# Oriented Overlays For Clustering Client Requests To Data-Centric Network Services

Congchun He and Vijay Karamcheti
Computer Science Department
Courant Institute of Mathematical Sciences
New York University, New York, NY 10003
{congchun, vijayk}@cs.nyu.edu

***Abstract*— Many of the data-centric network services deployed today hold massive volumes of data at their origin websites, accessing the data to dynamically generate responses. Such dynamic responses are poorly supported by traditional caching infrastructures and result in poor performance and scalability for such services. One way of remedying this situation is to develop alternative caching infrastructures, which can dynamically detect the often large degree of service usage locality and leverage such information to on-demand replicate and redirect requests to service portions at appropriate network locations. Key to building such infrastructures is the ability to cluster and inspect client requests, at various points across a wide-area network.**

**This paper presents a *zone-based* scheme for constructing *oriented overlays*, which provide such an ability. Oriented overlays differ from previously proposed unstructured overlays in supporting network traffic flows from many sources towards one (or a small number) of destinations, and vice-versa. A good oriented overlay would offer sufficient clustering ability without adversely affecting path latencies. Our overlay construction scheme organizes participating nodes into different zones according to their latencies from the origin server(s), and has each node associate with one or more parents in another zone closer to the origin. Extensive experiments with a PlanetLab-based implementation of our scheme shows that it produces overlays that are (1) robust to network dynamics; (2) offer good clustering ability; and (3) minimally impact end-to-end network latencies seen by clients.**

## I. Introduction

In recent years, the World Wide Web has undergone a transformation from its read-only, information-centric roots into an infrastructure that provides programmatic access to a variety of sophisticated services. Many of these services hold massive volumes of data at their origin web sites and serve requests by dynamically generating responses. Illustrative examples include Amazon Web Services [1], the Google Web APIs service [2], and imagery services such as Microsoft's MapPoint [3], TerraServer [4] and SkyServer [5]. Such *data-centric* services do not see much scalability or performance benefit either from traditional caching infrastructures (responses are considered as "uncacheble") or from content delivery networks (CDNs) (the massive volumes of data prevent replicating the whole website contents on edge servers). Fortunately, data-centric services present a high degree of locality in service usage patterns across several dimensions: *dataspace, network regions and timescales*. This characteristic offers the potential of developing alternative caching infrastructures, which can dynamically detect service usage locality and leverage such information to on-demand replicate service portions on and redirect requests to appropriate locations. Key to building such infrastructures is the ability to cluster and inspect client requests, at various points across a wide-area network, in other words to route client requests across an appropriately designed overlay network.

Building an efficient and scalable peer-to-peer (P2P) overlay network for data-sharing has been well studied by many researchers. Proposed approaches fall into two main categories: structured and unstructured overlays. Structured P2P overlays, like Tapestry [6], CAN [7], Chord [8], Pastry [9] and Coral [10], were designed principally to support data discovery and cooperative data storage. To do so, they make use of distributed hash tables (DHTs) whereby a data item is identified by a key and nodes are organized into a structured graph topology that maps each key to a responsible node where the data or a pointer to the data is stored. Unstructured P2P overlays, like Gnutella [11], Freenet [12] and Kazaa [13], organize nodes into a random graph topology and use floods or random walks for data discovery and other queries. However, data discovery in such overlays might travel arbitrarily long distances (for random walks) or use a lot of extra network bandwidth (for floods), resulting in inefficiencies. To address this problem, several researchers, [14]–[17], have proposed exploiting the network proximity among participating nodes to improve efficiency and scalability. These systems advocate the concept of locality-awareness: nodes that are relatively close to each other in the underlying network are clustered/grouped together to ensure that communication between two nodes in a group does not travel outside of this group. Although both structured and locality-aware unstructured overlays provide an efficient scheme for file-sharing in a system where any peer is likely to communicate with any other peer, they are not a good match to the requirement of data-centric services. The latter requires that the participating nodes be organized with an orientation "bias" towards an (or a small number of) origin server(s) such that (1) service usage locality can be detected dynamically by inspecting the underlying traffic flows; and (2) such locality can yield clustering and reuse benefits by replicating a small portion of data from the origin server(s) at a few locations.

In this paper, we present a *zone-based* scheme for constructing such *oriented overlays* to facilitate locality detection in data-centric service usage patterns. Oriented overlays differ from previously proposed unstructured overlays in supporting network traffic flows from many sources towards one (or a small number) of destinations, and vice-versa. A good oriented overlay would offer sufficient clustering ability without adversely affecting path latencies. The term "clustering" is used differently in our work than in locality-aware unstructured overlays: we are not attempting to providing connectivity among grouped nodes, instead, the nodes being clustered will redirect requests to and receive responses from the same set of parent(s)[1]. The intent is to provide a merge point in the network where requests from clients that are "close" to each other can be grouped together and inspected for service usage locality. Although the metric used in clustering can, in general, be application-specific, in this paper, we work with network latency.

Our overlay construction scheme organizes participating nodes into different zones according to their latencies from the origin server(s), and has each node associate with one or more parents in another zone closer to the origin. By clustering nearby nodes at different levels of the network, dynamic detection of service usage locality is possible at different granularities of network regions, enabling consequent replication of the associated service portions to be performed on-demand.

The rest of the paper is organized as follows. Section II introduces some illustrative data-centric services deployed in the Internet. In Section III and IV, we describe our zone-based scheme and parent selection algorithm for construction of oriented overlays. In Section V, we report our experiences on implementing oriented overlays on PlanetLab, a scalable, real-world network, and evaluate the characteristics of the resulting overlays. Finally, we discuss related work in Section VI, and summarize and discuss possible extensions in Section VII.

## II. Background

Clients of data-centric network services are typically geographically distributed, and access the services across a wide-area network. Service providers usually host the services at their origin web site and serve requests at one or a small number of web servers, which are responsible for dynamically generating responses by querying again a back-end database (virtual or physical). The volumes of data accessed by such data-centric services are usually extremely large, which prevent services from being replicated on CDN systems like Akamai's. For example, in the SkyServer service, which provides Internet access to the public Sloan Digital Sky Survey(SDSS) data, the magnitude of data is on the order of 10 terabytes. For other high energy and nuclear physics services, data volumes can be as large as on the order of petabytes.

[1]We use "parent" and "child", instead of neighbors, due to the asymmetric relationship between the nodes. In our oriented overlays, one node can be a parent of an other node only if it is closer to the origin server. Hence, a parent can only receive requests from and send responses to a child, but not vice-versa.

In previous work [18], we identified that there exists a high degree of locality in data-centric service usage across several dimensions: *dataspace, network regions and multiple timescales*. An illustrative example from a 4-month trace (Jan. 1 2004 - Apr. 30, 2004) of the SkyServer service shows that: (1) 10% of client IP addresses contribute to about 99.95% of requests, and (2) 84.04% of these requests hit on 30% of the regions in the data space. (The results came from an analysis where the astronomic database of the North American Sky was partitioned into 1024 by 1024 regions using the sky coordinate systems, and we accumulated client requests directed towards an individual region). In addition to such spatial and end-host locality, our analysis reveals that similar locality structures can found across multiple network levels. These results imply that a small subset of service data, which accounts for a large fraction of the overall request load, can be replicated at a small number of of network locations, where a large fraction of requests originate, to significantly improve overall system scalability and performance. Similar trends are also observed in the TerraServer service, and are expected in other services such as Microsoft's MapPoint, MapQuest, Google's Web API's, etc.

Our results suggest the possibility of building novel caching infrastructures where network intermediaries that can dynamically inspect traffic flowing between clients and services, infer models for service access patterns, and potentially improve service scalability by taking actions such as replication, request redirection, or admission control. We described the design and implementation of one such infrastructure in [19].

However, benefits from such infrastructures depend on how the underlying network is organized. Clearly, if requests are routed among intermediaries in a way that makes locality detection hard, the infrastructures can not take replication and redirection actions. Similarly, if all requests are forced to go directly through a central server, not much benefits can be expected even if requests exhibit locality at the network level and in the targeted data space.

