

Dynamic Distance Maps of the Internet

Wolfgang Theilmann, Kurt Rothermel
Institute of Parallel and Distributed High-Performance Systems (IPVR),
University of Stuttgart, D-70565 Stuttgart, Germany
email: [theilmann|rothermel]@informatik.uni-stuttgart.de

Fakultätsbericht (Technical Report) 1999/08
Department of Computer Science
University of Stuttgart, Germany
July 1999

Abstract

There is an increasing number of Internet applications that attempt to optimize their network communication by considering the network distance across which data is transferred. Such applications range from replication management to mobile agent applications. One major problem of these applications is to efficiently acquire distance information for large computer networks.

This paper presents an approach to create a global view on the Internet, a so-called network distance map, which realizes a hierarchical decomposition of the network into regions and which allows to estimate the network distance between any two hosts. This view is not only a single snapshot but is dynamically adapted to the continuously changing network conditions. The main idea is to use a certain set of hosts for performing distance measurements and to use the so gained information for estimating the distance between arbitrary hosts. A hierarchical clustering provides the notion of regions and allows to coordinate the measurements in such a way that the resulting network load is minimized.

An experimental evaluation on the basis of 119 globally distributed measurement servers shows that already a small number of measurement servers allows to construct fairly accurate distance maps at low costs.

1 Introduction

The Internet, today's most important and largest computer network, is suffering from serious performance problems. Apart from improving the protocols on the network layer, various

approaches have been undertaken to speed up communication at the application level by minimizing the distance across which data is transferred. For example, clients of replicated services or replicated data want to locate the nearest location of a replicated entity [3, 11]. Replication servers want to disseminate popular data items towards large client groups [12, 2, 16]. Hierarchical caches and distributed object repositories aim to reflect a decomposition of the network into a hierarchy of regions [4, 20]. Routing of user queries in distributed Digital Libraries is optimized at the application level [14]. Finally, in mobile agent systems the distance between client and server is needed to decide whether to ship the client or to load the data over the network [19].

A major problem of all these applications is to learn about network distances without probing the whole network. For many of them it would even be enough to learn about the coarse adherence of hosts to regions (e.g. [2, 12, 14, 20]). However, most current applications are based on ad hoc solutions, which do not really solve the problem. In addition, most solutions are only appropriate for a particular application scenario and cannot be shared by different applications.

This paper presents so-called *network distance maps*, which offer a global view on a computer network such that the distance between any two network hosts can be derived. Besides, distance maps provide a decomposition of the network into a hierarchy of regions. They can be constructed for any distance metric. Moreover, distance maps are dynamic in the way that they adapt continuously to changing network conditions. The main idea of this approach is to use a subset of hosts to perform distance measurements and to cluster these hosts hierarchically into regions of closely connected hosts. It is important to stress that the approach applies to any interconnected network that supports some form of distance measurement.

The remainder of this paper proceeds as follows. After a discussion of related work in Section 2, Section 3 introduces some general assumptions, goals and restrictions of our approach. Section 4 presents the approach of network distance maps in detail and Section 5 reports on an experimental evaluation. Finally, Section 6 summarizes our conclusions.

2 Related Work

The estimation of real network distances through geographic distances has been proposed by Gwertzman and Seltzer [12]. However, they found out that the correlation between geographic distances and network distances is rather poor, especially between different backbones. In addition, there is, so far, no possibility to automatically determine the geographic location of all Internet hosts.

Various approaches propose to perform local measurements, e.g. from clients to replication servers ([3, 11]) or vice versa [2]. This way, a host is able to learn the distance between itself and some remote hosts, but it cannot derive the distance between two remote hosts. This is especially required for replication servers, which need to discover the distance between possible replication locations and clients [2]. A second deficiency of these approaches is the high overhead that occurs if every single host is responsible for performing measurements. Consider, for example, two closely connected clients of a replicated service. Since they do not know of each other they have to do almost the same measurements.

Rabinovich et al. exploit the information available in the routing tables of Internet routers [16]. In comparison to probing approaches this is a very efficient way of learning network distances. On the other hand, such an approach is bound to the distance metrics available in the routing tables. This can be an important restriction since, for example, the distance metric used for the routing between autonomous systems (ASs) is simply the number of traversed ASs [17]. In addition, the access to such tables is not public. So in general, such an approach can only be followed by the operator of an autonomous system and is restricted to distances between hosts within this system.

Two protocols for the dissemination of distance information, namely the SONAR- and the HOPS-service, have been proposed by [15], [5] respectively. A SONAR-server offers distance information between itself and arbitrary remote hosts. The HOPS-service offers distances between arbitrary hosts, by distributing the information in a DNS-like hierarchy. However, the problem of acquiring distance information is not addressed.

To our knowledge, the first and only approach for a global, measurement based view on the Internet has been presented by Francis et al. [6]. They propose to use a set of servers that measure the distances between themselves and to other end systems. Shortest path algorithms shall prune the resulting data structure. Two different models are presented. The first one tries to discover the real Internet topology, i.e. determines autonomous systems (ASs) and inter-AS links as well as intra-AS links. The second model is simply based on the measurement of end-to-end distances. To determine the closest measurement server for every end systems, they propose a random driven approach in which each server repeatedly measures its distance to a randomly selected end system and checks whether it is closer than the so far closest known server. Unfortunately, the whole discussion is determined by the goal to minimize the amount of data needed to store the network distance information. The network load caused by the measurements is not considered. No concrete algorithms have been presented and the problem of updating the acquired data structures is poorly discussed. We differ from the proposed methods in that we try to achieve scalability (both in terms of network load and storage requirements) by clustering the set of measurement servers hierarchically.

3 Assumptions, Goals and Restrictions

This section introduces the major assumptions and goals of our approach and provides a discussion of the inherent restrictions, we have to face.

3.1 System Model

Network. Our model of a network consists of a set of *hosts* \mathcal{H} together with a function $\Delta(x, y)$ that assigns a distance to each pair of hosts $x, y \in \mathcal{H}$. We assume distances to be non-negative and symmetric. We will discuss the necessity of this symmetry assumption in Section 4 and will show the extent of its validity for our experimental validation in Section 5.

We neither require any special distance metric nor care about the method for performing a single distance measurement. Instead, our approach can be used with any distance metric, for example, the number of hops (i.e. the number of network routers existing on a path between

two hosts), the round trip time (the time needed to transmit a simple datagram packet to a remote host and back), packet loss rates, bandwidth or anything else.

Because of space limitations, we do not discuss how our algorithms deal with host or network failures. Instead, we assume for our presentation that any pair of network hosts is connected. Section 4.5 presents some basic principles how fault tolerance can be included into our approach.

A possible optimization, which we do not discuss in this paper but which could be easily integrated into our approach, is to reduce the granularity of the considered Internet to address prefixes, i.e. to group together all hosts with the same address prefix. A good discussion of this optimization idea can be found in [6].

