

Supporting Network Coordinates on PlanetLab

Peter Pietzuch, **Jonathan Ledlie**, and
Margo Seltzer

Hourglass Project at Harvard

<http://www.eecs.harvard.edu/~syrah/hourglass>



WORLDS '05

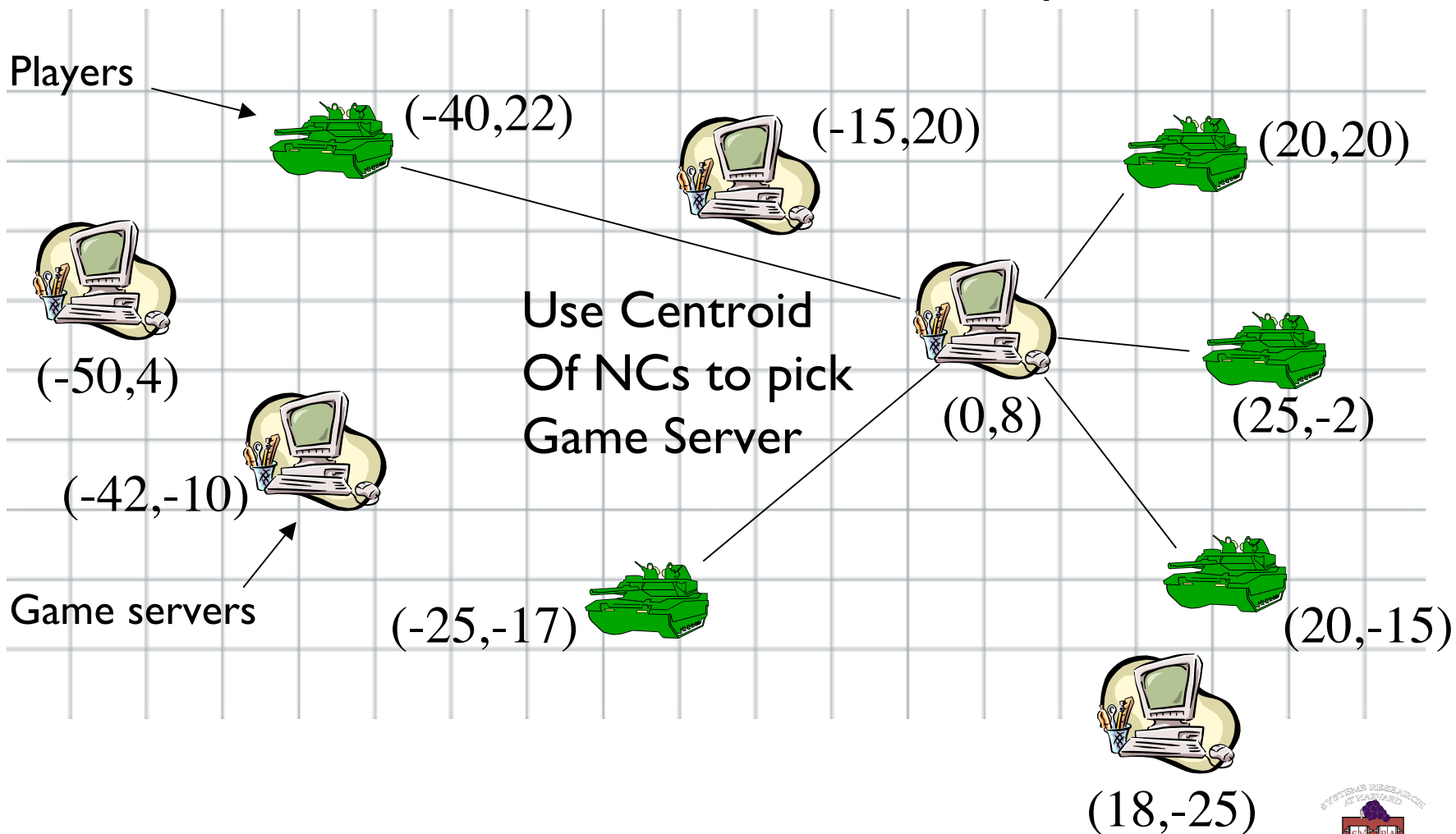
Outline

1. Review Network Coordinates
2. Discuss Two Implementation Problems and our Solutions
3. Conclude with plug for our NC service



