

Practical large-scale latency estimation

Michał Szymaniak^{a,*}, David Presotto^b, Guillaume Pierre^a, Maarten van Steen^a

^a *Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1081A, 1081HV Amsterdam, The Netherlands*

^b *Google Inc., Mountain View, CA, United States*

Received 17 July 2006; received in revised form 21 May 2007; accepted 20 November 2007

Available online 29 January 2008

Responsible Editor: I.F. Akyildiz

Abstract

We present the implementation of a large-scale latency estimation system based on GNP and incorporated into the Google content delivery network. Our implementation employs standard features of contemporary Web clients, and carefully controls the overhead incurred by latency measurements using a scalable centralized scheduler. It also requires only a small number of CDN modifications, which makes it attractive for any CDN interested in large-scale latency estimation.

We investigate the issue of coordinate stability over time and show that coordinates drift away from their initial values with time, so that 25% of node coordinates become inaccurate by more than 33 ms after one week. However, daily re-computations make 75% of the coordinates stay within 6 ms of their initial values. Furthermore, we demonstrate that using coordinates to decide on client-to-replica re-direction leads to selecting replicas closest in term of *measured* latency in 86% of all cases. In another 10% of all cases, clients are re-directed to replicas offering latencies that are at most two times longer than optimal. Finally, collecting a huge volume of latency data and using clustering techniques enable us to estimate latencies between globally distributed Internet hosts that have not participated in our measurements at all. The results are sufficiently promising that Google may offer a public interface to the latency estimates in the future.

© 2008 Elsevier B.V. All rights reserved.

Keywords: Large-scale distributed systems; Network modeling; Latency estimation

1. Introduction

Modern large-scale distributed applications can benefit from information about latencies observed between their various components. Knowing such latencies, a distributed application can organize its operation such that the communication delays

between its components are minimized [1–3]. For example, a content delivery network can place its hosted data such that its clients are serviced at their proximal datacenters [4,5]. In addition to improving the client-experienced latency, reducing the overall length of client-to-replica network paths allows one to localize the communication, leading to lower backbone and inter-ISP link utilization. Analogous benefits can be achieved for other large-scale distributed applications such as peer-to-peer overlays or online gaming platforms.

* Corresponding author. Tel.: +31 20 598 7748; fax: +31 20 598 7653.

E-mail address: michal@cs.vu.nl (M. Szymaniak).

The effectiveness of latency-driven techniques in improving the application performance depends on the accuracy of the latency information. A simple solution consists of periodically probing each latency the application needs to know [6]. However, such an approach makes sense only in relatively small systems, as continuous probing of pair-wise latencies is clearly not feasible when the number of nodes is very large. For example, re-directing clients to their nearest datacenters would require Google to maintain latency information from virtually every Web client in the Internet to each of its datacenters [7]. Also, the high dynamics of the Internet causes recently measured latencies to not always be a good indication of their current counterparts, as one latency measurement result is not a good predictor of a subsequent identical measurement. These two problems drive the need for scalable and accurate latency estimation techniques.

A promising approach to the problem of scalable latency estimation is GNP, which models Internet latencies in a multi-dimensional geometric space [8]. Given a small number of “base” latency measurements to a number of dedicated “landmark” nodes, GNP associates each node with its coordinates in that space. The latency between any pair of nodes can then be approximated with the Euclidean distance between their corresponding coordinates. What makes GNP scalable is the constant low number of measurements necessary to position each machine, which enables GNP to estimate all-pair latencies between a large number of machines at low cost.

The attractiveness of GNP has resulted in its various aspects being investigated for several years. However, whereas numerous theoretical properties of GNP have been described in detail [9–14], no GNP implementation has been demonstrated to work in a large-scale environment of a commercial content delivery network.

The common property of existing GNP implementations that hinders their deployment is active participation of positioned nodes, which are responsible for measuring and propagating their own base latencies [15–18]. Such an approach has several disadvantages. First, it introduces problems with malicious nodes lying about their base latencies. Handling such nodes is usually very hard, and typically comes at the expense of increased system complexity. Second, independent measurements of base latencies performed by many active nodes might overload both the network and the landmarks. This,

in turn, might lead to numerous measurement inaccuracies affecting the GNP performance. Finally, active participation typically requires that some special positioning software is deployed on a significant fraction of positioned nodes. This condition might be infeasible to meet, for example, in content delivery networks, where most nodes are unmodifiable third-party Web browsers.

This article presents a GNP implementation that addresses all these issues. Our solution is based on two key observations. First, instead of relying on remote nodes to measure and report their base latencies, one can simply trigger some standard application-level communication between these nodes and the landmarks, allowing the latter to measure latencies passively on their side. This eliminates the need for customizing the remote nodes and ensures the integrity of measurement results. Second, instead of allowing remote nodes to independently perform their measurements, one can trigger measurements individually using a central, yet scalable, scheduler. This prevents landmarks from overloads and reduces the overall network overhead in general, as the scheduler triggers only the measurements that are really necessary. We demonstrate the feasibility of our approach by incorporating GNP into the content delivery network operated by Google, which enables us to position millions of Google clients.

Compared to the previous GNP implementations, our approach has several advantages. First, it greatly facilitates system deployment, as only the landmarks and the scheduler need to be instrumented. Second, it removes the problem of malicious nodes, as all the instrumented nodes are kept under full control of Google. Third, it eliminates the risk of overloading the landmarks, as the scheduler effectively adjusts the measurement volume to the landmark capacity.

Implementing our system at the scale of millions of clients requires one to address a number of subtle issues. For example, it is necessary to transparently and efficiently schedule measurements such that they do not affect the client-perceived browsing performance. Also, implementing a centralized scheduler is far from trivial when millions of Web clients are serviced by thousands of globally-distributed Web servers [19]. Finally, producing GNP coordinates that can remain representative for a long time requires that some special preprocessing techniques are applied to base latencies.

Within the first 2 months of operation, our positioning system performed more than 75 million

latency measurements to more than 22 million unique Google clients. Using host clustering techniques allowed us to compute the coordinates of more than 200 million Internet hosts falling into more than 880,000 of /24 networks. To our best knowledge, this is the largest experiment involving network positioning performed so far.

Our study confirms many earlier results, and adds to them by extensively investigating the issue of coordinate stability over time. Stability of results is important from the perspective of maintaining a large-scale distributed system, in which decisions based on latency information often have long-lasting effects. This is what happens, for example, when clients are re-directed using DNS, which can cache and re-use re-directing decisions for a long time. We show that coordinates drift away from their initial values with time, making 25% of the coordinates to be off by more than 33 ms after one week. However, daily re-computations make 75% of the coordinates stay within 6 ms of their initial values. We also recommend to derive daily coordinates from base latencies measured until around 10 pm UTC, as it results in coordinates remaining representative throughout the most of the next 24 hours.

Our another contribution to understanding the practical applicability of GNP in real-life systems is the performance analysis of coordinate-based client re-direction. We demonstrate that using latency estimates to decide on client-to-replica re-direction leads to selecting replicas closest in term of *measured* latency in 86% of all cases. In another 10% of all cases, clients are re-directed to replicas offering latencies that are at most two times longer than optimal. Also, we show that positioning Google clients makes it possible to estimate latencies between globally distributed Internet hosts that have not participate in our measurements. We treat this result as an incentive to develop a new publicly available Google service providing pairwise latency estimates for Internet hosts.

The remainder of this article is structured as follows: We discuss a number of related research efforts in Section 2. Then follows the description of our system: Section 3 describes how we integrated GNP into the Google infrastructure, Section 4 shows how to compute stable coordinates, and Section 5 discusses our experience with GNP-based client re-direction. Section 6 evaluates the performance of our system as an application-independent latency estimation service. Finally, Section 7 concludes by summarizing our future development plans.

2. Related work

2.1. Internet node positioning

GNP was the first system to propose modeling the Internet as an N -dimensional geometric space [8]. Given such a space, GNP approximates the latency between any pair of hosts as the Euclidean distance between their corresponding coordinates in that space.

The space is determined by the coordinates of “landmark” hosts that GNP computes first. The number of landmarks k must be at least $N + 1$ to unambiguously determine the N -dimensional geometric space. Given the k landmark coordinates, GNP can compute the coordinates of any host X based on the measured latencies between X and each of the k landmarks. By treating these latencies as distances, GNP triangulates the coordinates of X relative to the landmark coordinates.

The landmark coordinates are computed as follows: First, GNP instructs the landmarks to measure their latencies to each other. Based on these latencies, GNP calculates all the landmark coordinates so that the distance between any pair of these coordinates is as close as possible to the latency measured between the corresponding pair of the landmarks (see Fig. 1a). The discrepancy between the distances and their corresponding latencies is minimized using a popular error-minimization algorithm called Simplex-downhill [20].

Once the landmark coordinates are known, GNP can determine the coordinates of any host X based on the measured latencies between that host and each of the landmarks. The coordinates of X are calculated so that the distance between these coordinates and the coordinates of each landmark is as close as possible to its corresponding measured latency (see Fig. 1b). This is again achieved by means of the Simplex-downhill algorithm. The GNP authors show that, in 90% of cases, the latency estimations produced by their system are within a relative error ratio of .53 compared to the real latency.

2.2. Positioning variants

A number of variants has been proposed to the original GNP concept. The PIC project suggested that at least some of the landmarks should be located close to the positioned hosts to improve the positioning accuracy [15]. When positioning a global community of Web clients, this suggestion

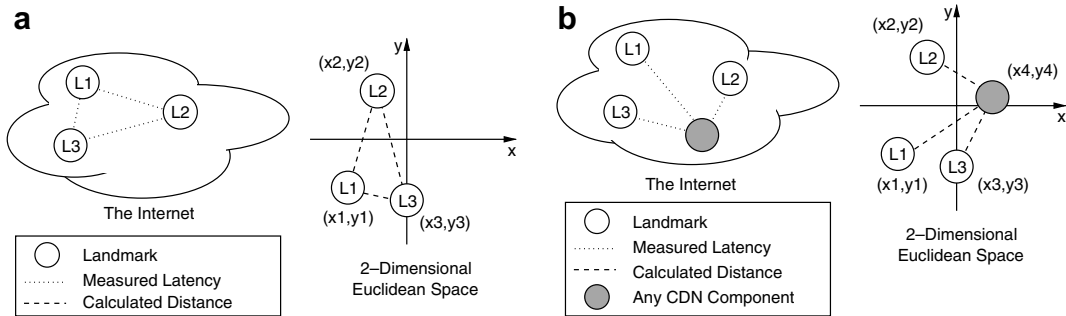


Fig. 1. GNP: landmark positioning (a), and host positioning (b).

is equivalent to that from another study, which recommends to globally distribute the landmarks in order to achieve higher positioning accuracy [10]. We discuss some practical implications of these suggestions in Section 3.1.1.

