An Internet Distance Prediction Service based on Passive Measurements

Matthias Scheidegger, Torsten Braun, Benjamin Zahler
Institute of Computer Science and Applied Mathematics
University of Bern
Neubrückstrasse 10, 3012 Bern, Switzerland
{mscheid, braun, zahler}@iam.unibe.ch

Abstract—**Distance estimation services support the adaptivity of distributed applications on the Internet by predicting the network characteristics between end systems. A common problem of such services is their need for large-scale deployment in order to operate properly. We propose a peer-to-peer-based distance measurement and prediction service that is able to predict distances from a single site to arbitrary end systems on the Internet while only requiring deployment at this specific site. In this paper we present the service architecture, and we demonstrate the viability of the approach with results from a prototype deployment of the service.**

Keywords: overlay networks; network measurements; Internet distance prediction

I. INTRODUCTION

Dynamic adaptation to network characteristics is an increasingly important topic for distributed Internet applications because it may increase the applications' performance and robustness. In server selection systems for example, clients get assigned to servers according to criteria such as their transmission latency to the server. More advanced adaptation schemes can be found in peer-to-peer networks.

Such adaptivity necessitates an efficient way to estimate the network characteristics between end systems on the Internet. Numerous so-called distance estimation schemes have been proposed to simplify this task and make it more efficient. Unfortunately, most of the existing approaches need global deployment in order to provide useful distance estimates, or even require additional infrastructure in the network. Moreover, they usually estimate the round-trip time between end systems only.

We have created a service that is able to predict distances from a single site to arbitrary end systems on the Internet while only requiring deployment at this specific site. Our proposed service provides a common distance measurement API to network applications, and it optimizes the measurement activities of the applications by coordinating them.

When an application requests a distance measurement to a known target, the service may return a prediction rather than making an actual measurement. The service architecture is made up of groups of nodes from the same network neighborhood. Inside these groups, the nodes form a small peer-to-peer network to exchange and store distance measurements and predictions. Measurements are mostly obtained by observing the regular application traffic of the group members. If needed the service may also actively measure distances. Nevertheless, we expect that passive measurements should be sufficient in most cases since the measurement requests from clients tend to concern the same hosts that the clients communicate with. In other words, a large amount of data will automatically be available for the most popular measurement targets. The focus on passive measurements also enables estimation for any type of distance measure. Moreover, the amount of measurements available for popular destinations even allows for distance predictions going several minutes into the future.

The remainder of this paper is structured as follows: Section II gives an overview of related work and previously published algorithms used in our architecture. In Section III we describe the architecture of the distance prediction service. We present results from test with a prototype implementation in Section IV. Section V concludes the paper.

II. RELATED WORK

Numerous different approaches to Internet distance estimation have been proposed during the last years. IDMaps [2] employs a network of measurement servers, so called tracers, placed at strategic locations in the network. These tracers continuously measure the round-trip times between themselves. IDMaps then estimates the round-trip time between two end systems using the sum of the round-trip times between the end sytems and their respective nearest tracers, plus the round-trip time between these tracers. Dynamic Distance Maps [7] is also based on measurement servers, but uses them to hierarchically decompose the Internet into regions.

Recently, coordinates-based distance estimation has received much attention. Such approaches embed network distances into a coordinate space so that the distance between the coordinates of two nodes is a good estimate of the distance between these nodes in the network. In GNP [3] the embedding is done in two steps. First, the coordinates of a set of landmark nodes are computed using the round-trip times between them. Then, any additional nodes only compute their coordinates with respect to these landmarks. GNP's fixed set of landmarks may pose scalability and robustness problems. More robust approaches like Vivaldi [1] address this problem. Vivaldi does

not use any landmarks. Instead, each node monitors its network traffic to other Vivaldi nodes in order to obtain round-trip time measurements and applies a distributed algorithm to iteratively adjust its coordinates.

