
**Sponsored by**
IEEE Communications Society
(Technical Committee on Computer Communications and
Technical Committee on Personal Communications)
ACM SigMobile
Create-Net
ICST

IEEE COMPUTER SOCIETY

IEEE

Los Alamitos, California

Washington • Brussels • Tokyo

# *Grido*- An Architecture for a Grid-based Overlay Network

Shirshanka Das, Alok Nandan, Michael G. Parker, Giovanni Pau and Mario Gerla
Computer Science Department
University of California Los Angeles
Los Angeles, CA 90095-1596
{shanky,alok,mgp,gpau,gerla}@cs.ucla.edu
*

## Abstract

*Grido is an architecture that targets a network operator intending to provide enhanced services to its customers. This is achieved by setting up a "backbone" overlay network. A backbone overlay is a set of Internet hosts dedicated to providing overlay services. A network operator can view Grido as a sandbox for rapid prototyping and market adoption assessment of novel services. In the past, overlay networks have been designed to mitigate deployment issues of functionalities such as multicast and QoS at the network layer. Grido provides a WS-Agreement based negotiation interface complying with the current Global Grid Forum (GGF) standards. We propose to use a novel virtual coordinates-assisted overlay construction and maintenance protocol.*

*We demonstrate using simulations, that Grido incurs a low latency overhead while maintaining sparse connectivity on the backbone overlay. Grido also incurs low overhead for virtual coordinates estimation and chooses the closest 5% overlay node to any IP address, 95% of the time.*

## 1 INTRODUCTION

Overlay network structures have been proposed to deploy new functionalities in Internet, such as multicasting [3], and to provide better services, such as fault-tolerance and QoS-aware routing to existing applications.

Large-scale deployment of overlay networks have been in the application-specific overlay domain. Application-specific overlays such as those for supporting multicasting, implement the functionalities exclusively at the end users, and thus do not require infrastructure support from intermediate nodes such as routers and proxies. However, there are serious scalability issues with increased adoption and usage of application specific overlays.

The alternative approach which alleviates the scalability problem are *backbone* overlays. Backbone *multicast* overlays for example, can be a set of strategically deployed proxy nodes that self-organize into an overlay network and deliver data packets on multicast distribution trees built on top of this overlay. End users subscribe to appropriate overlay nodes and receive data via unicast or local IP multicast. Hence, the communication overhead to maintain control and data delivery path is limited in scope.

The "backbone" overlay concept aims to share information between multiple overlays and consequently amortize the overlay maintenance across multiple overlays. Thus, the backbone overlay approach can fill the gap in the *present* by providing enhanced services.

The key contributions of our paper are as follows: (a) A novel overlay construction strategy that uses virtual coordinates. The joint use of Q-OSPF and virtual coordinates enables a fast and efficient overlay construction strategy for providing QoS support. (b) We devise an adaptive strategy on the overlay that mimics the power-law like topology of the underlying Internet to achieve scalability. (c) We also provide a scalable solution to determine the closest overlay node to a given Internet host with minimal probing overhead. (d) Grido uses a Grid-based interface for third-parties to negotiate and get a desired service level from the overlay service provider. We envision this standardized interface to be used between inter-provider agreement settlement as well.

The next few sections are organized as follows. Section 2 deals with our approach to construct and maintain the overlay in a distributed manner. In Section 3, we discuss the problem of isolating the best overlay node for any Internet host and detail our approach to set up routing within the overlay. In Section 4, we evaluate our construction strategy

*This work is partially supported by the Italian Ministry for Research via the ICTP/E-Grid Initiative, the National Science Foundation through grants CNS-0435515/ANI-0221528, and the UC-Micro Grant MICRO 04-05 private sponsor STMicroelectronics.*

and evaluate its performance with respect to certain well accepted parameters for overlay evaluation. Finally, Section 5 concludes the paper.

For brevity, we skip discussion about a possible QoS Call Admission Control (CAC) mechanism that can be used as a back-end for the Grid interface. We derive from our previous research [9] and take a network layer measurement based CAC algorithm and propose to use it at the overlay layer. We also skip the discussion about the details of the Grid interfaces that the overlay exposes for inspection and session set-up. We use a Grid standards (WSRF) compliant agreement protocol based mechanism for customer-provider interaction. The interested reader is referred to our technical report [12] for details on these issues.