Network View. Of course, we cannot assume to have access to every network’s host in order to perform distance measurements. However, we assume the availability of a set $\mathcal{M} \subseteq \mathcal{H}$ of *measurement servers* (called *mServers* for the remainder of this article) that allow to perform distance measurements to arbitrary hosts. Remark that this applies only to the “public” part of the Internet since we cannot measure distances to hosts behind firewalls or to hosts within private networks. But even the distance to a firewall should be quite useful for an application from the outside. In the following, we will call a host *simple host* if we want to emphasize that it is not an mServer.

3.2 Accuracy and Timeliness of Distance Information

An important aspect of network distance maps is the quality (i.e. the accuracy and timeliness) of the distance information they provide. However, this aspect is influenced by some factors, external to the development of network map algorithms: First of all, there is the extra communication overhead we want to spend. This tolerable overhead in turn depends on the number of applications that make use of the distance map. Given the tolerable overhead, accuracy and timeliness depend on two additional factors: One is the effort needed to perform a single distance measurement. For example, the measurement of the currently available bandwidth causes an overhead much higher than the measurement of the current round trip time [3]. The other is the variation scale of the respective metric. For example, #hops distances are supposed to be relatively static, while current round trip time is highly dynamic.

To enable a maximum of flexibility for the operator of a distance map, we developed algorithms that can be tuned to arbitrary degrees of accuracy and timeliness.

3.3 Scalability

There are three requirements in terms of scalability that must be satisfied by algorithms to distance maps. Firstly, the network load caused by the process of constructing and maintaining a map must be limited. Because large computer networks consist of millions of hosts, it is out of question to perform measurements from every host to every other host. Secondly, distance maps should have small storage requirements. This allows to replicate them to almost any ordinary server system. Finally, the derivation of distance values should be quickly feasible so that a distance map server can satisfy a lot of requests almost simultaneously.

4 Network Distance Maps

This section introduces the approach of network distance maps in detail. After an overview on the basic ideas of our approach, we proceed with the main data structure. Then, we present the algorithms for constructing and updating a map and finally, we discuss extensions for the treatment of failures.

4.1 Overview

The basic ideas of our approach are (1) to rely on a set of measurement servers (mServers), (2) to measure the distances between these mServers, (3) to assign simple hosts to their most closely connected mServer and (4) to estimate the distance between two hosts by the distance between their two assigned closest mServers.

The accuracy of this approach is limited by the number and distribution of the mServers. The more mServers we have the average distance between a host and its closest mServer becomes smaller, and so the estimation by the distance between mServers becomes more accurate. Figure 1 sketches this scenario. The distances between hosts 1 and 2 and between hosts 2 and 3 are both

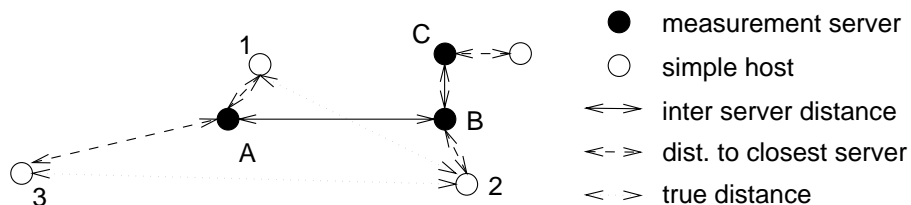


Figure 1: Distance estimation with measurement servers.

estimated by the distance between mServers A and B. While the first estimation is supposed to be relatively precise, the second one is not due to the large distance between host 3 and its closest mServer A.

Two problems in terms of scalability of the network load arise with this approach. Firstly, the computation of the closest assignment for simple hosts requires to measure the distance to this host from every mServer. Secondly, the number of distance measurements between mServers grows quadratically with the number of mServers. A lot of these measurements are redundant. For example in the scenario sketched in Figure 1, the distance between mServers A and C need not be measured if we know that C is close to B and B is far from A.

To solve these problems we cluster the mServers in a hierarchical manner, thus achieving a decomposition into regions in which each region is further refined into subregions. For each region/cluster we select a representative mServer. Then, the closest assignment for simple hosts can be done hierarchically by measuring the distance of the respective host to each representative of a toplevel cluster. The cluster with the closest representative is selected and the process is continued for its subclusters. Since we do not want to measure all distances between mServers for the initial clustering, we propose a mixed algorithm, that first computes a pre-clustering on a subset of the available mServers and then assigns the additional mServers to their most appropriate cluster.

It is important to remark that our cluster algorithms require symmetric distances. Otherwise they cannot decide whether to group together two entities or not, especially if the two distances significantly differ from each other.

4.2 Data Structure

The data structure of a network distance map is presented in Figure 2. It consists of a cluster tree and the additional assignment of simple hosts to their closest mServer. Inner nodes of this tree represent clusters of mServers, leaf nodes correspond to single mServers. The tree is

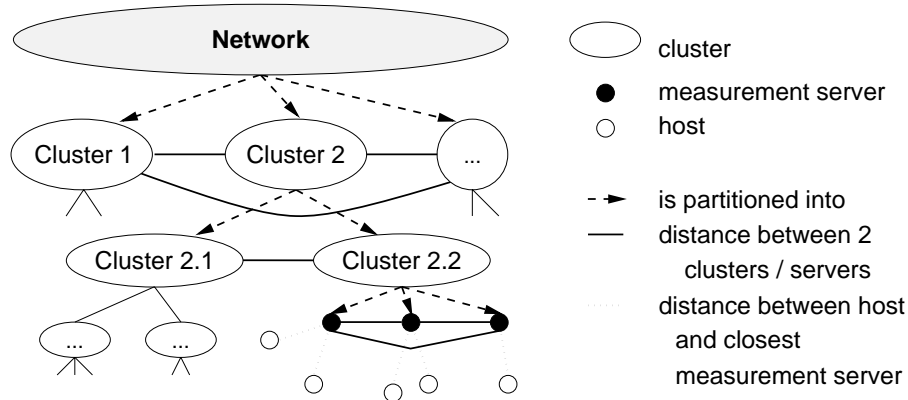


Figure 2: Representation of a network distance map.

extended by distance values between sibling nodes. The distance between two sibling clusters is an estimation for the average distance between arbitrary hosts belonging to these clusters. The distance between sibling mServers is directly derived from the corresponding network measurement. In addition, the assignment between simple hosts and their closest mServer is extended with the associated, measured network distance.