Coordinates Simplify Distributed Systems Problems

Pick server with lowest mean latency



Coordinates are a Powerful Abstraction

Apply well-understood geometric algorithms
to problems in distributed systems

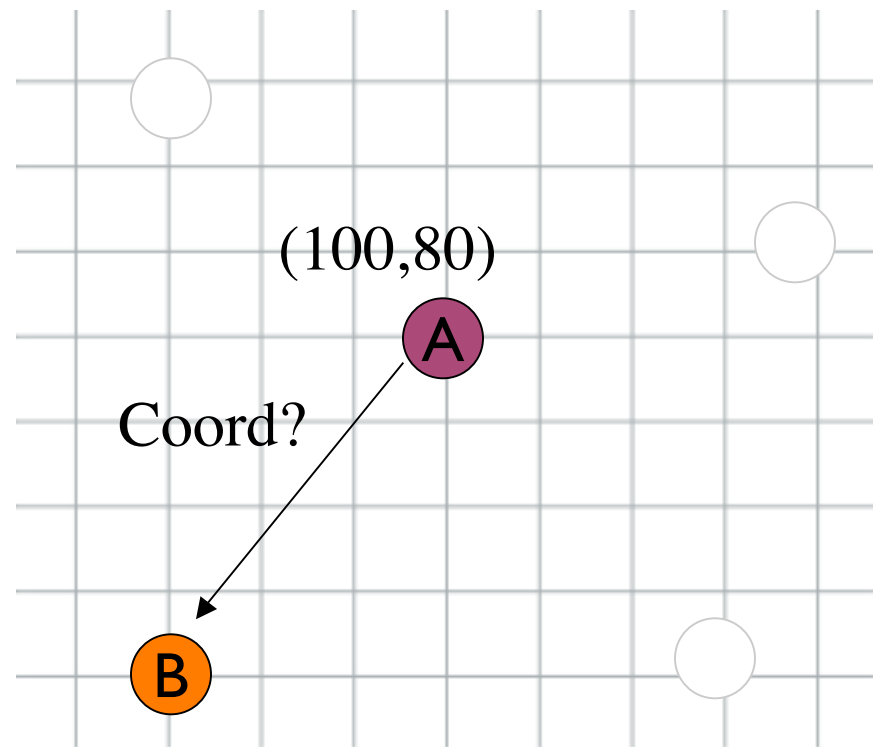
| Dist. Sys. Problem | Geometric Algorithm |
|---------------------------|--|
| Message Routing | Geographic routing Small world routing |
| Content distribution | Minimum spanning tree Cluster analysis |
| Resource location | Nearest neighbor query Geographic routing |
| Resource placement | Centroid calculation, Facility location, Steiner trees, Spring relaxation |
| Resource replication | Furthest neighbor query, Reverse nearest neighbor query, geographic quorums |
| Network analysis | Cluster analysis Principal Component Analysis |
| <i>Unknown Problem</i> | Travelling salesman problem Minimum length matching |



How Network Coordinates Work (I)

Goal: Minimize global latency prediction error

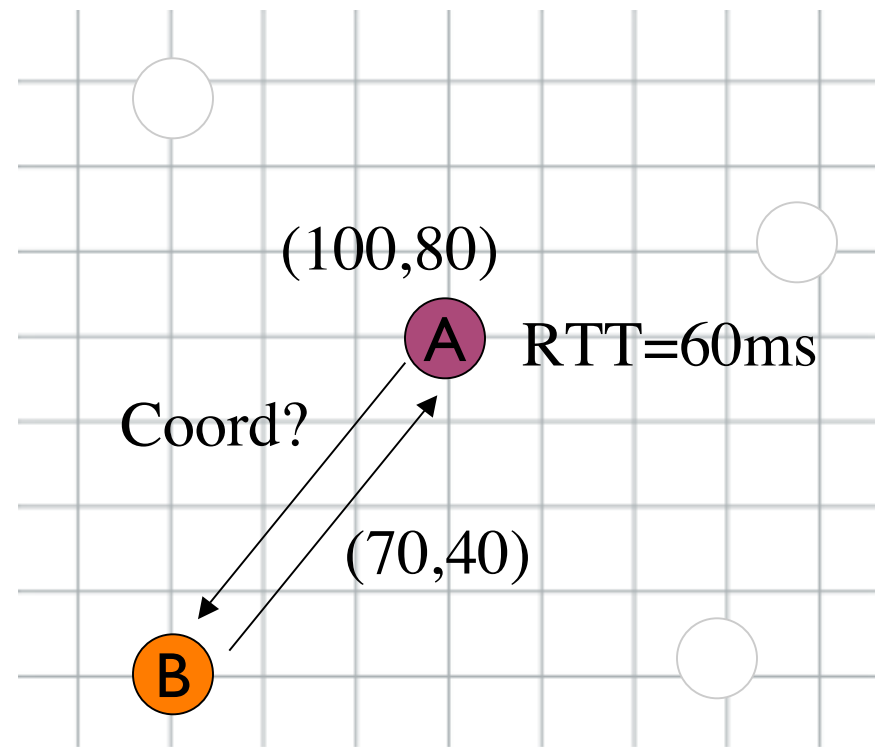
- I. **A** starts measurement to **B**.



How Network Coordinates Work (II)