## 2 Backbone Construction

Narada [3] addressed the problem of maintaining a mesh of end-systems by estimating latency between hosts and adding or dropping links based on the utility of the links. However estimating the utility of potential neighbors on the basis of ping results implies regular probing of nodes on the network, an activity that is not low in overhead. Virtual coordinates allow nodes to estimate latency in a distributed manner without the intensive overhead of regular pinging. We take an approach that is similar to Narada, but we use virtual coordinates to reduce the overhead involved in determining latency. We briefly cover the spectrum of virtual coordinate systems and discuss our use of Q-OSPF, a link-state protocol for disseminating QoS metrics and virtual coordinates in the next few paragraphs. For a detailed description of related work in this area, please refer to our technical report [12].

### 2.1 Background

There are several virtual coordinates systems proposed in literature such as GNP [10], IDMaps and PIC. A recent idea, Vivaldi [4] is an algorithm for assigning virtual coordinates to hosts such that the distance between the coordinates of two hosts accurately predicts the round trip time, or RTT, between them. Unlike previous methods to allow estimation of latency a priori, Vivaldi is distributed among participating hosts and thus scales efficiently as the size of the network grows.

Q-OSPF [2] is an extension to the standard OSPF protocol, a link-state routing protocol that is commonly run within an AS. Q-OSPF piggy-backs the QoS parameters of the links along with the link state advertisements, so that routers can make intelligent decisions while forwarding. In our architecture, we use Q-OSPF merely as a information-propagation protocol and that it runs at the application layer for our overlay nodes. The link-state advertisements correspond to overlay links which are themselves composed of multiple physical links. Thus the QoS parameters for a particular overlay link correspond to the measured metrics over the IP route between the two overlay neighbors. Each node in the overlay network builds a link state database which contains all the recent link state advertisements from other nodes. Routing decisions are made at the higher layers and the forwarding mechanism is set up based on these decisions.

### 2.2 Embedding the overlay on the coordinate system

The backbone overlay nodes use Vivaldi to embed themselves in a virtual coordinate system. While constructing the overlay (deciding which overlay links to keep and which to prune), we need to know the virtual coordinates of other nodes. Therefore we use Q-OSPF to propagate useful information about links, and piggyback the virtual coordinates of the overlay nodes in the advertisements that Q-OSPF generates. Other attributes that are propagated using Q-OSPF advertisements are discussed in [12]. Each overlay node maintains a list of overlay nodes that it sees the least distances to (in terms of virtual coordinates). Out of this set of nodes, the node decides which nodes to maintain overlay links with using a Narada-like mechanism. To encourage diversity and prevent network partitioning, each node also maintains overlay links with one or two nodes which are relatively far away. The overlay links that are constructed are used to forward overlay traffic as well as link state advertisements. The backbone overlay nodes ping their overlay neighbors to estimate virtual coordinates. In case, an overlay node does not have twelve neighbors, it will ping random nodes on the overlay to make up the remaining number. These ping messages are tunneled through the shortest path on the overlay.The virtual coordinates pings travel on the overlay links, and therefore mirror the structure of the overlay. [5] contends that Vivaldi technique suffers from small oscillations which move the coordinates indefinitely and stop it from converging if the nodes do not satisfy the triangle inequality. However, we realized that in the Vivaldi code, which is part of the Chord source code, you move a distance $\frac{local\_error^2}{(local\_error^2+remote\_error^2)}$ as opposed to $\frac{local\_error}{(local\_error+remote\_error)}$ as claimed in [4]. We evaluated a simple three-node example with triangle inequalities and found that the squaring leads to better stability and the pathological three node case does not keep changing coordinates indefinitely. However it is true that any coordinate system that tries to place nodes in Euclidean space will have to create errors in the actual estimates in order to satisfy the triangle inequality. This necessarily implies that using virtual coordinates to identify cases where a certain link $i, j$

is longer than a path $i, k, j$ is difficult if not impossible. On overlays, that is one of the most critical things that you need to know, whether its quicker to get to a certain node through the overlay or directly through the Internet.

### 2.3 Topology Construction Using Synthetic Coordinates

We assume that the overlay initially consists of a set of nodes, all of which know each others identities (IP addresses), say through a central administrator. For bootstrapping, the nodes start pinging each other on virtual coordinates over the Internet until the prediction error stabilizes. At this point, the nodes locally execute a $k$-means clustering [8] algorithm on the virtual coordinates and associate each node to a cluster. The value of $k$ (i.e. the number of clusters) is a system parameter and has been experimentally evaluated in the Section 4. Nodes that are deemed to be in the same cluster are called $CloseNodes$ and nodes that are in other clusters are called $FarNodes$. Once this is done, each node bootstraps itself with $ShortDegreeMin$ random nodes in its cluster and $LongDegreeMin$ nodes outside its cluster.