The main weakness of these approaches is that their distance estimations are limited to the part of the network in which the service has been deployed. IDMaps and Dynamic Distance Maps can only reliably estimate distances between those parts of the Internet in which measurement servers have been installed. Coordinates-based approaches can only estimate distances between end systems actively participating in the service, i.e., running a special software computing and updating the node's coordinates. IDMaps, Dynamic Distance Maps, and GNP also require additional infrastructure in the network. Furthermore, these approaches only support round-trip time as distance measure.

Our distance prediction service is based on groups of nodes located close to each other in the network. The groups must be formed such that distance measurements made by a group member to an end system outside the group are also valid for all other members of the group. In [5] we have presented a distributed algorithm for identifying such groups. It combines two existing approaches, mOverlay [9] and Meridian [8]. Each group has a leader that participates in a Meridian peer-to-peer network. Joining nodes then use this network to find the group closest to them and employ a grouping criterion related to the one used in mOverlay to check whether this group is suitable to join. We have shown that this algorithm creates groups with better properties than a related algorithm presented in [9]. Furthermore, the average time it takes for a node to join the system is smaller with our algorithm.

We use a peer-to-peer repository to store measurement data and prediction models. In order to reduce the number of entries in the repository we use the clustering algorithm presented in [6]. If we find that the measurements to two remote end systems are very similar over a given amount of time, we may combine them into a cluster in the repository and so save storage space. We use two criteria based on outlier and bias detection to identify such cases.

III. ARCHITECTURE

A. Overview

The architecture of our end-to-end distance prediction service is based on a peer-to-peer approach, where nodes that are close together in the underlying topology join the same groups to exchange and coordinate measurements. We use a two-layer architecture. On the loosely connected global layer, inter-group communication takes place. On the local layer, the members of each group are integrated into separate structured peer-to-peer networks that serve for efficient distributed storage of measurement data and prediction models.

Groups are largely independent of each other and can function without interaction to other groups, with the exception of the group locating procedure by which new nodes find a suitable group to join. Each member of a group provides an API to client processes on its end system. Client processes can request measurements and predictions through this API. In order to take advantage of the implicit measurements done by normal traffic, applications can also send such measurements back to the service entities. For example, when a web server has sent a large file to a client it can submit the achieved TCP throughput value to the service entity. The same API can also be used by network traffic monitors to submit additional measurements. The group maintains a repository of measurements and prediction models distributed among the peers.

Naturally, the repository would hardly be able to store information about every possible host on the Internet. The approach we take with our distance prediction service is related to the concept of proxy servers. Members of the same group are normally located at the same site, e.g. the local network of a single organization such as an enterprise. Since the users of an organization share common interests, their requests concentrate on a relatively small set of popular IP addresses. Therefore, if the service only stores information about the most popular clusters it can still give predictions for the majority of requests. Moreover, since our service relies heavily on measurements gained from monitoring application traffic, popular clusters tend to have more measurements available than unpopular ones, which in turn leads to better predictions.

Based on these observations we manage the size of the repository using a least-recently-used strategy. Each group member defines an upper limit for the amount of data it stores in its local storage (e.g., its hard disk). When this limit has been exceeded, it removes the least popular clusters from its part of the repository until enough storage space has been released. The popularity of a cluster is measured by the time elapsed since the last request for it has been received.

In order to further reduce the number of remote endpoints that the repository has to keep track of, we use clustering. Two or more remote nodes that are not members of the group are stored as a single unit (i.e., a cluster) in the repository, if the measurements to both nodes remain very similar over a certain time. We have described this clustering method in detail in [6]. Note the difference between the terms 'group' and 'cluster.' Groups are small peer-to-peer networks sharing a repository of distance measurement data. Clusters are an abstraction used in this repository to reduce the number of entries. Nevertheless, each group is a cluster (or part of a cluster) from the perspective of other groups. Consequently, we use three categories of clustering in the repository.

1. Remote nodes that are not members of the distance prediction service (so-called non-member nodes) are handled independently.
2. Remote nodes that belong to the same remote group are treated as a single unit.
3. Both remote groups and non-member nodes are further clustered using the algorithm from [6].