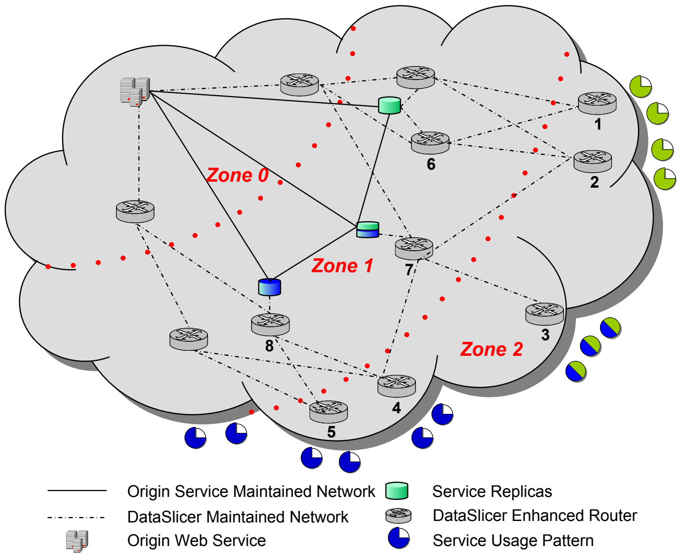
The first observation suggests that client requests must be routed through the network in a way that permit intermediate locations to identify and hopefully exploit request similarity. The second observation requires that these locations be distributed across the network as opposed to being clustered around the origin server(s). The challenge addressed in this paper is how to tradeoff between these two considerations.

## III. Design

Our solution to the above challenge is to build what we call "oriented overlays". We describe in turn their design, construction and maintenance, and the properties they offer. We assume that our overlays will involve on the order of $10^0$ — $10^1$ origin server(s) and $10^2$ — $10^3$ participating nodes.

### A. Overview

Overlay networks are constructed by participating nodes. Each node runs a protocol to communicate with other nodes and feeds the collected information into a centralized or



Zone 0
Zone 1
Zone 2
1
2
3
4
5
6
7
8
Origin Service Maintained Network
DataSlicer Maintained Network
Origin Web Service
Service Replicas
DataSlicer Enhanced Router
Service Usage Pattern

Fig. 1. Overview of an Oriented Overlay for Data-centric Network Services

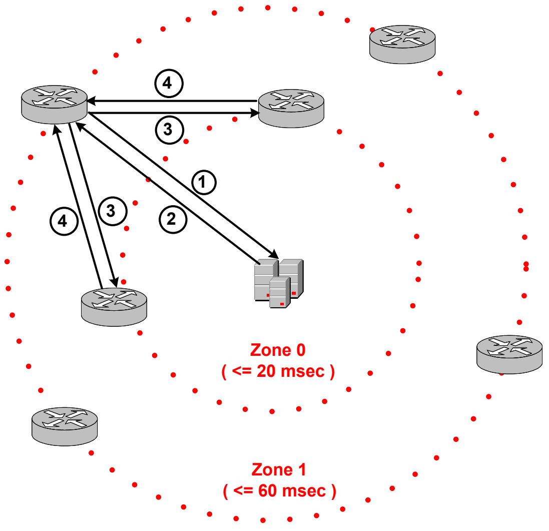


Fig. 2. Zone-based Scheme for Node Partitioning

distributed algorithm which organizes the participating nodes into a logical topology based on some metrics like latency, network bandwidth, etc.

In our oriented overlay networks, we assume there exist reliable origin servers that are physically close to the service web sites. The participating nodes consist of application-level routers that are responsible for relaying service requests and responses. Instead of pursuing optimal distances between nodes as other locality-aware overlays do, our primary goal is to build an overlay that can cluster and inspect client requests at various points in the network, without adversely affecting path latencies. To realize this goal, we propose a *zone-based* scheme.

A *zone* defines a range of distances (in terms of network latency) from an origin server: the higher the level of a zone, the farther away from the origin it is. According to their distances from the origin, the participating nodes are partitioned into different zones. Each participating node then selects one or more parents to connect to, forming an overlay. The parent selection has an orientation "bias" towards the origin: (1) a participating node $A$ can only select another node $B$ as its parent if $B$ resides in a lower level zone; (2) the candidate parents for node $A$ come from the nodes which reside in $A$'s next non-empty lower zone; and (3) to avoid adversely introducing additional overhead on latency for the path from $A$ to the origin, $A$ usually selects the closest node(s) from the candidates as its parent(s).

Figure 1 illustrates a desirable overlay for a data-centric service with a single origin server. In this illustration, nodes $1$ and $2$ that are close to each other share a common pattern when accessing the service. Similarly, nodes $4$ and $5$ share another pattern. Node $3$, located between these two groups, shares both patterns. Using our zone-based scheme, the participating nodes are partitioned into three zones: the first zone consists of nodes that are up to 20 msec away from the origin; the second zone consists of nodes whose distances from the origin are between 20 msec and 60 msec; and the third zone consists of nodes whose distances from the origin are larger than 60 msec. Ideally, nodes $1$ and $2$ will be directed to a node like node $6$ for relaying service requests and responses, because node $6$ provides a shorter path to the origin, compared to alternatives such as node $8$. On the other hand, node $3$ would not be selected as a parent for nodes $1$ and $2$ because it belongs to the same zone and could potentially adversely increase the latencies of the path between node $1$ or $2$ and the origin. The example overlay shown in the figure has the advantage that the intermediate nodes can easily detect service usage locality in client requests and hence allow actions such as service replication to be taken. For example, node $6$ is able to detect the similarity in service usage patterns from nodes $1$ and $2$ and create a replica nearby node $6$ to hold only a portion of the data corresponding to that usage pattern.

Our zone-based scheme can support building oriented overlays in situations where there are multiple origin servers. In this case, each origin server can have its own overlay which consists of a disjoint subset of participating nodes: each node selects the closest origin server and participates in only that origin server's overlay construction, assuming that this overlay potentially provides the best path latency. Since the network status changes dynamically, we allow a node to switch to another origin server's overlay from the current one if it detects that origin server to be closer.

In the rest of this section, we describe the construction and maintenance protocols for our zone-based oriented overlays, and discuss the properties our designed overlays possess.

## B. Construction Protocols

A participating node needs to take two important steps to join in our overlays: Node Startup and Parent Selection.

*1) Node Startup:* A node joins in the system by first registering itself to the closest origin server. To do so, the

```
Inputs:
    $S$: set of origin servers
    $K$: number of parents a node wants to connect to
    $N$: number of children an intermediate node allows
    $D$: threshold on times a node is reported as being dead

    $n, m$: overlay node
    $l_{n,m}$: round-trip latency between node $n$ and $m$

    $r_n$: zone rank of node $n$ assigned by an origin
    $C_n$: candidate parents for node $n$ as advised by an origin
    $P_n$: parent list selected by node $n$
    $L$: list of participating nodes maintained at an origin

Origin Server ($s$):
    upon a join/update request: ($n$, $l_{n,s}$)
    compute $r_n$ for node $n$ using $l_{n,s}$
    $r := r_n - 1$
    while $C_n$ is empty and $r \geq 0$
        add $m \in L$ into $C_n$ for all $m$ where $r_m = r$
        $r := r - 1$
    if $C_n$ is empty
        add the origin server into $C_n$
    send ($r_n$, $C_n$) to node $n$
    if $n$ is in $L$
        update the rank of $n$ with $r_n$
        reset $n.counter$ and $n.timer$
    else
        insert $n$ into $L$

    upon a leave request: ($n$)
    remove $n$ from $L$
    foreach $m$ where $r_m = r_n - 1$
        notify $m$ that $n$ has left

    upon a node_dead message: ($n$)
    increase $n.counter$ in $L$
    set $n.timer$ if not set
    if $n.counter > D$ or $n.timer$ expires
        remove $n$ from $L$
        foreach $m$ where $r_m = r_n - 1$
            notify $m$ that $n$ is left

Node Startup ($n$, $S$):
    probe the round-trip latencies $\{l_{n,s}\}$ for all $s \in S$
    select the closest $s$
    send a join request ($n$, $l_{n,s}$) to $s$
    receive a response ($r_n$, $C_n$) from $s$
    run parent selection

Parent Selection ($n$):
    probe $m \in C_n$ and sort $C_n$ by $l_{n,m}$
    $i := k := 0$
    while $k \leq K$ and $i \leq |C_n|$
        send a parent_sel request to $C_n[i]$
        if $C_n[i]$ grants the request
            $P_n := P_n \cup \{C_n[i]\}$
            $k := k + 1$
        $i := i + 1$
    establish connections to the selected parents

Participating Node ($n$):
    upon a parent_sel request from $m$
    if $m$ exists in child list
        grant $m$'s request
    else if number of children is less than $N$
        grant $m$'s request and add $n$ into child list
    else
        reject $m$'s request

    upon a parent_cancel request from $m$
    remove $m$ from child list

    upon a node_dead ($m$) message from $s$
    remove $m$ from $C_n$ and $P_n$

Overlay Switching ($n$, $s$, $s'$):
    send a leave request ($n$) to $s$
    send a join request ($n$, $l_{n,s'}$) to $s'$
    receive a response ($r_n$, $C_n$) from $s'$
    re-run parent selection

Node Maintenance ($n$, $s$):
    periodically, probe $s' \in S$
    if exists $s' \in S$, s.t. $l_{n,s'} < l_{n,s}$
        switch to the overlay oriented towards $s'$

    periodically, randomly select $C'_n \subset C_n$
    foreach $m \in C'_n$
        probe $l_{n,m}$
        if fail
            remove $m$ from $C_n$ and $P_n$
            send $node\_dead(m)$ to $s$
    sort $C_n$ in ascending order by $l_{n,m}$
    replace $P_n$ with first $K$ nodes in $C_n$ that can be $n$'s parent
    establish connections to the selected parents
```