Each overlay node has a window of the number of neighbors (both close and far) that it wants to have. It is not desirable to add links between overlay nodes that are too close, since that would lead to redundant overlay hops. Thus the system has a parameter called $minOverlayDistance$ which is the minimum distance on the virtual coordinate space between two nodes for them to be potential neighbors on the overlay. Refer to Figure 1 to see a sample of the system parameters used in the construction.

After the bootstrap phase, edges have been added to the topology. The nodes now go into maintenance phase, where they try to improve the mesh quality. During the maintenance phase, pings for estimating the virtual coordinates start travelling on the overlay edges. Thus, the virtual coordinates distance mirrors the overlay latency of the nodes. In the next subsection we describe the exact mechanism for addition and deletion of links to the overlay network.

### 2.4 Addition of Links

Every member receives link-state updates which inform it about other members' virtual coordinates. Based on these coordinates a particular node $i$ calculates its virtual coordinates distance to the nodes.

Each node $i$ estimates the real latencies between $i$ and a randomly picked $CloseNode\ j$ by pinging it. This pinging process is a periodic process that looks at the $CloseNodes$ table and pings a randomly chosen node. These real latencies are exponentially averaged, and once they are sufficiently old (5-10 samples spread out over a minute for example) they are compared with the shortest distance on the virtual coordinate space between $i\ and\ j$ on the graph using the overlay topology. The ratio of Internet latency to virtual distance is computed, and the link is considered a candidate for addition if this ratio is better than the worst ratio in the current neighbors list. The probability of choosing to add this particular link is inversely proportional to the number of times that the link has been added previously. Links have a high "added" count only if they've been deleted often. Therefore, making the probability inversely proportional to the number of times that the link has been added inhibits addition of links which have been deleted often (to prevent another potential deletion). Addition of short links happens freely until the threshold $ShortDegreeMax$ is reached. After this, a short link is added only if its Internet to virtual coordinates distance ratio is better than the best ratio among the current neighbors.

We experiment with an adaptive strategy in which a node increases its $ShortDegreeMax$ and $LongDegreeMax$, if it perceives its *Relative Delay Penalty* (RDP) to be greater than $\alpha$, a system specific threshold. Thus the number of neighbors that the node can have is increased based on the nodes local view of the RDP that it experiences. The RDP represents the suboptimality of the overlay network compared to the underlying topology. Adaptation based on RDP intuitively allows overlay nodes near routers with high degree, to attain higher degree in terms of their overlay edges.

RDP is a well-known metric for evaluating overlay topologies. The RDP between two overlay nodes is defined as the ratio of the overlay latency and the Internet latency between the two nodes. To the best of our knowledge, the exact policy that is assumed while calculating the latencies has not been well-specified. In our evaluations of Grido, we use the shortest delay path on the overlay (since nodes have Q-OSPF driven knowledge of the link delays on the topology) for calculating the overlay latency. The Internet latency is calculated assuming Min-Hop routing where ties are broken on the basis of shortest delay. The RDP of a topology is calculated by taking the 95% value of RDP for the sampled node pairs.

### 2.5 Deletion of links

A *least recently used* (LRU) cache of the active overlay links is maintained. Every time a link is used as part of a path set-up, the nodes on each side of the link update the overlay link time-stamp in their respective tables and place it at the top, thus marking it as freshly used. Similarly, when a link is just added, it is also marked as freshly used. When the threshold $ShortDegreeMax$ is crossed, the link which has been least recently used is dropped with a certain probability. This probability is inversely proportional to the number of times the link has been added. This leads to giving

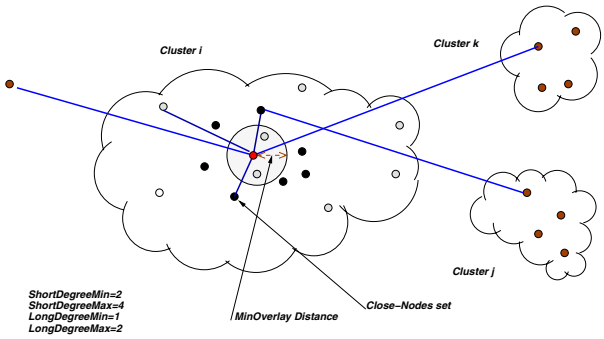


**Figure 1. A node, shown in red, selecting its overlay neighbors. Here it is constrained to the parameters as shown. Initially, each node starts out with $ShortDegreeMin = k - 1$, $ShortDegreeMax = 2(k - 1)$; $LongDegreeMin = 1$ and $LongDegreeMax = 2$.**