Another project established that the accuracy and stability of coordinates can be improved by statistical filtering of latency samples used for positioning [14]. The intuition is that long-term coordinates should not be affected by temporary and intermittent network conditions such as network congestion. This can be prevented by computing coordinates based on latencies typical for given landmark–host pairs. We verify these findings in our experiments presented in Section 4.2. Compared to [14], we rely on a much larger and more diverse trace of latencies. We also investigate the issue of how to determine typical latencies, and how often the resulting coordinates need to be re-computed.

The issue of positioning scalability has been addressed in the Lighthouses project [16]. It demonstrated that hosts can also be positioned relative to any previously positioned hosts, which in that case act as “local” landmarks. This eliminates the need for measuring latencies to the original landmarks each time a host is positioned, in turn leading to a distribution of the measurement effort resulting in higher positioning scalability. However, as we show below, one can position a huge community of Web clients by relying on the original landmarks only, as long as the measurements performed by the landmarks are appropriately scheduled. This also enables us to avoid the loss of accuracy that using local landmarks inherently incurs.

Following the idea of Lighthouses, our earlier SCoLE project showed that latencies estimated in completely different spaces are highly correlated [21]. Such correlation enables different hosts in a distributed system to construct their own spaces

and effectively run their private GNP instances. This improves system scalability, as there is no need for all the members of the distributed system to negotiate common GNP parameters. However, since our Google implementation uses only one set of GNP parameters, it does not benefit from these findings.

Other research efforts replace the Simplex-downhill computation used in GNP with simpler optimization schemes [11,13]. In fact, the selection of a particular positioning algorithm is orthogonal to the question of how to measure latencies required for positioning, as long as all the algorithms require the same set of latencies to be measured. We chose to compute all the coordinates using the Simplex-downhill algorithm recommended in the original GNP paper, as it has performed well when used in our other research projects.

The remaining efforts take a completely different approach and position all hosts simultaneously as a result of a global optimization process [9,17,22]. In that case, there is no need to choose landmarks, since every host is in fact considered to be a landmark. The respective authors claim that it leads to better accuracy. However, Google cannot generally rely on its clients to measure latencies to each other, which renders these techniques unattractive in our case.

2.3. Positioning implementations

A recent study by the authors of the original GNP paper describes how to implement a global Network Positioning System (NPS) based on GNP [18]. The authors identify four key system-building issues that must be addressed by any GNP implementation: maintaining a single global space, adapting to changes in Internet routes, handling fluctuations in network latencies, and computing positions as accurately as possible.

NPS addresses the key building issues by organizing hosts interested in positioning into a distributed infrastructure in which each host periodically recalculates its own coordinates. All the coordinates are calculated in the same geometric space, determined by a fixed set of global landmarks. NPS prevents these landmarks from becoming performance bottlenecks by allowing the hosts to position themselves relative not only to the landmarks, but also to any other “reference” hosts whose coordinates are already known. In that sense, NPS generalizes the concept of local landmarks introduced by Lighthouses. On top of that, NPS enables each of the landmarks to compute its coordinates locally by means of a special scheme for decentralized landmark positioning, and exploits some other distributed algorithms to synchronize positions computed by different hosts.

The distributed nature of NPS results in improved scalability. However, it also forces NPS to deal with a number of problems that result from the distribution itself, such as preventing malicious hosts from being used as positioning references, synchronizing distributed latency probing to prevent reference hosts from being overloaded, or triggering host repositioning to maintain global consistency of coordinates. Solving these problems makes NPS relatively complex. On the other hand, following our centralized approach enables one to avoid all these problems without limiting the system scalability. As a result, our solutions to the four key building issues identified by NPS are much simpler.

3. System architecture

Using GNP to position Google clients seems to be relatively simple. Essentially, the positioning process can be split into three phases (see Fig. 2): measuring base latencies, collecting the measurement results, and modeling latencies in the form of GNP coordinates. The coordinates can then be passed to any latency-driven applications, such as those responsible for client re-direction or replica placement.

However, as it turns out, naive implementations of either phase in a large-scale Internet service will easily show poor results. This is caused by a number of subtle problems that arise when deploying GNP in a real-world setting. The following sections discuss how we addressed these problems when implementing each phase of the positioning process.

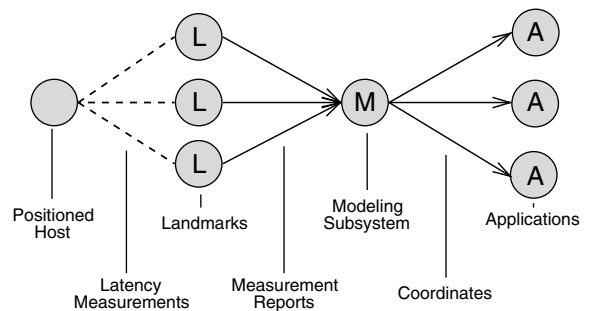


Fig. 2. High-level concept of positioning implementation.

3.1. Landmark infrastructure

3.1.1. Landmark deployment

GNP computes the coordinates of each host based on a number of so-called base latencies to that host. Base latencies are measured by landmarks, which must be deployed by the service. Deploying landmarks essentially consists of three steps: deciding on the number of landmarks, on their approximate location, and, finally, on the actual hosting facility where they should be installed.

The first step is to decide on the number of landmarks to deploy. Although GNP is able to compute coordinates using any number of landmarks, previous studies have recommended running at least seven landmarks to obtain good positioning accuracy [11,21]. Although we use that number of landmarks in our experiments, in practice we also run a number of redundant landmarks to increase the system’s resilience to landmark failures.

The second step is to choose approximate geographical locations for the landmarks. As mentioned in Section 2, the landmarks should be globally distributed. This is because GNP relies on the assumption that vectors of landmark-to-host latencies are different for hosts located in different parts of the Internet. Should we fail to meet this assumption, then the performance of GNP might turn out to be poor.

To confirm that global landmark distribution is indeed necessary in practice, we evaluated the accuracy of GNP offered by various combinations of landmarks located in different parts of the Internet. To this end, we chose 20 PlanetLab nodes [23] to act as candidate landmarks, and connected them to our positioning system. This allowed us to collect a large set of latencies between the candidate landmarks and a small fraction of Google clients.

The clients in the set turned out to originate from 113 countries, with the number of clients per

country varying from 1 to many thousands. To make the evaluation fair for all the countries, we randomly picked 10 clients from each country. For countries represented by less than 10 clients in our trace, all the clients were included. The resulting test set consisted of 616 clients.

Having generated the test set of clients, we iteratively positioned them relative to various combinations of seven landmarks. The subsequent combinations consisted of manually selected landmarks that were increasingly distributed in a geographical sense. For each combination, we evaluated its offered estimation accuracy based on the latencies measured between the clients and the 13 PlanetLab nodes that were not used for positioning. To this end, we calculated the relative estimation error $\varepsilon(\cdot)$ for each such latency similar to GNP:

$$\varepsilon(d_{CL}, \hat{d}_{CL}) = \frac{|\hat{d}_{CL} - d_{CL}|}{\min(d_{CL}, \hat{d}_{CL})},$$

where d_{CL} and \hat{d}_{CL} , respectively, denote the measured and estimated latencies between client C and landmark L . The distribution of estimation errors observed for four example landmark combinations is depicted in Fig. 3.

As can be observed, estimation accuracy is lowest when all the landmarks are located in the US. The combination consisting of four American- and three European landmarks offers better accuracy, which improves even further when three of the seven landmarks are located in Asia (Tokyo, Singapore, and China). The best accuracy is offered by the fourth combination, wherein the landmark in Tokyo is replaced with a Brazilian one. This confirms the importance of global landmark distribution, and allows for reaching estimation accuracy close to those reported in our previous study [21]. Note that this estimation accuracy might be improved further by means of applying other embedding algorithms [9,24].

The last step of landmark deployment is to choose the actual hosting facilities where the landmarks should be installed. It may seem attractive to deploy landmarks in existing service datacenters to benefit from hardware that is already in place. However, the number or locations of such datacenters may not meet the global landmark distribution requirement. In that case, we need to decouple the placement of landmarks from the locations of the datacenters by constructing an infrastructure of dedicated landmarks rented from third-party host-

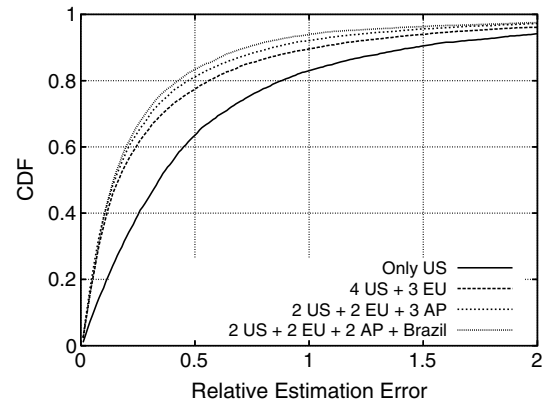


Fig. 3. Importance of landmark distribution.

ing facilities worldwide. In our experiments, we used the best set of PlanetLab nodes as our landmark set.

3.1.2. Latency collector

All the latencies measured by the landmarks must be collected and passed to some modeling component for processing. However, the modeling component typically runs in one of the datacenters. Given that datacenters are normally tightly firewalled, the landmarks deployed outside the datacenters cannot contact the modeling component directly.

One solution to that problem would be to reconfigure the datacenter firewalls to allow incoming traffic from the landmarks. However, doing so potentially exposes the service to attacks initiated from the landmarks. The potential problem becomes even worse when the landmarks are operated by external organizations such as PlanetLab. This solution should therefore be avoided unless there are no other options available.

We therefore decided to follow another approach, in which latencies are collected using network connections opened from some dedicated component residing in one of the datacenters to the landmarks. This component, called a *collector*, retrieves latencies from the landmarks and stores them in measurement logs accessed by the modeling component. The collector-to-landmark connections are protected with SSL for secure communication.

3.2. Latency measurements

3.2.1. Measurement types

Once the landmark infrastructure has been deployed, we can start collecting latencies. There are essentially three kinds of latencies to be mea-

sured. First, the landmarks must measure latencies between each other, as GNP requires this information to construct its geometric space. This can easily be achieved by means of periodical active probing, which is the simplest way of discovering latencies between any two machines under our control.

