Vivaldi:

A Decentralized Network Coordinate System

Authors:
Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris
MIT

Published at SIGCOMM '04

Presented by:
Emmanouel Kyriakakis

Key tool: Synthetic Coordinates

- ***Content distribution & File Sharing systems***:

KaZaA, BitTorrent, CoDeeN, CFS, DNS etc.

All of these application could benefit from network coordinates.

Designing a Synthetic Coordinate System

- Finding a metric space that embeds the Internet with little error.
- Scaling to a large number of hosts.
- Decentralizing the implementation
- Minimizing probe traffic
- Adapting to changing network conditions

Vivaldi: Features

- Decentralized, no landmarks required
- Simple: low-overhead
- Adaptive to network dynamics

Vivaldi was developed for & used by Chord

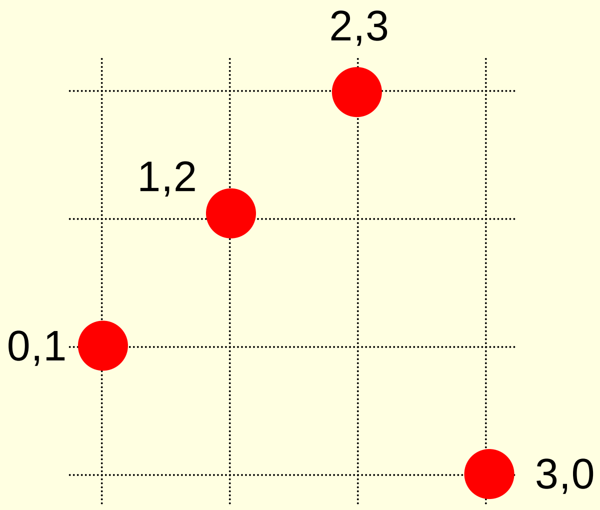
- Vivaldi is a simple, adaptive, distributed algorithm for computing network coordinates that accurately predict Internet latencies
- Internet Hosts compute their coordinates in some coordinate space such that the distance between themselves and other host's coordinates predicts the RTT between them

Vivaldi Synthetic Coordinates

- Each node estimates its own position
- Position = (x,y): “synthetic coordinates”
- x and y units are time (milliseconds)
- Distance predicts network latency
- Key point: predict w/o pinging first

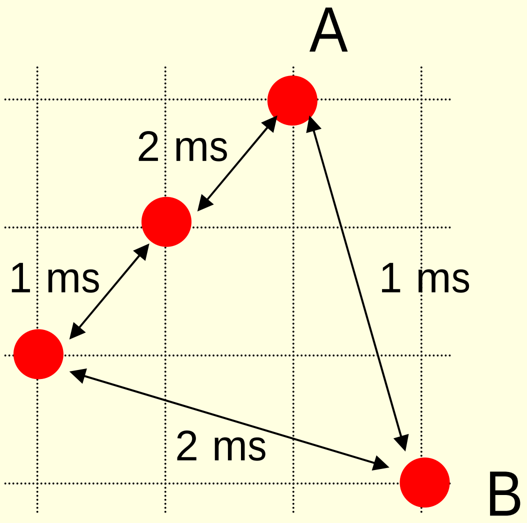
Vivaldi Synthetic Coordinates

- Each node starts with a random incorrect position



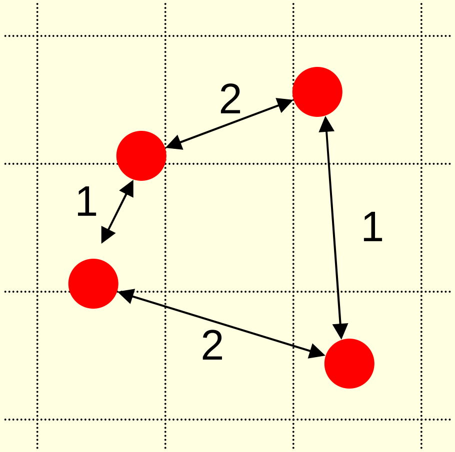
Vivaldi Synthetic Coordinates

- Each node "pings" a few other nodes to measure network latency (distance)