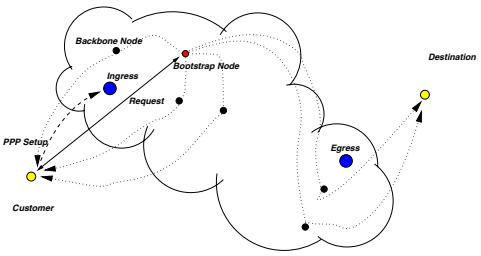


**Figure 2. The $k$-pinger algorithm: The customer contacts its bootstrap node, who dispatches 3 pinging nodes. From their RTTs its virtual coordinates are calculated, and the shown ingress is found as the closest node.**

links that have been added often more time to prove their utility. If the link is not chosen for deletion this time, the threshold remains unchanged, and the node has more short neighbors than $ShortDegreeMax$ for the time being. The next time a table modify operation occurs, the process of trying to delete a link will repeat. Thus links that are added often (since they show an apparent gain in the latency inspite of being deleted often) are given more time to prove their utility. This helps in dampening oscillations of both good links which do not seem useful in the short run as well as bad links which just seem attractive in the short run but are not useful on the overlay. Long degree links are also handled in the same way, except that the initial thresholds are lower as mentioned earlier.

## 3 Overlay Routing

In most of the backbone overlay solutions proposed, all the end hosts involved in the exchange need to subscribe to the overlay, or pick the closest overlay node to them. In other cases, the overlay uses DNS records to try to resolve the same hostname to the closest server, depending on where the request originated.

In this paper we address simple unicast scenarios where only one of the participants needs to subscribe to the overlay service. Thus, we have situations such as, User A subscribes to Grido to get enhanced services when A is watching a stream from MovieStream Inc. Grido has to determine the path to use for routing the traffic for this session on the overlay. This implies that the overlay nodes have to figure out the best ingress and egress nodes in the overlay for routing from the source to the destination IP.

### 3.1 Routing through the Overlay

Pseudo-code 1 shows the steps involved in deciding on the ingress and egress nodes.

**Pseudo-code 1 This pseudo-code describes the events that ensue when a customer wants to use the backbone overlay service. If the closest ingress node is cached in the DHT, we use that; otherwise we attempt to "geo-locate" the customer node by computing its virtual coordinates. For this purpose, we first select $k$ backbone nodes that ping the customer, as shown in Fig 2. After this is done, we pick a backbone node with the shortest virtual distance to the customer node.**

```
FindClosestNode(IPAddress){
  DnsIP = FindDnsServer(IPAddress);
  ClosestNode = DhtLookup(DnsIP);
  If(ClosestNode eq ∅){
    OverlayPingers = Select_K_Pingers();
    Coords = CalculateVivaldi(OverlayPingers, DnsIP);
    Guardian = FindClosestOverlayNode(Coords);
    DhtInsert(DnsIp, Guardian);
    }
  return(Guardian);
  }

ComputePaths(SourceIP, DestIP, QoSParams){
  Ingress = FindClosestNode(SourceIP);
  Egress = FindClosestNode(DestIP);
  Paths = QoSFeasiblePaths(Ingress, Egress, QoSParams);
  return(Paths)
  }
```

We adopt the idea first proposed in [6] that the RTTs to the local DNS server for a particular host gives you a reasonable approximation about the RTTs to the host itself. Also DNS servers typically respond to pings (individual hosts may be firewalled), and keeping state regarding DNS servers instead of the actual hosts reduces the total state overhead. Each overlay node maintains a set of DNS servers to which

it maintains virtual coordinates with. The overlay node is called the Guardian for the DNS servers that it is responsible for. This information is stored in a Distributed Hash Table (DHT) composed of the backbone overlay nodes. Given the IP address of a DNS server, a lookup in the DHT results in information about its Guardian. In our architecture the Guardian is the closest node in terms of virtual distance for the particular IP address. If the DHT lookup for the DNS server results in a failure, $k$ overlay nodes are pressed into action to ping the DNS server and return back the RTTs that they experience. Keeping the information organized on the basis of the DNS servers in a DHT leverages two aspects of design: First, executing $k$-pings everytime a client wants to route through the overlay is not scalable, because it creates a lot of traffic and hammers too aggressively at the DNS servers. Second, DNS servers are typically well-connected and their distance to the overlay is not likely to change very often. Thus, the Guardian infrequently pings the DNS servers that it is responsible for, to ensure liveness.