Second, the landmarks must measure their latencies to each datacenter so that the datacenters can be positioned as well. Computing the coordinates of datacenters is necessary to estimate client–datacenter latencies, which can then be used during client re-direction. Given that datacenters are operated by the service, the landmarks can discover their latencies to the datacenters by actively probing them just like they probe each other.

Third, the landmarks must determine their latencies to Google clients so that the coordinates of these clients can be computed as well. However, we cannot use active probing this time, as it is likely to trigger various intrusion–detection systems deployed on the client side. This could result in numerous client complaints affecting the service reputation.

Rather than actively probing clients, the landmarks can measure their latencies to the clients without initiating any traffic to these clients. To this end, the landmarks must rely on passive latency discovery, wherein latency measurements can be obtained by monitoring the service traffic and deriving the client latencies from the dynamics of packets constituting that traffic.

A well-known technique for passive latency discovery is the SYNACK/ACK method [25]. It enables a server to estimate its round-trip time to a client when the client initiates a TCP connection to the server. The round-trip time can then be estimated during the TCP hand-shake phase as the delay between sending the SYNACK packet and receiving its corresponding ACK packet (see Fig. 4). We chose this technique for its natural applicability in Web systems, wherein network traffic is typically carried over TCP connections.

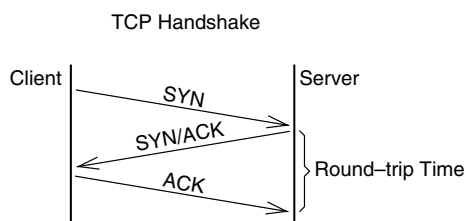


Fig. 4. Passive latency discovery with SYNACK/ACK.

3.2.2. Measurement triggering

Using SYNACK/ACK to measure the latency between a client and a landmark requires that the client opens a TCP connection to the landmark. However, the clients issue requests only to datacenters, which are separated from the landmark infrastructure. We must therefore implement a mechanism causing clients to open additional TCP connections to the landmarks.

In general, Google clients are regular Web browsers. A natural way to make them open TCP connections to the landmarks consists of deploying Web servers on the landmarks and instructing the clients to fetch content from these Web servers.

We can easily instruct Web servers to fetch content from the landmarks by embedding some small landmark-delivered objects inside Google Web pages. A classical example of such objects is a tiny image, which is commonly used by the providers of Web site statistics to track site accesses [26]. However, the major drawback with such an approach is that it makes the client experience dependent on the landmark performance, as Web pages can be displayed in their final shape only after all their parts have been retrieved. Datacenters are typically tuned to offer reliable service of high quality to a huge number of clients, but the landmarks are likely to be incomparably less reliable and powerful. Should any landmark face reliability or performance problems, then these failures may become visible to users, and in turn compromise the overall service performance.

Solving this problem requires that the landmark-delivered objects are embedded in such a way that the client-perceived service performance does not depend on the landmarks. In particular, a Web browser should be able to display complete service responses even if the embedded objects cannot be downloaded.

This transparency can be achieved in two ways. First, the service might rely on JavaScript code included in a response to retrieve a number of objects from the landmarks after the response has been displayed [27]. This approach is appealing because JavaScript is supported by most Web browsers. However, the semantics of retrieval failures varies across different JavaScript implementations, which makes it hard to guarantee that running JavaScript code never results in unexpected browser behavior [28]. Since one of our priorities was to keep the user's perception of Google untouched, we decided not to risk compromising it by using JavaScript.

Another transparent way of embedding objects is to use server-directed prefetching capabilities of certain browsers [29]. This technique enables a Web server to instruct browsers to retrieve a given object *after* the entire response has been displayed. Prefetching is typically used to accelerate the download of Web documents that clients are likely to be requesting next [30]. However, it can also be used to trigger the retrieval of landmark-delivered objects.

The service can pass prefetching instructions to Web browsers in the form of special HTTP headers or HTML tags embedded inside its responses [31]. Each such instruction contains the URL of an object that a Web browser should retrieve. In contrast to regular object retrieval, however, Web browsers keep their users unaware of any delays or failures that might occur during prefetching. This guarantees that prefetching does not affect client-perceived service performance.

We decided to employ prefetching to trigger the retrieval of landmark-delivered objects. To this end, we modified Google Web servers to embed prefetching instructions inside their responses such that each tag points at an object hosted by some landmark. This causes the clients to open HTTP connections to the landmarks, which can then perform passive latency discovery.

A potential limitation of prefetching is that it is currently supported only by the Mozilla Firefox Web browser [32]. This means that Google can only trigger prefetching requests from approximately 13% of its clients [33]. However, prefetching features are planned to be supported by the future releases of Internet Explorer browser as well [34]. Also, measuring latencies to a fraction of all the clients might turn out to be enough to position all Internet hosts, as we discuss next.

3.3. Measurement scheduling

The above sections have discussed two mechanisms that enable the service to trigger latency measurements: active probing and embedding of prefetching instructions. Whereas the configuration of active probing is relatively straightforward, deciding on how to trigger measurements with prefetching is much harder.

Obviously, the service needs to trigger all the measurements necessary to position its clients. However, while doing so, it should respect the following three conditions: First, it should trigger only as many measurements as each of the landmarks can handle, as

overloaded landmarks cannot measure latencies accurately. Second, it should also keep the total number of measurements low to reduce client-side overhead. Third, it should avoid triggering redundant measurements to minimize network usage.

The following sections describe how our system meets each of these three requirements using a centralized scheduling policy. We then propose how such a policy can be implemented in a large-scale system in which responses are simultaneously generated by the thousands of Web servers that constitute the Google infrastructure [19].

3.3.1. Landmark load

In a naive approach, the service could include prefetching tags in all its responses to perform as many measurements as possible. However, doing so would most likely lead to overloading the network connections to the landmarks, resulting in latencies being measured with high inaccuracies.

Overloading the landmarks can be avoided by limiting the number of measurements performed by each landmark. To this end, the service can enforce some delay between subsequent measurements scheduled to each landmark such that the landmark capacity is never exceeded. The distinguishing property of this time-sharing scheme is that it can be easily distributed over multiple scheduling components, which we benefit from below. It is also very easy to implement, as it only needs to maintain a timestamp of the most recent measurement scheduled to each landmark.

3.3.2. Client clustering

Scheduling individual measurements should ultimately result in collecting all the latencies necessary to position all the clients. However, since the clients might consider measurements to be an unnecessary burden, the service should strive to minimize that burden by reducing the number of measurements.

We decided to reduce the number of measurements issued to the clients by means of clustering, which is a popular technique for reducing the number of operations performed in a distributed system. In principle, clustering groups machines into so-called clusters, and performs the operations on a per-cluster rather than on a per-machine basis. In our case, clustering reduces the number of measurements by grouping clients whose latencies to a given landmark are very similar.

Efficient scheduling requires that clustering is fast, which limits the selection of clustering schemes to

very simple ones. An example of such a scheme is clustering of machines whose IP addresses share the same 24-bit prefix. We call each such cluster a /24 network, and identify each such network with its 24-bit prefix. Given that each /24 network can contain up to 254 machines, /24 clustering can reduce the number of measurements by up to two orders of magnitude.

However, relying on /24 clustering when performing latency measurement is possible only if latencies measured to the clustered machines are similar. To validate whether this condition is met in the Internet, we calculated 10–90 percentile ranges for latencies measured to different clients in the same /24 networks.

The percentile ranges were calculated based on the latency trace collected by our system. First, we extracted latencies measured by the landmark running at MIT during a two-week period. The duration of two weeks was chosen to limit the impact of routing changes on the observed latencies. Second, we identified all the /24 networks containing at least three different clients in the two-week trace. The number of such networks turned out to be 28,540. Third, we obtained an indication of the landmark’s latency to each client by calculating a median for each landmark–client pair. Finally, for all the clients in each network, we evaluated how close their median latencies are to each other. To this end, we calculated the 10–90 percentile range over the set of medians, and divided that range by the mean median latency for that network. The resulting distribution of 10–90 percentile range coefficients is depicted in Fig. 5.

As can be observed, in over 91% of /24 networks, the coefficient of the 10–90 percentile range is lower than .2. This means that, in 91% of /24 networks,

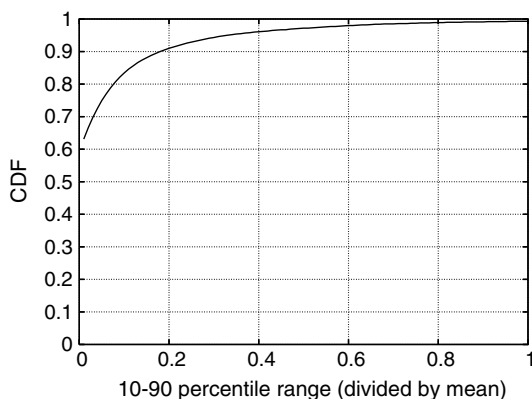


Fig. 5. Variation of latencies to hosts within a /24 network.

median latencies to 80% of clients differ by at most 20%. Such a low variation enables the landmarks to measure their latencies to any client in a network, and treat these latencies as representative for any other clients in that network. Note that /24 clustering enables to position all the clients in a given /24 network only if at least one of them supports pre-fetching. According to our data, this condition is met by about 85% of /24 networks containing Google clients. The remaining clients can be positioned when a more aggressive clustering scheme is used, as we discuss in Section 6.

3.3.3. Redundant measurements

Positioning a /24 network requires measuring latencies between that network and all the landmarks. This can be achieved by triggering measurements from a given /24 network to the landmarks in a round-robin fashion. To this end, subsequent service responses sent to each network contain pre-fetching tags pointing at objects hosted by subsequent landmarks.

A potential problem is that starting all the round-robin sequences from the same landmark is likely to cause that landmark to be fully loaded. In that case, the mechanism responsible for limiting the landmark load will prevent many measurements from being performed. The service can avoid this problem by using random initial landmarks in round-robin sequences specific to different /24 networks.

Another problem with round-robin scheduling is that it keeps triggering measurements from a given network even after a complete set of landmark latencies to that network has been collected. The redundant measurements are of little use to the positioning system and might prevent the service from triggering more useful latencies when the landmark load increases.

We chose to avoid triggering redundant measurements by simply limiting the number of round-robin sessions to a given network. For example, once a complete set of latencies has been collected for a given network, no other measurements are triggered to that network for some time. The duration of the interval between sessions generally depends on how often new coordinates are being computed. In the current setup, we allow only one round-robin session per /24 network every hour.

3.3.4. Scheduling policy

The complete scheduling policy consists of three steps taking place every time a measurement can

be triggered to some client. First, the policy determines the client's /24 network by dropping the last 8 bits of the client's IP address.