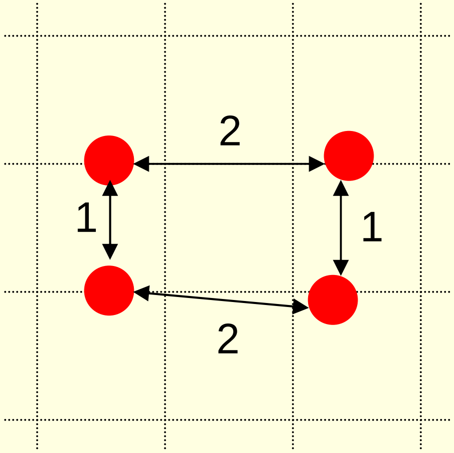


Vivaldi Synthetic Coordinates

- Each nodes "moves" to cause measured distances to match coordinates



1.



2.

Vivaldi: Algorithm

- Use synthetic distance between nodes to accurately map to latencies (RTT) between nodes.
  - Can not create an exact mapping due to violations of triangle inequality
- Tries to minimize the error of predicted RTT values
- Observation
  - Minimizing the square error function of predicted RTT between two nodes is analogous to minimizing the energy in a mass-spring system

$$E = \sum_i \sum_j (L_{ij} - \| x_i - x_j \|)^2$$

Where:

L_{ij} = Actual Measure RTT between Node i and Node j

x_i = Synthetic coordinates of Node i

x_j = Synthetic coordinates of Node j

Vivaldi: Algorithm

Hooke's Law:

$$F_{ij} = (L_{ij} - \| x_i - x_j \|) \times u(x_i - x_j)$$

Force vector Fij can be viewed as an error vector

- Forces

$$F_{ij} = (L_{ij} - \| x_i - x_j \|) \times u(x_i - x_j)$$

$$F_i = \sum_{i \neq j} F_{ij}$$

- Movement

$$x_i = x_i + F_i \times t$$

Vivaldi: Centralized Algorithm

```
// Input: latency matrix and initial coordinates
// Output: more accurate coordinates in x
compute_coordinates(L, x)
  while (error (L, x) > tolerance)
      foreach i
         F = 0
         foreach j
            // Compute error/force of this spring. (1)
            e = L_ij − ||x_i − x_j||
            // Add the force vector of this spring to the total force. (2)
            F = F + e × u(x_i − x_j)
         // Move a small step in the direction of the force. (3)
         x_i = x_i + t × F
```

- Calculate net Force on node i
- Move a step in the direction of the net Force

Vivaldi: Simple Algorithm

- Algorithm

```
// Node i has measured node j to be rtt ms away,
// and node j says it has coordinates x_j.
simple_vivaldi(rtt, x_j)
  // Compute error of this sample. (1)
  e = rtt − ||x_i − x_j||
  // Find the direction of the force the error is causing. (2)
  dir = u(x_i − x_j)
  // The force vector is proportional to the error (3)
  f = dir × e
  // Move a a small step in the direction of the force. (4)
  x_i = x_i + δ × dir
```

- Update rule: $x_i = x_i + \delta \times \left(rtt - \|x_i - x_j\|\right) \times \mathrm{u}(x_i - x_j).$

Vivaldi: Difficulties in simple algorithm

- Whether it convergences to the coordinate that predict the distance well
- Whether it convergences fast
- Both relate to the movement timestep: δ

- Adaptive timestep ($c_c < 1$)

$$\delta = c_c \times \frac{\text{local error}}{\text{local error} + \text{remote error}}$$