B. The Global Layer

The global layer's main purpose is bootstrapping of new nodes. In our architecture we use the method described in [5], which combines the existing approach from mOverlay [9] with the nearest node search from Meridian [8]. In this

bootstrapping method one node per group (the group leader) becomes a Meridian node, allowing new nodes to find the group nearest to themselves. In order to avoid a single point of failure, each group maintains one or several backup leader nodes.

The second purpose of the global layer is identification of remote groups. When a node receives a request referring to an IP address that is not known in its local repository, it will try to send a message to that address on a well-known port, asking for the ID of the remote node's group. If the destination node is a member of the system it will answer, allowing us to look for existing information about that group ID in the local repository. If the remote node is a non-member, the message will time out after a short delay and the request fails.

C. The Local Layer

On the local layer nodes are organized into groups, each of which is made up of a Pastry [4] ring. Using its Pastry ring, each group maintains a distributed repository of identified clusters, measurement data, and prediction models. The members of a group are controlled by its group leader. When a new node joins the group, the leaders insert it into the ring. This is done by computing a Pastry ID that uniquely identifies the new node and determines the node's position in the ring. IDs are computed using an MD5 hash on a string containing the node's IP address. A given IP address always results in the same ID. There are four types of IDs: Node IDs designate a member node of the local group, Cluster IDs designate identified clusters of non-member hosts and remote groups, Group IDs designate known remote groups, and Host IDs designate known remote hosts (including nodes belonging to a remote group). Node, cluster, and host ID need only be unique within the repository. Group IDs however are globally unique, because they are used for identifying different groups on the global layer.

Information in the repository is always stored by cluster ID. The Pastry node with the numerically closest node ID is responsible for storing the data. If the group learns of a new remote host or group that cannot be assigned to a cluster, a new cluster is created that contains only this host or group. Later, when enough measurement data is available to run the clustering algorithm the new cluster may be merged with another.

The Pastry ring maps each host and group ID to its assigned cluster ID. Accordingly, a lookup for a given ID first resolves the assigned cluster ID and then locates the Pastry node responsible for storing the relevant data. When a node wants to retrieve the latest measurement data for a given IP address, it computes the host ID of that address and performs a lookup on the Pastry ring. If the ID is known it receives the ID of the host's cluster. The requesting node then sends a message to the node responsible for that cluster ID to retrieve the desired data. Finally, once the message reaches the responsible node, it returns the data (or an error message if the data is not available) directly to the requester. Figure 1 illustrates this procedure.

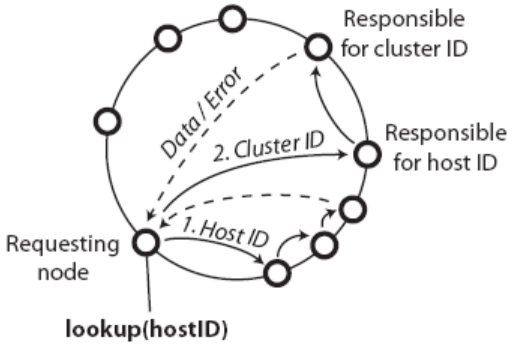

Figure 1. A node retrieves information about a known IP address

If the local repository has no information about the remote host, the initial lookup fails. In this case, the requesting node will send a message to the remote host on a well-known port. If the remote host is a member of the service, the requesting node receives the ID of the remote host's group and performs a second lookup for this group ID. If this also fails, it returns an error message to the client.

Client applications will quite frequently send queries about unknown host IDs to the local group. Consequently, we have to add new entries to the repository. If the repository does not contain the host ID and we fail to obtain a group ID from the remote host, we create a new cluster containing only the new host and update the Pastry ring accordingly. Such small clusters are inefficient, so we try to merge it with another cluster as soon as possible. Accordingly, we schedule a series of round-trip time measurements to the remote host in order to obtain the data necessary to run the clustering algorithm.