Goal: Minimize global latency prediction error

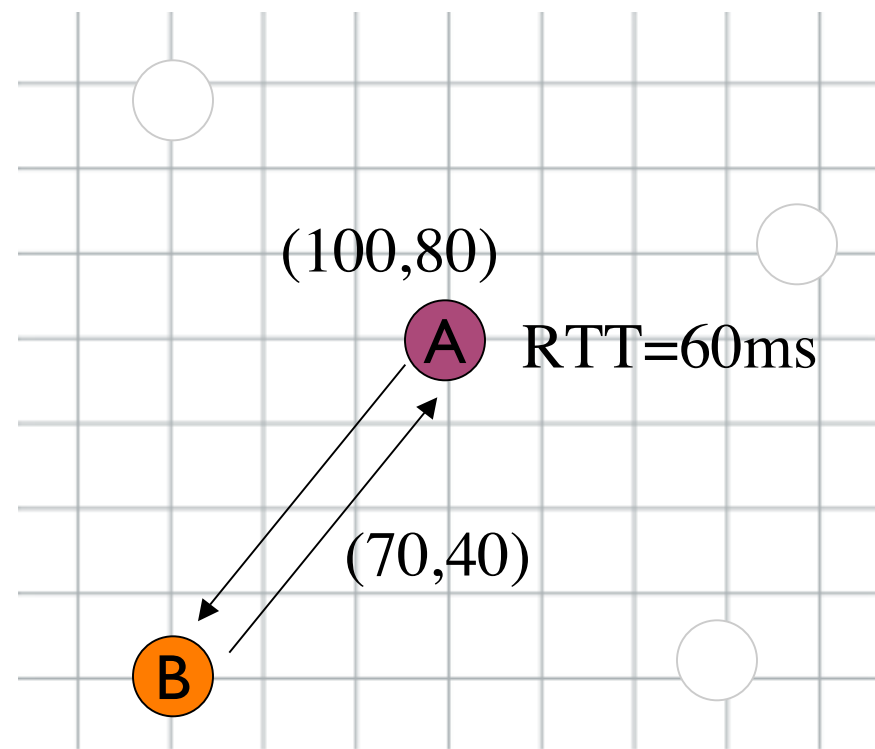
1. **A** starts measurement to **B**.
2. **B** replies with its coord.
A deduces RTT.



How Network Coordinates Work (III)

Goal: Minimize global latency prediction error

1. **A** starts measurement to **B**.
2. **B** replies with its coord.
A deduces RTT.
3. **A** computes estimate and error.



$$\text{Estimate} = |(100,80)-(70,40)|=50\text{ms}$$

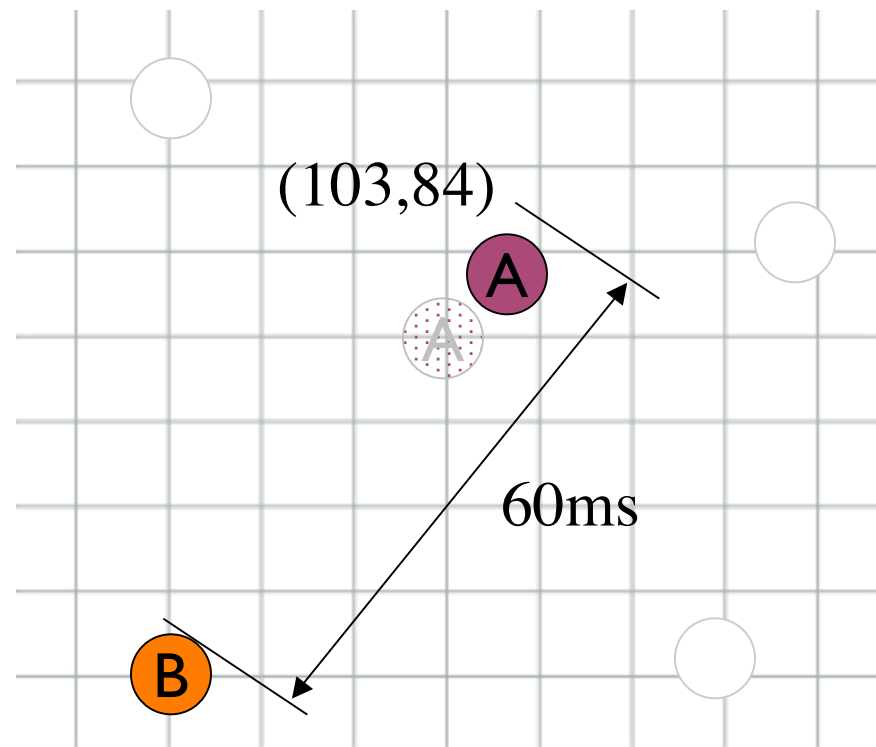
$$\text{Error} = (60 - \text{Estimate}) = 10\text{ms}$$



How Network Coordinates Work (IV)

Goal: Minimize global latency prediction error

1. **A** starts measurement to **B**.
2. **B** replies with its coord.
A deduces RTT.
3. **A** computes estimate and error.
4. **A** moves toward ideal coord, relative to **B**.



$$\text{Estimate} = |(100,80) - (70,40)| = 50\text{ms}$$

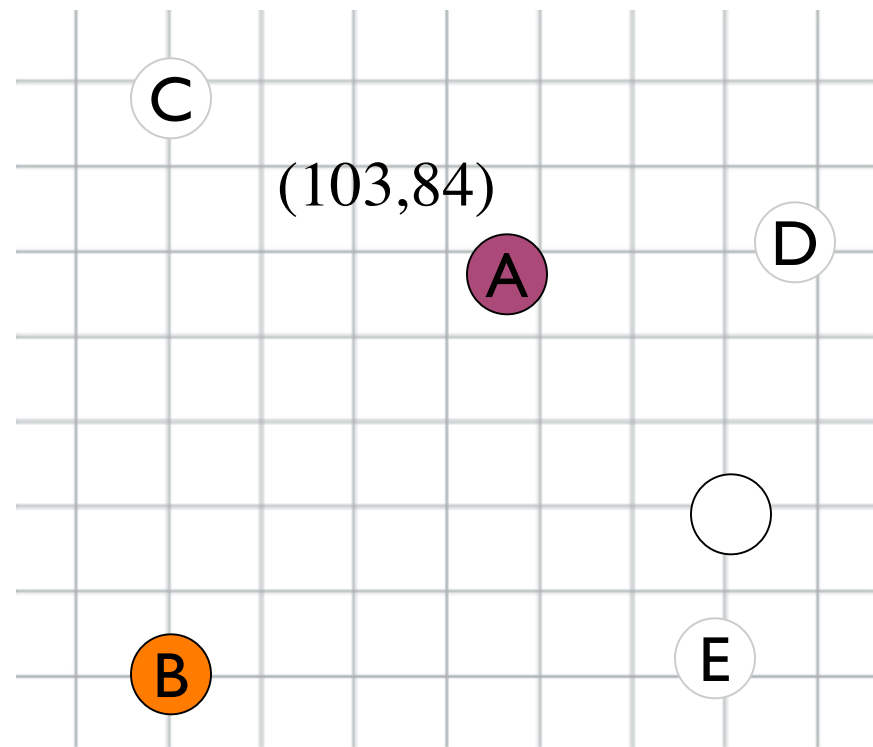
$$\text{Error} = (60 - \text{Estimate}) = 10\text{ms}$$