Vivaldi: Adaptive algorithm

// Incorporate new information: node j has been
// measured to be rtt ms away, has coordinates x_j,
// and an error estimate of e_j.
//
// Our own coordinates and error estimate are x_i and e_i.
//
// The constants c_e and c_c are tuning parameters.

vivaldi(rtt, x_j, e_j)

// Sample weight balances local and remote error. (1)

$w = e_i / (e_i + e_j)$ ← Confidence in remote node

// Compute relative error of this sample. (2)

$e_s = \left| \|x_i - x_j\| - rtt \right| / rtt$

// Update weighted moving average of local error. (3)

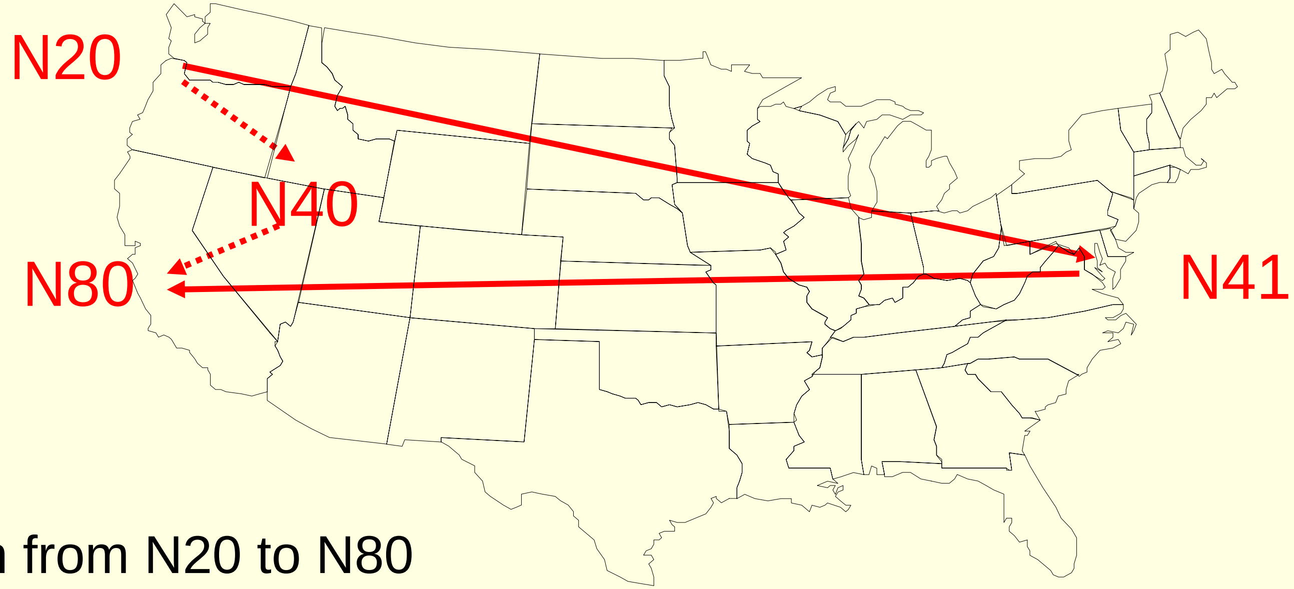
$e_i = e_s \times c_e \times w + e_i \times (1 - c_e \times w)$ ← Confidence in self

// Update local coordinates. (4)

$\delta = c_c \times w$ ← Adjust time step

$x_i = x_i + \delta \times \left(rtt - \|x_i - x_j\|\right) \times u(x_i - x_j)$

Exploiting proximity



- Path from N20 to N80
  - might usually go through N41
  - going through N40 would be faster
- In general, nodes *close* on ring may be *far apart* in Internet
- Knowing about proximity could help performance