If the remote host returns the ID of its group, but this group ID is not known either, we have to create an additional entry for this group in the repository. Both, the host ID and the group ID will point to the newly created cluster. However, if the repository contains information about the remote host's group, we can simply add a Pastry entry that links the new host ID to the same cluster as the group ID resolves to. No additional measurements are necessary in this case.

D. Repository Mechanisms

Every group member stores and manages all clusters with IDs that are in its ID range on the Pastry ring. This includes storing measurements made to hosts in these clusters, computing distance prediction models, and realigning clusters where necessary. To save space, old data may be discarded. It may also entirely remove inactive clusters from the system. The repository also stores measurements for the individual nodes and groups, because they may be used later, if clusters have to be merged or split. In this case, these measurements serve as a basis for recalculating the clustering. The node also creates prediction models for its clusters and updates them regularly when it receives new measurements in order to improve them and fit them to recent data. We use auto-regressive (AR) models for predicting all available kinds of measurements. Each group member also creates and distributes backups of cluster data based on Pastry's key replication strategy.

The clustering of hosts and groups may be flawed due to inaccurate measurements or because of routing changes in the network. Accordingly, the repository may realign clusters if it detects the need to do so. This operation, called cluster splitting, takes place on the node responsible for the cluster. We construct time series for each of the cluster's hosts and groups and re-apply the clustering algorithm from [6]. If the algorithm only identifies a single cluster, the node continues as before. If it identifies two or more clusters, the node retains the biggest one and migrates the remaining ones to other members of the group for storage.

Cluster merging, the reverse operation of cluster splitting, becomes necessary when there are too many small clusters in the repository. Unlike cluster splitting, merging often concerns clusters stored on separate group members. Accordingly, the peers have to detect possibilities to merge clusters and coordinate the merge operations. From time to time each group member creates short time series describing the most recent round-trip time data for all of its clusters and sends them to all other group members using a Pastry broadcast message. In addition to the time series these messages also contain the sizes of the respective clusters. The receivers try to match these time series with the clusters they store, using the neighborhood criteria from [6]. If a receiver detects a possible match it will contact the sender to initiate cluster merging. The merging algorithm is always executed at the member storing the larger cluster (in terms of the number of nodes). The actual merging is done by running the clustering algorithm on the combined set of members of both clusters.

The bandwidth used for this broadcasting of time series is controlled by two mechanisms. On the one hand, the longer a cluster remains in the repository, the longer the time between its broadcasts. Thus, new clusters quickly find existing clusters to merge with and old ones do not use bandwidth unnecessarily. On the other hand, the broadcast rate for each cluster is reduced when the number of group members grows. This keeps the bandwidth per node constant. Since a group's size is restricted by the network topology, the resulting rate of cluster information broadcasts never reaches zero.

IV. Performance Results

A. Test Setup

We have deployed a prototype implementation of the distance prediction service on 11 PlanetLab nodes at the University of California, Berkeley. All 11 nodes formed a single peer-to-peer group that received prediction requests generated by a controlling node. Each node had a Pentium 4 CPU running with 3 GHz.

Since our distance prediction service relies heavily on user requests and observing user traffic, a fundamental problem for testing a system like this is to generate realistic user requests and traffic. We believe that the request pattern observed by a real deployment of our service would resemble the request pattern observed by a web proxy, in that a large part of the requests pertain to a very small set of popular destination addresses. Nevertheless, the number of requests to a proxy should be significantly higher since loading a web page normally involves several requests sent shortly after each other, while a client would request a distance prediction only once for a given network connection.

