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 set of Internet hosts dedicated to providing overlay services. A network operator can view Grido as a sand-box 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.

I. INTRODUCTION

Overlay network structures have been proposed to deploy new functionalities in Internet, such as multicasting [10] [15], and to provide better services, such as fault-tolerance and QoS-aware routing [4] [23] 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 [15] 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. [8] suggest two approaches towards a synergistic co-existence of overlay networks and underlying IP networks.

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.

The first approach seeks to obviate the need for an overlay by augmenting the services of the underlay. This approach might eventually be deployed along with IPv6.

The second approach proposes to share information either between overlays and underlays or between multiple overlays. Sharing state or performance information across such boundaries would alleviate the lack of awareness on each side. One solution that takes this approach is [9] which proposes the use of a routing underlay. The "backbone" overlay concept aims to share information between multiple overlays and consequently amortize the overlay maintenance across multiple overlays. 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) We present 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 II deals with our approach to construct and maintain the overlay in a distributed manner. In Section III, 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 IV, we present 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 [22] and take a network layer measurement based CAC algorithm and propose to use it at the overlay layer. Section V outlines the Grid interfaces that the overlay exposes for inspection and session set-up. We present a Grid standards (WSRF) compliant agreement protocol based mechanism for customer-provider interaction. We elaborate on the use cases of our overlay architecture and talk about possible applications and economic incentives for its feasibility. In Section VI, we evaluate our construction strategy and evaluate its performance with respect to certain well accepted parameters for overlay evaluation. Finally, Section VIII concludes the paper and outlines the road map for the realization of this architecture on an Internet-wide testbed.

II. Backbone Construction

Given a set of overlay nodes, we want to construct the backbone overlay to have a well-connected mesh with "good" overlay links. The backbone mesh serves as the base for the overlays that may be constructed on top of it. Thus, if latency is the primary concern, it is important for the mesh to have links which are very low latency. Since the overlay in its most connected form can be a complete mesh, and a complete mesh may not scale in terms of the overhead involved in maintaining the structure, we would also like to constrain the overlay to not use all possible overlay links. For example, the message overhead of link state protocols like Q-OSPF is directly proportional to the number of edges. We conclude from our observations that maintaining a good backbone mesh that is not too densely connected is one of the crucial design goals.

Narada [10] 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.

A. Background

There are several virtual coordinates systems proposed in literature such as GNP [30], IDMaps [33] and PIC [34]. A recent idea, Vivaldi [11] 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. Furthermore, coordinates change quickly in reaction to adverse traffic patterns such as congestion.

Q-OSPF [7] 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.

B. 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 Section IV. 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.[1] However, 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. However 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.

C. 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). 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 [27] 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

[1] [13] contends that Vivaldi technique suffers from small oscillations which move the coordinates indefinitely and stop it from converging [12] 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 [11]. The squaring leads to better stability and the pathological three node case does not keep changing coordinates indefinitely.

evaluated in the Section VI. 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.

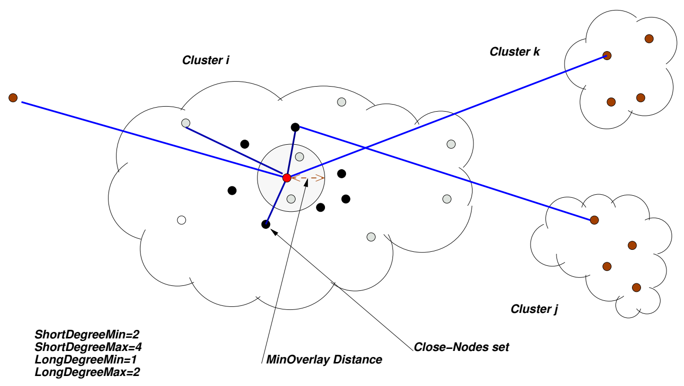


Fig. 1. **A node, shown in red, selecting its overlay neighbors. Here it is constrained to the parameters as shown. Initially, each node starts out with** $ShortDegreeMin = k - 1$**,** $ShortDegreeMax = 2(k - 1)$**;** $LongDegreeMin = 1$ **and** $LongDegreeMax = 2$**.**

D. 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 α. Thus the number of neighbors that the node can have is increased based on the nodes local view of the RDP that it experiences. This 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.

E. 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 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.

F. Overlay Management

We now summarize some related functions associated with managing the backbone, both in terms of the members belonging to the backbone overlay as well as the structure of the backbone. We are assuming in a backbone overlay architecture that nodes will not be joining and leaving very often, the only events we need to handle are node failures and mesh partitioning.

- *Mesh performance:* When an overlay node first joins the backbone, the overlay links it creates can be quite sub-optimal, however after a dozen pings to other nodes the node should be able to establish good coordinates. Nodes maintain overlay links with close neighbors with low latency and also a few links with neighbors slightly far away. This helps in getting better coordinates as well as preventing mesh partitioning. Nodes also drop their existing overlay links if they are being used very rarely.
- *Node failure:* In the event of a backbone node failure, neighbors of the backbone node detect it due to the absence of Q-OSPF keep alive messages. As a result, the failed node gets deleted from their routing tables and link state databases. Eventually, this node's entry will be purged from the databases of all other backbone nodes.
- *Mesh partitioning prevention:* To prevent mesh partitioning, each node maintains one or two overlay links to nodes which do not measure up as the closest nodes on the network. Thus, given the virtual coordinates of all the nodes, the node will try to maintain overlay links with one or two nodes which are relatively far away in the coordinate space. This allows nodes to maintain nodes across AS'es.
- *Mesh repair:* In the unlikely event of a mesh partitioning, nodes observe the timeouts in the link state database of an unusually high number of links. This leads to nodes probabilistically probing the nodes whose entries expired, leading to the healing of the mesh.