Evaluation Methodology

- **Environment**
  - Packet-level network simulator using measured RTT values from the Internet
- **Latency data**
  - Matrix of inter-host Internet RTTs
  - Compute coordinates from a subset of these RTTs
  - Check accuracy of algorithm by comparing simulated results to full RTT matrix
  - 2 Data sets (Measured Data)
    - 192 nodes Planet Lab network, all pair-ping gives fully populated matrix
      - Median RTT = 76 ms
    - 1740 Internet DNS servers
      - Median RTT = 159 ms
      - populate full matrix using the King method
      - Continuously measure pairs over a week take median (other schemes just keep minim measured RTT since King can give estimates that are lower than actual RTT need to take median)
      - During collection of data need to make sure unwanted forwarding of name request did not occur (give RTT for the wrong name server)

Evaluation Methodology

- 2 Data sets (Synthetically generated Data)
  - Grid
    - Vivaldi accurately recovers RTT values but coordinates are translated and rotated from the original grids coordinates
  - ITM topology generation

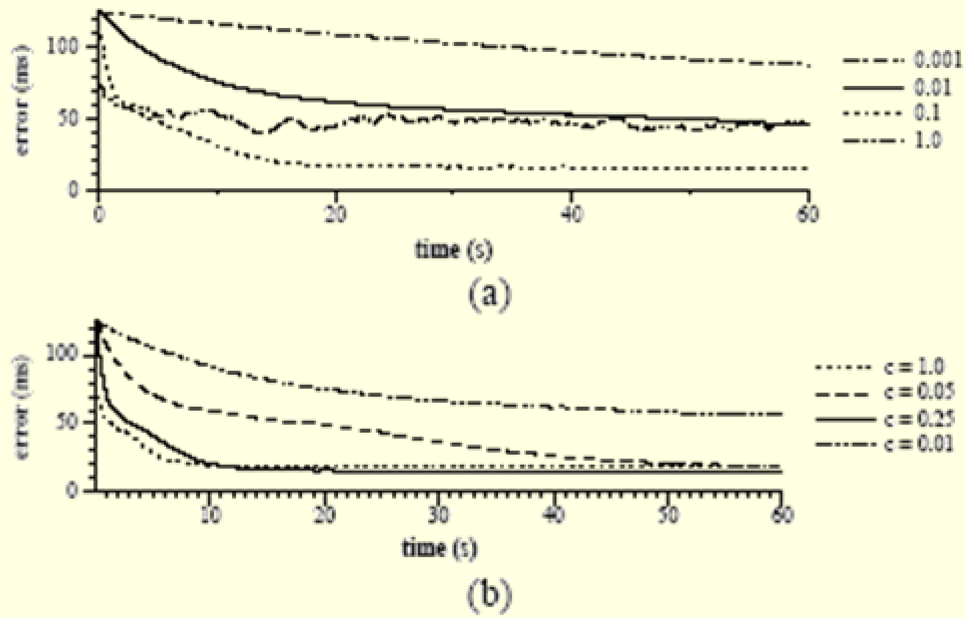
Using the Data

- Simulation test setup
  - Input RTT matrix
  - Send a packet one a second
  - Delay by ½ RTT time
    - Send RPC packet
    - Uses measured RTT of RPC to update coordinates
- Error definitions
  - Error of Link
    - Absolute difference between predicted RTT (coordinate math) and measured (RTT Matrix element)
  - Error of Node
    - Median of link errors involving this node
  - Error of System
    - Median of all node errors