Distances (denoted by $\|*,*\|$) can be derived from this tree representation as follows. The distance $\|m_1, m_2\|$ between two mServers m_1 and m_2 is estimated as the distance between the children of the least common ancestor¹ of m_1 and m_2 . Based on this, the distance $\|h_1, h_2\|$ between two arbitrary hosts h_1 and h_2 is estimated as

$$\max\{\|cl(h_1), cl(h_2)\|, \Delta(h_1, cl(h_1)), \Delta(h_2, cl(h_2))\},$$

where cl denotes the closest assignment and Δ the associated, measured distances. Note that the first argument in the max-function is estimated from the cluster tree while the other ones are gained by network measurements. The max-function guarantees accurate distance estimations also for hosts, which are connected to the same mServer, and for hosts, which are not closely connected to any mServer, i.e. which are not well covered by the cluster tree. The derivation of any distance is feasible in linear time, i.e. linear to the tree's depth.

The storage requirements of a cluster tree can be estimated as follows: Let k be the maximum number of sub-clusters or mServers within one cluster and d the depth of the cluster tree

¹The *ancestor*-relation is the transitive extension of the *parent*-relation.

(the root's depth is defined as 0). Then, a complete cluster tree consists of $(k^{d+1} - 1)/(k - 1)$ nodes. Since each node must contain distance values for its siblings (at most $k - 1$) and assuming a well balanced tree with $|\mathcal{M}| = k^d$ we can specify an upper bound for the amount of data required for this representation of a network to

$$O\left(|\mathcal{H}| + \frac{k^{d+1} - 1}{k - 1} * (k - 1)\right) = O(|\mathcal{H}| + |\mathcal{M}| * k).$$

The parameter k determines the tradeoff between the accuracy of the network's representation and the amount of required data. Setting k to $|\mathcal{M}|$ would result in an exact representation of a network view except for the fact that the simple hosts are only linked to their closest mServer. Taking an example network with 10.000 mServers and 1 million hosts and assuming $k = 10$ we need only $1.1 * 10^6$ data items which is better by orders of magnitude than the 10^{10} items required for a complete representation of a network view.

Cluster Representative. A cluster representative should satisfy two conflicting goals: On the one hand, it should be "representative" in the way that distances from hosts outside the cluster to the cluster representative should be similar to the distances to the other cluster's hosts. On the other hand, the election of a representative should not induce a tremendous additional network load. This is especially important for the updating process.

We decided to take the *cluster centre* as the representative, i.e. the host for which the maximum distance to the other hosts of a cluster is minimal. This choice has several advantages: It does not require any additional network measurements, neither for the initial computation nor for the updating. If we compute the centres hierarchically, that is if the centre of each cluster is computed as the centre of its child-clusters' centres, we can use the distances derived from the cluster tree. In addition, cluster centres, which are characterized by their good internal connectivity, are supposed not to have an unrepresentative bad connectivity to the outside. Therefore, they are quite appropriate for the insertion and updating of simple hosts (compare to the algorithms in the next paragraphs).

4.3 Initial Computation of a Distance Map

Cluster Criteria. The first question is how to compute an effective clustering. Francis et al. discuss the value of clustering according to autonomous systems (ASs) in the Internet [6]. However, they abandon this idea since ASs may be very widespread and hosts from different AS may be close to each other (in terms of latency).

According to our way of deriving distances from the cluster tree, the optimal cluster criterion would be the *Min k-Avg-Error* criterion, which minimizes the average error that occurs from our distance estimation, i.e. minimizes the expression

$$\sum_{\substack{i,j \in [1..k] \\ i < j}} \sum_{\substack{x \in C_i \\ y \in C_j}} |\Delta(x, y) - \Delta(C_i, C_j)|.$$

In this formula, k denotes the given number of resulting clusters, C_i a single cluster and $\Delta(C_i, C_j)$ the average distance between hosts in the two clusters C_i, C_j . Unfortunately, we do not know any polynomial solution or approximation to this optimization problem.

Therefore, we evaluated some heuristic criteria that either seek to maximize the distances between two different clusters or to minimize the distances that occur within one cluster. By either optimizing the average or the extreme value we have four possible criteria, which we discuss in the following:

A *Max k-Separation* maximizes the minimum distance between any two clusters, i.e. maximizes the expression

$$\min_{\substack{i,j \in [1..k], i < j \\ x \in C_i, y \in C_j}} \Delta(x, y).$$

According to [18], it is computable in $O(n^2 * \log n)$.

A *Max k-Cut* maximizes the average distance between any two clusters, i.e. maximizes the expression

$$\sum_{\substack{i,j \in [1..k] \\ i < j}} \sum_{\substack{x \in C_i \\ y \in C_j}} \Delta(x, y).$$

It is NP-complete to optimize and the best known approximation ratio [7] is $1/(1 - 1/k + 2 \ln k/k^2)$, which is rather bad if we consider that a random clustering already achieves an approximation ratio of $1/(1 - 1/k)$.

A *Min k-Clustering* minimizes the maximum distance between any two hosts within the same cluster, i.e. minimizes the expression

$$\max_{i \in [1..k], x, y \in C_i} \Delta(x, y).$$

An approximation for this NP-complete problem with the optimal ratio 2 can be computed in $O(n * k)$ if we assume the triangle inequality²[9].

A *Min k-Clustering Sum* minimizes the average distance between any two hosts within the same cluster, i.e. minimizes the expression

$$\sum_{i=1}^k \sum_{x, y \in C_i} \Delta(x, y).$$

The fastest known approximation with ratio 2 for this NP-complete problem has a computation complexity of $O(n^k)$, which is too expensive for our application [10].

The experiments in the next section are performed with the two extreme cluster criteria, *Max k-Separation* and *Min k-Clustering*. Optimization of extreme values can sometimes lead to unexpected results since the process can be completely determined by one single data item. For a better understanding of such effects we present two example clusterings in Figure 3. Distances in these examples are defined through planar geometry. The remarkable effects are that a *Max k-Separation* may lead to unbalanced clusters and a *Min k-Clustering* may result in closely connected hosts that reside in different clusters.

The total processing time of a cluster tree computation depends on the cluster criteria and the extent to which the computed clusters are well balanced, i.e. how often and on which set sizes the partitioning has to be repeated. Especially for the *Max k-Separation*, where the computation has quadratic order, the time significantly depends on the balancing effectiveness.

²The triangle inequality is satisfied iff for any three hosts a, b, c the distances obey $|a, c| \leq |a, b| + |b, c|$.

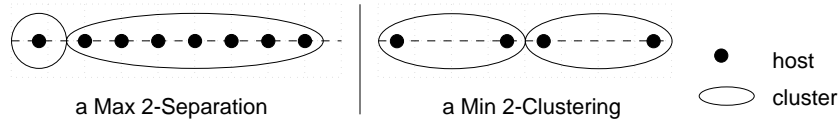


Figure 3: Two extreme clustering results.