III. 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 [6].

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.

In the unicast scenario, the customer contacts a central server, say `Grido.com` and the overlay automatically figures out the best ingress node for the customer. The customer then executes a WS-Agreement handshake with the ingress node and the best egress node and the overlay path is determined by the QoS provisioning mechanism. Therefore both the source and the destination are totally oblivious to the structure of the backbone overlay and the identity and proximity of the overlay nodes. As long as they possess a valid IP address and know of one way of contacting the overlay, the overlay will optimize their experience. The next section deals with routing within the overlay once the route has been decided.

A. Intra-backbone Overlay Routing

As discussed earlier, the nodes on the backbone overlay run Vivaldi, thus after some time the nodes on the backbone overlay are relatively stable in terms of their synthetic coordinates. The nodes also maintain tunnel-ids which correspond to paths on the backbone overlay. Thus similar to label switching schemes, the overlay will route traffic within the overlay by encapsulating the packets with the tunnel-ids. Tunneling packets from one overlay node to the next is accomplished by setting up Generic Routing Encapsulation (GRE) tunnels. We also want to do source routing on the overlay. Therefore we do label switching similar to Multi-Protocol Label Switching (MPLS), on top of the GRE tunnels. We use globally unique labels, each node picks a monotonically increasing sequence number for the tunnel id and the `<ingress-overlay-node-ip,seq_no>` tuple is used to create the path. There is a tunnel set-up phase executed in a manner similar to RSVP, where the labels of the route are set up on the path. We do not need all the complexity of RSVP since we do not let downstream nodes decide for us whether they can support the overlay tunnel or not. Once created, the intra-overlay path is never broken down, thus future traffic through that path will not need a tunnel set-up phase at all.

B. 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($DnsIP$);
  $If(ClosestNode\ eq\ \emptyset)${
    $OverlayPingers$ = 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 [14] 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.

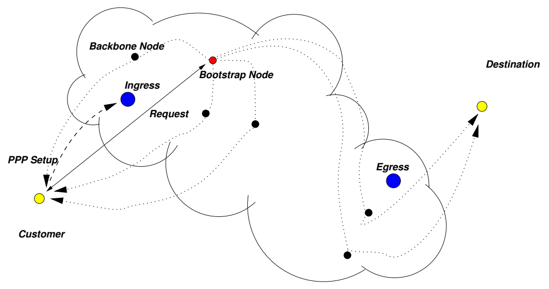


Fig. 2. **The k-pinger algorithm: The customer contacts its bootstrap node, who dispatches 3 pinging nodes. From their RTTs its virtual coordinates are calculated, and the shown ingress is found as the closest node.**

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**[2]

```
//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
    $F = F + e \times u(x_i - x_j)$
    //move a small step in the direction of the force
    //$x_i = x_i + t \times F$
```

Given the source IP and the destination IP, the overlay can compute the ingress and the egress node for each session. WS-Agreement negotiations are carried out with the ingress, which computes the best path to the egress taking into account the intra-backbone conditions. The routing is adhered to by encapsulating the packets of that session with the tunnel-id corresponding to the path that has been calculated. When the packets are finally decapsulated at the egress node, they are simply routed to the destination IP using regular IP routing. In case of multiple-paths, depending on the application, it may desire packet scattering in which packets are sent round robin over the candidate paths or mesh routing [26] in which multiple copies of the same packet are sent over the paths. The ingress node functions as a round-robin scheduler in the former case and a multiple packet copy generator for the latter case. The packets are either striped through the different tunnels or merely copied out onto multiple tunnels.

Bootstrapping the system: The system starts out with no knowledge of any DNS servers. As requests start coming in, the tables of the overlay nodes start filling up with DNS servers and their virtual coordinates. The DHT starts filling up with DNS servers and their Guardians. It is important to note that the information that is stored in the DHT is merely the identity of the Guardian for the DNS server. The virtual coordinates themselves are stored at the Guardian and are modified periodically by it based on the updated RTTs that it sees from the DNS server. Thus the virtual coordinates are being maintained in a two-tier process. At the top tier, the backbone overlay nodes maintain good coordinates on the overlay. At the lower tier, the overlay nodes are infrequently pinging DNS servers close to them and executing the k-pinger procedure to re-"place" the DNS server, if the delay changes significantly. Figure 3 gives a good overview of the whole system.

State Analysis

It takes $O(\log n)$ hops to do a DHT lookup (where n is the number of backbone overlay nodes) and figure out whether a particular DNS server is there in the system or not.