Evaluation

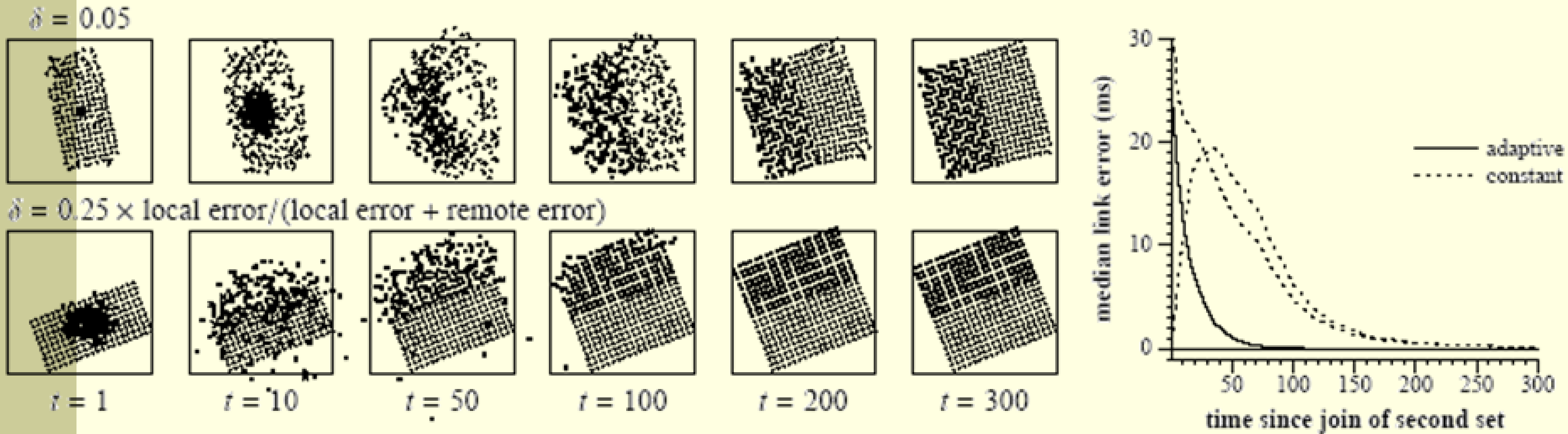
- Time-step choice

Empirically $c_c = 0.25$



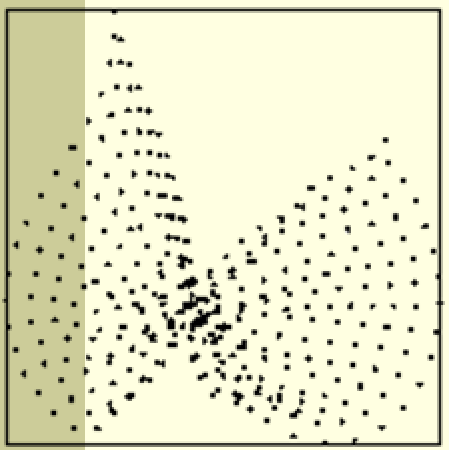
The effect of δ on rate of convergence. In (a), δ is set to one of the range of constants. In (b) δ is calculated with c_c values ranging from 0.01 to 1.0. The adaptive δ causes errors to decrease faster.

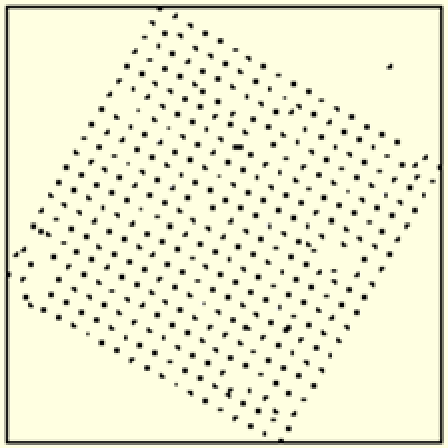
Evaluation (robustness against high-error nodes)