Accordingly, we have based our tests on a 24-hour log from the web proxy server of the University of Bern. The log records the time and destination of each web request on a single line in a text file. During each hour of the test, the controlling node generated distance prediction requests according to the distribution of the corresponding hour of the web proxy's log. Unfortunately, the log entries do not contain any usable information about the observed round-trip time or throughput. We have thus replaced the missing passive measurements by active ping measurements to the requested destination IP addresses. We have scheduled such measurements once every 30 seconds for destinations that were lacking recent measurement data. In the tests, each group member received a prediction request once every 20 seconds. Each successful distance prediction also triggered an additional round-trip time measurement to determine the error of prediction. Cluster information was broadcasted by each group member every 60 seconds to detect possibilities to merge clusters (the prototype did not include the rate adaption methods described in Section III.D). Group members sent backup data to their neighbors every 5 minutes. We have run two tests. In the first test we have focused on the predictions and clustering made by the 11 nodes. In the second test we have monitored the network traffic and CPU loads of 3 nodes. Both ran for 4 hours, using the daytime hours 8–12 from the proxy log. Measurement data in the repository was stored for 2 hours before discarding it.

B. Results

The first test focused on the predictions and clustering made by the system. Naturally, one of the most important aspects of a distance prediction service is the error of prediction. Two widely used figures for this error are the relative error ($|x'-x|/\min(x', x)$, where x' is the predicted value) and the directional relative error ($(x'-x)/\min(x', x)$), both of which are defined in [3]. Furthermore, we look at the absolute error of prediction $x'-x$.

Figures 2 and 3 show the corresponding cumulative distribution functions. The observed results are very encouraging. Only very few predictions significantly over- or underestimate the measured round-trip time, and there is no noticeable bias. The results are significantly better than those achieved with GNP or IDMaps in [3]. Unfortunately, they are not fully comparable since the scenarios differ and our service only provides distance predictions, if it has recent measurement data available. If there is insufficient data in the repository, the service is forced to actively measure the distance.

In fact, the group members were only able to make predictions in 55.08% of the cases. In 11.26% of the cases, end systems were unreachable. Nonetheless, the main reason for this small ratio is the low rate of requests used in the test. Simulations based on the web proxy logs have shown that the ratio of successful predictions increases with the rate of requests. In other words, if the service receives a large number of requests over a short period of time, it will be able to answer

a high percentage of the subsequent requests because it has stored information about a large number of measurement targets. We could already achieve a success ratio of ca. 80% with 2 requests per second, which corresponds to a group of 40 members if each node receives a request every 20 seconds. 90% can be achieved with 10 requests per second.

A further result of interest is the number and the size of clusters the system has been able to identify during the test using our algorithm from [6]. Figure 4 shows the number of hosts assigned to clusters of various sizes. We can see that even though the majority of hosts are assigned to small clusters, clusters containing more than 80 hosts have been identified. In comparison to storing data for each destination independently, this clustering resulted in 62.06% fewer repository entries. This demonstrates that clustering may in fact significantly reduce the number of repository entries.