Next, the policy inspects the round-robin state specific to that network and checks whether any more measurements should be performed to it in its current round-robin session. If not, then no measurement is triggered. Otherwise, the policy identifies the next landmark that should perform the measurement.

Finally, the policy verifies the approximate load of the selected landmark. If that landmark is currently overloaded, then no measurement is triggered. Otherwise, the policy updates both the round-robin state and the landmark load information, and instructs the service to trigger a measurement between the client and the landmark.

3.3.5. Scheduler separation

Although the scheduling policy is conceptually simple, it is not obvious how to implement it in a large-scale Web system. This is because it requires the service to maintain state for round-robin landmark selection and an approximation of landmarks' load. The service needs this information to decide which of its generated Web pages should contain prefetching instructions.

Unfortunately, given the large number and wide-area distribution of Web servers in a large-scale Web system, it is unlikely that they can efficiently share state among them [19]. This is because of frequent updates that make the state difficult to keep consistent without degrading the scheduling performance, even though the state itself is relatively small (about 8 bytes for round-robin information per cluster, plus another 8 bytes per landmark for the load information).

We decided to solve this problem by splitting the measurement-triggering mechanism into two parts (see Fig. 6): First, while responding to regular client requests, the Web servers implementing the service include *static* prefetching tags into a small fraction of their responses. Static prefetching tags do not point at any particular landmark. Instead, they point at a dedicated cluster of Web servers taking care of measurement scheduling.

The second part of the triggering mechanism is implemented by the scheduling cluster. Each machine in the cluster maintains its local scheduling state, and processes an even share of all the requests triggered by the static prefetching tags. For each such request, it invokes the scheduling policy to select the target landmark for the measurement that the request can potentially trigger.

The scheduling policy might sometimes decide not to trigger any measurement for a given prefetching request, for example when all the landmarks are overloaded. In that case, the prefetching request is serviced locally by the scheduling cluster. Note that the scheduling cluster could even exploit the performance-neutral nature of prefetching requests and drop them completely.

Typically, however, the scheduling policy returns the address of some landmark. The scheduling cluster can then re-direct the prefetching request to that landmark using an HTTP-302 response [31]. This causes the clients to re-issue the prefetching request to the landmark exactly as if the landmark address was put in the prefetching tag embedded inside the original service response. Note that although the content prefetched from the landmarks is never displayed to the users, it can still contain some brief information about the measurements being performed. This helps preventing users from becoming

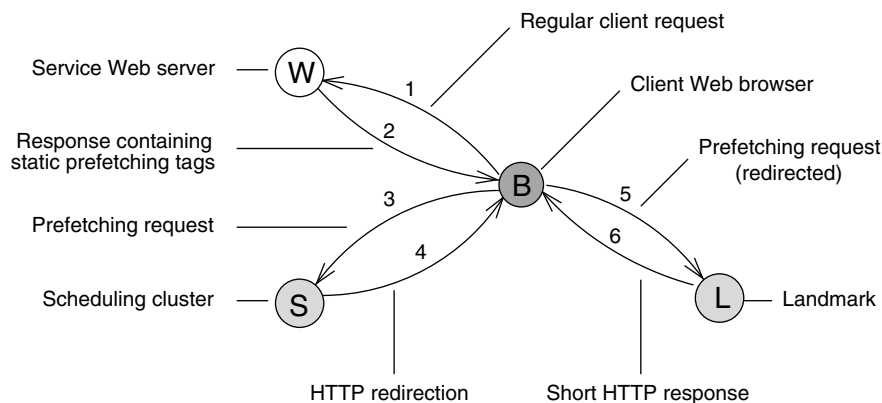


Fig. 6. Two-phase measurement triggering.

suspicious about the prefetching requests after they are detected by client-side firewalls.

3.3.6. Web server logic

Embedding static prefetching tags prevents the regular Web servers from maintaining any scheduling state, as all the prefetching tags always point at the URL of the scheduling cluster. However, the Web servers must still be able to decide whether a given response should carry a static prefetching tag, or not. For now, we assume that Web servers insert at most one prefetching tag per response.

One way of enabling the Web servers to decide on insertion of prefetching tags would be to rely on client identifiers embedded in service cookies. In that case, the Web servers would include prefetching tags in responses sent to clients holding cookies with identifiers meeting the condition that

$$ID_{client} \% X = 0,$$

where X denotes some divisor value, which can be used to adjust the number of generated prefetching tags to the capacity of the scheduling cluster. An attractive property of this approach is that it keeps triggering measurements from the same clients, which should intuitively result in quickly collecting multiple measurements required to position these clients.

However, triggering measurements from the same group of clients results in only a small fraction of all the /24 networks being ultimately positioned. This can be observed in Fig. 7a (the ‘Cookie’ line), which indicates that only about 250,000 out of the total 1.2 millions of client /24 networks were positioned after 300 hours.

The positioning coverage can be improved by inserting static prefetching tags purely at random. To this end, the Web servers include prefetching tags when

$$random() \% X = 0,$$

where X can again be adjusted to the capacity of the scheduling cluster. As can be observed in Fig. 7a (the ‘Random’ line), this approach results in a larger number of /24 networks being positioned in the long run (500,000 after 300 hours), even though relying on cookies might initially seem to perform better.

An interesting question is how many static prefetching tags should be embedded in a single service response once the decision has been made that there should be any. Clearly, inserting more tags results in triggering more measurements at the cost of increasing the load at the clients and the landmarks. On the other hand, collecting more measurements should also result in a larger number of /24 networks being positioned the same time, as the seven measurements necessary to position each network are collected faster.

Fig. 7b depicts the dependency between the number of positioned networks and the number of static prefetching tags embedded in a single response. As can be observed, inserting more prefetching tags in a single response indeed helps to collect measurements faster. For example, inserting four tags per response allows for positioning more than 800,000 of /24 networks after about 450 hours, instead of 1200 hours necessary to position these networks when only one prefetching tag is embedded. Inserting seven tags per response, in turn, allows for reducing that time to 160 hours, which is less than one week.

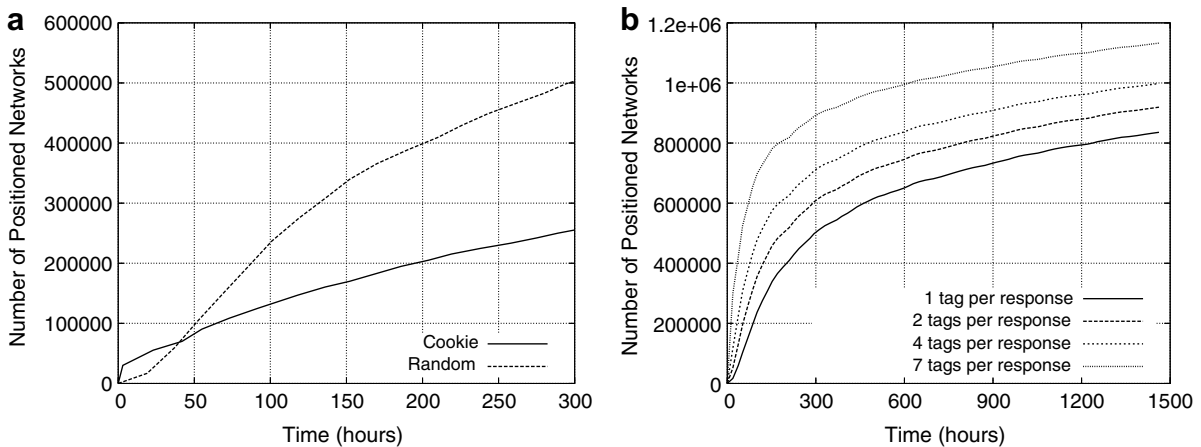


Fig. 7. Impact of different tag-embedding strategies (a), and different numbers of tags (b).

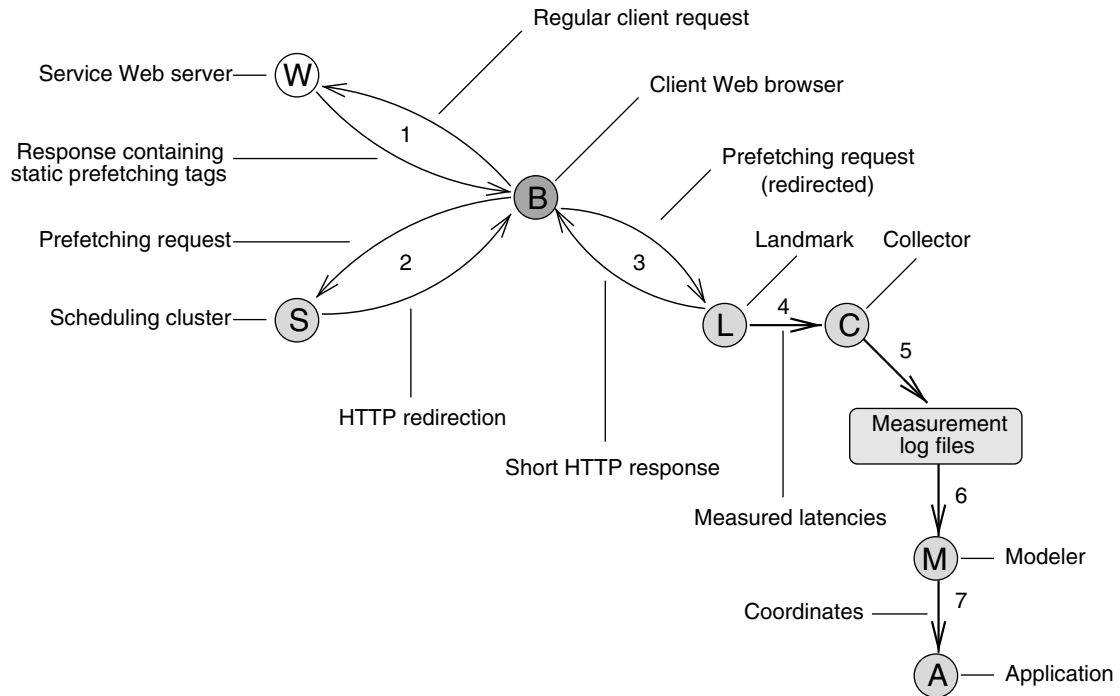


Fig. 8. Final system architecture.

3.4. Final architecture

The final system architecture is depicted in Fig. 8. Latency measurements to the clients are triggered by service Web servers, which embed static prefetching tags inside a fraction of their responses.