How Network Coordinates Work (V)

Goal: Minimize global latency prediction error

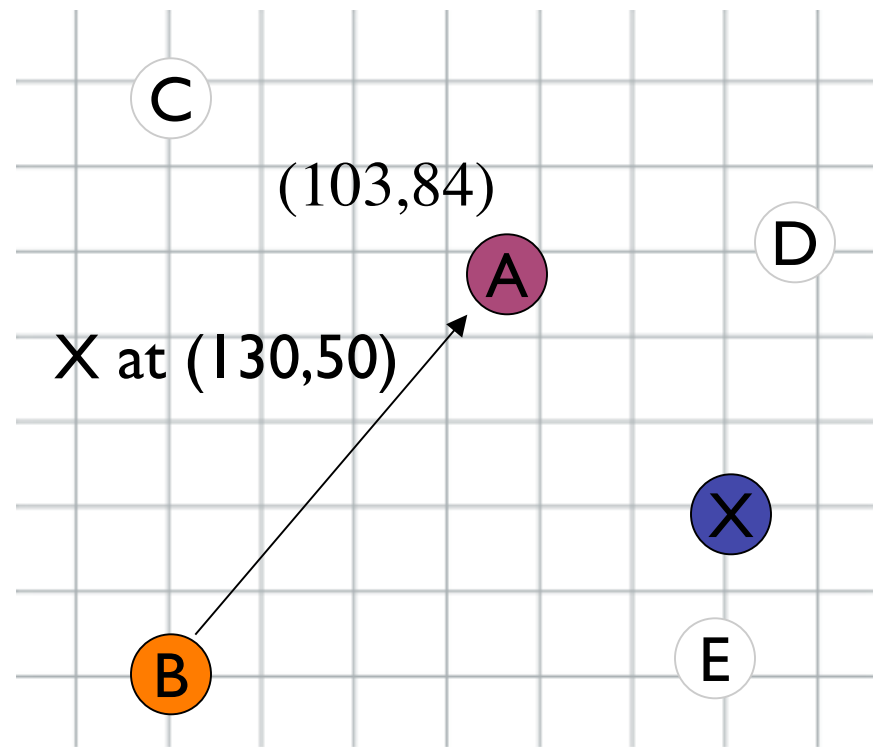
1. **A** starts measurement to **B**.
2. **B** replies with its coord. **A** deduces RTT.
3. **A** computes estimate and error.
4. **A** moves toward ideal coord, relative to **B**.
5. Repeat with C, D, E.



How Network Coordinates Work (VI)

Goal: Minimize global latency prediction error

1. **A** starts measurement to **B**.
2. **B** replies with its coord. **A** deduces RTT.
3. **A** computes estimate and error.
4. **A** moves toward ideal coord, relative to **B**.
5. Repeat with C, D, E.
6. Predict to **X**



Outline

1. Review Network Coordinates
2. Discuss Two Implementation Problems and our Solutions
3. Conclude with plug for our NC service