Fig. 3. Distributed algorithm for construction of oriented overlays (Independently run for each origin server)

node probes the round-trip latencies between itself and all of the origin servers, selects the one with the smallest latency, and passes this information to that chosen origin in the *node_join* request. Upon receiving a *node_join* request, an origin server extracts the round-trip latency information from the request message, computes the rank of the zone that this node belongs to, and assigns the rank to this node. As a response, the origin server sends the assigned rank, along with an advised candidate parent list, back to the node. In Figure 2, steps 1 and 2 demonstrate this procedure.

Each origin server maintains a table of participating nodes and their assigned ranks.

*2) Parent Selection:* The parent selection algorithm is designed to ensure that paths are chosen with an orientation "bias" towards an origin server. When the origin receives a *node_join* request from a node, it responds with an assigned zone rank and advises that node of a list of parent candidates with lower ranks. The node then probes the round-trip latencies between itself and these parent candidates and selects $K$ nodes with minimum latencies as its parents, where $K$ is a threshold on the maximum number of parents that a node can have.

To avoid overload on some intermediate nodes, i.e., a situation where a large number of nodes select the same node as their parent, we also impose a restriction on the maximum number of children that an intermediate node can have. Hence, a node needs to communicate with its selected parent node first to confirm that indeed that node can serve as its parent. In Figure 2, steps 3 and 4 demonstrate this procedure.

Figure 3 shows the detailed actions taken on overlay nodes for our oriented overlay construction.

## C. Maintenance Protocols

A good overlay should adapt itself to changes of the underlying network conditions as well as nodes joining and leaving. Key to this adaptation is the ability to effectively detect the changes and efficiently propagate such information.

*1) Origin Server:* In our oriented overlay networks, the origin server receives four kinds of messages: *node_join*, *node_update*, *node_leave* and *node_dead*. The first is sent by a new joining node, which registers itself to join in the overlay; the second is sent by a node which is already participating in the overlay and periodically updates the probed round-trip latency to the origin; the third is sent by a node in the overlay which has determined that it wants to switch to another overlay; and the fourth comes from a node to report that another node is "dead" when it tried to probe that node and failed.

The origin server handles the first and the second type of messages by inserting a new record into the maintained list of participating nodes if the sender does not exist, otherwise, it just updates the node information appropriately (e.g., update the assigned rank for the sender). The origin then sends the rank of the sender and an advised list of candidate parents back to the sender. For the third type of message, the origin server removes the node from the maintained list and notifies all other nodes at zones immediately higher than the one of the node which has left. For the fourth type of message, the origin does not eagerly remove the node reported as dead. Instead, it marks that node by setting a timer and increases a counter which keeps track of number of times that node has been reported as dead. In the case that either the counter exceeds a threshold or the timer expires, the node then is removed and notifications are sent to all nodes with a rank one higher than the removed one's. The counter and the timer will be reset if either a *node_join* or a *node_update* message is received from the suspected node before the timer expires.

*2) Participating Node:* Each node maintains a list of all origin servers and the round-trip latencies between itself and these origin servers. At startup, a node selects the closest origin server to participate in its overlay construction. Periodically, a node probes all of the origin servers to update the round-trip latencies and switches to another overlay if there exists a closer origin server. In the case that a switch happens, a node sends a *node_leave* to the origin server in its current participating overlay, and then sends a *node_join* to the new selected origin server. The node then needs to re-run the parent selection algorithm in the new overlay.

After a node participates in a particular overlay, the node maintains a candidate parent list advised by the origin server in that overlay and the round-trip latencies between itself and these candidate parents. Periodically, a node probes the origin for the round-trip latency and sends this latest information in a *node_update* message to the origin. Upon receiving a response from the origin, the node merges the advised candidate list in the response with its own copy by (1) removing nodes from the current list that are not in the new one; (2) for nodes that are in the new list but not in the old one, probing these nodes and inserting them into the current list.

Each node maintains the round-trip latencies between itself and its candidate parents by periodically probing a random subset of the candidates, and updates its parent selection if there exists any candidate that can still accept new children and has smaller round-trip latency than any of its chosen parents. If any such probes fails, the node reports the failure to the origin with *node_dead* messages.

There are four types of messages used to exchange information between participating nodes: *parent_sel*, *parent_cancel*, *parent_grant* and *parent_reject*. A node can only select a candidate as a parent by first sending a *parent_sel* message to and receiving a *parent_grant* message from that candidate. In the case that the contacted candidate finds that its number of children has exceeded a threshold, it responds to the *parent_sel* request with a *parent_reject* message. If a node updates its parent selection, as discussed above, a *parent_cancel* message needs to be sent to the parent node that was chosen not to be its parent. Upon receiving a *parent_cancel* message, a node just removes the corresponding node from its child list.

## D. Overlay Properties

Our zone-based overlay construction scheme is (1) relatively simple — no support from any external measurement infrastructure is needed; (2) efficient — an origin server acts as a rendezvous point by maintaining participating nodes and advising about candidate parent lists such that a node joining in the system needs only query the origin once, following a small number of probes; (3) distributed — parent selection and maintenance are pair-wise distributed algorithms; and (4) incurs minimal communication cost — the traffic contributed to our overlay construction and maintenance is light-weight compared with other unstructured overlays which rely on network floods.

Our oriented overlays are also robust in the face of high-network-churn because a node that has left the system can be detected quickly with high probability and reported to the origin, which in turn propagates this information to all of the affected nodes. Node leaves don't really impact the connectivity of our overlays because a node can not reach the origin server only if all of its paths are broken.

The impact of overlay paths on the latency of propagating a request from a participating node to an origin server (and vice-versa) is minimal because a node's candidate parents always reside in a lower level zone and our parent selection algorithm selects the closest candidates as parents. In this way, we ensure that a path is constructed with strong orientation "bias" towards the origin with minimal latency overhead being introduced.

Our overlays approximately position the participating nodes in the network using the measured latencies between the nodes. Given the lack of accuracy in network proximity, nodes that are clustered in our built overlay could be rather far away from each other. Such inaccuracies can affect the goodness of clustering of our overlay construction. Obviously, if additional information such as node coordinates is available (for example, a participating node can provide its position to the origin at the registration step), the clustering in our overlay can be further improved. Similarly, some other application-specific information, if provided, can also help our overlay clustering. Such information can also reduce the traffic used for our overlay construction and maintenance: the origin can advise joining nodes of more accurate parent candidate lists and hence reduce the amount of probes needed in parent selection.

## IV. Implementation

We have implemented our proposed oriented overlay construction algorithm in a wide area network environment. The communication protocols described above are implemented in C code, using the UDP protocol. The total length of our C programs is about 3,000 lines.