The prefetching tags cause the clients to issue prefetching requests to the scheduling cluster, which re-directs these requests to the landmarks according to the scheduling policy. This causes the clients to re-issue the requests to the landmarks, which perform latency measurements while delivering a short system description. All the measured latencies are reported to the collector.

Once the latencies have been collected, they are stored in measurement logs. These logs are periodically retrieved by a special component called *modeler*, which processes the latencies and computes new sets of coordinates, as we discuss next.

4. Latency modeling

The modeler essentially performs two types of tasks: First, it creates a geometric space by computing the landmark coordinates. Second, it computes all the other coordinates relative to the landmark

coordinates. Since both these tasks require some set of latencies as the input, it is tempting to directly apply the positioning algorithm to the base latencies stored in the measurement logs.

However, GNP requires its input to contain only one indication of latency between a given pair of machines. On the other hand, the measurement logs produced by the controller are likely to contain multiple such indications, as each landmark typically measures its latency to the same /24 network many times. Since subsequent latency measurements between the same pair of nodes are likely to return fluctuating results, the modeler must preprocess the measurement logs before their contents can be passed to the GNP implementation.

4.1. Stable latencies

In principle, latencies measured between a given landmark–node pair can fluctuate for two types of reasons. The first types are temporary intermittent conditions that do not affect long-term latencies between landmarks and nodes, such as network congestion and high CPU load. The second types are route changes, which can permanently change latencies between nodes. The goal of a good latency

preprocessor is to eliminate fluctuations caused by the intermittent conditions while remaining reactive to permanent latency changes.

Clearly, network congestion can affect the observed latencies. If the path between the landmark and the node is saturated, the measurement packets are delayed by routers on the path, causing the observed latency to be longer. Note that the service should strive to reduce the impact of network congestion by avoiding it on the landmark side. This can be achieved by deploying the landmarks in hosting facilities providing hard bandwidth guarantees.

Apart from network congestion, latencies can also fluctuate because of high CPU load on either the node or the landmark. The problem with high CPU load on the node is that it might prevent the node from immediately responding to packets sent by the landmark. This can result in observed latencies being longer than they really are. On the other hand, since the packets exploited by both ICMP probing and SYNACK/ACK are handled entirely by the operating system kernels, the delay caused by high load of the node's CPU is likely to be negligible.

High load on the landmark presents a bigger problem, as it can prevent the packet sniffer running on the landmark from timestamping measurement packets accurately. The resulting inaccuracies strongly depend on sniffer implementation. We therefore assume that the observed latencies can not only be higher, but also lower than they really are.

Given that temporary intermittent conditions occur only occasionally, their resulting measurement inaccuracies can be eliminated through statistical filtering. To this end, the modeler could maintain a history of latencies measured between each landmark–node pair, and identify the real latency for that pair as the one occurring most commonly in the history. This could be achieved by means of medians, for example.

However, median latencies can change over time as well. This is caused by long-lasting conditions, such as route changes. As the route between the landmark and the node changes, its corresponding history of latencies contains more and more groups of latencies, each measured for a different route. In that case, medians calculated over complete latency histories are not guaranteed to indicate current real latencies.

We decided to detect route changes by applying the sliding percentile concept to the latency history

[14]. To this end, it keeps only a specific number of most recent measurements in each history, which should result in history medians being closer to the actual observed latencies.

We verified the impact of sliding percentiles on measurements used for positioning. To this end, we applied them to the latency trace collected by our system, and evaluated their performance. The trace spanned a period of six weeks and contained latencies to Google clients measured by one of our PlanetLab landmarks located in MIT. To ensure fair comparison, we analyzed latencies to only the 10,000 networks that occurred most frequently in the trace (57 times on average). The performance of sliding percentiles was evaluated by calculating the relative error between observed latencies and their corresponding values after filtering with sliding percentiles. The resulting error distribution for various configurations of sliding percentiles is depicted in Fig. 9.

As can be observed, using sliding percentiles indeed enables one to identify current latencies more accurately, although the improvement is not very high. However, these small improvements result in significantly higher stability of coordinates, as we demonstrate next.

4.2. Stable coordinates

Computing the coordinates for a given /24 network enables one to estimate latencies between hosts in that network and those in any other network whose coordinates are already known. This allows our request re-director to identify the data-center that is closest in terms of latency to clients in a given network, and re-direct these clients accordingly.

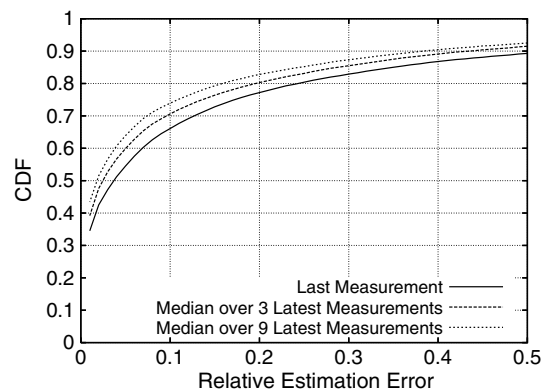


Fig. 9. Stabilization of measured latencies.

However, latency fluctuations cause the coordinates to change over time. The degree of these changes determines how useful the coordinates are to make long-term decisions, which are important for the above applications. For example, when client requests are re-directed using DNS, it can cache the responses produced by the re-directing DNS servers for several hours, which causes the re-directing decisions based on coordinates produced by our system to remain in effect for a relatively long time.

To investigate the influence of latency fluctuations on GNP coordinates, we evaluated the stability of coordinates produced by our system. We used the trace of latencies between the landmarks and the 10,000 most popular /24 networks selected for the previous experiment. We split the six-week trace into two parts. The first part was two-weeks long and was used as a basis to compute the initial coordinates of all the /24 networks. The remaining part of four weeks was used as a test trace, based on

which we investigated how the coordinates of /24 networks change over time in terms of distance to their initial counterparts. To this end, for every test trace hour, we re-computed the coordinates of all the /24 networks for which latency measurements were performed within that hour. This resulted in re-positioning on average 1271 networks every hour.

Ideally, at each hour, we would compute the distance between the current- and initial coordinates of each /24 network. In many cases, however, due to the lack of latency measurement within the last hour, it is impossible to compute the current coordinates directly. However, this does not mean that these coordinates did not change during that hour, but just that we did not measure latencies frequently enough.

Fig. 10 depicts how we approximated the missing coordinates for each network. Essentially, for each pair of coordinates computed during subsequent re-positioning operations, we assume that the missing coordinates between them change linearly. This enabled us to calculate the coordinates of all the networks for each test trace hour.

We evaluate the changes in coordinates during subsequent hours by calculating the median distance between the 10,000 coordinates calculated for a given hour and their initial counterparts. As shown in Fig. 11a, the coordinates change significantly when computed based on the most recent measurements (line ‘Last Measurement’). They also seem to increasingly deviate from their initial values over time, as the median distance between current and initial coordinates generally increases with time. However, this hypothesis was not confirmed by a

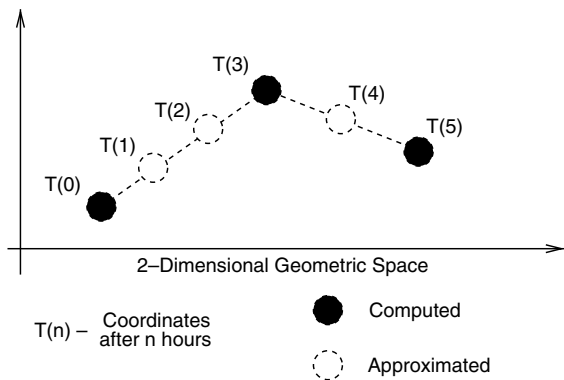


Fig. 10. Approximating missing coordinates.

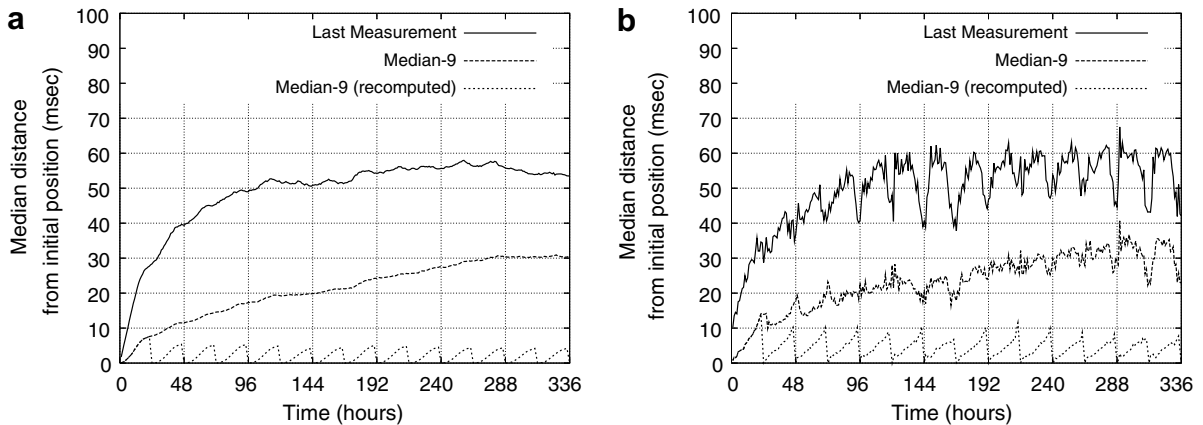


Fig. 11. Coordinate stability including approximated intermediate coordinates (a) and without them (b).

number of case studies we performed for individual networks. We therefore believe that the increasing trend is caused not by large deviation in coordinates, but by changes in the *number* of relatively small deviations. This number increases with time, as latencies to more and more networks become affected by route changes, leading the coordinates of these networks to be significantly different. The result of aggregating such differences calculated for 10,000 networks is the increasing trend in the median distance between the current- and initial coordinates.

Having observed the instability of coordinates computed based on the most recent latency measurements, we investigated whether the coordinate stability can be improved by computing coordinates based on latency measurements stabilized with sliding percentiles. To this end, we performed an experiment that was very similar to the previous one. The only difference was that the networks were re-positioned based on latencies filtered using sliding percentiles. We used median latencies calculated over the set of nine most recent measurements, including the approximated ones. The results are depicted in Fig. 11a (line ‘Median-9’).

As can be observed, sliding percentiles significantly improve the stability of coordinates. However, they do not eliminate the increasing trend, which limits the maximum time for which coordinates can be relied upon. To overcome this problem, each application would need to periodically re-compute coordinates so that they meet its requirements with respect to positioning accuracy. Line ‘Median-9 re-computed’ in Fig. 11a shows that daily re-computations can keep current coordinates within 8 ms of their initial counterparts. Note that we obtained similar results when performing the above experiments without approximating intermediate node coordinates (Fig. 11b).