There are about 130K IP prefixes in the Internet as of now, out of which about 70K are unique. The logical organization of DNS servers and IP prefixes tend to mirror each other, so we

[2]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

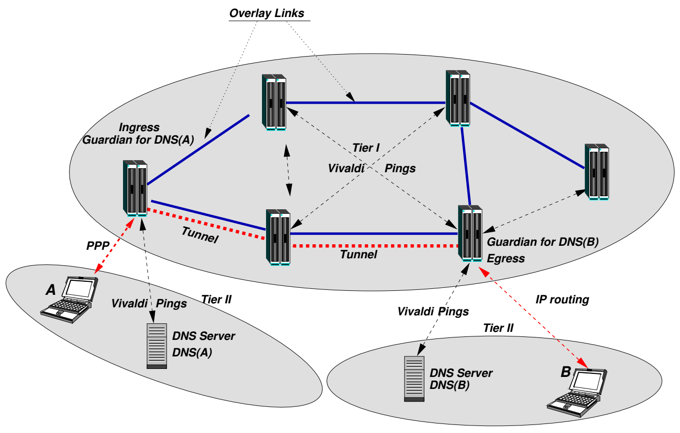


Fig. 3. **Overall architecture of the backbone overlay routing. Customer A wants to set up communication with Customer B. (1)Ingress and egress nodes on the overlay are discovered using the K-Pinger algorithm described earlier. (2) Point-to-Point Protocol (PPP) tunnel is established to the ingress, followed by the tunneled path on the backbone overlay and is finally routed to the customer B through the egress over the Internet.**

can assume that on an average if there are about 500 overlay nodes, each node will take care of about 260 DNS servers. This means that if the DHT is filled completely, each node will be a Guardian for about 260 DNS servers.

IV. QoS Provisioning on the Overlay

We derive from our previous research [22] and propose to use a measurement-based QoS Call Admission Control mechanism to provide feasible load balancing on the overlay. We contend that the Internet is overprovisioned (i.e. there is plenty of capacity), but is prone to hot spots (i.e. sections may become quite overloaded). This is in part because Internet routing cannot balance loads on alternate paths. Thus, one important advantage of the overlay lies in better routing (being able to figure out good paths) and better network utilization (being able to perform load balancing over the network to avoid hot spots). It would be interesting to evaluate whether these benefits can be provided by a simple measurement based approach such as the one proposed in [22]. This mechanism was originally developed to address QoS concerns at the network layer. Simulation results in the papers and practical implementation results in [28] show that the technique is light-weight and capable of providing enhanced single-path and multi-path provisioning for QoS-sensitive flows. The mechanism is purely measurement based, and therefore it is attractive to use it at the overlay layer only if we employ scalable and accurate measurement techniques for the metrics of interest. We are primarily concerned with measuring the capacity of overlay links, the latency of the links and having an estimate of the level of link utilization. The following subsections elaborate on the details of the solution.

A. Propagating QoS information

The overlay nodes will be running Q-OSPF at the application layer using the neighbor info in the tables to broadcast overlay link state info to their overlay neighbors. The link state advertisements will contain the following information.

- Capacity of the virtual link as reported by CapProbe [32].
- Load or Congestion level on the virtual link.
- Delay on the link as experienced by pings involved in CapProbe.
- Current Vivaldi coordinates of the node generating the advertisement.

B. Qos-Based Path Selection

When a WS-Agreement is being executed with a user, the overlay consults the underlying QoS routing algorithm (i.e., Q-OSPF with the enhanced routing algorithms in our case) for feasible paths. If no feasible path is found, the offer is rejected. Depending on the logic at the higher layer, this may lead to a new offer by the user, which may be accepted by the overlay if the metrics are satisfied. As discussed in [22], we propose to use two different QoS routing algorithms in our system; the conventional single path algorithm and a multiple path algorithm.

The single QoS path computation algorithm with multiple QoS constraints derives from the conventional Bellman-Ford algorithm as a breadth-first search algorithm minimizing the hop count and yet satisfying multiple QoS constraints.

The proposed multiple QoS path algorithm is a heuristic solution. We do not limit ourselves to strictly "path disjoint" solutions. Rather, the algorithm searches for multiple, maximally disjoint paths (i.e., with the least overlap among each other) such that the failure of a link in any of the paths will still leave (with high probability) one or more of the other paths operational. [5] has shown that with "blind" multi-path overlay routing, packet losses on "overlay-edge-disjoint" paths are often correlated. Therefore inferring topology characteristics from virtual coordinates would be very helpful, and is the subject of our ongoing research. The interested reader is referred to the detailed descriptions of the single and the multiple path algorithms in [22].

As a consequence of running the single-path or the multiple-path computation algorithms, we get one or a set of overlay paths. These paths are set up using the label set-up mechanism, and the encapsulation at the ingress is set up like we discussed in Section III.

The metrics of interest and the toolset that we plan to evaluate are the following

- *Capacity estimation:* CapProbe [32] is a capacity estimation technique that has proven to be very accurate and light-weight. CapProbe derives from the packet-pair estimation technique and sends packet trains of increasing size. It estimates the capacity by picking the samples which experience the minimum RTT since intuitively these samples experience the least queuing delay at intermediate nodes. We propose to use CapProbe to estimate the capacity of an overlay edge. Since the capacity of wired links are highly unlikely to change, CapProbe can be run very sparingly, further reducing the overhead involved.
- *Available bandwidth calculation:* Computing the "*exact*" load on a particular link is challenging and is often

of no real use, because by the time the data reaches the party concerned, the load may have changed. Thus it is important that the estimating technique should have low latency. Techniques such as Spruce [35] and Pathrate [29], [31] have proven effective in providing approximate estimates with reasonable latency. Our system implementation would incorporate an evaluation of how scalable these schemes are for a real overlay based implementation.

- *Latency calculation:* Latencies can be estimated from the RTTs observed during CapProbe pings and Spruce/Pathrate probes. Changes in the delays will be propagated through Q-OSPF only when they change significantly from the previous values.