Since the DNS server which is being pinged will not typically be running a virtual coordinate system, but just responding to the pings, the overlay nodes which ping it must exchange the RTTs that they experience and fix the DNS server in the virtual space as described in Pseudo-code 2. Given the coordinates, the closest overlay node to those coordinates is identified and assigned to be the Guardian for that DNS server, and this information is inserted in the DHT.

**Pseudo-code 2** **Centralized $k$-pinger virtual coordinate calculation**[1]

```
//Input: latency matrix and initial coordinates
//Output: accurate coordinates in x
compute_coordinates(L, x)
while(error(L, x) > tolerance)
  F = 0
  foreach j
    // compute error force of this spring
    e = L_{i,j} − ‖ x_i − x_j ‖
    //Add the force vector of this spring to the total force
    F = F + e × u(x_i − x_j)
    //move a small step in the direction of the force
    //x_i = x_i + t × F
```

# 4 Simulations

In this section we evaluate some of the features of our architecture using simulation. The primary goal is to evaluate our protocol with respect to some key parameters that might affect the overall performance of the system.

[1]Compare with the original Centralized Vivaldi algorithm the only thing that changes is that we the ingress node that runs this $k$-Pinger to "Geo-locate" a new node in the virtual Coordinate space does not need to iterate over all $i$ in the original algorithm

a) *Latency Overhead*: To achieve the goal of providing quality of service, constructing a overlay network that can achieve the desired levels of QoS is the primary goal. We evaluate the performance of our overlay network construction by computing the Relative Delay Penalty (RDP) metric as defined previously.

b) *Stability*: Overlay construction mechanisms and virtual coordinate systems that run on the overlay can lead to *unstable* feedback loops in which both the systems are trying to adjust to each others view. It is important to measure whether the overlay construction strategy is stable, and whether the virtual coordinate system exhibits low error estimates when it is run on top of the overlay. This assumes further importance in a backbone overlay network, since an unstable backbone can potentially affect more overlay sessions than an application specific overlay. We evaluate the edge churn rate in terms of edges added and deleted at each sampling interval and the error estimate of our virtual coordinates system to measure the stability of our system.

## 4.1 Simulation Setup

We have written a custom event driven simulator in C++ to simulate our protocol. We have not simulated all dynamic network conditions like queueing delay, packet losses, congestion. We use the Brite topology generator [1] to generate the network topologies used in our simulations. The different topology construction methods are run in parallel, so that they see the same chain of events during a particular run.

We use the Barabasi, preferential connectivity model to obtain graphs that more closely resemble the Internet hierarchy compared to a pure random graph. Unless otherwise specified, a topology of 10,000 nodes, 100 routers in each of the 100 AS'es is used for the simulations. Latencies to links in the physical topology are assigned by the topology generator. We've experimented with different fractions from 1% to 10% of the underlying topology to form the overlay network. The frequency of the Vivaldi pings is 1 ping per node per second. Thus the overlay as a whole generates about $n$ pings per second where $n$ is the number of nodes in the overlay. We used the default delay values generated by Brite and did not introduce triangle inequality violations in our evaluation. Triangle inequality violations (TIVs) are an important current area of research and patterns are just emerging with regard to the percentage of TIVs in the research networks [11]. We plan to address this issue in the near future as clearer results and consensus emerges about a feasible model for simulating TIVs.