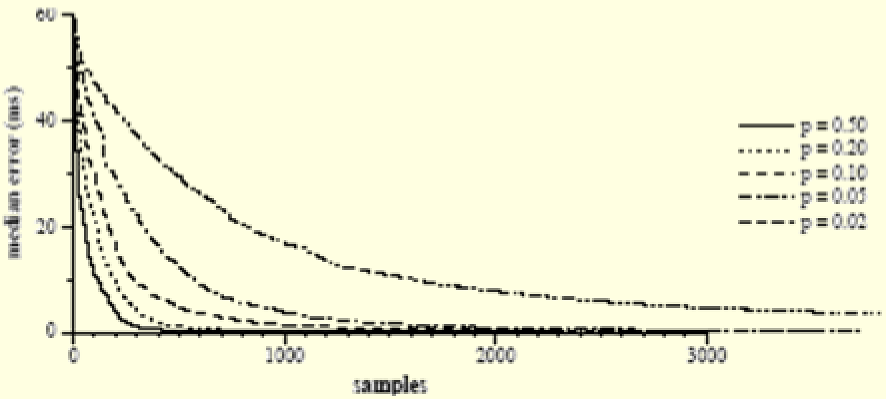


- Adding many new nodes that do not know their coordinates s, so are very uncertain (200 stable, then 200 new)
  - Constant delta, already certain node get knock away from there good coordinates
  - Adaptive delta, already certain nodes stay stable while new nodes move relatively quickly to their correct coordinates

Evaluation (Communication Patterns)

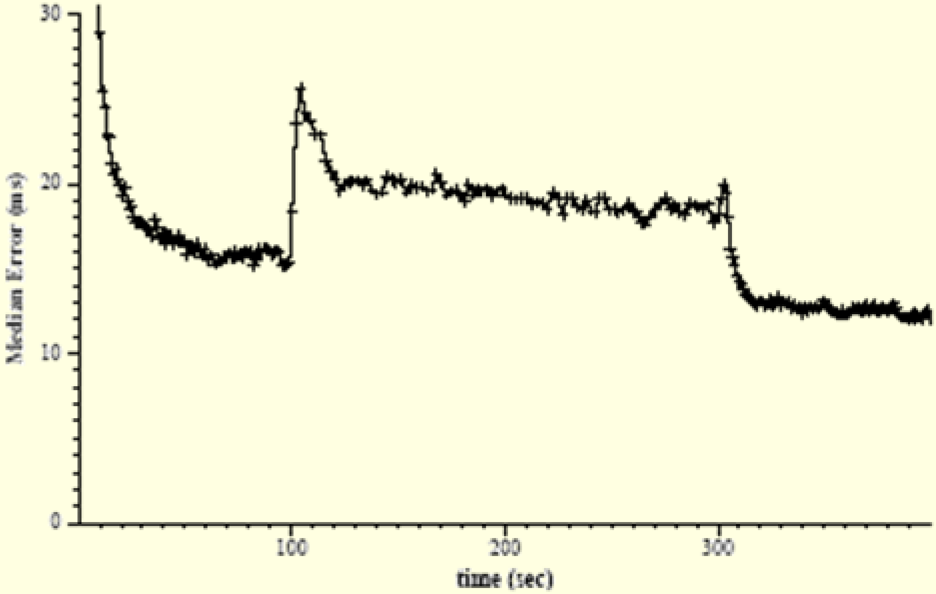






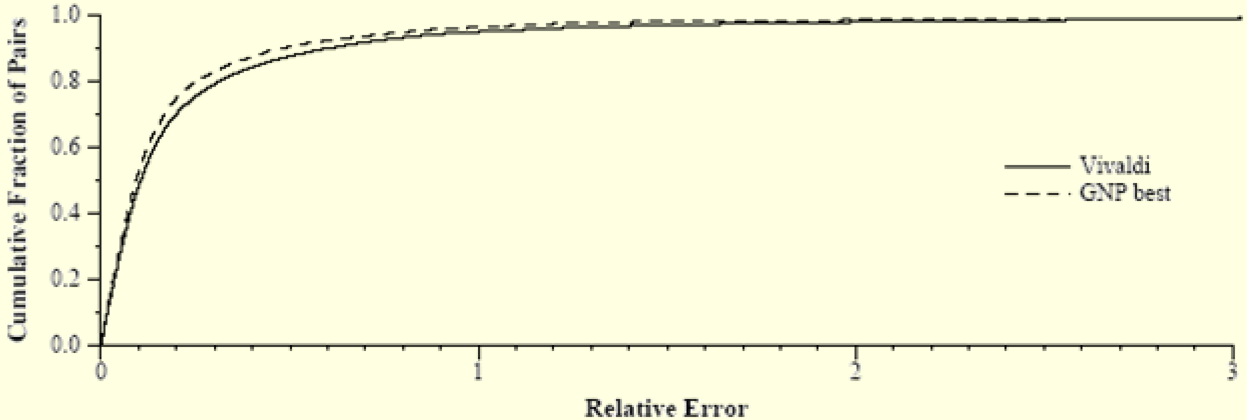
- In 21 (localization in sensor networks) shown that sampling only low latency nodes gives good local coordinates but poor global coordinates.
- 400 node sim (set 4 close neighbor, set 4 far neighbor) chose from far neighbor set is a probability p.
  - p = .5 quick convergence
  - p > .5 convergence slows
  - p < .5 convergence slows
  - no distant communication