How often coordinates should be re-computed depends on a trade-off between the positioning accuracy and the cost of computing and propagating the coordinates to the applications. To investigate this trade-off, we ran the above experiment with *initial* coordinates re-computed every X days, for X between 1 and 7. For each of the resulting seven simulations, we computed both the median- and the 75th percentile of distances between current coordinates and their most recently computed “initial” counterparts.

The results in presented in Fig. 12. They indicate that daily re-positioning reduces the median dis-

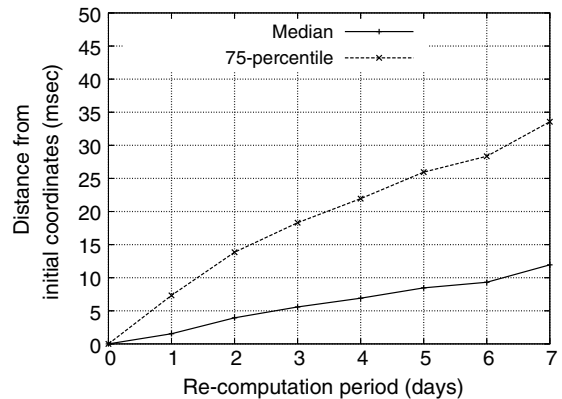


Fig. 12. Impact of different re-computation periods.

tance between current- and initial coordinates to only 1.53 ms, whereas re-computing coordinates every week results in that distance being 11.94 ms. However, the corresponding 75th percentiles of distances to initial coordinates are already 7.33 ms and 33.56 ms, respectively. In our experiments, we decided to re-compute all the coordinates on a daily basis, which, apart from offering very good stability, also makes the system very responsive to changes in network conditions.

When re-computing coordinates every day, an interesting question is whether the coordinate stability depends on the actual time of day when re-computations take place. To answer this question, we performed 24 simulations of daily re-positioning based on our 4-weeks-long test trace. Each simulation was configured to re-compute coordinates at a different trace hour. We evaluated the resulting stability by computing the 75 percentile of distances between current- and initial coordinates observed throughout the 24 trace hours after each re-positioning. The results are depicted in Fig. 13a.

As can be observed, the coordinates are most stable when re-computed around 10 pm UTC. In that case, the coordinates are computed based on measurements collected during busy Internet hours, which account for day time in the US and evening in Europe – the two continents where most of the 10,000 test networks are located. Our results indicate that such coordinates remain representative for the most of the 24 hours following the re-computation, which results in 30%, or 2.5 ms, improvement compared to re-computing at 10 am UTC, when the stability is the worst. We believe that this is because Internet latencies are most of the time affected by network congestion commonly observed during busy network hours.

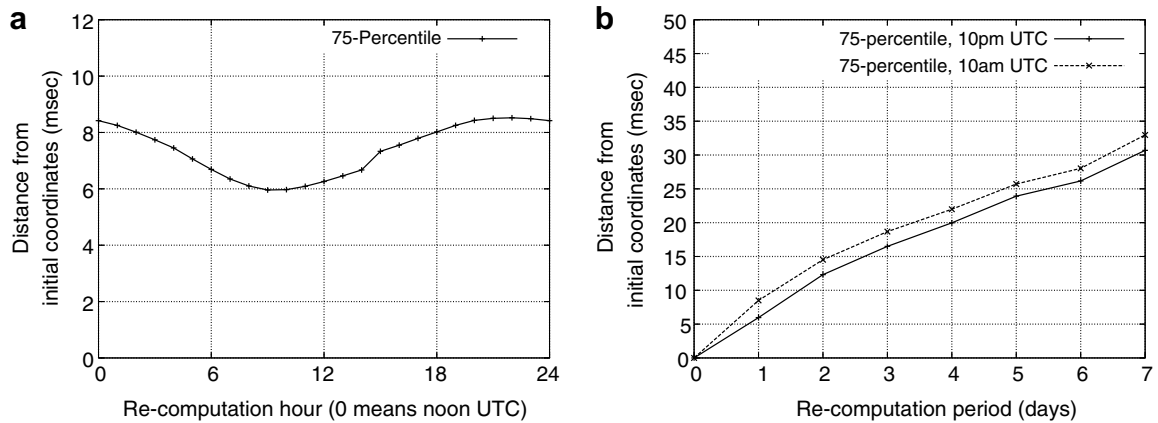


Fig. 13. Impact of re-positioning hour on daily coordinate stability (a), and on coordinate stability in general (b).

For the sake of completeness, we also checked how different re-positioning hours influence the stability of coordinates re-computed every 2 days or more. To this end, we again simulated coordinate re-computations every N days (for N from 1 to 7) based on our test trace. We performed two simulations, each configured to re-compute positions at a different hour: 10 pm UTC and 10 am UTC. The results are presented in Fig. 13b. As can be observed, the improvement of 2.5 ms remains roughly constant irrespective of how often coordinates are re-computed, which reduces its impact to approximately 10% when re-computing coordinates every 4 days or more.

5. Coordinate-based client re-direction

Deploying the positioning system enables Google to implement various latency-driven applications that shall improve access latency for its clients. One of such applications is client re-direction: based on the coordinates produced by GNP, we can re-direct each client to a replica that is closest to that client in terms of latency. To this end, we calculate the distance between the coordinates of the client and the coordinates of each replica, and select the replica with the shortest distance to the client.

5.1. Absolute performance

We verified the efficiency of coordinate-based re-direction. To this end, we positioned the 10,000 /24 networks based on the median latencies measured between these networks and 20 candidate landmarks deployed on PlanetLab nodes over a period

of six weeks. Each network was positioned relative to the best set of seven landmarks identified in Section 3.1.1. We chose 10 of the 13 remaining candidate landmarks to form a globally distributed set of replicas. Next, for each replica, we calculated its median measured latency to each network. Finally, for each network, we determined its closest replica based on the median measured latencies, and matched that choice against that made based on latencies estimated with coordinates. The results are depicted in Fig. 14 (line 'Popular').

As can be observed, clients from 86% of /24 networks are re-directed to the replica closest to them in terms of median measured latency. Also, clients from another 10% of networks are re-directed to replicas offering latencies at most two times longer than the closest ones. Finally, only about 2% of networks are re-directed to replicas further than three times than the closest ones. Note that the results

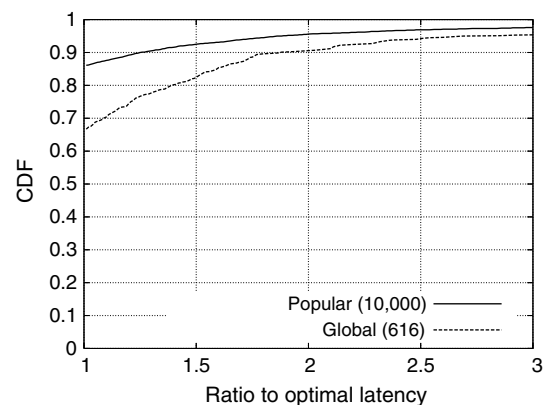


Fig. 14. Efficiency of GNP-based replica selection.

presented in [24] indicate that employing alternative embedding schemes might further improve the performance of coordinate-based client re-direction.

We have also performed the above experiment for the set of 616 globally-distributed clients that we constructed in Section 3.1.1. The results are also depicted in Fig. 14 (line ‘Global’). It shows that coordinate-based re-direction selects the closest replica for clients in about 67% of globally-distributed /24 networks, and replicas offering latencies at most two times higher than optimal in for clients in another 24% of such networks. We believe that the suboptimal replica selection in the remaining cases is caused by node mispositioning. Nodes are typically mispositioned when they have long latencies to all the landmarks, or when the latencies of their network paths to the landmarks are self-inconsistent from the perspective of GNP, for example, because of multi-homing [35]. Another reason for suboptimal client re-direction decisions might be short latencies between the client and more than one replica. As discussed in Section 6.2, GNP positioning has a lower accuracy for low-latency estimations. However, for such clients located close to the replicas, one may consider that any re-direction decision is acceptable.

5.2. Relative performance

Although GNP-based re-direction seems to perform well in terms of absolute latency values, it has recently been suggested that absolute metrics are not enough to completely evaluate re-direction efficiency [36]. This is because re-directed clients sometimes care more about *relative* dependencies

between latencies to different replicas, rather than about their absolute values. Such situations might occur when relative ordering of nodes influences the application behavior in terms of, for example, data propagation strategy (e.g., gossiping order or multi-cast tree structure).

The relative performance of GNP-based re-direction can be measured by means of another metric, called *Relative Rank Loss* (*rrl*). For each client, it creates two replica rankings: one calculated based on measured client-to-replica latencies, and another based on latency estimates provided by GNP. Given the two rankings, the *rrl* of each client C can be computed according to the following formula:

$$rrl(C, R) = \frac{|\{(x, y) | x \neq y \text{ and swapped}(x, y)\}|}{|R|(|R| - 1)},$$

where R is the set of replicas, (x, y) are elements of $R \times R$, and $\text{swapped}(x, y)$ is true when the relative ordering of x and y is different in the two rankings created for client C . *rrl* can also be interpreted as the probability that $\text{swapped}(x, y)$ is true for any two different replicas.

We have computed *rrl* values based the client latencies from the test sets used to evaluate the absolute performance of client re-direction. The results are presented in Fig. 15a.

As can be observed, *rrl* is lower than .2 for about 92% of frequent clients, and for about 65% of globally-distributed clients. For these clients, any pair of replicas has only 20% chance to be re-ordered when client-to-replica latencies are estimated with GNP. Furthermore, according to our data, misordering happens mostly when two replicas have very similar latencies to the client. This can be observed in

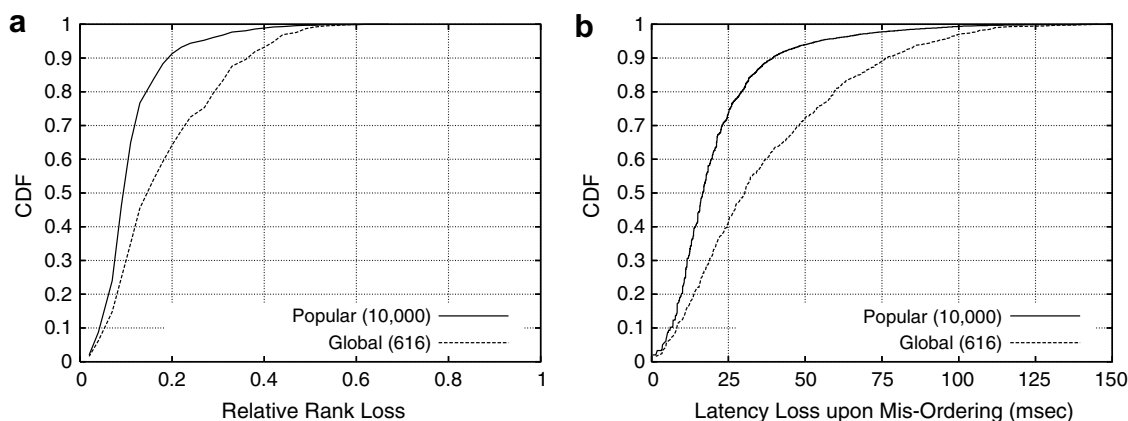


Fig. 15. Relative performance of client re-direction in terms of *rrl* (a), and latency (b).