Mixed Clustering. As discussed in Section 4.1 the algorithm for computing an initial clustering follows a mixed strategy by first computing an optimal clustering on a certain mServer subset (according to a selected criterion) and then assigning the additional mServers to the most appropriate clusters. Therefore, the algorithm is determined by two parameters: The *refinement factor* k describes the number of clusters to be computed at each level of this recursive process. The *selectivity* s denotes the number of mServers that are considered for the optimal clustering step. The algorithm consists of the following steps:

-
- (1) select a set S of s mServers randomly from \mathcal{M}
 - (2) measure distances between all hosts in S
 - (3) do optimal clustering of S into k clusters C_1, \dots, C_k
 - (4) compute cluster distances as average value
 - (5) compute cluster centres
 - (6) for each additional mServer $m \in \mathcal{M} \setminus S$
 - (7) measure distances to all cluster centres
 - (8) assign m to cluster with closest centre
 - (9) update distances between clusters
 - (10) for each cluster C_i
 - (11) perform step (1) recursively for the mServers in C_i
-

The distance update in step (9) is done for the distance estimations between the closest cluster and all the other clusters. Since we measured the distance between mServer m and all cluster centres we have for each such cluster pair a new measurement, which allows to refine our distance estimation. If C denotes the cluster selected for mServer m and d_i denotes the distance between m and the centre of cluster i then the distance between C and C_i ($C_i \neq C$) is updated according to

$$|C, C_i| := |C, C_i| + (d - |C, C_i|) * \frac{1}{|C| + |C_i|}.$$

The refinement factor k determines the granularity of the cluster tree and so the accuracy of distance estimations. The selectivity s determines the probability for an effective clustering, i.e. the probability that the randomly determined set S enables the computation of “representative” clusters.

The number of network measurements resulting from the first recursion level of the initial clustering is $s * (s - 1)/2 + k * (|\mathcal{M}| - s)$. Remark that symmetric distances only need to be measured once. If we assume $s = k^n$ and $|\mathcal{M}| = k^d$ ($0 < n < d$) we can specify the total number of measurements required for a complete, recursive clustering that leads to a well balanced tree

to

$$\sum_{i=0}^{d-n} k^i * \left(\frac{s * (s - 1)}{2} + k * (k^{d-i} - s) \right). \quad (1)$$

If, for example, we have 1000 mServers a clustering with $k = 10$ and $s = 100$ requires 63450 measurements, which is about 8 times less than measuring the distances between all mServers (499500).

Insertion of Simple Hosts. The insertion of simple hosts into a cluster tree is similar to the insertion process described above. At each level, distances between the new host and the respective cluster centres are measured. The process is recursively continued for the cluster with the closest centre. The insertion of a single host into an optimal balanced tree of depth d and refinement factor k requires $d * k$ network measurements.

We also realized an extended version of the above algorithm in which the insertion of simple hosts is performed more sophisticatedly. In this version, we check if several cluster centres have a similar distance to the new host than the closest cluster centre. For each such cluster, we concurrently continue the insertion process until one cluster turns out to be significantly better than the other ones. This extended version is driven by a *similarity threshold* that bounds the range up to which additional centres are further examined. The performance evaluation of this version is done empirically in Section 5.3.

4.4 Map Updating

The updating of a distance map shall adapt this map to changing network conditions. A trivial and expensive solution is, of course, to recompute the whole map at regular time intervals. However, we also developed mechanisms that allow a less expensive and continuous updating.

Updating of Simple Hosts. There are two basic possibilities for updating the closest assignment of simple hosts. Either we re-insert each host from higher nodes in the cluster tree or we perform a kind of radial search that frequently checks if other mServers in the neighbourhood of the closest mServer have become closer than this original one. With both approaches we can limit the range within updates shall be done. This allows to perform local updates more frequently than global ones, i.e. checks in the near neighbourhood (from direct super-clusters) can be done more often than in the further neighbourhood (further super-clusters). The underlying heuristic of this method is that small changes in the network conditions are more likely to occur than large changes.

We developed an algorithm that combines the above introduced policies and that is driven by two parameters: the radius, i.e. the number of neighbored mServers or clusters, which are examined in the order of their proximity, and the similarity threshold, which is used for the insertion process into neighbored clusters. The closest assignment of a simple host is changed if another, more closely connected mServer is detected by this algorithm.

Updating of Cluster Trees. Cluster trees are updated by reclustering parts of them. The main parameter that determines this process is the tree level at which the reclustering shall be

done. We define the tree level of the root node as 0. Reclustering at level l effects that for every cluster at level l its associated mServers are collected and a new clustering is performed on the basis of this set of mServers. A reclustering at level 0 is identical to a complete clustering.

The effort, both in times of computing complexity and network load, for reclustering a well balanced tree at level l is k^l times the effort for clustering a set of $|\mathcal{M}|/k^l$ mServers. Continuing the example we presented for equation 1, i.e. 1000 mServers and $k = 10$, an optimal reclustering at level 1 requires 49500 measurements which is 10 times less than an optimal reclustering at level 0.

4.5 Treatment of Failures

The consideration of network or host failures can be easily integrated into our algorithms. Basically, three methods are needed: During the clustering process, not measurable connections are reflected by the value of infinity. If during the insertion or update of hosts the distance to a cluster representative cannot be obtained, other members of the respective cluster have to be taken instead. If the distance to none of these other cluster members can be obtained, the distance between the host and the cluster is set to infinity. Finally, crashed mServers can be easily retracted from the cluster tree by assigning its associated simple hosts to its closest neighboured mServer.

5 Experimental Evaluation

In this section, we present an experimental validation of our approach to compute network distance maps on the basis of data acquired from Internet measurements. Various distance metrics might be interesting, for example the number of hops, the round trip time, packet loss rates, bandwidth and many others. We performed experiments for the #hops and round trip time metric because of their distinct “nature” and because we were able to measure them in a large scale.

5.1 Methodology

We performed network measurements on the Internet by using the tool *traceroute*, which allows to trace the routing path between two Internet hosts [13]. The resulting data allows to derive the number of hops and a very rough estimation of the round trip time (rtt). As Acharya and Saltz [1] have shown, rtt has a large temporal variation and a value that well characterizes a rtt distribution requires a larger sample of single rtt measurements. However, we decided to take these rough estimations for evaluating the behaviour of our algorithms for different distance metrics. A second source of imprecise and incomplete measurements is that traceroute does not always succeed since it is not supported by every host or router. A detailed discussion of traceroute’s caveats can be found in [8].

Various web servers on the Internet offer the service of performing traceroute measurements from their location to an arbitrary Internet host. We compiled a set of 119 such web servers (mServers) and 460 simple hosts, which are distributed among 5 continents and numerous

autonomous systems. A first data collection ($D1$) was gathered from February 1 to February 11, 1999. We performed measurements from every mServer to every other host, in total 68782 measurements. Since we did not want to overload the mServers we provided an interval of 2–5 minutes between two consecutive traceroute’s from a single mServer. In addition, we took care that every host is target of a traceroute measurement at most every 2–5 minutes. These mechanisms ensure that we do not have a significant interference between the distributed measurement activities.