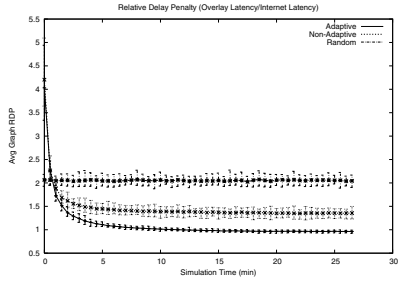


**Figure 3.** **Comparison of the Relative Delay Penalty. VC dim=5, K-means $k = 7$. 1,000 routers, 100 overlay nodes.**

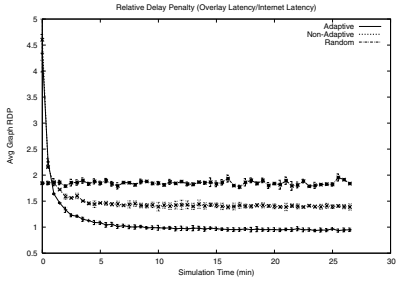


**Figure 4.** **Comparison of the Relative Delay Penalty. VC dim=5, K-means $k = 7$. 10,000 routers, 100 overlay nodes.**

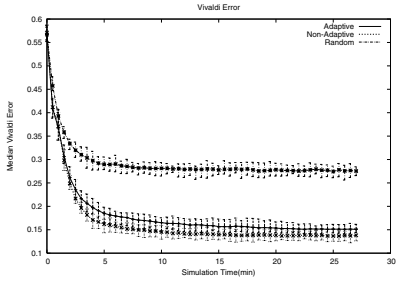


**Figure 5.** **Comparison of Vivaldi error. VC dim=5, K-means $k = 7$. 10,000 routers, 100 overlay nodes.**

## 4.2 Latency Overhead

We measure the RDP of a particular overlay topology by picking $10 * n$ random overlay node pairs and comparing the ping RTTs on the overlay and on the underlying graph. In the results presented, the overlay pings are tunneled on the shortest latency path on the overlay, whereas the underlying network pings travel on the default IP min-hop path[2]. The 95% (percentile) RDP for these pairs is used to calculate the final RDP for that topology. We evaluated the performance of the adaptive neighbor selection algorithm where the overlay nodes adaptively increase the initial values of $LongDegreeMax$ and $ShortDegreeMax$ to make the overlay more "dense" in order to achieve the desired amount of RDP. In our simulations, nodes adaptively increase their degree if they see their local RDP estimate to be higher than 1.5. The other approaches we evaluated are *"Random"*,where the neighbors are randomly selected, and *"Non-Adaptive"*, where the maximum and minimum node degree is merely a system parameter and overlay nodes do not adjust their degree based on the instantaneous RDP. Based on the averages of a 100 independent runs over an underlying topology of a 1,000 nodes and a 100 overlay node network, we observe in Figure 3 that *Adaptive* works better than the *Non-Adaptive* and *Random*. *Random* stabilizes at a sub-optimal level of an RDP value of $\approx 2$ with the same degree bound as *Non-Adaptive*.

[7] simulates Narada and Kudos to conclude that with an underlying topology of 3600 nodes and 200 overlay nodes, Narada achieves an RDP of 4, while their enhanced Narada achieves an RDP of 2.9. This actually remains the best that they achieve in the paper over various topologies.

Figures 3 and 4 show RDP values over different underlying topology and overlay node fractions. Over the wide range of values experimented with, we found that our algorithm leads to significantly lower latency overhead. Our protocol achieves RDP of 1.1 because it is a combination of good neighbor choices, and the capability for better path selection due to the OSPF driven QoS information propagation. The adaptive scheme adds more edges if it perceives the RDP to be high. Thus, there is a tradeoff between the "denseness" of the overlay graph and the RDP that we can achieve on it. "Denseness" can be defined as the ratio of the number of edges in the overlay to the maximum possible edges(which for an $n$-node graph is $n(n-1)/2$). We measured the denseness of the graph during our runs and observed values around 0.15, so for example if there are 100 nodes, this implies there are around 700 edges or on the average 3.5 edges/node. We believe the "denseness" of the graph is an important feature of an overlay network because it signifies the tradeoff between better RDP and more overhead. Using virtual coordinates allows us to minimize one aspect of this overhead. For instance, brute force pinging in a complete graph is $O(n^2)$, whereas using a virtual coordinates system, we can reduce the frequency and number of pings, and scale linearly as $O(kn)$ where $k$ is a constant[3] number of nodes pinged at random by each node. Results for other relevant parameters like link stress, and similarity to the underlying topology have been omitted for brevity and can be obtained in [12].

## 4.3 Stability

We evaluate two measures of stability of the overlay network setup algorithms, namely, virtual coordinates stabilization and the churn of edges in the overlay network.

[2]In the conclusions, we also discuss the results obtained by other methods of calculating RDP by permuting the different combinations of Min-Hop and Shortest Delay with the overlay latency and the Internet latency.

[3]a good value of $k$ is 12 as pointed out by Dabek *et al* [4] to yield best stability.