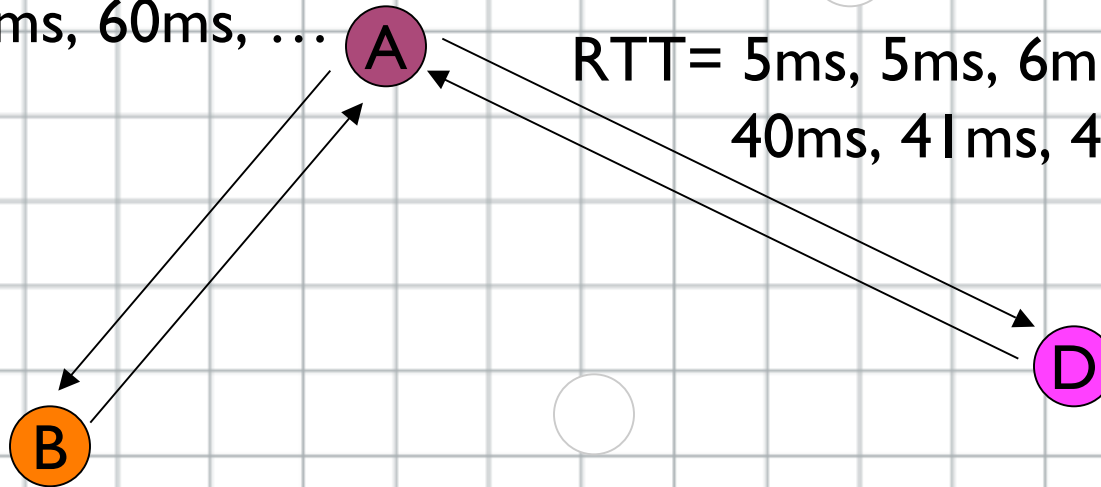


Problem #1: Latencies are not Static

Latencies measurements have errors and change

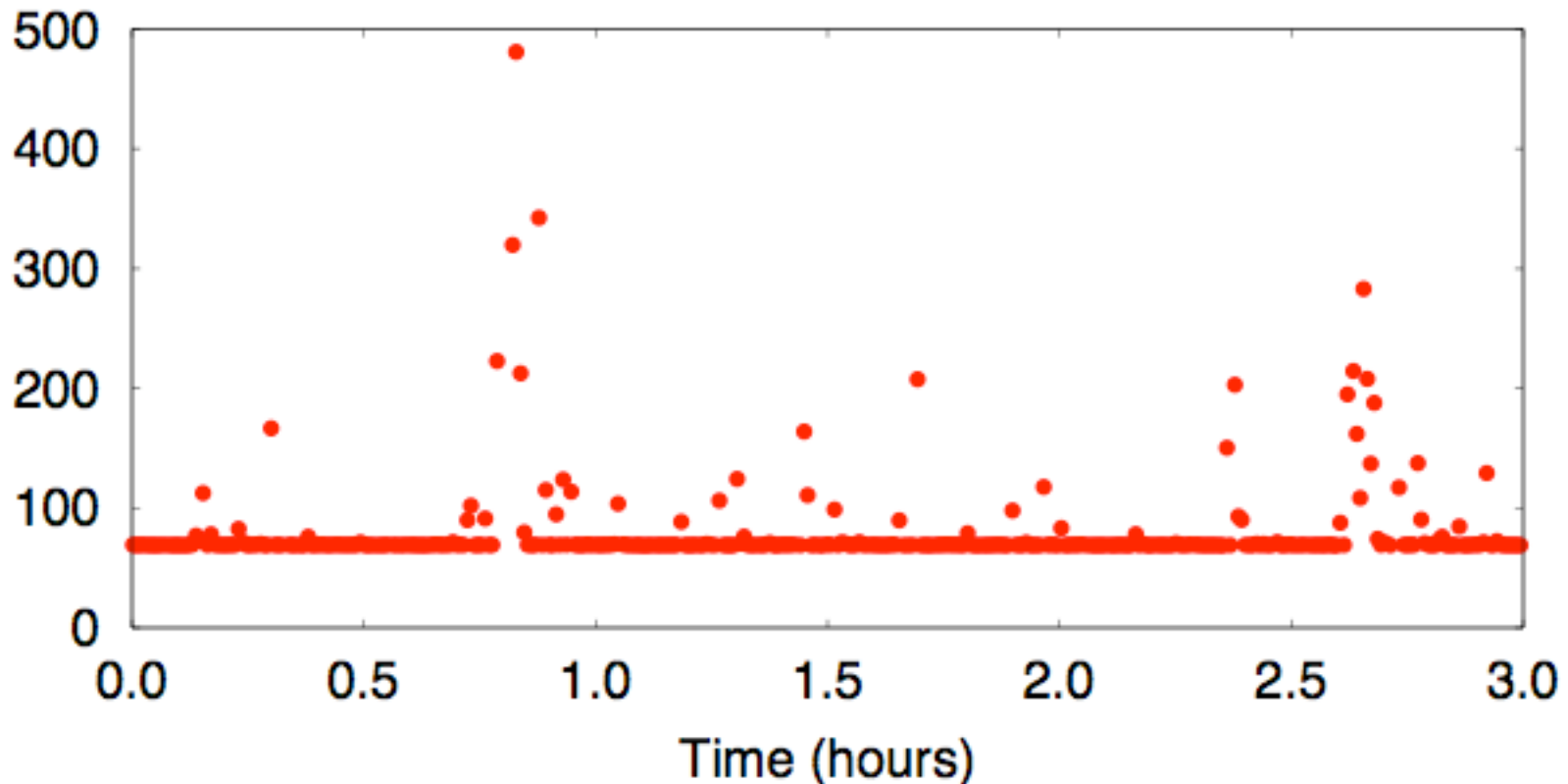
RTT = 60ms, 61ms, 59ms,
1000ms, 70ms, 60ms, ...

RTT = 5ms, 5ms, 6ms,
40ms, 41ms, 40ms, ...