On the server side, our server program listens on a public port to receive messages from participating nodes and processes these messages in an event-driven fashion. There is a tradeoff between notifying the participating nodes of the up-to-date status of the overlay and reducing the communication cost in overlay maintenance. In the implementation, the server does not respond to every *node_update* request from a participating node. Instead, it updates the information about the participating nodes with the incoming requests. The server then periodically (every 5 minutes) sends its advised candidate parent list to each participating node based on the latest information of its overlay. The advantage of doing so is clear: the traffic contributed to overlay maintenance is significantly reduced from $O(N^2)$ to $O(N)$, where $N$ is the number of nodes that participate in the overlay.[2]

On the participating node side, our client program also listens on a public port to receive messages from the server(s) and other nodes, using an event-driven model. The client program responds to messages immediately. Every 30 seconds, the client program probes all of the origin servers and a subset of its candidate parents using the system tool “PING”. Specifically, the client program is configured to send out 10 ICMP ECHO_REQUEST packets to another node within 2 seconds, one for each 200 ms interval. If the client program does not receive any ICMP ECHO_RESPONSE packets within this 2 seconds period, the corresponding node is considered as “dead”. Based on the probing results, the client program takes appropriate actions such as switching overlays, reporting dead nodes or changing its parent selection. Each participating node is configured to allow a maximum of 25 children nodes and connect itself to up to 3 parents.

To support the client and server programs above, we rely upon a coordinator program running on a node that is assumed reliable. The coordinator is responsible for starting up the client and server programs (using SSH), periodically checking for liveness, and restarting the programs as required after individual nodes fail and recover.

[2]Assuming that nodes are evenly partitioned into each zone, for a *node_update* request from a node in an intermediate zone, the origin server needs to sends out $O(N)$ notification messages. The communication cost is therefor $O(N^2)$. In our implementation, an origin server integrates all of the changes that happen to the overlay during a period, and only needs to send out $O(N)$ messages.

## V. Evaluation

To study to what degree our zone-based oriented overlays demonstrate the desired properties discussed in Section III, we experimented with a prototype of our implementation on the PlanetLab network [20]. Our testbed on PlanetLab consists of 195 hosts distributed across North America, South America and Europe. In the rest of this section, we present the results of our experiments in building zone-based oriented overlays using both a single origin server with and without network churn, and using multiple origin servers.

### A. Single Origin Server

*1) Experiment Setup:* We ran multiple experiments to construct our zone-based oriented overlays using different sets of PlanetLab hosts. In each experiment, we used the same origin server located at New York University in New York, USA. However, we randomly chose half of the hosts to participate in overlay construction. Each experiment lasted for 30 minutes and the origin server was reset to cope with the overlay construction on a different set of PlanetLab hosts.

Our zone-based scheme partitioned nodes into different zones in terms of their round-trip latencies from the origin server: *Zone0* corresponds to $0 \sim 20$ ms, *Zone1* corresponds to $20 \sim 60$ ms, *Zone2* corresponds to $60 \sim 100$ ms, *Zone3* corresponds to $100 \sim 200$ ms, and *Zone4* corresponds to more than 200 ms. Intuitively, we expect nodes in north-east United States to fall into Zone0, nodes in the central portions of the United States to fall into Zone1, nodes on the west coast of the United States to fall into Zone2, and nodes in Europe or South America to fall into Zone3 or Zone4.

*2) Results:* We evaluate our oriented overlays according to the following aspects: (1) the nature of the overlay (how nodes are partitioned into zones, and what kind of connectivity the overlay provides), (2) the performance of the overlay (to what extent is network latency improved/impaired), and (3) the ability of the overlay to cluster client requests.

We ran 50 experiments, 9 randomly selected ones of which are presented in this paper. In the rest of this section, we focus our discussion on one illustrative example highlighted in the tables. In this experiment, 95 PlanetLab hosts were chosen to build the overlay. Among these, 6 hosts terminated our programs and rejected all further SSH connections during the experiment. Therefore, only 89 nodes (including the origin server at NYU, which is not shown in the tables) were actually used.

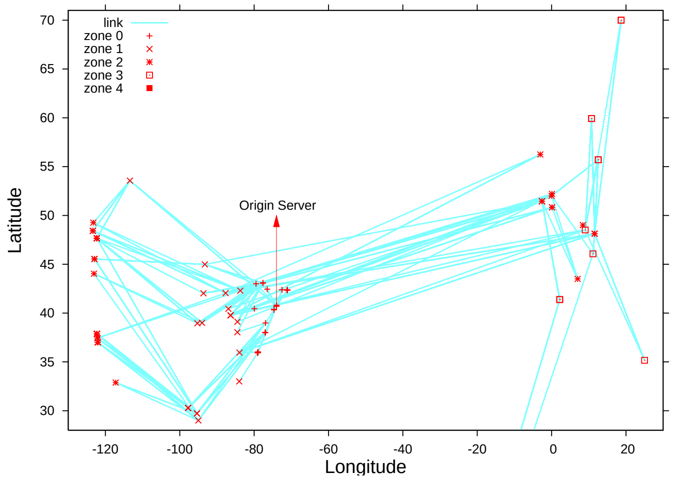


Fig. 4. An Oriented Overlay with a Single Origin Server Constructed on PlanetLab. Notice that one node (belonging to Zone 4) in South America is not shown to better show the remaining nodes in the network.

Figure 4 shows the oriented overlay that got built in the illustrative experiment. In this overlay, nodes in the northeast of the United States were partitioned into Zone0, nodes in the midwest and southeast regions of the United States were partitioned into Zone1, nodes in the west of the United States and some nodes in Europe were partitioned into Zone2, the rest of the nodes in Europe were partitioned into Zone 3, and finally, the nodes in South America were partitioned into Zone4. Note that the latter are not shown in the figure to obtain a better view of the constructed overlays.

**Overlay Nature** Table I shows how nodes were distributed into different zones in the constructed overlays: each column shows the number of nodes partitioned into the corresponding zone and the average number of parents (out-degree) and children (in-degree) of these nodes. Each row corresponds to an overlay constructed in a particular experiment.

The illustrative example (the highlighted row) shows that 25 (28.41%) nodes were partitioned into Zone0, 18 (20.45%) nodes into Zone1, 34 (38.64%) nodes into Zone2, 10 (11.36%) nodes into Zone3 and only 1 (1.14%) node into Zone4. For the nodes in Zone0, the out-degree was always 1 since these nodes can only select the origin web server as their parent. The average out-degree of nodes in other zones was 3, the maximum allowed for nodes in parent selection. This implies that all of the nodes were able to select 3 parents to connect to. Notice that the average in-degree of nodes in Zone0 and Zone1 were higher than those for the other three zones, because Zone1 and Zone2 contained more nodes.

The results validate our proposed zone-based scheme by demonstrating that (1) the participating nodes can be appropriately clustered using the network latency metric; and (2) the overlay provides good connectivity for participating nodes.

**Impact on Latency** To understand what kind of impact our overlay construction has on a node's network latencies, we compare the average latency seen by a node (computed by taking the average of the latencies of all paths from the node to the origin) in the constructed overlay with that experienced by a direct connection between the node and the origin server. The ration of these values is called $Latency\_Dilation$. Table II summarizes the latency dilations for participating nodes in each constructed overlay. An entry in the table shows that (1) the number of nodes whose latency dilations fall into a particular range; and (2) how these nodes were partitioned among the zones in the overlay. For example, the first entry in the third column of the table, "$23(14/4/5/0/0)$", should be read as "there are $23$ nodes in the overlay whose latency dilations are between $0.8 \sim 0.95$; $14$ of these nodes were partitioned into Zone0, $4$ into Zone1 and $5$ into Zone2".

In the illustrative example, most of the nodes in Zone0 and Zone1 did not get affected because of the overlay: for nodes in Zone0, the latency is the same as would be seen by a direct connection to the origin. [3] Surprisingly, a few of the nodes in Zone1 (7 out of 18) and Zone2 (5 out of 34) achieved a better latency. On the other hand, 22 out of the 34 nodes in Zone2, and all of the nodes in Zone3 and Zone4 found their performance impaired by a factor of up to 2. Due to the extra hop(s) introduced by the constructed overlay, this is not surprising. In fact, if such "far" nodes can be clustered together and connected to the same intermediate node(s), a service replica can then be created near these intermediate nodes to significantly improve their performance.

By computing the average latency dilation of all nodes in an overlay, we found that among all 50 overlays constructed in our 50 experiments, the overhead introduced by our overlay construction was in the range $5\% \sim 15\%$, with an average of $9\%$. Our results show that our overlay construction scheme does not adversely impact node-perceived latency.

**Clustering Ability** One of the primary goals of our overlay construction was to provide an ability to cluster participating nodes in terms of their service access patterns.