From March 24 to April 1, 1999 we gathered a second data collection ($D2$), based on 110 mServers and 467 simple hosts. Unfortunately, 9 mServers shut down their service and 2 simple hosts were not reachable anymore. So this second collection is based on 110 mServers and 467 additional hosts (we used the shut down mServers at least as simple hosts). Comparisons between these two collections are always done on the basis of the common mServers and simple hosts, which are exactly those used in $D2$.

5.2 Data Analysis

Figure 4 shows the histograms of the 67961 successfully measured distances in collection $D1$. The rightmost column of both histograms presents cumulative values for the open distance interval that starts at the rightmost histogram’s argument. These distributions are quite similar

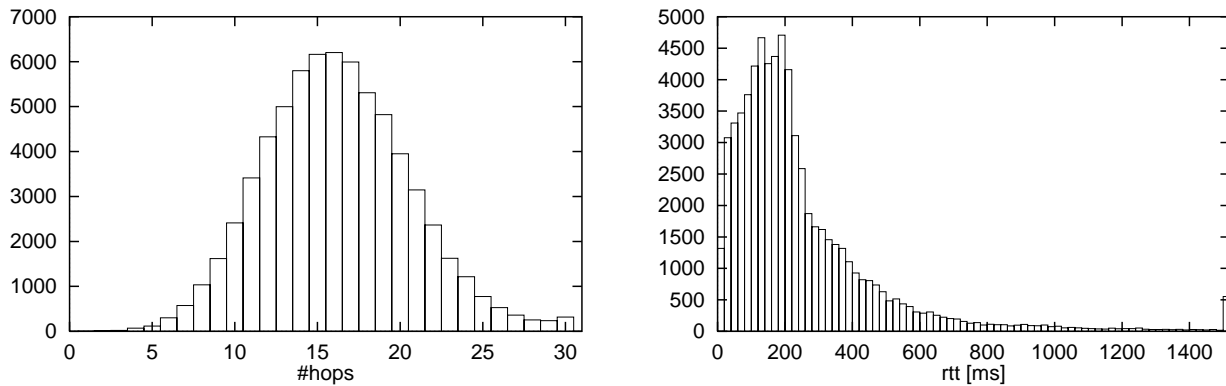


Figure 4: Histograms of the measured distances.

to those measured elsewhere, for example [3].

Our first examination targeted the validation of the symmetry assumption made in our network model (compare to Section 3.1). On the basis of 6960 successful bidirectional measurements we computed the average symmetry error of the hops measurements to 2.0 hops (standard deviation 2.3). The average symmetry error of the rtt measurements is 119ms (std.dev. 266ms). Though the definition of rtt effects symmetric distances in theory it is not surprising that this property is not well fulfilled by our measurements because of the poor measurement method.

Next, we examined the validity of the triangle inequality, which is assumed by some of the cluster criteria, presented in Section 4.3. From more than 7 million host triplets we derived that the triangle inequality is valid in 99.5% for the #hops metric and in 87.6% for the rtt metric.

Finally, we computed for every simple host its most closely connected mServer. The average value for the #hops metric is 7.1, which is about two fifth of the average of #hops distances (16.2). The average value for the rtt metric is 27ms, which is about one tenth of the average of rtt distances (255ms). This shows that in the sense of rtt our set of mServers covers the Internet quite well. For every simple host there is a comparatively closely connected mServer. This property is fulfilled much worse for the #hops distances.

The same evaluations for the second data collection $D2$ brought out very similar results. We additionally computed the divergence of distances between the two collections, i.e. for every pair of distances $d1 \in D1$ and $d2 \in D2$ we computed the ratio $\frac{\max(d1,d2)-\min(d1,d2)}{\max(d1,d2)}$. This is depicted in Figure 5. The average divergence for the #hops metric is 8.2% (standard deviation

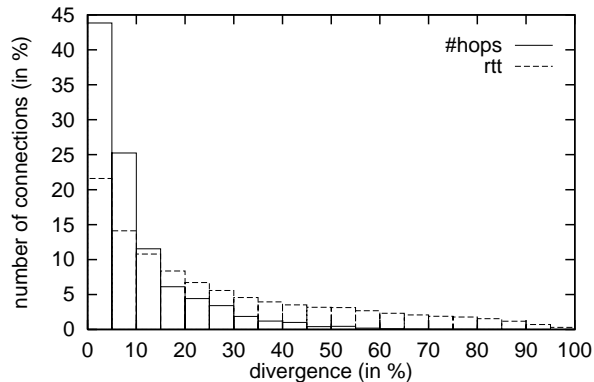


Figure 5: Divergence between $D1$ and $D2$.

10.7%) and 24.8% for the rtt metric (std.dev. 24.0%). Unsurprisingly, the #hops distances turn out to be much more stable than the rtt distances.

5.3 Initial Map Computation

The following experiments on cluster trees are based on averaged values for bidirectionally measured distances. This is necessary because our cluster algorithms require symmetric distances. We used data collection $D1$ for the evaluation of the initial map construction. Collection $D2$ was used for the update evaluation.

Optimal Clustering. Our first experiments target a comparison of the two cluster criteria *Max k-Separation* and *Min k-Clustering*. We computed cluster trees for both criteria, for both distance metrics and varied k between 2 and 119. The selectivity s was always set to $|\mathcal{M}|$. The accuracy of a cluster tree is described by the estimation error between measured and estimated distances. The relative estimation error between a measured distance m and the associated estimation e is computed as $\frac{\max(m,e)-\min(m,e)}{\max(m,e)}$, thus it is a normalized value between 0 and 1. A second parameter that describes cluster trees is the tree’s depth. It is an important factor since deeper trees effect a higher effort (i.e. a larger number of network measurements) needed for the insertion and updating of simple hosts. Figure 6 shows the relative estimation error (the average for all pairs of measured and estimated distances) and the tree depth for each computed cluster tree.

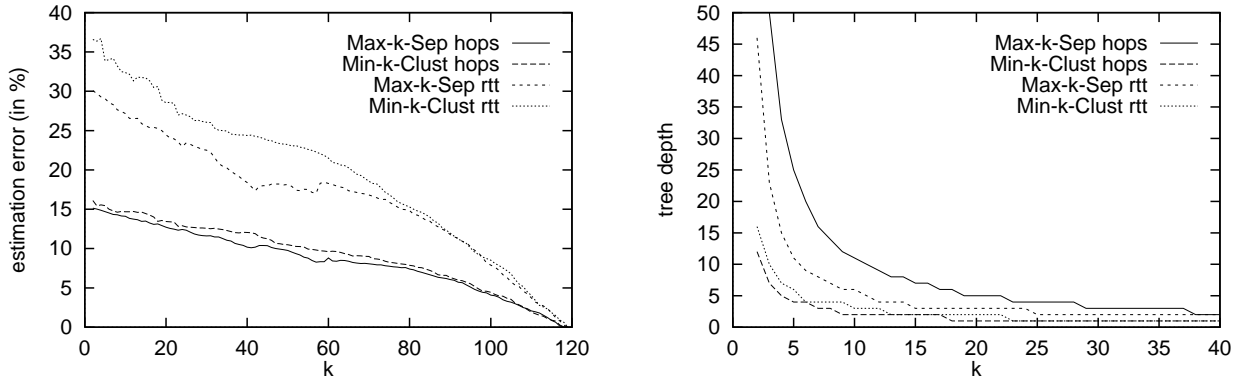


Figure 6: Characteristics of cluster trees.