Evaluation (Adapt to network changes)

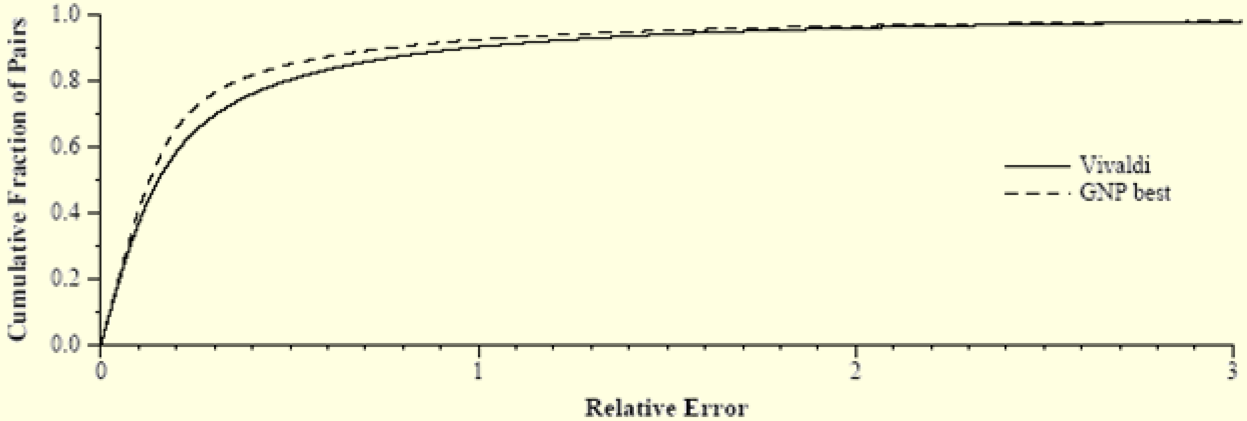


- Ability to adapt to changes in the network (tested with "Transit-Stub")
  - extend one stub by 10x
  - Put stub back

Evaluation (Accuracy vs. GNP)