Problem #1 (a): Errors are Unpredictable

Three hours of measurements from berkeley to uvic.ca



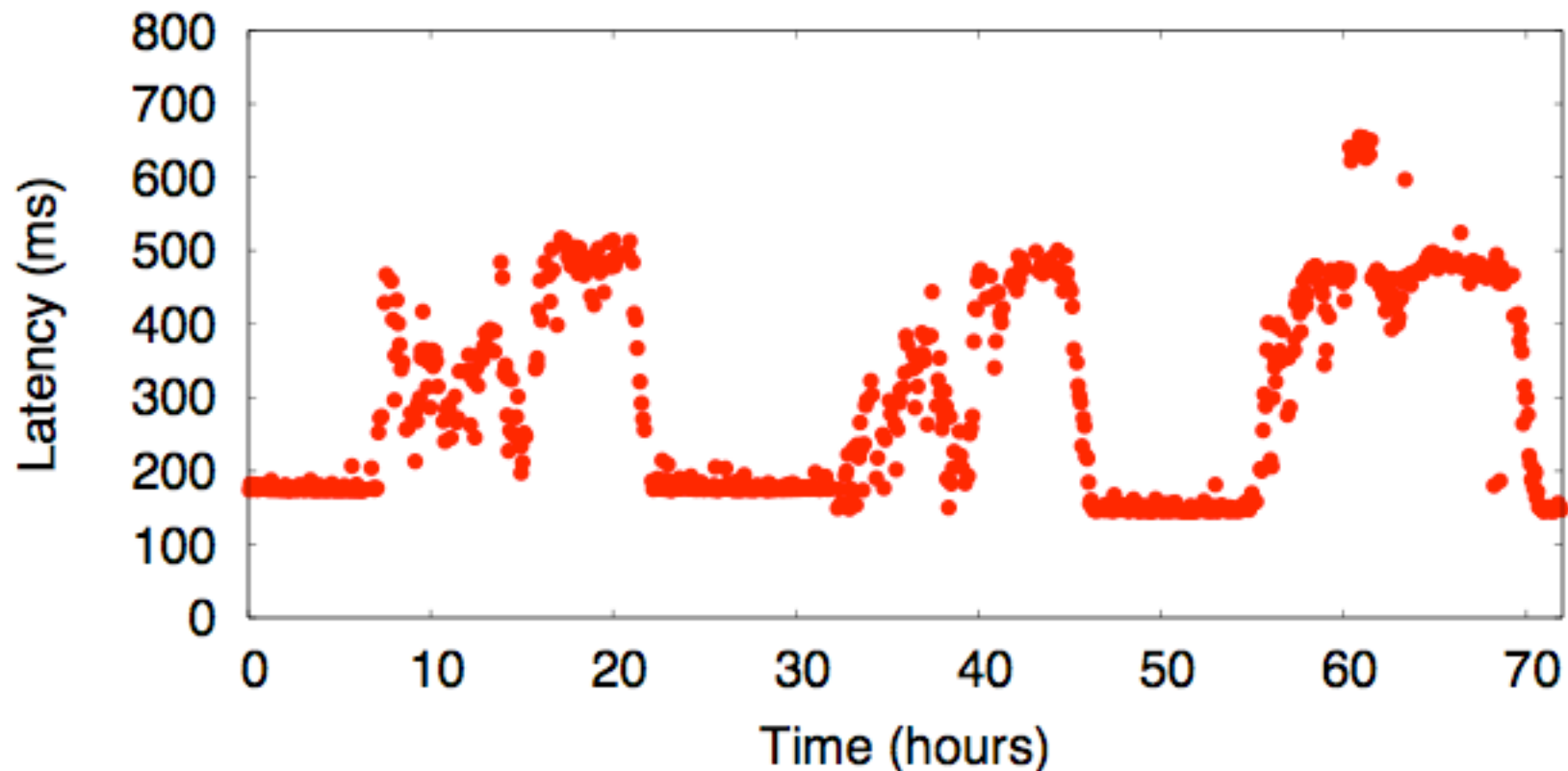
82% of measurements within 1ms of median



WORLD'S '05

Problem #1 (b): Latencies can change

Three days of measurements from ntu.edu.tw to 6planetlab.edu.cn



Need to remove noise, but remain adaptive



Solution #1: Moving Minimum as Latency Filter

- Remove outliers and respond to latency change
- Minimum of previous four samples worked best

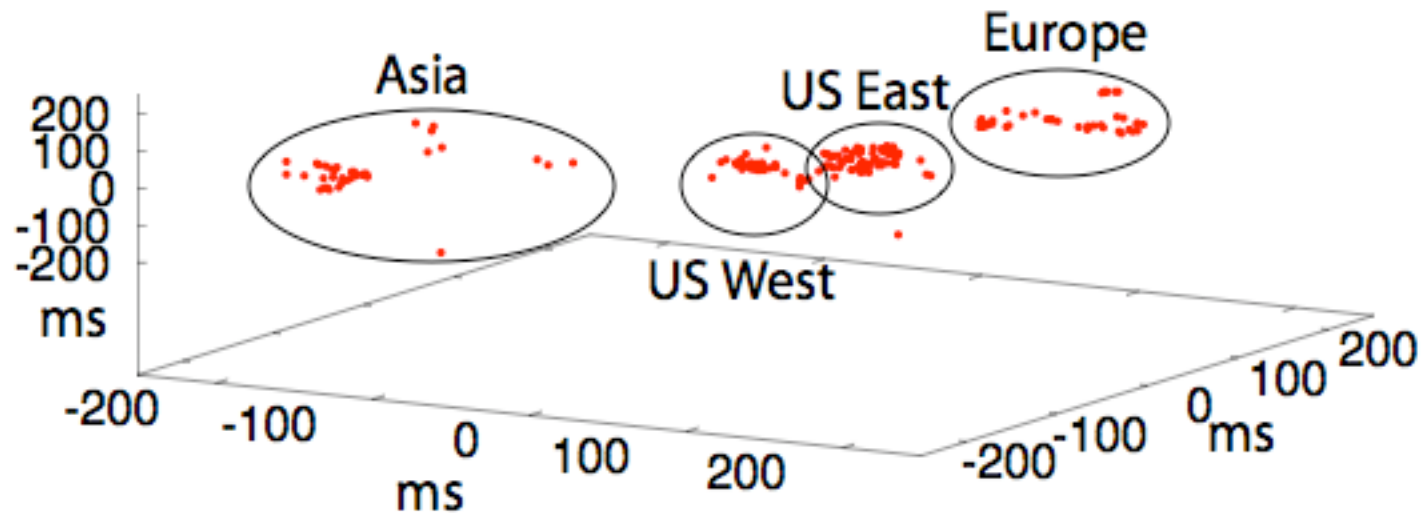
| Time | age | newest | ← | → | oldest |
|-------|-----------------------|--------|---|---|--------|
| t_0 | ms | 60 | | | 62 |
| t_1 | Receive 1000ms RTT | 1000 | | | 59 |
| t_2 | Receive 70ms RTT | 70 | | | 61 |