Fig. 15b, which depicts the distribution of differences in client-to-replica latencies when a misordering occurs. It shows that loss in client-to-replica latency resulting from misordering is less than 50 ms for 95% of frequent- and 73% of globally-distributed clients.

5.3. DNS considerations

A potential problem when using client coordinates for request re-direction is that large-scale Internet services typically re-direct their clients using DNS. In that case, the re-directing decisions are made based on the addresses of client-side DNS servers rather than on those of end clients themselves [37]. Meanwhile, our positioning system can compute coordinates only for /24 networks that contain at least one Web client, which is relatively uncommon for networks used by client-side DNS servers. This means that the service might be unable to determine the coordinates of a DNS server, which in turn makes it impossible to select the best replica for the service clients which access that server.

We solve this problem by associating networks containing Google clients with networks containing DNS servers these clients use. To this end, one could rely on network-aware clustering, which identifies co-located /24 networks as those falling within the same BGP prefix [38]. However, this solution implicitly assumes that clients typically use DNS servers that belong to the same BGP prefix, which has been shown to be false in most cases [39]. We therefore exploit a proprietary mechanism that precisely discovers which DNS server is used by each Google client. The details of this mechanism are out of scope of this article.

6. Generic latency estimation service

Given that our system collects latency information about millions of Internet hosts, it can potentially be used to predict latencies between arbitrary machines in the Internet, which are not necessarily Google clients. Such a generic latency estimation service could be useful for any application that needs to estimate end-to-end latencies between Internet hosts, such as a peer-to-peer overlay or a third-party content delivery network.

In this section, we investigate to what extent our system succeeds in predicting such latencies. To this end, we evaluate the accuracy of latency estimates predicted for hosts that have never been involved

in any operation performed by our system. Such a non-involvement means that these hosts have never been instrumented by our system, and that we have never measured their base latencies in any way. Instead, we determine the coordinates of these hosts by simply taking the coordinates of their co-located Google clients.

A potential problem at this stage is that our system can estimate latencies only between /24 networks containing Google clients. However, while using /24 clustering allows us to position a huge number of Internet hosts, there are also many hosts that cannot be positioned when such an approach is followed. This is true for network servers, for example, which are typically deployed in different networks than user machines. We circumvent this problem by clustering Google clients into BGP prefixes, and not into /24 networks. Such coarse-grain clustering enables us to position more hosts at the expense of potential loss in estimation accuracy, as latencies to machines located in the same BGP prefixes are likely to be more diverse than those to machines located in the same /24 networks.

Fair accuracy evaluation requires that latency estimates produced by our system are compared against their corresponding measured latencies. We use two datasets of measured latencies derived from third-party latency traces, called PlanetLab and RIPE. Both these datasets contain matrices of all-pair latencies measured between a number of machines during subsequent hours in November 2006. Each matrix is specific to a different hour and contains minimum latencies observed for given pairs of machines throughout that hour. We chose to use minimum latency values because they correspond to the “empty path” latencies that our system is striving to estimate.

The estimation accuracy is evaluated by measuring the relative difference between latencies found in each dataset against their estimated counterparts. To ensure the fairness of comparison, all the estimates are computed based on the data collected before their corresponding measurements were performed.

6.1. PlanetLab latencies

The PlanetLab dataset contains latencies measured between 489 PlanetLab nodes. It was derived from the latency trace collected by Jeremy Stribling for his “all-pair pings” project [40]. To this end, we aggregated the original latencies (measured every 15 min) into hourly all-pair matrices.

We compare the dataset latencies against their estimates provided for 327 (out of 489) PlanetLab nodes whose coordinates we were able to derive from base latencies measured to Google clients. The total number of latencies analyzed is 39.6 million (more than 50,000 per hourly matrix). For each such latency, we calculate the relative latency estimation error according to the formula introduced in Section 3.1.1. The resulting distribution of error values is depicted in Fig. 16a.

As can be observed, relative estimation error is lower than .5 in approximately 83% of the cases, which comes close to that reported in the original GNP paper for hosts instrumented with GNP software. More importantly, the high estimation accuracy is preserved over time. This can be observed in Fig. 16b, which shows how the fraction of good estimates (with error lower than .5) changes over time.

Although the overall system performance for the PlanetLab dataset is very good, a further investigation of error values reveals that the estimation accuracy varies depending on latency magnitude. This can be observed in Fig. 16c, which depicts the distributions of estimation errors for four different intervals of measured latencies. The differences between these distributions indicate that precise estimation of very short latencies (25 ms or less) is very hard, as opposed to predicting long latencies (100 ms or more). These results make us believe that the high estimation accuracy achieved for PlanetLab latencies is partially caused by their favorable distribution, as more than 65% of them are longer than 100 ms. Another reason for such good results might be that academic networks are generally less dynamic than their commercial counterparts, which makes latency estimation in academic networks easier.

6.2. RIPE latencies

The favorable properties of PlanetLab latencies are not present in our second dataset. It contains latencies measured by the infrastructure of 70 diagnostic stations deployed for the RIPE Test Traffic Measurements project (TTM) [41]. The diagnostic stations, called test-boxes, are deployed on the backbones of various Internet Service Providers, and used for evaluating and streamlining the communication between these backbones. Given that most of TTM ISPs are located in Europe, most of the latencies between test-boxes are very short, which makes them very hard to estimate accurately.

We evaluate the performance of our system based on the RIPE dataset just as we did with PlanetLab. First, we use BGP clustering to position the test-boxes, which ultimately led to determining GNP coordinates for 47 of them. Given these coordinates, we calculate relative estimation errors for latencies measured between these stations, which leads to analyzing more than 1100 measurements per hourly matrix. The resulting distribution is error values depicted in Fig. 17a and c.

As can be observed, short latencies are indeed very hard to estimate accurately. This severely affects the overall performance, as 61.8% of RIPE latencies are shorter than 50 ms. However, the accuracy of long latency estimates is far better: 70% of them are off by less than .5. Also, similar as in the case of PlanetLab, the estimation accuracy for RIPE latencies is preserved over time (see Fig. 17b).

Based on the analysis performed with our two datasets, we conclude that our system could be used as a generic latency estimation service. It performs very good when estimating long latencies, which makes it particularly suitable for predicting latencies between globally distributed hosts. As for short

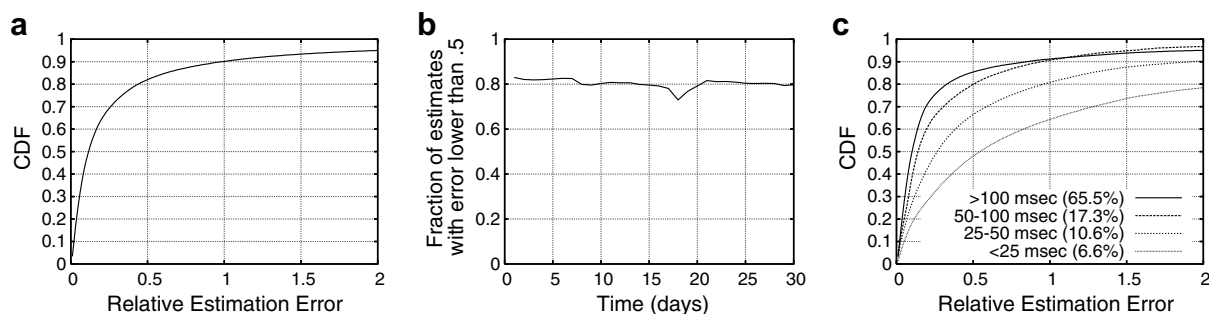


Fig. 16. PlanetLab latency estimation: relative error (a), accuracy over time (b), and accuracy for different latency intervals (c).

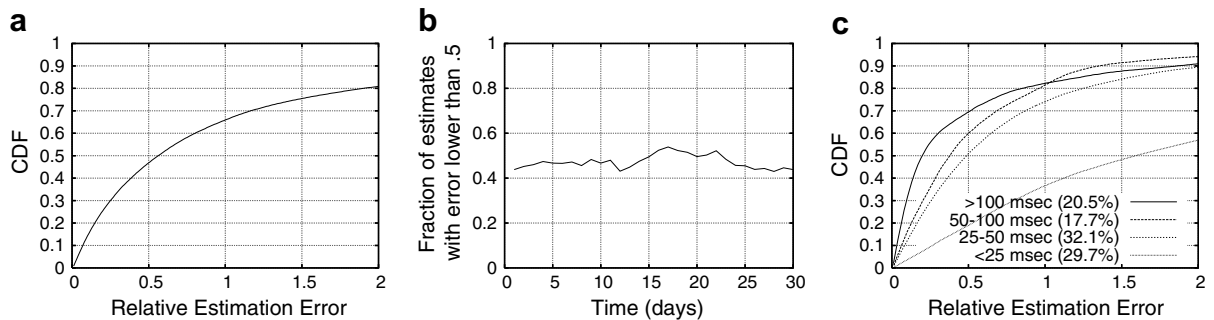


Fig. 17. RIPE latency estimation: relative error (a), accuracy over time (b), and accuracy for different latency intervals (c).

latencies, such as those found in the RIPE dataset, they are very hard to estimate precisely, which is an inherent problem with positioning-based latency estimation. However, our system can still estimate at least some of them with a reasonable degree of accuracy.

7. Conclusions and future work

We have presented an implementation of GNP incorporated into the Google content delivery network. In contrast to all its previous counterparts, our implementation does not rely on active participation of Web clients, as all the latency measurements are performed passively by the landmarks. The overhead incurred by latency measurements is carefully controlled by a scalable centralized scheduler, which prevents both the landmarks and the network from becoming overloaded. Deploying our solution requires only a small number of CDN modifications, which makes it attractive for any CDN interested in large-scale latency estimation.