While clustering effectiveness is necessarily influenced by application metrics, we use a model that is based on the analysis of access patterns of imagery services, such as SkyServer or TerraServer, and is also likely to be seen for map services such Microsoft's MapPoint or MapQuest. For such services, client nodes are geographically distributed. However, nodes that are geographically close tend to share some commonness in requesting services. In our model, we approximate such locality using geographical proximity of participating nodes. Our model for service usage is as follows. Each node is associated with a geographic region where it itself sits at the center. This region represents the portion of the service data accessed by client requests originating at that node. A measure of request clustering on an intermediate node is the overlap between the geographic regions of its child nodes. We define the overlap_ratio to be the ratio of the area of the union of

[3]The fact that some Zone0 nodes are shown with latency dilation values smaller than 1 is attributable to small (expected) measurement perturbations because of dynamic network conditions. Given the low absolute values of latencies in these cases, these perturbations sometimes result in large variations in the latency dilation value.

| Zone 0 (0 ∼ 20 ms) | | | Zone 1 (20 ∼ 60 ms) | | | Zone 2 (60 ∼ 100 ms) | | | Zone 3 (100 ∼ 200 ms) | | | Zone 4 (200+ ms) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nodes | inDeg. | outDeg. | nodes | inDeg. | outDeg. | nodes | inDeg. | outDeg. | nodes | inDeg. | outDeg. | nodes | inDeg. | outDeg. |
| **25** | **2.16** | **1.00** | **18** | **5.67** | **3.00** | **34** | **0.74** | **3.00** | **10** | **0.80** | **3.00** | **1** | **0.00** | **3.00** |
| 18 | 4.00 | 1.00 | 25 | 4.68 | 3.00 | 40 | 0.75 | 2.93 | 9 | 0.33 | 3.00 | 1 | 0.00 | 3.00 |
| 25 | 1.56 | 1.00 | 13 | 3.69 | 3.00 | 23 | 1.09 | 2.22 | 10 | 0.30 | 2.20 | 1 | 0.00 | 3.00 |
| 24 | 1.88 | 1.00 | 15 | 5.20 | 3.00 | 29 | 0.86 | 2.69 | 12 | 1.17 | 3.00 | 1 | 0.00 | 3.00 |
| 19 | 2.58 | 1.00 | 15 | 4.73 | 3.00 | 29 | 0.52 | 2.48 | 5 | 0.60 | 3.00 | 1 | 0.00 | 3.00 |
| 22 | 2.86 | 1.00 | 21 | 3.29 | 3.00 | 23 | 1.17 | 3.00 | 9 | 0.00 | 3.00 | 0 | 0.00 | 0.00 |
| 18 | 3.50 | 1.00 | 21 | 5.14 | 3.00 | 37 | 0.81 | 3.00 | 9 | 1.33 | 3.00 | 4 | 0.00 | 3.00 |
| 19 | 2.53 | 1.00 | 18 | 4.11 | 2.67 | 30 | 0.57 | 2.60 | 6 | 0.50 | 2.17 | 1 | 0.00 | 3.00 |
| 22 | 1.91 | 1.00 | 15 | 4.80 | 2.80 | 25 | 0.96 | 2.88 | 8 | 0.00 | 3.00 | 0 | 0.00 | 0.00 |

TABLE I

NODE DISTRIBUTION ON ZONE-BASED OVERLAYS WITH A SINGLE ORIGIN SERVER

| Distribution of Latency Dilation (Average Overlay Latency / Direct Latency) Values | | | | | | |
|---|---|---|---|---|---|---|
| [0, 0.5) | [0.5, 0.8) | [0.8, 0.95) | [0.95, 1.05] | (1.05, 1.2] | (1.2, 1.5] | (1.5, 2.0] |
| **3 (3/0/0/0/0)** | **3 (0/3/0/0/0)** | **23 (14/4/5/0/0)** | **26 (8/11/7/0/0)** | **5 (0/0/5/0/0)** | **11 (0/0/6/4/1)** | **17 (0/0/11/6/0)** |
| 5 (4/0/1/0/0) | 1 (0/1/0/0/0) | 12 (9/3/0/0/0) | 43 (5/19/19/0/0) | 13 (0/2/8/2/1) | 11 (0/0/5/6/0) | 8 (0/0/7/1/0) |
| 11 (3/0/6/2/0) | 2 (0/1/1/0/0) | 13 (11/2/0/0/0) | 19 (11/8/0/0/0) | 12 (0/2/9/0/1) | 9 (0/0/5/4/0) | 6 (0/0/2/4/0) |
| 7 (3/0/3/1/0) | 3 (0/3/0/0/0) | 19 (15/4/0/0/0) | 28 (6/6/15/1/0) | 15 (0/2/4/8/1) | 6 (0/0/4/2/0) | 3 (0/0/3/0/0) |
| 9 (3/0/5/1/0) | 0 (0/0/0/0/0) | 10 (9/1/0/0/0) | 18 (7/11/0/0/0) | 18 (0/3/13/1/1) | 10 (0/0/10/0/0) | 4 (0/0/1/3/0) |
| 4 (4/0/0/0/0) | 4 (0/4/0/0/0) | 14 (11/3/0/0/0) | 31 (7/12/12/0/0) | 3 (0/2/1/0/0) | 9 (0/0/5/4/0) | 10 (0/0/5/5/0) |
| 2 (2/0/0/0/0) | 3 (1/2/0/0/0) | 28 (9/5/14/0/0) | 20 (6/13/1/0/0) | 9 (0/1/8/0/0) | 15 (0/0/7/4/4) | 12 (0/0/7/5/0) |
| 3 (3/0/0/0/0) | 1 (0/1/0/0/0) | 12 (9/3/0/0/0) | 23 (6/13/4/0/0) | 15 (1/1/13/0/0) | 12 (0/0/7/4/1) | 8 (0/0/6/2/0) |
| 2 (2/0/0/0/0) | 2 (0/2/0/0/0) | 28 (13/4/11/0/0) | 25 (9/9/7/0/0) | 12 (0/5/2/5/0) | 12 (0/0/6/4/2) | 8 (0/0/5/3/0) |

TABLE II

LATENCY DILATION IN ZONE-BASED OVERLAYS WITH A SINGLE ORIGIN SERVER.

| Distribution of Overlay-Score Values for a Node Region of 3° Longitude by 3° Latitude | | | | |
|---|---|---|---|---|
| [0, 0.2) | [0.2, 0.4) | [0.4, 0.6) | [0.6, 0.8) | [0.8, 1] |
| **6** | **10** | **14** | **6** | **0** |
| 14 | 16 | 2 | 7 | 2 |
| 5 | 9 | 6 | 3 | 0 |
| 5 | 9 | 4 | 3 | 3 |
| 4 | 5 | 5 | 9 | 0 |
| 2 | 10 | 9 | 7 | 0 |
| 9 | 8 | 14 | 9 | 1 |
| 8 | 12 | 6 | 4 | 4 |
| 5 | 3 | 19 | 7 | 3 |

(a)

| Distribution of Overlay-Score Values for a Node Region of 5° Longitude by 5° Latitude | | | | |
|---|---|---|---|---|
| [0, 0.2) | [0.2, 0.4) | [0.4, 0.6) | [0.6, 0.8) | [0.8, 1] |
| **5** | **9** | **6** | **16** | **0** |
| 13 | 10 | 8 | 7 | 3 |
| 5 | 9 | 6 | 3 | 0 |
| 5 | 9 | 8 | 3 | 5 | 3 |
| 4 | 5 | 3 | 11 | 0 |
| 2 | 9 | 7 | 9 | 1 |
| 7 | 9 | 10 | 8 | 7 |
| 4 | 15 | 2 | 8 | 5 |
| 2 | 4 | 19 | 8 | 4 |

(b)

TABLE III

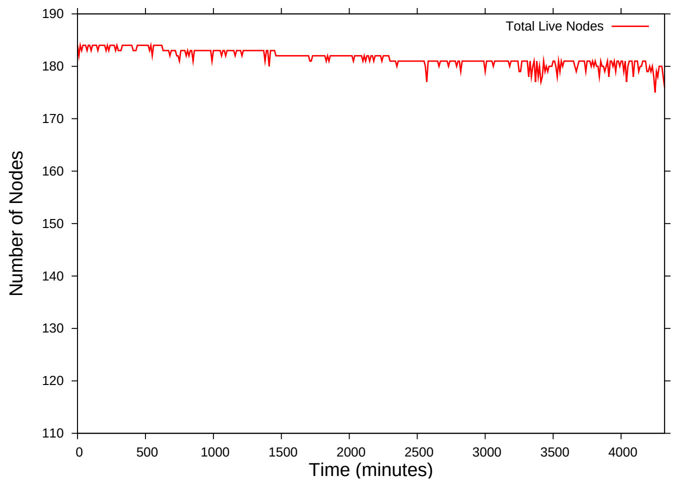
CLUSTERING IN ZONE-BASED OVERLAYS WITH A SINGLE ORIGIN SERVER.