Other simple techniques did not work.



Network Coordinates on PlanetLab

Network Coordinates of 226 PlanetLab Nodes



- Points represent locations of PL nodes in 3-d relative coordinate space
- We used Vivaldi, but both problems inherent in any NC implementation



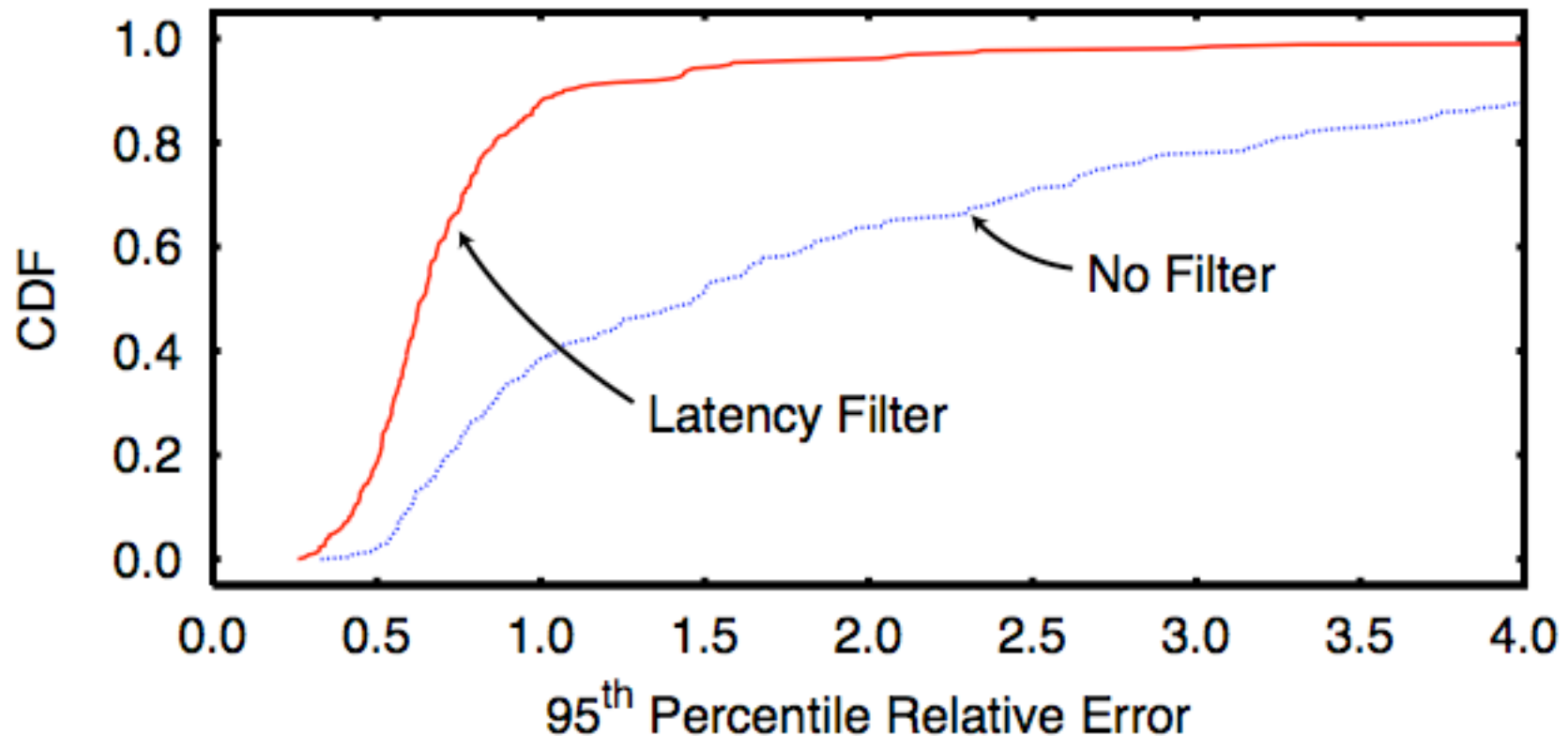
Latency Filters in Practice: Video Comparison

- Raw Vivaldi Coordinates vs. Latency Filter
- Latency Filters eliminate outliers that cause distortion of many coords all at once (e.g. minute 38 of the video)