### Virtual Coordinates Stability

One of the measures of the stability of the system is the virtual coordinates computed by the Vivaldi algorithm. Figure 5 illustrates that the error stabilizes to around 0.15 for both non-adaptive and adaptive schemes. The random scheme stabilizes at 0.35. We compare this result to a run where we just have the overlay nodes pinging each other over the Internet paths (not the overlay links) and using Vivaldi. In this case, we find that the error levels off at around 0.12. Thus our overlay construction strategies do not impact Vivaldi's accuracy to any significant degree.

### Edge Churn

We evaluated the edge churn of the overlay network in the various clustering schemes. We observe that the adaptive strategy undergoes slightly more churn in terms of edges added and deleted. This is understandable since the overlay network is being optimized in a more aggressive manner, and is the tradeoff of obtaining a better overlay network. However, the number of edges added to the global overlay graph per sampling instance (every 3 mins) finally settles down to a nominal number of around 5. This is reasonable considering an overlay network of $\approx$ 700 edges. The churn in edges deleted is also comparable for adaptive and non-adaptive strategies.

## 4.4 Overlay Routing Simulations

We now discuss how well we can find, using $k$-pinging, the closest node in a cluster to some outside node. To determine our $k$-pingers, we randomly selected with replacement $g$ groups of size $k$ from within the cluster. To determine which of the $g$ groups will be our group of $k$-pingers, we used three heuristics: First, we choose the group which minimizes the maximum Vivaldi error. Second, we used the group whose minimum pair-wise distance was maximized. Third, we ranked groups independently by the first and second heuristics, and used the group that had the best joint ranking. Such a group should have a good balance of low-error nodes that are far apart from one another. Each $k$-pinger in the selected group pings the outside node, yielding a vector of $k$ round-trip times to the outside node. We then run the algorithm presented in Pseudo-Code 2 over $m$ iterations, thereby simulating each node in the group pinging the outside node $m$ times (without the overhead of repeatedly pinging), and noting the same RTT each sample.

In our simulations, our cluster size was 100, and the number of outside nodes we generated was 1000. To model the RTTs we used the King Data set consisting of a full matrix of pair-wise RTTs between 1740 DNS servers, collected by using the King method [6]. Before any k-pinging of nodes outside the cluster begins, each node in the cluster goes through 100 rounds of pinging 10 randomly selected nodes inside the cluster to establish its own Vivaldi coordinates. Note that we do not model nodes joining or leaving the overlay during this time, because we expect the nodes inside the overlay to be stable. Then, when an outside node is generated and pinged, using the real pair-wise RTT data in the King data set we order all nodes in the cluster by increasing RTTs to the outside node. We then find the rank of the node we predicted was closest. In an ideal scheme, the rank of the predicted node would always be one, meaning the node we predicted was closest was really the node with the smallest RTT.

In our experiments, we found that increasing $m$ beyond 10 did not give us significant benefits in terms of prediction accuracy. Figure 8 shows the results with $k = 7$, $g = 5$, $m = 10$ for the three heuristics. From it we can gather that it is important to find nodes with a good balance of pair-wise distance and low-error before pinging and establishing the outside node's coordinates. In our trials, the rank of the predicted closest node is equal to one approximately 60% of the time, and we consistently choose a closest node in the top 5% approximately 95% of the time. It will be interesting to investigate in detail the interactions of the parameters $k$, $g$, and $m$.

# 5 Conclusions and Future Work

In this paper we presented an architecture for a Grid-based backbone overlay network. One of our main contributions is the usage of virtual coordinates to construct and maintain the backbone overlay mesh. We evaluated various strategies for topology construction and maintenance. We also proposed an accurate prediction mechanism for identifying the closest overlay node to any given Internet host. Our key conclusions are:

a) We observed that our protocol gives RDPs of around 1.1 compared to earlier reported results of 2.9 for Narada-like protocols. However, as we pointed out in Section 2.4, the exact semantics of overlay latency and Internet latency have not been unambiguously specified across the existing literature on this topic. Therefore, we have evaluated our strategies using all possible interpretations of overlay latency and Internet latency (Shortest Delay and Min-Hop Shortest Delay). Our evaluations show, that Adaptive still achieves the desired RDP value, at the cost of higher "denseness". Non-Adaptive is not sensitive to the desired RDP value, and therefore retains the same "denseness" while achieving a worst-case RDP value of 1.8. It would be relevant to point out the worst-case RDP value is attained when the overlay latency is computed as Min-Hop Shortest Delay and the Internet latency is computed as Shortest Delay.