the child regions to the sum of the areas of these regions. The goodness of clustering is measured by a score that compares this ratio to the ideal case — where all of the child nodes reside at the same location (and hence the overlap ratio is $1/\text{number\_of\_children}$):

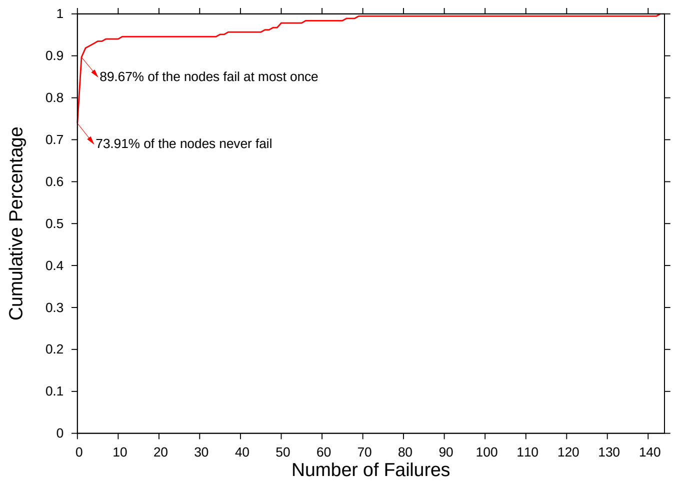
$$\text{Overlap\_Score} = (1 - \text{overlap\_ratio})/(1 - (1/\text{number\_of\_children}))$$

The closer the overlap-score value to 1, the better the clustering. Table III shows the scores computed on intermediate nodes. We first set the region size as $3.0°$ longitude by $3.0°$ latitude. The results of the illustrative example show that 34.89% of intermediate nodes (14 out of 36) score between $0.4 \sim 0.6$, and 16.67% of intermediate nodes score higher than 0.6. As we increase the size of region to $5.0°$ by $5.0°$, the percentages change to 16.67% and 44.44%, respectively. In our other experiments, we also found that some nodes can score as high as 0.95, very close to the ideal case.

Considering the approximation based on node's coordinates and the fact that nodes are rather geographically diverse, our overlay construction scheme demonstrates good ability to cluster nodes that are geographically close together. Such clustering provides ample opportunity to inspect the traffic flows between clients and the origin server to detect service usage locality.

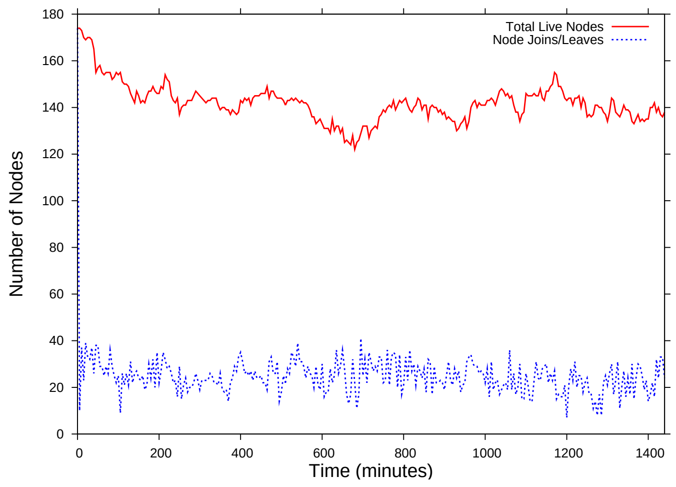


(a)

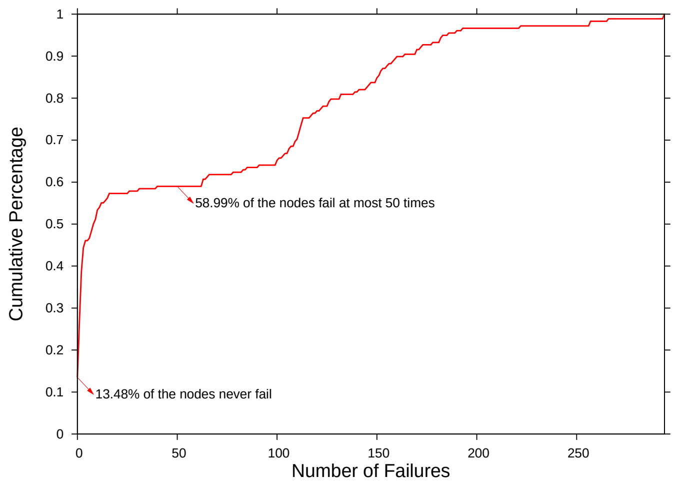


(b)

Fig. 5. Measurement of Network Churn on PlanetLab over a 3-day period.

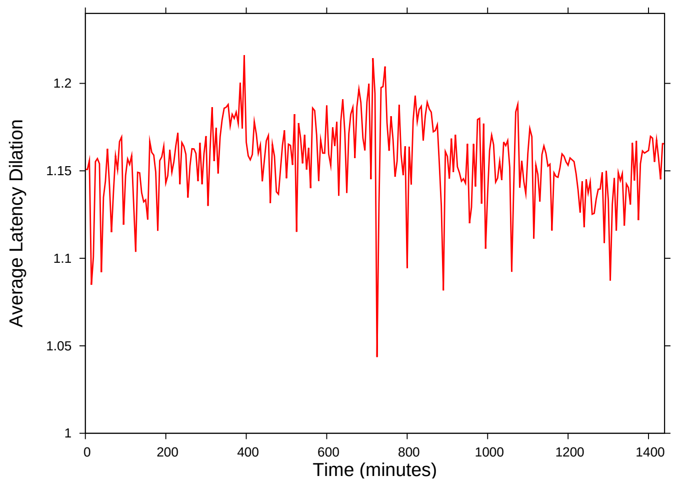


(a)

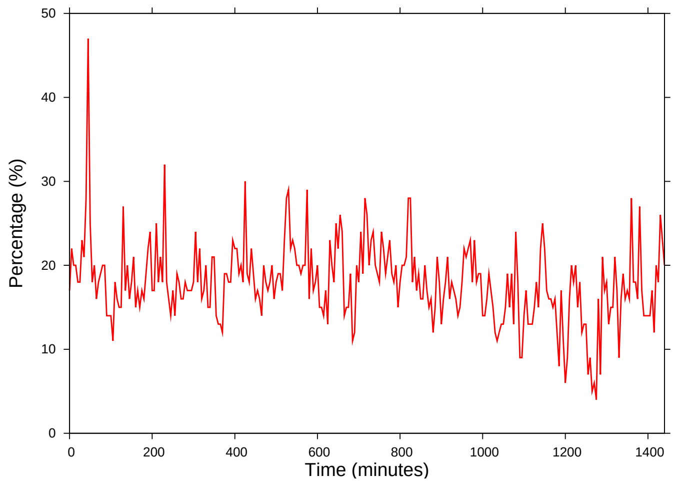


(b)

Fig. 6. A Simulated Network with High Churn on PlanetLab over a 24-hour period.



(a) Latency Dilation



(b) Communication Cost

Fig. 7. Overlay Performance on the Simulated Network with High Churn.

## B. Effect of Network Churn

In this experiment, we study the impact of network churn on our constructed overlays.

*1) Churn on the PlanetLab Network:* To better understand what kind of churn happens on the PlanetLab network, we measured the liveness of nodes over a 3-day period. Our program first identified 184 nodes out of the overall 195 nodes that were alive before the measurement, and then probed each node for its liveness every 30 minutes from a reliable node using the system tool PING. Our results show that during this 3-day period, the churn on the PlanetLab network was rather low. Figure 5(a) shows that the number of nodes that were live at a probe point was in the range of $175 \sim 184$. The number of nodes that were down increased a little bit as the experiment progressed: $1 \sim 4$ at the end of the first day and $3 \sim 9$ at the end of third day. Figure 5(b) provides addition detail about the node failures over the 3-day period. The figure, which plots the CDF of the number of failures seen by a node, shows that 73.91% of the nodes stayed up over the entire 3-day period, and 15.76% of the nodes went down just once.