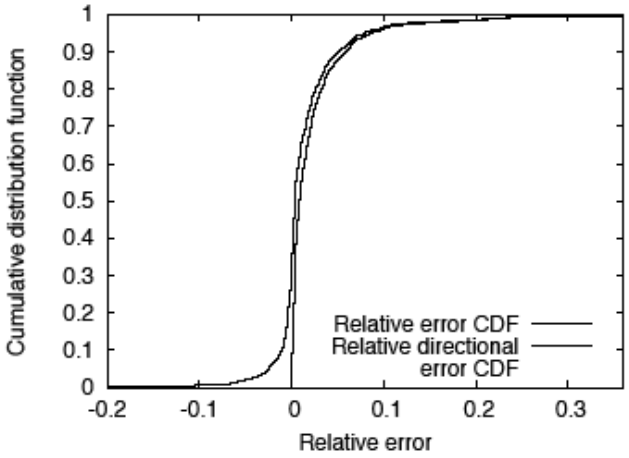

Figure 2. CDF of the relative error and the directional relative error

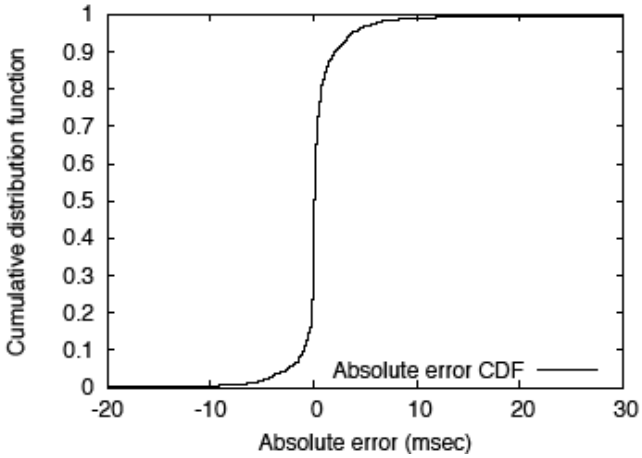

Figure 3. CDF of the absolute error

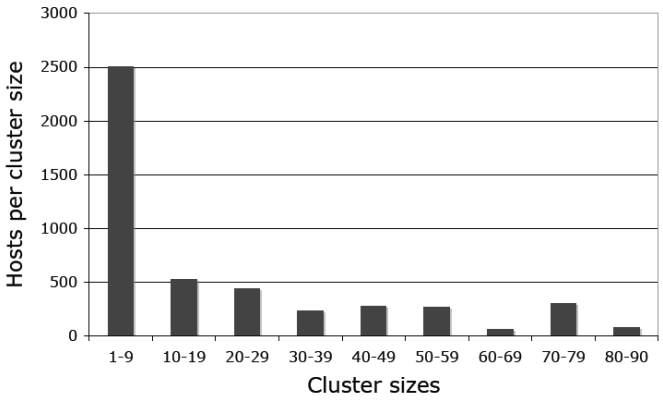

Figure 4. The number of nodes assigned toclusters of a given size

V. CONCLUSIONS

We have presented an architecture for a peer-to-peer based distance measurement and prediction service that does not require global deployment in order to provide useful distance predictions. In this architecture, end systems that are close to each other form small peer-to-peer groups in order to exchange and store distance measurements to remote end systems. The group members obtain measurements by passively monitoring existing traffic.

We have also presented results from a prototype deployment of the service on PlanetLab, showing that the service can really provide predictions for a majority of the requests. Moreover, the predictions are very good. In fact, they appear to be better than the estimates provided by established distance estimation services, although this comes at the price that not every request can be answered.

[1] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris, “Vivaldi: A decentralized network coordinate system,” ACM Sigcomm, Portland, USA, August 2004.

[2] Paul Francis, Sugih Jamin, Cheng Jin,Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang, “IDMaps: A global internet host distance estimation service”, IEEE/ACM Transaction on Networking, 9(5), pages 525-540, October 2001.

[3] T. S. Eugene Ng and Hui Zhang, “Predicting internet network distance with coordinates-based approaches”, IEEE Infocom, 2002.

[4] Antony Rowstron and Peter Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),,Heidelberg, Germany, November 2001.

[5] Matthias Scheidegger and Torsten Braun, “Improved locality-aware grouping in overlay networks”, KiVS 2007, Bern, Switzerland, Informatik Aktuell, pages 27–38. Springer, February 2007.

[6] Matthias Scheidegger, Torsten Braun, and Florian Baumgartner, “Endpoint cluster identification for end-to-end distance estimation”, IEEE ICC 2006, Istanbul, Turkey, June 2006.

[7] Wolfgang Theilmann and Kurt Rothermel, “Dynamic distance maps of the internet”, IEEE Infocom 2001, Tel-Aviv, Israel, March 2001.

[8] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer, “Meridian: A lightweight network location service without virtual coordinates”, ACM Sigcomm CCR, 35(4), pages 85–96, October 2005.

[9] Xin Yan Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, Wenwu Zhu, “A construction of localityaware overlay network: mOverlay and its performance”, IEEE Journal of Selected Areas in Communications 22(1), pages 18–28, January 2004.