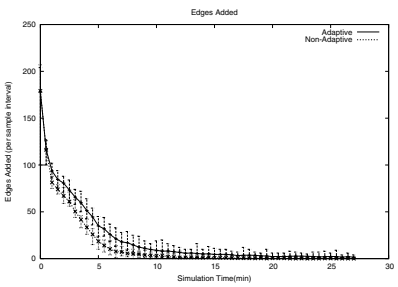


**Figure 6.** **Edges added per sampling instance**

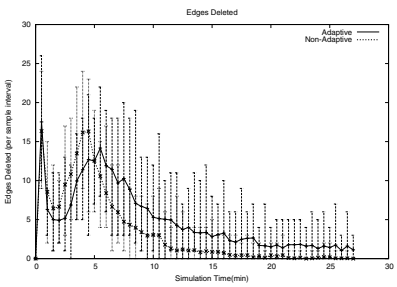


**Figure 7.** **Edges deleted per sampling instance**

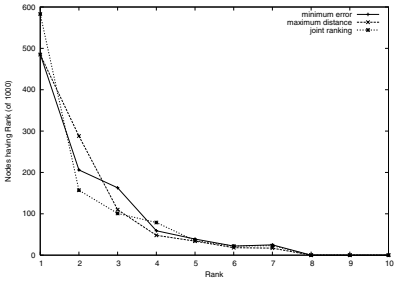


**Figure 8.** **The effect of different heuristics on the rank of 1000 nodes.**

b) Vivaldi assisted overlays are more scalable and retain desirable optimality characteristics compared to standard overlays. The dimension of the virtual coordinate space does not have much impact beyond a certain number. In our case, we found that beyond 5, the dimension of the space has negligible impact of the performance of the overlay network.

c) It is possible to identify the closest overlay node to a given Internet host with high accuracy and low overhead. Active probing with 7 overlay nodes predicts the 5% closest overlay node 95% of the time. This is among a 100 backbone overlay nodes and with a set of 1000 Internet hosts. Our approach of caching this information in a DHT indexed by the DNS server will reduce the control overhead when future lookups map to the same DNS server.

d) Using a GGF standards compliant agreement structure (WS-Agreement) aims at providing a common standard interface for automating complex customer-provider and provider-provider negotiations of quality of service parameters.

## References


[1] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In Proc. MASCOTS '01, Cincinnati, Ohio, August 2001.

[2] George Apostolopoulos, Sanjay Kamat, and Roch Guerin. Implementation and Performance Measurements of QoS Routing Extensions to OSPF. New York, March 1999.

[3] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case For End System Multicast. In Proceedings of ACM SIGMETRICS, Santa Clara,CA, June 2000.

[4] F. Dabek, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In Proceedings of ACM SIGCOMM'04, Portland, Oregon, USA (2004)

[5] C. de Launois, S. Uhlig, and O. Bonaventure. Scalable Route Selection for IPv6 Multihomed Sites. To appear in Networking'05, Ontario, Canada, May 2005.

[6] K. Gummadi, S. Saroiu and S. Gribble. King: estimating Latency between Arbitrary Internet End Hosts. In Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002).

[7] S. Jain, R. Mahajan, D. Wetherall and G. Borriello, Scalable Self-Organizing Overlays UW-CSE TR 02-06-04

[8] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, An efficient k-means clustering algorithm: Analysis and implementation, IEEE Trans. Pattern Analysis and Machine Intelligence, 24 (2002), 881-892.

[9] S.S. Lee, S. Das, K. Yamada, H. Yu, G. Pau and M. Gerla, Practical QoS Network System with Fault Tolerance Computer Communications Journal 2003, Volume 25, Number 11-12: Advances in Performance Evaluation of Computer and Telecommunications Networking

[10] T. S. E. Ng and H. Zhang, Predicting Internet network distance with coordinates-based approaches. In Proceedings of IEEE Infocom, pages 170179, 2002.

[11] H. Zheng, E.K. Lua, M. Pias and T. G. Griffin. Internet Routing Policies and Round-Trip-Times. , In Proc. PAM 2005, Boston, MA, USA, March 31 - April 1,2005

[12] S. Das, A. Nandan, M. G. Parker,G. Pau and M. Gerla. Grido-An Architecture for a Grid-based Overlay Network, UCLA CSD TR#050020