We can see that the estimation of #hops distances is significantly better than for rtt distances. However, this was already indicated by our data analysis in the previous section. Both errors decrease with an increasing k . Of course the error is 0 for $k = 119$ since in this case the tree contains all measured distance values between pairs of mServers. We can also see that the distance estimations are slightly better if we use the *Max k-Separation* criterion, especially for the rtt metric. On the other hand, the tree depths resulting from the *Min k-Clustering* are significantly smaller. For example, the tree depth for the #hops metric and $k = 10$ is only 2 while it is 11 for the *Max 10-Separation* tree. This indicates that the problem of unbalanced trees (described in Section 4.3) occurs for the *Max k-Separation* trees.

Another observation is that the correlation between k and the estimation error is roughly linear. This allows the conclusion that there is no *natural clustering*, which should be preferred for getting an optimal balance between k and the resulting estimation error. Of course, it is possible that there will be a natural clustering for larger numbers of mServers, for example, according to autonomous systems or backbones in the Internet. However, our experiments do not allow such a conclusion.

Next, we analysed the quality of the hierarchical refinement of clusters. Table 1 contains some characteristic parameters that describe the three highest tree levels for some example trees. Due to space limitations we only present results for trees with $k = 10$. However, these are also characteristic for trees with other refinement factors. For each described tree level l , we show the average measured distance between mServers that share the same cluster at level l and the associated absolute estimation error. Round trip time distances are always presented in milliseconds. We can see that the average measured distance decreases for deeper

criterion	network distances / abs. estimation error		
	level 0	level 1	level 2
10-Sep. #hops	15.6 / 2.47	15.1 / 2.46	14.7 / 2.45
10-Clu. #hops	15.6 / 2.65	12.7 / 1.52	1.52 / 0.0
10-Sep. rtt	249 / 91	163 / 70	145 / 67
10-Clu. rtt	249 / 109	147 / 39	80 / 16

Table 1: Cluster tree quality at levels 0–2.

tree levels, thus hierarchical clustering is useful. The absolute estimation error also decreases for deeper tree levels, which proves that the cluster tree distances well reflect this increased proximity. Both effects are more clearly for the *Min 10-Clustering* criterion, which shows that the corresponding trees are better balanced.

Mixed Clustering. Another aspect of investigation was the extent to which we can use the mixed clustering strategy, described in Section 4.3. We performed test series for cluster trees with a refinement factor $k = 10$ and varied the selectivity s between 10 and 119. The resulting estimation errors for each constellation are depicted in the graph of Figure 7. We can see that

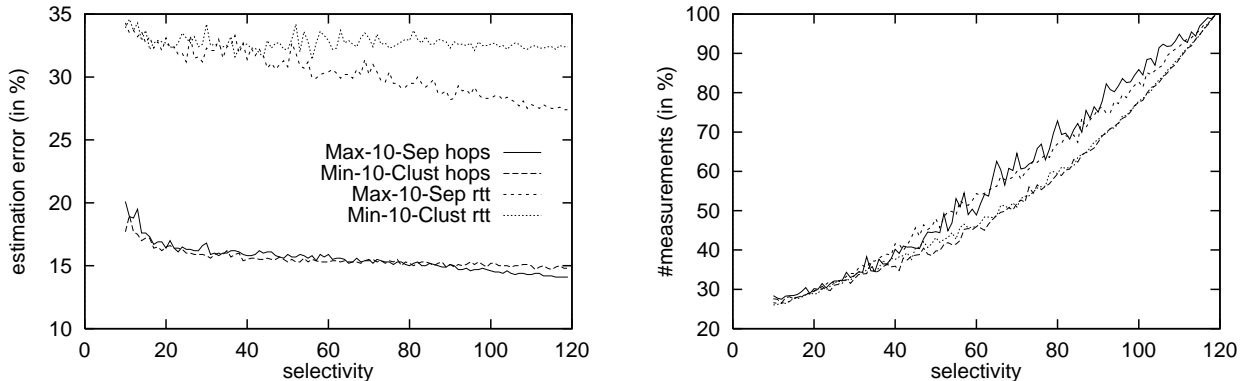


Figure 7: Mixed clustering.

the mixed algorithm can provide fairly good results at low costs. This is especially the case for the *Min k -Clustering* criterion where a selectivity of 20 allows to approximate the optimum, which is represented by the rightmost value and the maximum selectivity, up to 1% (#hops) resp. 0.3% (rtt). This approximation is less fast for the *Max k -Separation* criterion but still very useful. The number of network measurements, shown at the right side of Figure 7 grows roughly linear with the selectivity, starting with about 1800 (26%) network measurements up to the maximum of 7021 (100%). A selectivity of 20 reduces the required number of network measurements to 28%. Due to space limitations, we do not present the corresponding graph.

Insertion of Simple Hosts. Next, we analysed the algorithm for the hierarchical insertion of simple hosts into a cluster tree. For the clarity of the presentation, we do not show the different settings of the radius and similarity parameters (see Section 4.3). Instead, Figure 8 presents for each setting the resulting number of network measurements and the *closest approximation* which is the ratio of the optimal distance to the closest mServer and the distance to the actually assigned mServer. Both parameters are presented as the average for all simple hosts. The leftmost value of each line represents the case of performing the basic insertion algorithm that does not consider similar clusters concurrently.

We can see that the approximation is quite good for the #hops metric and the *Min 10-Clustering*. An average of 30 measurements per simple host, which is one fourth of the maximum number of measurements, leads to a closest approximation of about 83%. The approximation for the *Max 10-Separation* tree is much worse due to its unbalanced shape. The approximations

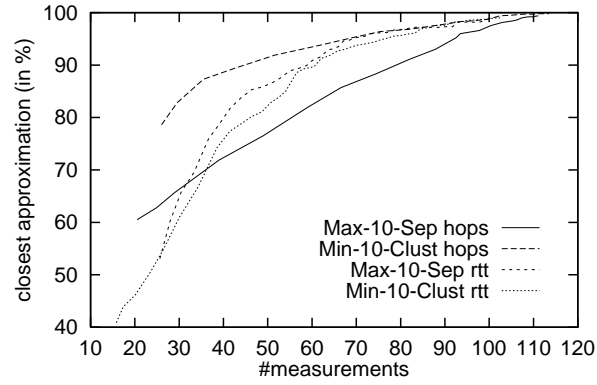


Figure 8: Insertion of simple hosts.

for the rtt metric are quite similar for both criteria. Naturally, they are worse than for the #hops metric. However, an average of 40 measurements leads to an approximation of 76%–79%.