*2) A Simulated Network with High Churn:* Our goal is to study the impact of high churn on our overlay construction. Since the real network churn we observed on the PlanetLab network was rather low, we ended up simulating a high-churn network at application-level: we extended our client and server programs so they could be either "up" or "down". A "down" node does not participate in overlay construction or maintenance, nor does it respond to liveness enquiries.

In a real wide area network, a relatively large fraction of nodes might stay up for a long time, while the others might go down at any time. Once nodes go down, some might take a short time to recover, while others might take an arbitrary long time to do so. Nodes might also join/leave the network arbitrarily. To model such kind of churn, we identified 178 nodes that were up when our experiment started and then randomly selected 60% of these nodes (107) that would stay up throughout the experiment. For the remaining 71 nodes, each node was able to switch its status between "down" and "up" (the initial status was "up"). Whenever a node wanted to change its status, it succeeded in doing so with a probability of 0.25. As part of the status change, a node would also determine how long it would stay in its new status: this period was randomly selected to be in the range $0 \sim 60$ minutes during the first and third quarters of the experiments, and in the range of $0 \sim 20$ minutes during the second and fourth quarters.

The origin server kept track of constructed overlay snapshots every 5 minutes, giving a total of 288 snapshots over a 24-hour period. Figure 6(a) shows that the number of nodes live at a snapshot point was in the range of $122 \sim 178$. [4] Not surprisingly, we observed a higher amount of network churn: in each 5-minute period, the number of events of churn (node joins/leaves) was in the range of $7 \sim 39$. The ratio of the number of node joins/leaves compared to the number of live nodes in each snapshot was about $17.66\%$ on average. The CDF in Figure 6(b) also supports these measurements. Only 13.48% of the nodes stayed up throughout the 24-hour experiment. [5] As expected, 40% of the nodes failed multiple times (50 times or higher).

For each snapshot-overlay recorded at the origin server, we computed the average latency dilation of all participating nodes. Figure 7(a) shows that such latency dilation is in the range of $1.1 \sim 1.2$. The average of all 288 snapshots is about 1.16. Not surprisingly, the latency dilations we observed in this experiment are slightly higher (by $7\%$) than those observed in the experiments without simulating churn. This is because in the presence of churn, a node might lose the connection(s) to its selected parent(s) and have to reconnect itself to some other nodes which are farther away. However, on the whole, this result is fairly positive: despite the relatively high amount of network churn ($40\%$ of nodes go down and up frequently), node-perceived latencies are affected by relatively small amounts. This behavior attests to the robustness and adaptability of our overlay maintenance protocols.

To evaluate the impact of network churn on the communication cost in our overlay maintenance protocols, we distinguished the messages resulting from churn from other messages of overlay maintenance. Such messages include $node\_join$, $node\_leave$ and $node\_dead$. Our results show that over the 24-hour period, the extra communication cost attributable to network churn is $18\%$ on average, as shown in figure 7(b). This number is approximately equal to the ratio of events of network churn, implying that our algorithms do not introduce any unnecessary communication costs.

## C. Multiple Origin Servers

Our oriented overlay construction algorithm can also support a network with multiple origin servers. The idea in this case is to cluster the participating nodes as close as possible to the origin servers. The advantage of such a strategy is that it provides potentially lower path latencies for participating nodes. We restrict that a node can participate in only one overlay oriented towards its closest origin.

Figure 8 shows an overlay constructed with three origin servers: one is in the east coast of the United States (NYU), the second is in the west coast of the United States (UCSB), and the last is in France (INRIA).

The results show that most of the nodes end up participating in the overlay oriented towards an origin server which is geographically closest. Not surprisingly, there exist a few nodes that violate the geographical proximity rules: four nodes in Europe selected NYU instead of INRIA to participate in because of the smaller round-trip latency between the node and NYU. Since the number of such violations is very small, it does not affect the metrics of our constructed overlays.

[4] There is a sharp decrease of this number within the period of [0, 10] minutes. This is because all the 71 nodes simulating network churn are initially configured with an "up" status so the number of nodes switching to "down" is more than those going in the reverse direction.

[5] We had expected this number to be closer to 60%, but suspect that the PlanetLab slice scheduling policy might manifest itself in some of our "up" nodes being classified as being "down" over certain intervals. Note that these nodes rarely suffer more than 8 failures.

| Distribution of Latency Dilation (Average Overlay Latency / Direct Latency) Values | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Origin | [0, .5) | [.5, .8) | [.8, .95) | [.95, 1.05] | (1.05, 1.2] | (1.2, 1.5] | (1.5, 1.7] | *(1.7, 2.0]* |
| NYU | 1 (1/0/0/0/0) | 0 (0/0/0/0/0) | 26 (21/5/0/0/0) | 23 (11/12/0/0/0) | 1 (1/0/0/0/0) | 1 (0/0/1/0/0) | 3 (0/0/1/2/0) | — |
| UCSB | 1 (1/0/0/0/0) | 6 (0/6/0/0/0) | 26 (19/7/0/0/0) | 25 (8/17/0/0/0) | 0 (0/0/0/0/0) | 0 (0/0/0/0/0) | 0 (0/0/0/0/0) | — |
| INRIA | 1 (1/0/0/0/0) | 1 (0/0/1/0/0) | 0 (0/0/0/0/0) | 30 (0/21/8/1/0) | 2 (0/0/2/0/0) | 2 (0/0/0/0/2) | 3 (0/0/0/0/3) | — |
| ALL | 3 (3/0/0/0/0) | 7 (0/6/1/0/0) | 52 (40/12/0/0/0) | 78 (19/50/8/1/0) | 3 (1/0/2/0/0) | 3 (0/0/1/0/2) | 6 (0/0/1/2/3) | — |

TABLE IV
LATENCY DILATION IN ZONE-BASED OVERLAYS WITH MULTIPLE ORIGIN SERVER.

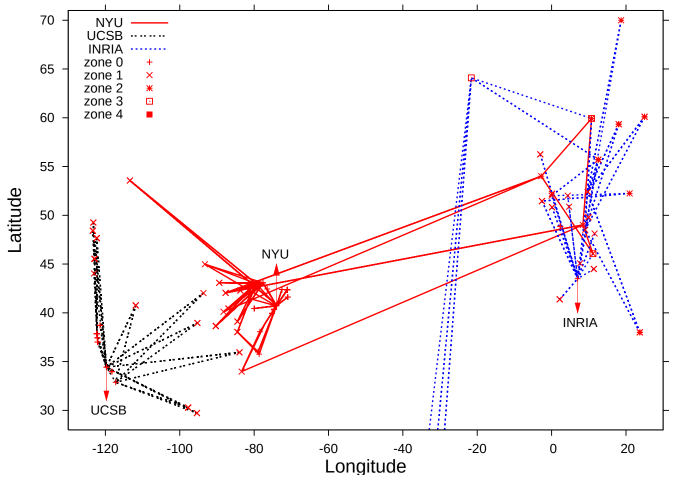


Fig. 8. An Oriented Overlay with Multiple Origin Servers Constructed on PlanetLab. Notice that 5 nodes (belonging to zone 4) in South America are not shown to better show the remaining nodes in the network.

| Distribution of Overlay-Score Values for a Node Region of 5° Longitude by 5° Latitude | | | | | |
|---|---|---|---|---|---|
| Origin | [0, 0.2) | [0.2, 0.4) | [0.4, 0.6) | [0.6, 0.8) | [0.8, 1] |
| NYU | 6 | 1 | 1 | 5 | 0 |
| UCSB | 3 | 2 | 1 | 3 | 2 |
| INRIA | 2 | 1 | 7 | 1 | 0 |
| ALL | 11 | 4 | 9 | 9 | 2 |

TABLE V
CLUSTERING IN ZONE-BASED OVERLAY WITH MULTIPLE ORIGIN SERVERS.

The resulting overlays end up exhibiting better quality in terms of latency dilation (Table IV) and node clustering ability (Table V) as compared to those built with a single origin server. In the overlays shown in Figure 8, we found that (1) for latency dilation, only 7.89% of nodes score larger than 1.05 while 85.53% of nodes score between $0.8 \sim 1.05$, as compared to 37.5% and 55.68% for the corresponding intervals in the single origin server case (see Table II); and (2) for clustering ability, 25.71% of the intermediate nodes have an overlap score between $0.4 \sim 0.6$, while 31.43% of the intermediate nodes score higher than 0.6, as compared to 16.66% and 44.44% respectively in the the single origin server case (see Table III). Notice that the number of intermediate nodes is smaller than the one in an overlay with single origin server. This is because by partitioning nodes into different overlays, the number of leaf-nodes in the resulting overlays is increased significantly.