Our system has been collecting latency information about millions of Google clients for several months. The analysis of these data enabled us to confirm many of the results presented in earlier research on GNP, and add to these results by investigating the issue of coordinate stability over time. We have shown that coordinates drift away from their initial values with time, making 25% of the coordinates to become inaccurate by more than 33 ms after one week. However, daily re-computations make 75% of the coordinates stay within 6 ms of their initial values.

Apart from analyzing the behavior of GNP coordinates over time, we have also discussed our experience with their practical applicability. We have demonstrated that using coordinates to decide on

client-to-replica re-direction leads to selecting replicas closest in term of *measured* latency in 86% of all cases. In another 10% of all cases, clients are re-directed to replicas offering latencies at most two times longer than optimal.

Collecting a huge volume of latency data has enabled us to estimate latencies between globally distributed Internet hosts that have not participated in our measurements. We have been able to determine the coordinates of such hosts by applying network-aware clustering. The results are sufficiently promising that Google may offer a public interface to the latency estimates in the future. Such an interface could be useful for any large-scale distributed applications, including peer-to-peer overlays and other content delivery networks. We plan on developing our system further by improving its scalability using multiple schedulers, and by reducing the delay between measuring base latencies and converting them into fresh coordinates.

Acknowledgements

We would like to express our gratitude to all the people who have made conducting this research possible. In particular, Sean Knapp and James Morrison, both from Google, helped us modify the code of Google Web Search servers. Marius A. Eriksen, another Google engineer, pointed us to the prefetching functionality of Firefox. Larry L. Peterson from the PlanetLab Consortium enabled us to deploy our landmarks on port 81 of PlanetLab nodes. Henk Uijterwaal from the RIPE Network Coordination Centre provided us with the latency data from the Test-Traffic Measurement project. Finally, we would like to thank all the anonymous reviewers for their useful comments that have helped us improve this article.

References

- [1] M. Zari, H. Saiedian, M. Naeem, Understanding and reducing Web delays, *IEEE Computer* 34 (12) (2001) 30–37.
- [2] F. Dabek, J. Li, E. Sit, J. Robertson, F. Kaashoek, R. Morris, Designing a DHT for low latency and high throughput, in: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, USENIX, March 2004.
- [3] J. Pereira, L. Rodrigues, A. Pinto, R. Oliveira, Low latency probabilistic broadcast in wide area networks, in: *Proceedings of the 23rd International Symposium on Reliable Distributed Systems*, IEEE Computer Society, New York, October 2004.
- [4] M. Szymaniak, G. Pierre, M. van Steen, Latency-driven replica placement, in: *Proceedings of the 2005 International Symposium on Applications and the Internet*, IEEE Computer Society, New York, February 2005.
- [5] L. Amini, H. Schulzrinne, Client clustering for traffic and location estimation, in: *Proceedings of the 24th International Conference on Distributed Computing Systems*, IEEE Computer Society, New York, March 2004.
- [6] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, *Future Generation Computer Systems* 15 (5–6) (1999) 757–768.
- [7] S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine, *Computer Networks and ISDN Systems* 30 (1–7) (1998) 107–117.
- [8] T.S. Eugene Ng, H. Zhang, Predicting Internet network distance with coordinates-based approaches, in: *Proceedings of the 21st INFOCOM Conference*, IEEE Computer Society, New York, June 2002.
- [9] Y. Shavitt, T. Tankel, Big-bang simulation for embedding network distances in Euclidean space, in: *Proceedings of the 22nd INFOCOM Conference*, IEEE Computer Society, New York, April 2003.
- [10] S. Srinivasan, E. Zegura, An empirical evaluation of landmark placement on Internet coordinate schemes, in: *Proceedings of the International Conference on Computer Communications and Networks*, IEEE Computer Society, New York, October 2004.
- [11] L. Tang, M. Crovella, Virtual landmarks for the Internet, in: *Proceedings of the Internet Measurement Conference*, ACM, New York, October 2003.
- [12] R. Cox, F. Dabek, F. Kaashoek, J. Li, R. Morris, Practical, distributed network coordinates, in: *Proceedings of the 2nd Workshop on Hot Topics in Networks*, ACM, New York, November 2003.
- [13] H. Lim, J. Hou, C. Choi, Constructing Internet coordinate system based on delay measurements, in: *Proceedings of the Internet Measurement Conference*, ACM, New York, October 2003.
- [14] P. Pietzuch, J. Ledlie, M. Seltzer, Supporting network coordinates on PlanetLab, in: *Proceedings of the 2nd Workshop on Real, Large Distributed Systems*, USENIX, December 2005.
- [15] M. Castro, M. Costa, P. Key, A. Rowstron, PIC: Practical Internet coordinates for distance estimation, Technical Report MSR-TR-2003-53, Microsoft Research, September 2003.
- [16] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, S. Bhatti, Lighthouses for scalable distributed location, in: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, February 2003.
- [17] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a decentralized network coordinate system, in: *Proceedings of the SIGCOMM Conference*, ACM, New York, August 2004.
- [18] T.S. Eugene Ng, H. Zhang, A network positioning system for the Internet, in: *Proceedings of the 4th Symposium on Internet Technologies and Systems*, USENIX, June 2004.
- [19] L. Barroso, J. Dean, U. Holzle, Architecture, Web search for a planet: the Google cluster, *IEEE Micro* 23 (2) (2003) 22–28.
- [20] J. Nelder, R. Mead, A simplex method for function minimization, *The Computer Journal* 4 (7) (1965) 308.
- [21] M. Szymaniak, G. Pierre, M. van Steen, Scalable cooperative latency estimation, in: *Proceedings of the 10th International Conference on Parallel and Distributed Systems*, IEEE Computer Society, New York, July 2004.
- [22] M. Waldvogel, R. Rinaldi, Efficient topology-aware overlay network, in: *Proceedings of the 1st Workshop on Hot Topics in Networks*, ACM, New York, October 2002.
- [23] The PlanetLab Project Web page, <<http://www.planet-lab.org/>> (accessed: 21.05.07).
- [24] Y. Shavitt, T. Tankel, The curvature of the Internet and its usage for overlay construction and distance estimation, in: *Proceedings of the 23rd INFOCOM Conference*, IEEE Computer Society, New York, April 2004.
- [25] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, F. Zane, Clustering and server selection using passive monitoring, in: *Proceedings of the 21st INFOCOM Conference*, IEEE Computer Society, New York, June 2002.
- [26] The Site Meter Web page, <<http://www.sitemeter.com/>> (accessed: 21.05.07).
- [27] R. Kokku, P. Yalagandula, A. Venkataramani, M. Dahlin, NPS: A non-interfering deployable Web prefetching system, in: *Proceedings of the 4th Symposium on Internet Technologies and Systems*, USENIX, March 2003.
- [28] C. Wootton, JavaScript Weirdness, *Web Developer's Journal*, 1999, <<http://www.webdevelopersjournal.com/>>.
- [29] D. Fisher, G. Saksena, SYNOPSIS: Link prefetching in Mozilla: a server-driven approach, in: *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, IBM Research, September 2003.
- [30] D. Duchamp, Prefetching hyperlinks, in: *Proceedings of the 2nd Symposium on Internet Technologies and Systems*, USENIX, October 1999.
- [31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext transfer protocol – HTTP/1.1. RFC 2616, IETF, June 1999.
- [32] The Mozilla Firefox Web page, <<http://www.mozilla.com/firefox/>> (accessed: 21.05.07).
- [33] N. Brinkman, Mozilla's browsers global usage share is still growing, Onestat.com, July 2006.
- [34] V. Kudallur, IE7 networking improvements in content caching and decompression, MSDN Blog, October 2005.
- [35] H. Zheng, E. Keong Lua, M. Pias, T. Griffin, Internet routing policies and round-trip times, in: *Proceedings of the*

Passive and Active Measurement Workshop, Springer, New York, March 2005.

- [36] E. Keong Lua, T. Griffin, M. Pias, H. Zheng, J. Crowcroft, On the accuracy of embeddings for Internet coordinate systems, in: Proceedings of the Internet Measurement Conference, ACM, New York, October 2005.
- [37] A. Shaikh, R. Tewari, M. Agrawal, On the effectiveness of DNS-based server selection, in: Proceedings of the 20th INFOCOM Conference, IEEE Computer Society, New York, April 2001.
- [38] B. Krishnamurthy, J. Wang, On network-aware clustering of Web clients, in: Proceedings of the SIGCOMM Conference, ACM, New York, August 2000.
- [39] Z. Morley Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, J.A. Wang, Precise and efficient evaluation of the proximity between Web clients and their local DNS servers, in: Proceedings of the Annual Technical Conference, USENIX, June 2002.
- [40] J. Stribling, All-pairs Ping Data for PlanetLab, <http://pdos.csail.mit.edu/~strib/pl_app/> January 2006.
- [41] F. Georgatos, F. Gruber, D. Karrenberg, M. Santcroos, A. Susanj, H. Uijterwaal, R. Wilhelm, Providing active measurements as a regular service for ISP's, in: Proceedings of the Passive and Active Measurements Workshop, RIPE, April 2001.



Michal Szymaniak is a Ph.D., candidate in the Computer Systems group at the Vrije Universiteit Amsterdam. His research focuses on large-scale distributed systems for content delivery in the Internet. He is a student member of the IEEE and the ACM. He holds a double MSc in Computer Science from Warsaw University (Poland) and from the Vrije Universiteit Amsterdam.



Dr David Presotto holds the title of Fashion Icon at Google. He is head wrangler for the Google flamingo herd and in his spare time is responsible for Google's load balancing, DNS, and CDN. He obtained a high school diploma from The Boston Latin School. David has the twin distinction of being deported from both Switzerland and Yugoslavia (pre break up).



Guillaume Pierre is an assistant professor in the Computer Systems group at the Vrije Universiteit Amsterdam. He has been working in the field of Web-based systems for many years. He is a member of the IEEE and an editorial board member of IEEE DSONline. He holds an MSc and a Ph.D., in Computer Science from the University of d'Evry-val d'Essonne (France).



Maarten van Steen is a full professor in the Computer Systems group at the Vrije Universiteit Amsterdam. He is head of a research team developing large-scale distributed systems. Besides Web-based systems, his research interests include peer-to-peer and gossip-based distributed systems. He is senior member of the IEEE and member of the ACM. He holds an MSc in Applied Mathematics from Twente University and a Ph.D., in Computer Science from Leiden University.