Distance Estimation for Simple Hosts. Since we cannot measure the distance between simple hosts we computed various distance maps, always keeping exactly one mServer unused for the cluster tree computation. Distances between this unused mServer and other (real) simple hosts allow to evaluate the quality of distance estimations between simple hosts. Figure 9 shows the absolute rtt estimation errors at the first two tree levels for various constellations.

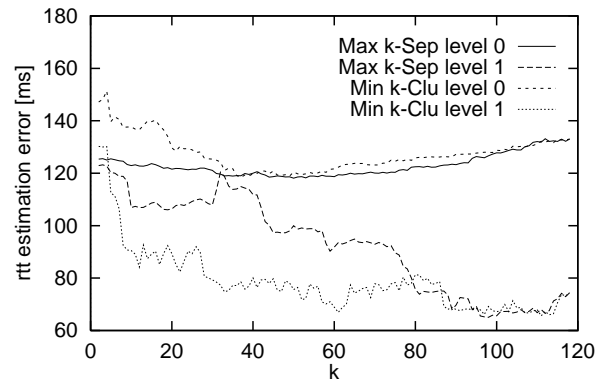


Figure 9: Simple host distance estimation (rtt).

We can observe that the level-0 curves decrease towards the middle of the graph (the optimal relative estimation error is 32.5%) and then increase for larger values of k . The first of these two effects well conforms to our previous evaluations. However, the latter one needs further explanation: For large values of k the number of mServers within each cluster becomes very small. Therefore, the estimation of distances between clusters becomes less robust. Considering the extremely simple method of measuring rtt distances (compare to Section 5.1) this effects the worse estimation of distances between simple hosts. The level-1 curves show that distance estimations at the next tree level are significantly better and become more precise if k increases.

Figure 10 shows the absolute #hops estimation errors for the same scenarios as above. Surprisingly, the estimation error does not decrease or even increases for larger values of k .

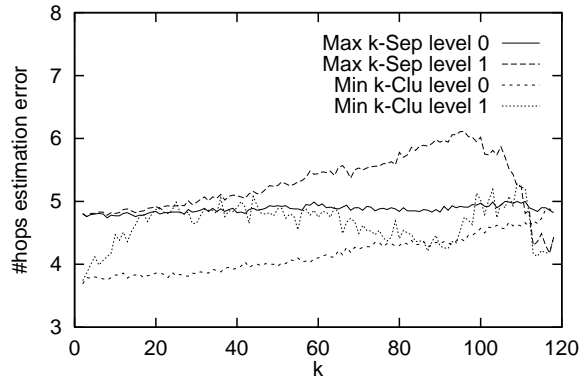


Figure 10: Simple host distance estimation (#hops).

The reason for this effect can be found in the large average distance between simple hosts and their closest mServer, which is 7.1 hops (2/5 of the average of hops distances). Even if the estimation of distances between mServers becomes more precise there remains a path of about 7.1 hops that is not considered by the distance estimations for the simple hosts. The optimal relative estimation error for the #hops metric is 19.9%.

5.4 Map Updating

The experiments in this section are based on both data collections, $D1$ and $D2$. The first one is used to compute an initial distance map. This map is updated with measurements gained from the second collection. The resulting map is compared to the optimal one, which results from a complete computation on the basis of the collection $D2$.

Updating of Simple Hosts. We present the analysis of the algorithm for updating the closest assignment of simple hosts in the same way than we did for the insertion algorithm. Once more, we show the closest approximation as a function of the required number of measurements. Both parameters are presented as the average for all simple hosts. Figure 11 presents the results for trees with parameter $k = 10$. For the purpose of comparison, we also plotted the results of a random algorithm that checks the distance to randomly chosen mServers.

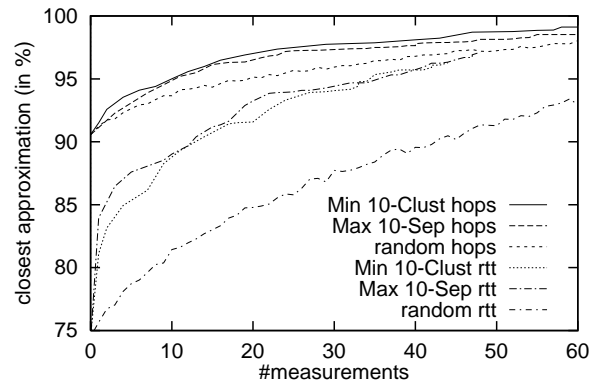


Figure 11: Update of simple hosts.

We can see that the initial approximation for the #hops metric is much better than for the rtt metric. This corresponds to the distinct divergence distributions presented in Section 5.2. The approximation quality is fairly similar for both cluster criteria and clearly better than for the random algorithm. An approximation of 95% for the #hops metric is possible with an average of 10 measurements (20 for the random algorithm) and rtt can be approximated to 90% with 13 measurements (42 for the random algorithm).

Updating the Cluster Tree. We have also performed update experiments for cluster trees with various settings for parameter k . As a representative example, we show the results for $k = 10$ in Table 2. We took the cluster trees originating from data collection $D1$ and updated them at the levels 2 and 1 by applying the complete cluster algorithm. For each of these trees we computed the estimation error of distances between mServers with respect to collection $D2$. This error was computed for the tree levels 0–2 and is shown in columns 3–5 of Table 2. Column 2 shows the number of network measurements needed for the reclustering operations. For comparison purposes, we also show the corresponding parameters of the actual tree, computed on the basis of collection $D2$.

	#network measurements	absolute estimation error		
		level 0	level 1	level 2
Max 10-Sep #hops				
$D1$ -tree	-	2.69	2.7	2.69
recluster level 2	4186	2.55	2.53	2.48
recluster level 1	5050	2.51	2.48	2.46
$D2$ -tree	5995	2.47	2.46	2.44
Min 10-Clu #hops				
$D1$ -tree	-	2.76	1.83	0.94
recluster level 2	109	2.74	1.74	0.0
recluster level 1	1060	2.71	1.56	0.0
$D2$ -tree	5995	2.66	1.49	0.0
Max 10-Sep rtt				
$D1$ -tree	-	119	103	99
recluster level 2	1369	115	94	83
recluster level 1	2385	110	82	80
$D2$ -tree	5995	109	85	83
Min 10-Clu rtt				
$D1$ -tree	-	134	81	34
recluster level 2	433	134	81	32
recluster level 1	1227	128	53	9
$D2$ -tree	5995	139	55	0

Table 2: Cluster trees updated at level 1 and 2.