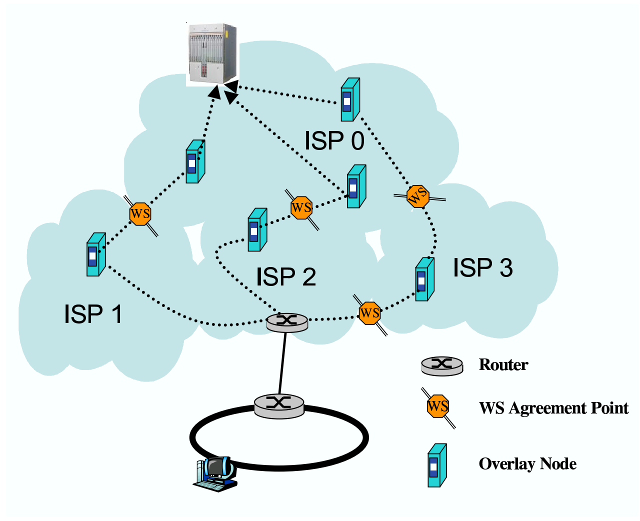
V. Grid interfaces for the backbone

The future of the Internet pricing architecture is likely to be based on a flat-fee service at the network layer complemented by a number of enhanced services at the application layer [39] [40] [41] [42]. For example, an ISP may provide a *virtualized* multi-homed service that uses the backbone overlay to deliver traffic while adhering to certain QoS metrics even in the event of BGP-failures (see figure 4.) Recent studies show that in terms of latency, standard overlay routing outperforms or equals any multi-homed BGP-based routing [38]. In particular, a virtualized multi-homed service can provide the same latency as physical multi-homing and would cost only a fraction of it. It is expected that an actual multi-homed domain is reliable to failures in the access link while a virtualized multi-homed is obviously not. Although in absolute terms, a multi-homed domain performs slightly better than a single backbone overlay in terms of throughput (2-12%) due to the completely independent access to the network [38], the two solutions are comparable in terms of cost. Finally, it is worth noting that all the current studies are based on RON/Narada technologies and they don't consider the increased scalability and optimality introduced with our scheme.

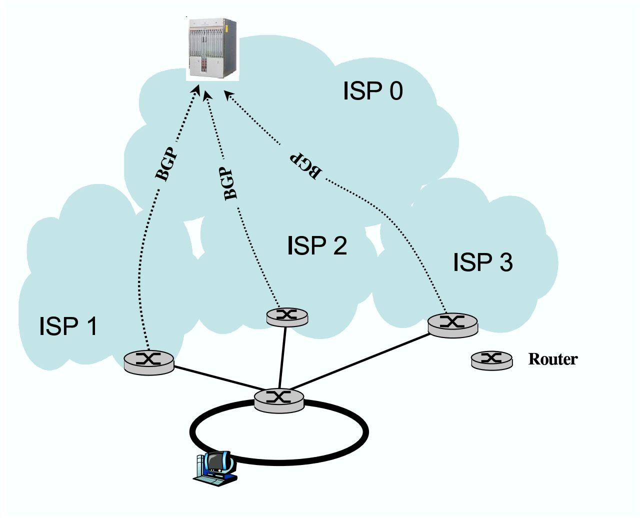
We envision that standardized Grid-based interfaces can become key enablers for seamless negotiation and usage of such services by enterprise customers. The overlay nodes on the backbone will have interfaces for applications to make WS-Agreement based negotiations with the ingress overlay node. Customers will be able to make queries of the form of "I need B-Mbps bandwidth and D-ms delay, with multi-path support going to final destination IP Address Y". Using WS-Agreement instead of any other specific structure, allows us to be compliant with the current GGF standards, thus making it easy for other "grid-enabled" overlays, resource managers and end systems to interact with the backbone seamlessly.

The objective of the WS-Agreement specification is to define a language and a protocol for advertising the *capabilities* of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime.

WS-Agreement is an XML language for specifying an agreement between a resource/service provider and a consumer, and a protocol for creation of an agreement using



(a) Virtual multi-homing service is provided by ISP 2 who can provide cheaper rates to customers since it negotiates bulk agreements with peer ISPs



(b) Multi-homing service through 3 different ISPs. The customer is required to pay 3 different access fees.

Fig. 4. **The figure shows two different solutions to provide reliability and enhanced performance: (a) shows our proposed schema in which the advanced services are supported by a single provider through the deployment of backbone overlays and WS-agreements; (b) shows a traditional multi-homed site, in which the customer can choose between different ISPs through different physical connections.**

agreement templates. The specification consists of three parts to be used in a composeable manner: a schema for specifying an agreement, a schema for specifying an agreement template, and a set of port types and operations for managing agreement life-cycle, including creation, termination, and monitoring of agreement states.

An agreement between a service consumer and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on the availability of resources and/or service qualities. For example, an agreement may provide assurances on the bounds on service response time and service availability. Alternatively, it may provide assurances on the availability of minimum resources such as bandwidth.

To obtain this assurance on service quality, the service consumer or an entity acting on its behalf must establish a service agreement with the service provider or another entity acting on behalf of the service provider. Because the service objectives relate to the definition of the service, the service definition must be part of the terms of the agreement or be established prior to agreement creation. The specification in the appendix provides a schema for defining overall structure for an agreement document for the overlay path set up. An agreement includes information on the agreement parties and references to prior agreements, one or more discipline specific service definition terms, and one or more guarantee terms specifying service level objectives and business values associated with these objectives.

VI. 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.

- *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.
- *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. This was also pointed out in [8]. 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.

A. 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 [3] 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 [2] 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.

