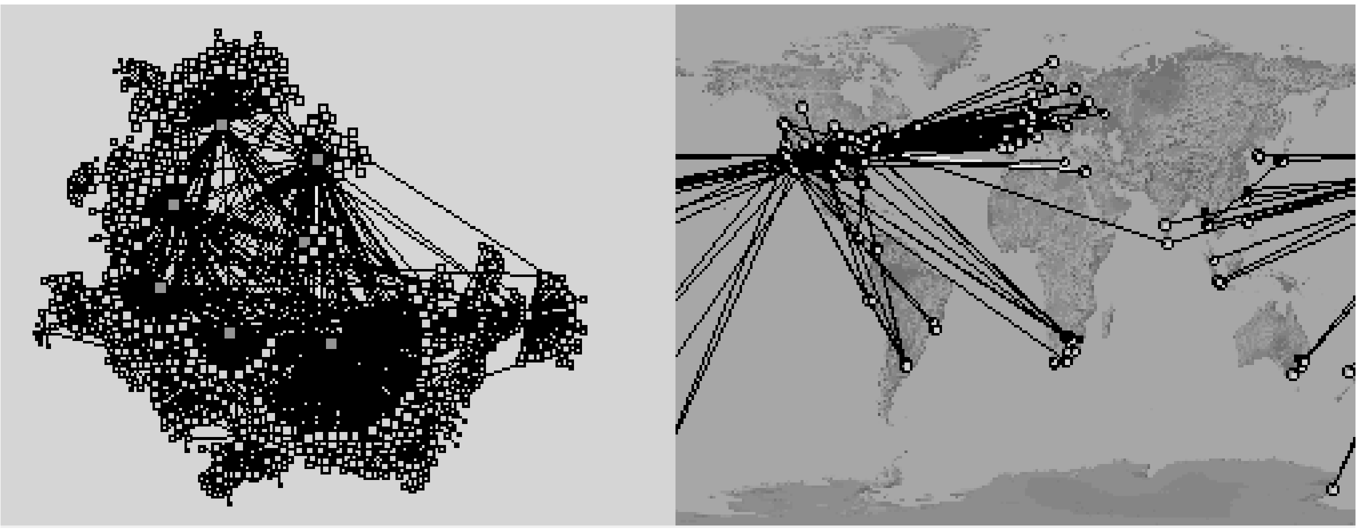

Figure 4: Topological and geographic depiction of caching hierarchy taken from June 10, 1998

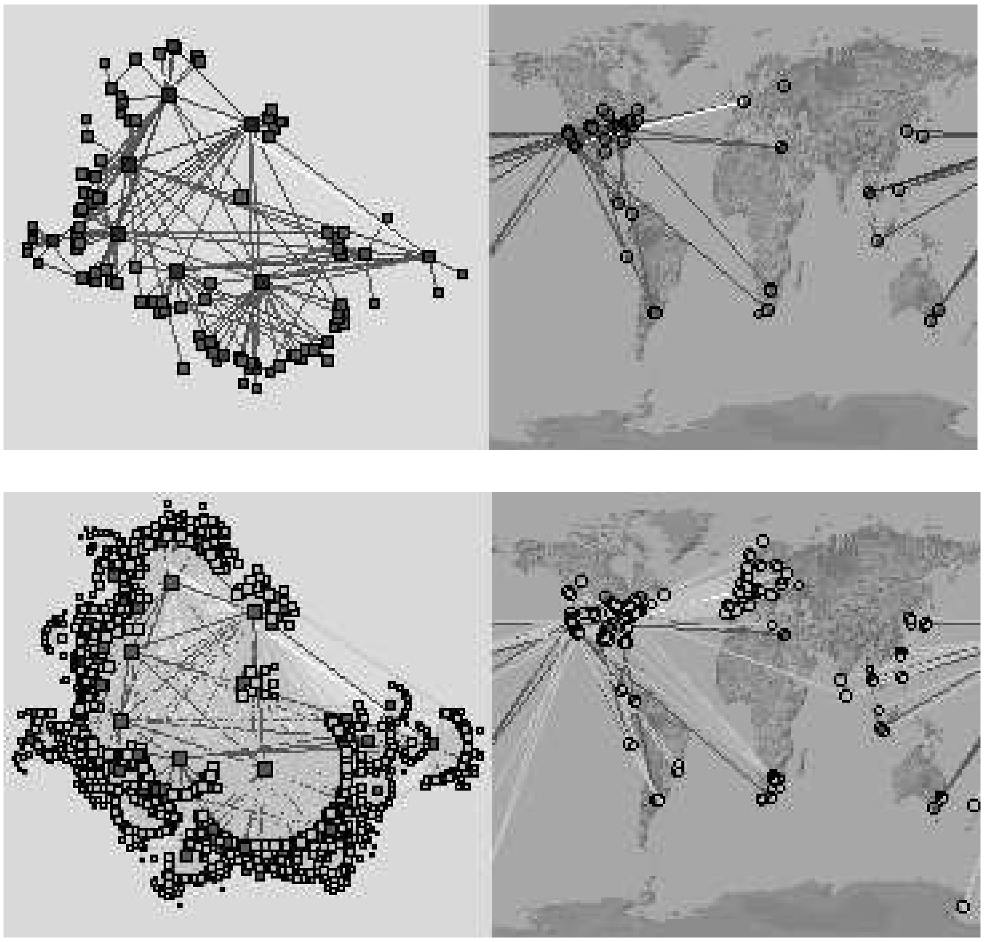

Figure 5: Mask/Color by HTTP requests. The top figure shows only objects having more than 10000 HTTP requests per day, which can be assumed substantial users of the system. In the bottom figure, those nodes and links with heavy traffic load are highlighted by a different color.