We can see that reclustering at tree levels lower than the root level is quite effective and that it even improves distance estimations at higher levels. The reclustering is most effective for the

Min k-Clustering trees where a fairly good approximation of the actual tree is achieved with about 1200 measurements, which is only one fifth of the number of measurements required for the complete recomputation. The reclustering of *Max k-Separation* trees is more costly in terms of network measurements due to the more unbalanced shape of these trees.

5.5 Discussion

The experiments have shown that the construction of cluster trees and the determination of closest mServers, i.e. the hierarchical decomposition of a network into regions, works very well and at reasonable costs.

However, the distance estimation for simple hosts is not yet satisfying for two reasons: Firstly, the distribution of #hops distances has a small deviation around the mean. Therefore, the average distance between simple hosts and their closest mServer is rather high (compared to the general average of #hops distances). To overcome this deficiency, a larger set of mServers would be needed. Secondly, our method for measuring round trip times is too simple for achieving characteristic rtt values. As Acharya and Saltz [1] have shown, the mode value of a larger sample of rtt measurements is a good characterisation of rtt distribution and remains valid for about 45 minutes. Despite these deficiencies, the achieved distance estimations with an error of 19.9% (#hops) resp. 32.5% (rtt) are quite useful (see [6] for a discussion of useful ranges of estimation errors).

We have already successfully deployed distance maps for the coordination of mobile agents [19]. There, we were able to reduce the network load up to 90%.

6 Conclusions

This contribution has presented the concept of network distance maps, which allow to estimate the network distance between arbitrary Internet hosts. Besides, network maps provide a decomposition of the network into a hierarchy of regions of closely connected hosts. By means of hierarchical clustering, we have been able to achieve a highly scalable solution in terms of network load, storage requirements and distance computing complexity. Algorithms for the initial construction of network maps and for their updating have been presented and the approach has been successfully validated for two completely different distance metrics.

Future work concentrates on the development of fault tolerant protocols for the coordination of the measurement activities and for the dissemination of the distance information. In addition, we intend to improve the algorithm for updating cluster trees, realizing this process in a more distributed way.

References

- [1] A. Acharya, J. Saltz: *A Study of Internet Round-trip Delay*. Technical report CS-TR-3736, Department of Computer Science, University of Maryland, USA, December 1996

- [2] A. Bestavros: *WWW Traffic Reduction and Load Balancing through Server-Based Caching*. IEEE Concurrency, Special Issue on Parallel and Distributed Technology, vol. 5, Jan-March 1997, pp. 56-67
- [3] R.L. Carter, M.E. Crovella: *Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks*. Proc. 16th IEEE Infocom'97, Kobe, Japan, 1997, IEEE Press
- [4] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwarz, K.J. Worrel: *A Hierarchical Internet Object Cache*. Proc. USENIX Annual Technical Conference (USENIX'96), San Diego, CA, USA, January 22-26, 1996, USENIX
URL: <http://www.usenix.org/publications/library/proceedings/sd96/>
- [5] P. Francis: *A Call for an Internet-wide Host Proximity Service (HOPS)*. White paper, March 1997, URL: <http://www.ingrid.org/hops/wp.html>
- [6] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, Y. Jin: *An Architecture for a Global Internet Host Distance Estimation Service*. Proc. 18th IEEE Infocom'99, New York, USA, March 21-25, 1999, IEEE Press
- [7] A. Frieze, M. Jerrum: *Improved approximation algorithms for MAX k -CUT and MAX BI-SECTION*. Proc. 4th Int. Conf. on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, vol. 920, Springer-Verlag, 1995, pp. 1-13
- [8] G. Gardner: *Understanding Traceroute*. GeoNet Communications, white paper, 1998, URL: <http://www.noc.geo.net/support/traceroute.html>
- [9] T.F. González: *Clustering to minimize the maximum intercluster distance*. Theoretical Computer Science 38, North-Holland, pp. 293-306, 1985
- [10] N. Guttmann-Beck, R. Hassin: *Approximation algorithms for min-sum p -clustering*. Discrete Applied Mathematics, Elsevier, 1999 (to appear)
- [11] J.D. Guyton, M.F. Schwarz: *Locating Nearby Copies of Replicated Internet Servers*. ACM SIGCOMM'95, in: Computer Communication Review, 25 : 4, October 1995, ACM SIGCOMM
- [12] J. Gwertzman, M. Seltzer: *The Case for Geographical Push-Caching*. Proc. 5th Conf. on Hot Topics in Operating Systems (HotOS'95), Orcas Island (WA), USA, May 1995
- [13] V. Jacobsen: *traceroute*. December 1988. Documentation and software available from URL: <ftp://ftp.ee.lbl.gov/pub/traceroute.tar.Z>
- [14] C. Lagoze, D. Fielding, S. Payette: *Making Global Digital Libraries Work: Collection Services, Connectivity Regions, and Collection Views*. Proc. 3rd ACM Conf. on Digital Libraries (DL'98), Pittsburgh (PA), USA, June 23-26, 1998, ACM Press
- [15] K. Moore: *SONAR - A Network Proximity Service*. Internet-Draft, August 1998, URL: <ftp://ftp.isi.edu/internet-drafts/draft-moore-sonar-03.txt>

- [16] M. Rabinovich, I. Rabinovich, R. Rajaraman, A. Aggarwal: *A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service*. Proc. 19th IEEE Int. Conf. on Distributed Computing Systems, Austin (TX), USA, May 31–June 5, 1999, M. Gouda (Ed.), IEEE Computer Society, pp. 101–113
- [17] Y. Rekhter, T. Li: *A Border Gateway Protocol 4 (BGP-4)*. RFC 1771, March 1995
- [18] W. Theilmann, K. Rothermel: *Efficient Dissemination of Mobile Agents*. Proc. 19th IEEE Int. Conf. on Distributed Computing Systems Workshop on Web Based Applications, Austin (TX), USA, May 31–June 5, 1999, W. Sun et al. (Eds.), IEEE Press, pp. 9–14
- [19] W. Theilmann, K. Rothermel: *Disseminating Mobile Agents for Distributed Information Filtering*. Proc. Joint Symposium ASA/MA'99 of 1st Int. Symp. on Agent Systems and Applications (ASA'99) and 3rd Int. Symp. on Mobile Agents (MA'99), Palm Springs (CA), USA, October 3–6, 1999, IEEE Press (to appear)
- [20] M. van Steen, P. Homburg, A.S. Tanenbaum: *The Architectural Design of Globe: A Wide-Area Distributed System*. Internal report IR-422, Vrije University, Netherlands, March 1997