B. 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[3]. 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 5 that *Adaptive* works better than the *Non-Adaptive* and *Random*. *Random* stabilizes at a sub-optimal level of an RDP value of ≈ 2 with the same degree bound as *Non-Adaptive*.

While, this seems intuitively clear, one thing to note is that this makes our protocol *scalable* and *flexible*. The adaptive scheme enables the protocol to pick a degree for a node based on the underlying Internet's degree distribution. It is well known that the Internet topology forms a power-law like degree distribution. In the simulations, our adaptive scheme tries to mimic this at the overlay network construction layer. We evaluate the adaptiveness of the schemes to the underlying topology by taking the ratio of the overlay degree of the backbone node and the Internet degree of the underlying router. It is pertinent to note that the nodes start out with a minimum overlay degree, so we ignore those samples where the underlying routers have a lower degree than this minimum. Also, cases when the overlay degree is greater than the Internet degree are normalized to 1.0. We calculate the $TopologySimilarity$ by taking the average of the ratios over the set of backbone overlay nodes and averaging this

[3]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.

value over the 100 independent runs. The Non-Adaptive strategy shows a similarity of 63% and Adaptive does better as expected with a similarity of 89%. We also evaluate the $LinkStress$ of the schemes by calculating both the average, and the 95 percentile value of the link stress over all the Internet links (omitting links with zero stress), again averaged over the 100 runs. $Random$ has a $LinkStressAverage$ of 3.22 and a $LinkStressPercentile$ of 9, Non-Adaptive has an average of 2.22 and percentile of 5.5 while Adaptive has an average of 2.5 and a percentile of 6. Thus, we see that Adaptive incurs a low $LinkStress$ overhead compared to Non-Adaptive while giving much better RDP.

[20] 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 5 and 6 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[4] number of nodes pinged at random by each node.

C. 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.

Virtual Coordinates Stability

One of the measures of the stability of the system is the virtual coordinates computed by the Vivaldi algorithm. Figure 7 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.

[4]a good value of k is 12 as pointed out by Dabek *et al* [11] to yield best stability.

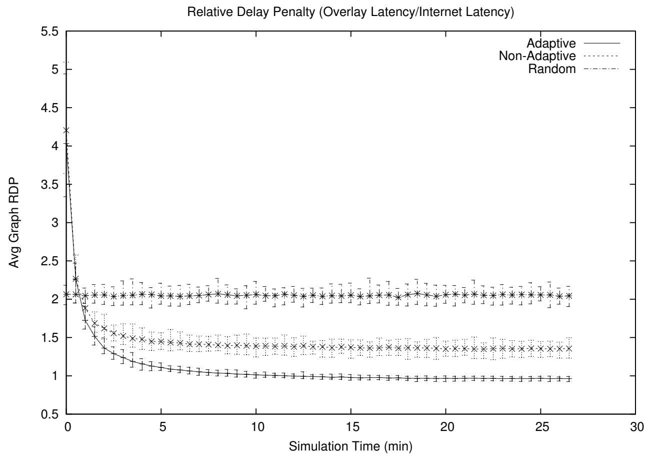


Fig. 5. **Comparison of the Relative Delay Penalty. Virtual coordinate dimension is 5, with k of the clustering as 7. We use the underlying graph with 1,000 nodes and an overlay network of 100 nodes.**

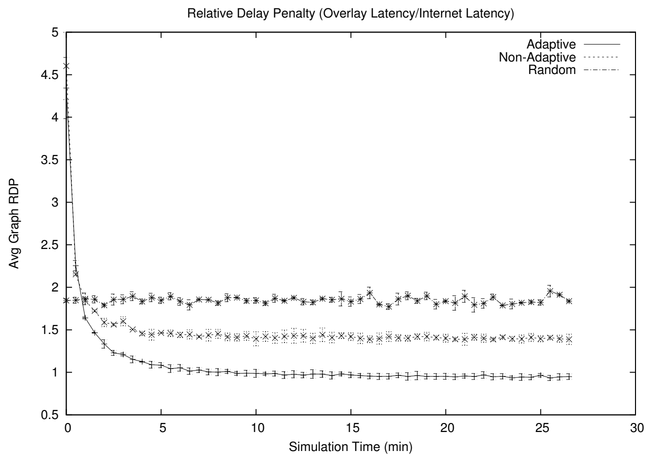


Fig. 6. **Comparison of the Relative Delay Penalty. Virtual coordinate dimension is 5, with k of the clustering as 7. We use the underlying graph with 10,000 nodes and an overlay network of 100 nodes.**

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 ≈ 700 edges. The churn in edges deleted is also comparable for adaptive and non-adaptive strategies.

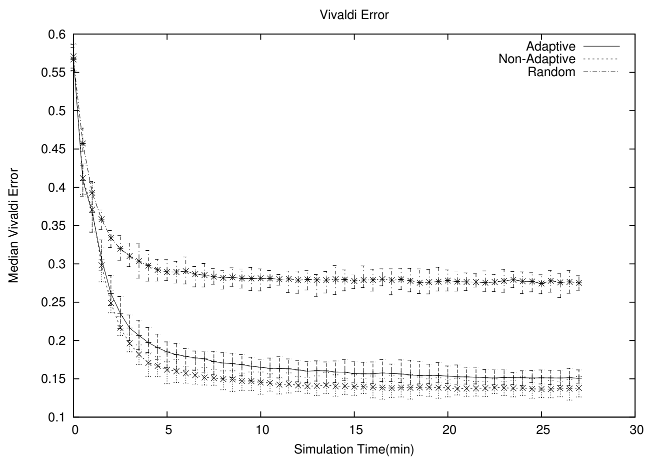


Fig. 7. **Comparison of Vivaldi error. Virtual coordinate dimension is 5, with k of the clustering as 7. We use the underlying graph with 10,000 nodes and an overlay network of 100 nodes.**