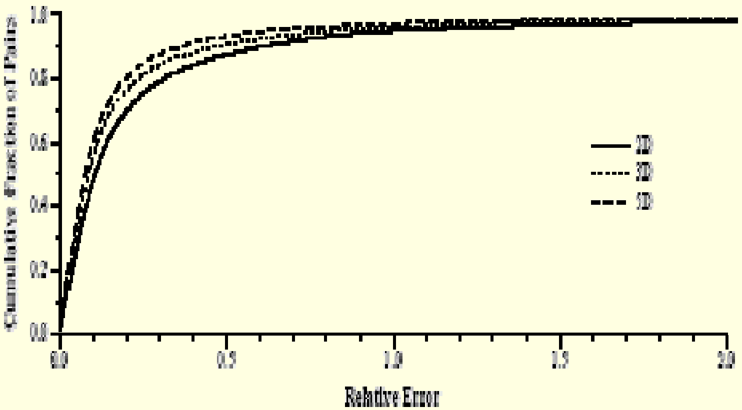


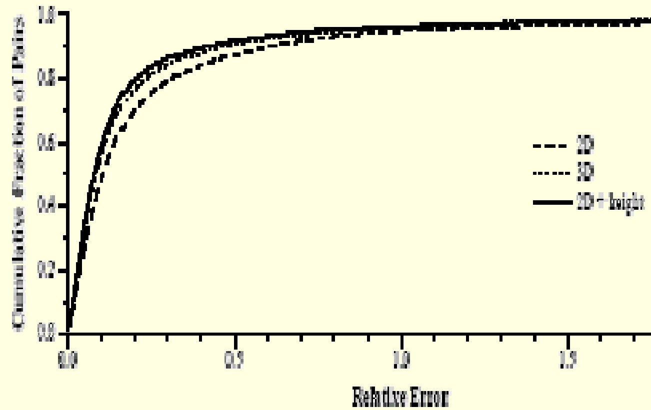
Planet Lab



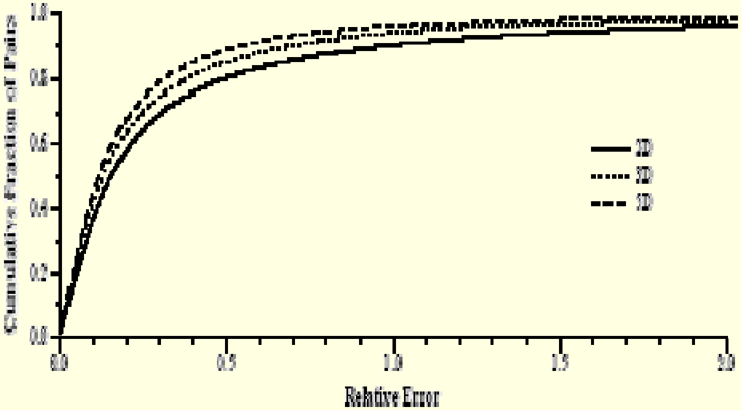
King

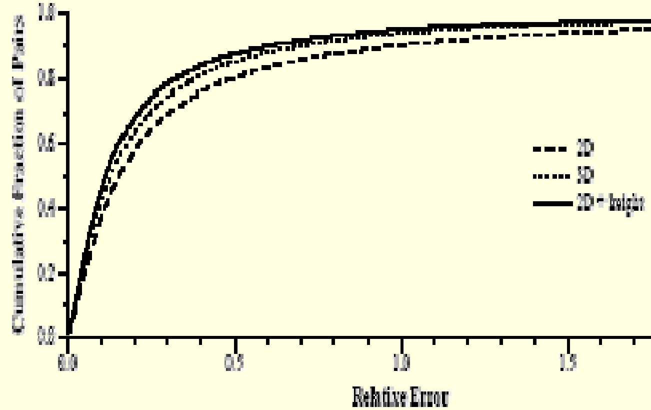
Model Selection





Planet Lab





King

Related Work

- Centralized Coordinate Systems
  - GNP
  - NPS
- Decentralized Internet Coordinate Systems
  - PIC
  - NPS
- Coordinate Systems for Wireless nets
  - AFL

Vivaldi: Points

- **Strong points**
  - Presents a simple, adaptive, decentralized algorithm for computing synthetic coordinates for Internet hosts to estimate latencies
  - Requires no fixed infrastructure, all nodes run the same algorithm
  - Converges to an accurate solution quickly
  - Maintains accuracy even as a large number of new hosts join the network that are uncertain of their coordinates
- **Bad points**
  - Limited scope of application area due to its dependency on traffic pattern
    - Applications communicating neighbors are less benefited from Vivaldi
  - The implication of delta(δ) is profound but no guidance provided
  - No proposed architecture for managing coordinates