## VI. RELATED WORK

Our zone-based oriented overlays address the challenge of building a "good" overlay network for data-centric services to flow requests from geographically distributed clients towards one or more origin servers. The main ideas underlying our work are that (1) nodes with network proximity exhibit similar service usage behaviors; and that (2) clustering nearby nodes with an orientation "bias" towards the origin server(s) provides ample opportunity to detect and dynamically leverage such service usage locality. Although addressing a somewhat different goal, our work is related to prior work that has looked at building overlay networks and at enhancing their performance with different kinds of information about network proximity.

In addition to the work on structured [6]–[10] and unstructured [11]–[17], [21], [22] P2P overlay networks that we discussed in Section I, researchers have also examined construction of overlay networks to support multicast flow patterns [23]–[26]. The focus in the former case is on supporting an all-to-all flow pattern in the context of data sharing, and unlike our medium-scale focus, the emphasis in such systems is typically on supporting efficient routing in extremely large-scale systems. The multicast networks address a more related problem, that of delivering a content stream from a single source to multiple locations. Unlike the bandwidth-centric focus of these systems, our target applications are more latency sensitive. Additionally, the reason for merging routes in the network has less to do with elimination of redundant communication, and more to do with discovering and leveraging service usage locality.

Researchers have also looked into enhancing the performance of the above overlay networks using some information about network proximity. Krishnamurthy et al. [21], [22] looked at topology-aware clustering of web clients using border gateway protocol routing information. At the application level, work on topology-aware unstructured overlays has proposed a landmark clustering scheme [14], [16], which rely upon the existence of a small number of carefully selected landmark nodes that serve as location beacons for the other (usually larger number of) nodes that participate in the overlay. Given the smaller scale of our networks, we have relied upon direct measurements of the latency between participating nodes and origin servers and likely parent candidates. Recent work on incorporating network locality considerations into structured overlays (e.g., Coral [10]) have also pursued a similar direct measurement approach. Unlike these systems, most of which use network latency as an indicator of proximity, recent work on topology-aware multicast networks [25], [26] has looked into mechanisms for estimating and optimizing use of network bandwidth. The latter is harder to measure directly, and reasoning about its shared use requires a better model of network utilization than our target applications provide.

Finally, a number of recent systems such as IDMaps [27], GNP [28], WNMS [29] and more recently, Vivaldi [30] have been proposed to map nodes on the Internet onto locations in a cartesian coordinate system. These systems provide a global distance estimation service at the infrastructure level, and if available and accurate enough can substitute for some of the measurements our algorithms make currently. Given that there are many applications where accurate geographical location information (as opposed to merely proximity indicators) yields a substantially better model of service usage, the wider availability of such systems will end up further improving the performance of our overlays.

## VII. Summary and Discussion

In this paper, we have presented a zone-based scheme to construct oriented overlays and shown using extensive experiments with a PlanetLab-based implementation that it produces overlays that (1) are robust to network dynamics; (2) offer good clustering ability; and (3) minimally impact end-to-end network latencies seen by clients.

Our overlay construction algorithms attempt to define common routing paths for requests originating at nodes that are close geographically, observing that for several data-centric services, clients that are geographically close demonstrate similarity in service usage patterns. Our approach determines geographical proximity using network latency measurements, an approximation shown in our experiments to be relatively good. As noted earlier, more direct geographical location indicators can be easily incorporated. More generally, the zone partitioning and parent selection steps of our algorithms can be extended to accommodate other, possibly application-specific, scoring systems to influence request and response routing. For instance, recognizing that clients with different levels of network connectivity typically exhibit different service usage patterns, one may wish to route requests from low-bandwidth clients along different paths as compared to those from higher bandwidth clients. Such differentiation opens up many interesting possibilities for service specialization.

As noted in Section II, one of the motivations for this work was to enable construction of alternative caching infrastructures for data-centric network services whose responses are generated dynamically. We are currently building one such infrastructure, extending an earlier design [19]. In this context, a long-term challenge is to better understand the interactions between the characteristics of the overlay network and the routing and scheduling decisions taken at the level of the infrastructure, particularly in situations where multiple services are being co-hosted on the same resources.

## Acknowledgment

This research was sponsored by DARPA agreements N66001-00-1-8920 and N66001-01-1-8929; by NSF grants CAREER:CCR-9876128, CCR-9988176, and CCR-0312956; and Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Labs, SPAWAR SYSCEN, or the U.S. Government.

## References

[1] Amazon Web Services. [Online]. Available: http://www.amazon.com/gp/aws/landing.html

[2] Google Web APIs. [Online]. Available: http://www.google.com/apis/

[3] Microsoft Corporation. Microsoft MapPoint Web Services. [Online]. Available: http://www.microsoft.com/mappoint/default.mspx

[4] TerraServer.com. [Online]. Available: http://www.terraserver-usa.com/

[5] Sloan Digital Sky Servey. SkyServer Projects. [Online]. Available: http://skyserver.sdss.org/

[6] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Univ. of California, Berkeley, CA, Tech. Rep. TR-UCB/CSD-01-1141, Apr. 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," presented at the ACM SIGCOMM, San Diego, CA, Aug. 2001.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at the ACM SIGCOMM '01, San Diego, CA, Aug. 2001.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," presented at the 18th IFIP/ACM Int. Conf. Distributed Systems Platforms (Middleware 2001), Heidelberg, German, Nov. 2001.

[10] Coral: The NYU Distribution Network. [Online]. Available: http://www.scs.cs.nyu.edu/coral/overview.html

[11] Gnutella Website. [Online]. Available: http://gnutella.wego.com/

[12] Freenet Website. [Online]. Available: http://freenet.sourceforge.net/

[13] Kazaa Website. [Online]. Available: http://www.kazaa.com/

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM'02*, 2002.

[15] M. Waldvogel and R. Rinaldi, "Efficient topology-aware overlay network," in *Proc. ACM HotNets 2002, SIGCOMM/CCR 2003*.

[16] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," in *Proc. the 23rd ICDCS*, Washington, DC, May 2003.

[17] X. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu, "A construction of locality-aware overlay network: mOverlay and its performance," in *IEEE JSAC Special Issue on Recent Advances on Service Overlay Networks*, Washington, DC, Jan. 2004.

[18] C. He and V. Karamcheti, "An analysis of usage locality for data-centric web services," New York University, Tech. Rep. TR-2005-866, 2005.

[19] C. He and V. Karamcheti, "Improving scalability of data-centric services using in-network traffic inspection," in *Proc. IEEE 10th International Workshop on Web Content Caching and Distribution (WCW)*, Sophia Antipolis, France, September 2005.

[20] PlanetLab. [Online]. Available: http://www.planet-lab.org/

[21] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *ACM SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[22] B. Krishnamurthy and J. Wang, "Topology modeling via cluster graphs," in *ACM SIGCOMM IMW '01*, San Francisco, CA, Nov. 2001.

[23] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Proc. of ACM Sigmetrics*, Santa Clara, CA, June 2000.

[24] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proc. OSDI'00*, San Diego, CA, 2000.

[25] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in a cooperative environment," in *Proc. SOSP'03*, Lake Bolton, New York, October 2003.

[26] L. Garces-Erice, E. W. Biersack, and P. A. Felber, “Multi+: Building topology-aware overlay multicast trees,” in *Proc. of the Fifth International Workshop on Quality of Future Internet Services (QofIS’04)*, Barcelona, Spain, September 2004.

[27] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jin, “An architecture for a global internet host distance estimation service,” in *Proc. IEEE INFOCOM ’99*, New York, NY, 1999, pp. 210–217.

[28] T. S. E. Ng and H. Zhang, “Predicting internet network distance with coordinates-based approaches,” in *IEEE INFOCOM ’02*, 2002.

[29] Y. Chen and R. Katz, “On the placement of network monitoring sites,” 2001. [Online]. Available: http://www.cs.berkeley.edu/yanchen/wnms/

[30] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: a decentralized network coordinate system,” in *Proc. the 2004 conf. on Applications, technologies, architectures, and protocols for computer communications*, Portland, Oregon, 2004.