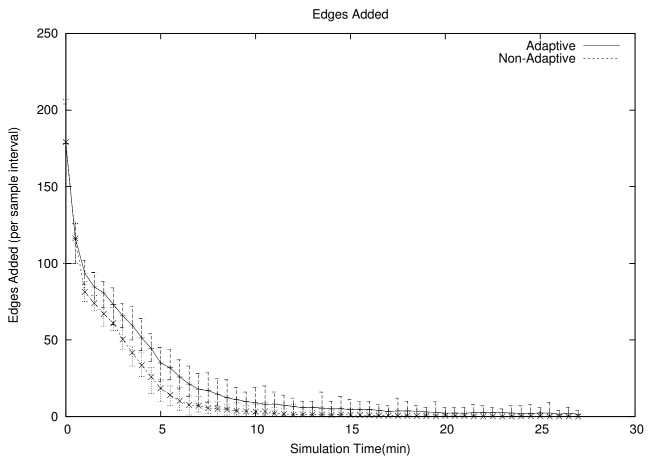


Fig. 8. **Edges added per sampling instance**

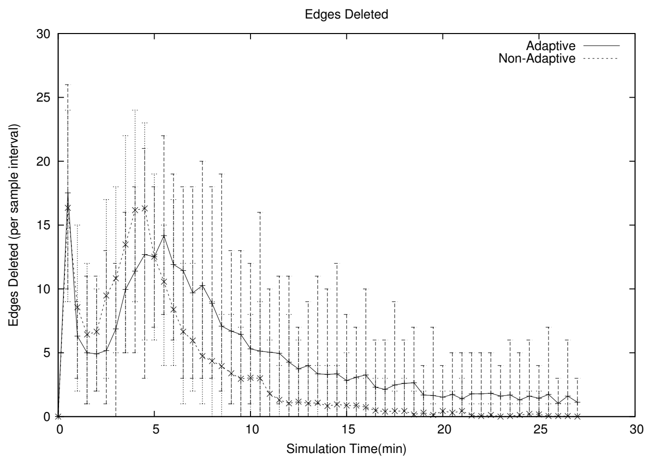


Fig. 9. **Edges deleted per sampling instance**

D. Overlay Routing Simulations

In Section III we discuss an approach to solving the general problem of routing to any given IP address and estimating the closest node on the overlay to it. We formally specified the k-pinger algorithm where k backbone overlay nodes are pressed into service to ping the DNS server of the target host and place it in the coordinate space.

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 to some node outside the cluster, 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 used the group whose maximum Vivaldi error was minimized. Intuitively, choosing a group of pingers with low aggregate Vivaldi error should establish more accurate coordinates for an outside node. Second, we used the group whose minimum pair-wise distance was maximized. Again, drawing on intuition, nodes that are farthest apart in the coordinate space should be able to better "triangulate" the position of an outside node. 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 simulation, 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 provided by MIT. The King Data set consists of a full matrix of pair-wise RTTs between 1740 DNS servers, collected by using the King method [14]. 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.

Figure 11 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. Figure 10 shows the results of increasing m. Intuitively, by increasing m we refine the position of the outside node's coordinate in the coordinate space, reducing its error. Thus we increase the likelihood of correctly choosing the closest node among the nodes in the cluster. 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.

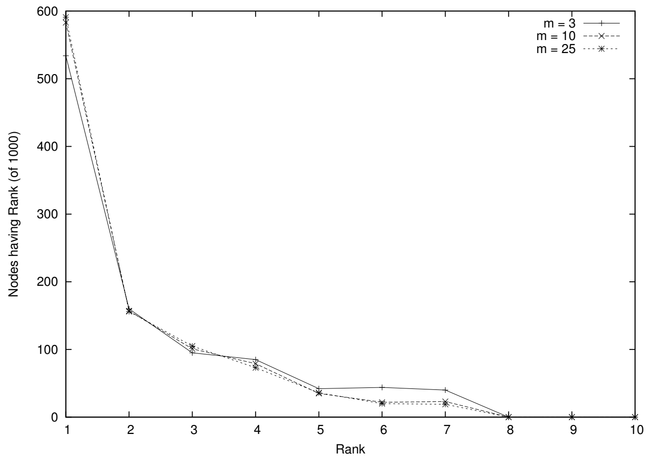


Fig. 10. **The rank of 1000 nodes as the number of Vivaldi iterations, m, varies.**

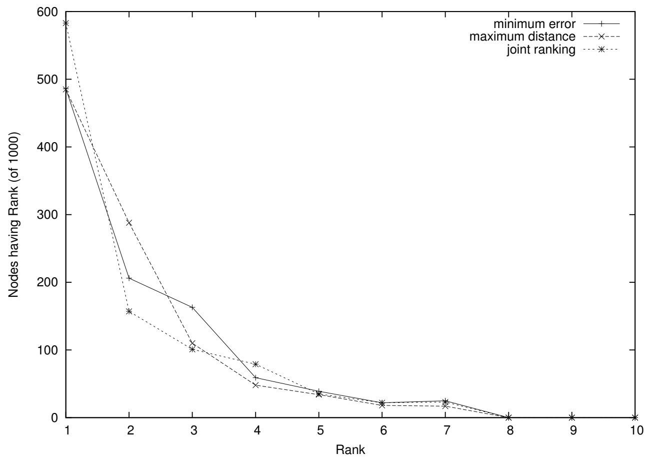


Fig. 11. **The effect of different heuristics on the rank of 1000 nodes.**

VII. Related Work

Several Internet overlays have been designed in the past for various purposes, including providing OSI network-layer connectivity [10], easing IP multicast deployment using the MBone, and providing IPv6 connectivity using the 6-Bone.

Application level multicast approaches using overlay networks have been recently proposed as a viable alternative to IP multicast. In particular, End System Multicast (ESM) [10] has gained considerable attention due to its success in conferencing applications. The main idea of ESM (and its Narada protocol) is that end hosts exclusively handle group management, routing information exchange, and overlay forwarding tree construction. The efficiency of the constructed overlay multicast trees, in terms of both performance and scalability, is the primary subject of this paper. [16] proposes a simple heuristic algorithm, which they call Topology Aware Grouping (TAG), to exploit underlying network topology data in constructing efficient overlays for application level multicast. Unlike ESM, TAG is tailored to applications with a large number of members, which join the session at different times and regard delay as a primary performance metric and bandwidth as a secondary metric. NICE [36], on the other hand, is designed to support applications with very large receiver sets and relatively low bandwidth requirements. It recursively arranges group members into a hierarchical overlay topology, which implicitly defines a source-specific tree for data delivery. It has been shown that NICE scales to large groups in terms of control overhead and logical hops. On similar lines, TOMA [21] uses a two level hierarchy to create a scalable distribution overlay for aggregated multicast.

QoS has received its fair share of overlay papers. Schemes such as QRON [23] try to search for QoS-satisfied overlay paths using link-state protocols for QoS metric propagation and source-based provisioning of the path. Service Overlay Networks [17], [18] have been proposed to purchase bandwidth with certain QoS guarantees from network domains using SLAs and stitch them to provide end-to-end QoS guarantees. Such an architecture would still rely on the underlying domains to meet their specified QoS requirements. We believe that service composition based techniques such as those mentioned in [19] would form an ideal mechanism to perform WS-Agreement negotiations across backbone overlays. OverQoS [25] applies QoS enhancements within the overlay network as opposed to end-to-end. Streaming media flows in OverQoS can be shaped as part of a larger aggregate as opposed to being treated as separate flows.

Significant work has also been done on using the overlay to do "indirect" routing. The Detour framework [24] was motivated by the potential long-term performance benefits of indirect routing. It is an in-kernel packet encapsulation and routing architecture designed to support alternate-hop routing, with an emphasis on high performance packet classification and routing. It uses IP-in-IP encapsulation to send packets along alternate paths.

While RON [4] shares with Detour the idea of routing via other nodes, it seeks to prevent disruptions in end-to-end communication in the face of failures. However, RON incurs high overhead due to its constant probing of its neighbors and consequently doesn't scale to more than 50 overlay nodes.

Routing within the overlay has also come under the microscope. [5] tries to address the important issue of using multipath on an overlay. The main conclusion of the paper is that failure independence of Internet paths is "reasonable, but not large" and hence the usage of multipath might not be useful.

Inspite of the enormous amount of research that has been done on overlays, each scheme has its own caveats. A lot of performance oriented overlay proposals do not address key issues such as identifying the best ingress and egress node on the backbone for a given source and destination pair, and the accuracy and scalability of the measurement techniques involved in the decisions. Given that so much of the overlay value lies in making good decisions quickly, scalable and accurate measurement techniques are very desirable.

VIII. 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:

- 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 II-D, 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.
- 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.
- 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.
- 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.

Implementation Issues

We are in the process of implementing the architecture on an Internet-wide test-bed. We expect to see some variations from expected results due to cross-traffic, the difficulty of accurate measurements and the time lag incurred by Q-OSPF in the dissemination of information. So the results of our simulations are somewhat optimistic considering that the lag in the dissemination and consequent selection of good neighbors will be dependent on the Q-OSPF timers. This is one aspect of the architecture which we will affect the results in the real implementation.

References

[1] Y. Amir, C. Danilov, and C. Nita-Rotaru. High Performance, Robust, Secure and Transparent Overlay Network Service. In Proceedings of International Workshop on Future Directions in Distributed Computing (FuDiCo), June 2002

[2] R. Albert and A. Barabasi, Statistical Mechanics of Complex Networks, Review of Modern Physics 2002

[3] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS '01, Cincinnati, Ohio, August 2001.

[4] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In Proceedings of 18th ACM SOSP, October 2001.

[5] D. G. Anderson, A. C. Snoeren, and H. Balakrishnan. Best-Paths vs. Multi-Path Overlay Routing. In Proceedings of ACMSIGCOMM/USENIX Internet Measurement Conference (IMC2003), October 2003.

[6] www.akamai.com

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

[8] D. G. Anderson, A. C. Snoeren, and H. Balakrishnan. Overlay Overheating. In Proceedings of HotNets-II.

[9] L. Peterson. A Routing Underlay for Overlay Networks. In Proceedings of ACMSIGCOMM/USENIX Internet Measurement Conference (IMC2003), October 2003

[10] 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.

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

[12] C. de Launois, http://www.info.ucl.ac.be/people/delaunoi/svivaldi/ (November 2004)

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

[14] 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).

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In Proceedings of USENIX Symposium on Operating Systems Design and Implementation, October 2000.

[16] M. Kwon and S. Fahmy. Topology aware overlay networks for group communication. In Proceedings of NOSSDAV'02, May 2002.

[17] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning. In Proceedings of 10th IEEE International Conference on Network Protocols, Paris, France, November 2002.

[18] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In Proceedings of The IEEE 23rd International Conference on Distributed Computing Systems (ICDCS 2003), Providence, Rhode Island, May, 2003.

[19] X. Gu and K. Nahrstedt. Distributed Multimedia Service Composition with Statistical QoS Assurances. To appear in IEEE Transactions on Multimedia, 2005.

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

[21] L. Lao, J. H. Cui, and M. Gerla. TOMA: A Viable Solution for Large-Scale Multicast Service Support. To appear in IFIP Networking 2005, Waterloo, Ontario, Canada, May 2005.

[22] Scott Seongwook Lee and Mario Gerla. Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths. *Lecture Notes in Computer Science*, 2092:155–, 2001. International Workshop on Quality of Service (IWQoS).

[23] Z. Li and P. Mohapatra, "QRON: QoS-aware Routing in Overlay Networks," Special issue on Service Overlay Networks in the IEEE Journal on Selected Areas in Communications, January 2004.

[24] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a Case for Informed Internet Routing and Transport. In IEEE Micro, pp. 50-59, v 19, no 1, January 1999.

[25] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverQoS: An Overlay based Architecture for Enhancing Internet QoS. In First Symposium on Networked Systems Design and Implementation (NSDI'04), March 2004.

[26] A.C. Snoeren, K. Conley, and D.K. Gifford. Mesh-based content routing using XML. In Proc. 18th ACM SOSP (Banff, Canada, Oct. 2001), pp.160-173

[27] Richard C. Dubes and Anil K. Jain, Algorithms for Clustering Data, Prentice Hall, 1988

[28] 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

[29] C. Dovrolis, D.Moore and P.Ramanathan. What Do Packet Dispersion Techniques Measure? In Proceedings of IEEE INFOCOM, Anchorage AK, April 2001.

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

[31] M.Jain and C. Dovrolis, End-to-End Available Bandwidth Measurement Methodology, Dynamics and Relation with TCP Throughput, In Proceedings of ACM SIGCOMM, August 2002, pp295-308.

[32] R. Kapoor, L.J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. CapProbe: A Simple and Accurate Capacity Estimation Technique. ACM SIGCOMM 2004, Portland, USA, 2004.

[33] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, IDMaps: A global Internet host distance estimation service. IEEE/ACM Transactions on Networking, Oct. 2001.

[34] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. In International Conference on Distributed Systems, Tokyo, Japan, March 2004.

[35] J. Strauss, D. Katabi and F. Kaashoek. A measurement study of available bandwidth estimation tools. In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement[IMC'03], Miami, USA 2003, pp39-44

[36] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In Proceedings of ACM SIGCOMM 2002, Pittsburgh, Pennsylvania, August 2002.

[37] A. Nakao, L. Peterson, and A. Bavier. A Routing Underlay for Overlay Networks. In Proceedings of ACM SIGCOMM 2003, Karlsruhe, Germany, August 2003.

[38] A. Akella, J. Pang, B., S. Seshan, A. Shaikh, A comparison of overlay routing and multihoming route control, In Proceedings of ACM SIGCOMM 2004, Portland, OR, 2004.

[39] A. Odlyzco,The economics of the Internet: Utility, utilization, pricing, and Quality of Service, http://www.dtc.umn.edu/˜odlyzko/doc/Internet.Economics.pdf

[40] A. Odlyzko, Data networks are lightly utilized, and will stay that way, Review of Network Economics, 2 (no. 3), September 2003, pp. 210-237.

[41] A. Odlyzco, Privacy, economics, and price discrimination on the Internet, ICEC2003: Fifth International Conference on Electronic Commerce, N. Sadeh, ed., ACM, 2003, pp. 355-366. Reprinted on pp. 187-211 of Economics of Information Security, L. Jean Camp and S. Lewis, eds., Kluwer, 2004. Also reprinted on pp. 39-61 of The Icfaian Journal of Management Research, vol. 3, no. 12, December 2004.

[42] A. Odlyzko, The Internet and other networks: Utilization rates and their implications, Information Economics & Policy. To appear. (Presented at the 1998 Telecommunications Policy Research Conference.)

Appendix

Definition 1: Domain Specific Service Description Language is as follows:

```
 <xsd:schema
targetNamespace="http://www.cs.ucla.edu/namespaces/OverlayNetwork"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:qos="http://www.cs.ucla.edu/namespaces/OverlayNetwork"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xsd:complexType name="QoSType">
        <xsd:sequence>
            <xsd:element name="parties"
             type="tns:AgreementPartiesType" minOccurs="1"/>
            <xsd:element name="serviceDesription"
             type="tns:serviceDescriptionType" minOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="qos" type="qos:QoSType"/>
    <xsd:element name="executable" type="xsd:anyType"/>
    <xsd:complexType name="AgreementPartiesType">
        <xsd:sequence>
            <xsd:element name="client"
            type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="provider"
            type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="serviceDescriptionType">
        <xsd:sequence>
            <xsd:element name="Source"
            type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="Destination"
            type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="Bandwidth"
            type="xsd:int" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="Delay"
            type="xsd:int" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="Multipath"
            type="xsd:bool" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
<xsd:element name=" SetupOverlayPath"
type="qos:serviceDescriptionType"/> </xsd:schema>
```