Latency Filters: Accuracy

270 PlanetLab nodes; last two of four hour trial.
Graph shows relative error experienced by each node.

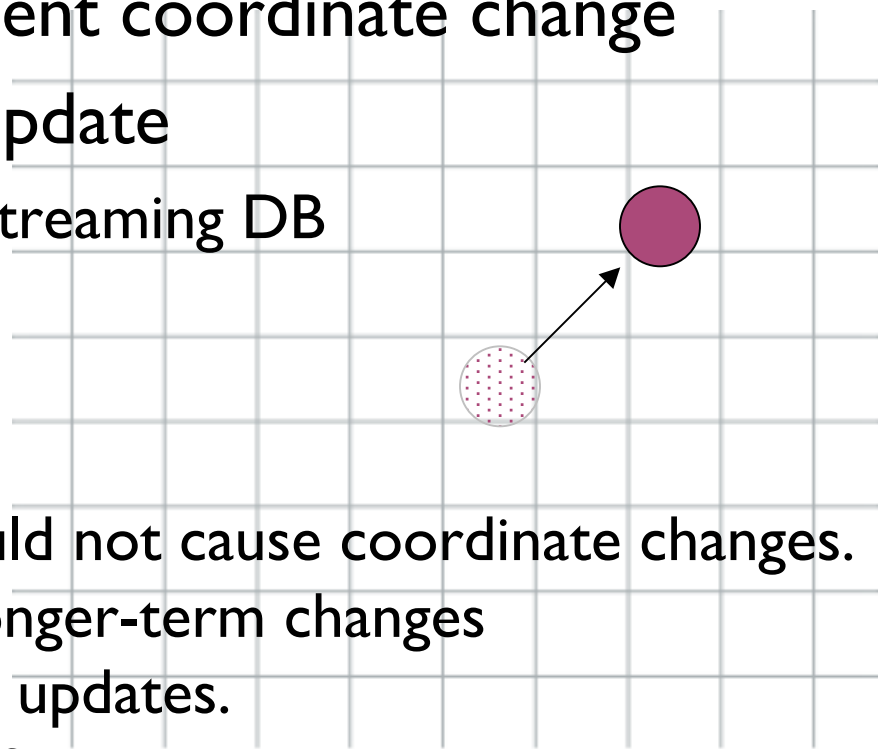


$$\text{Relative Error} = |\text{RTT-prediction}|/\text{RTT}$$



Problem #2: Changing a Coordinate is Expensive

- Even with latency filter, frequent coordinate change
- App-specific cost for coord update
 - Cause operator migration in streaming DB
 - Routing table change
- Short-term variations in RTT should not cause coordinate changes.
 - However, still need to track longer-term changes in network topology, e.g., BGP updates.
- Is it possible to tell apps less frequently and retain high accuracy?



Solution #2: Coordinate Windows as Update Filters

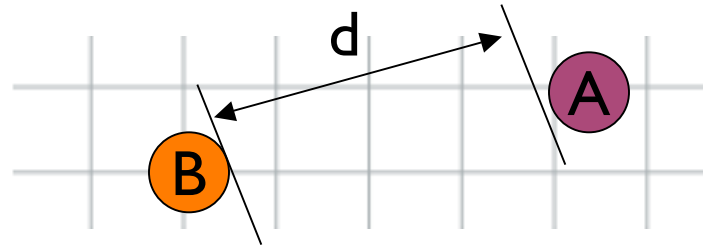
1. Keep history of recent coordinates
 2. Divide history into two windows (sets):
current (newest) and *start* (oldest)
 3. When *current* and *start* diverge (by some metric), update application with new coordinate
- Two metrics:
 - Local Relative Distance
 - Energy



Update Filters: *Local Relative Distance* Heuristic (I)

Base update on distance moved relative to nearest neighbor

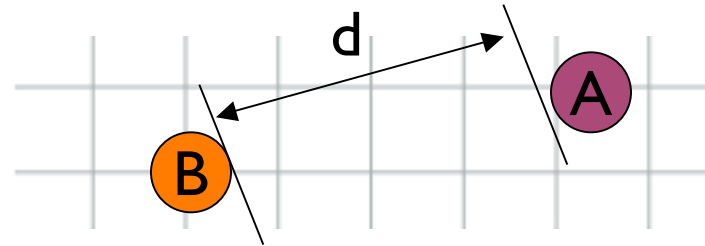
I. Remember nearest known neighbor



Update Filters: *Local Relative Distance* Heuristic (2)

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s



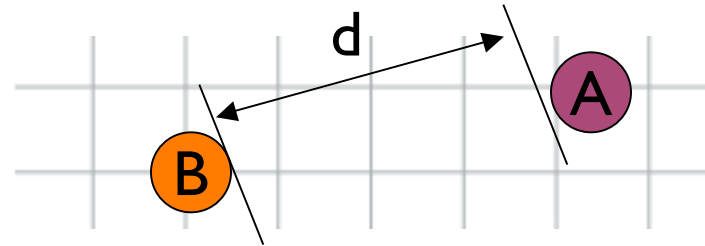
Current W_c



Update Filters: *Local Relative Distance* Heuristic (3)

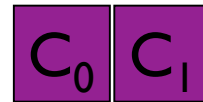
Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s



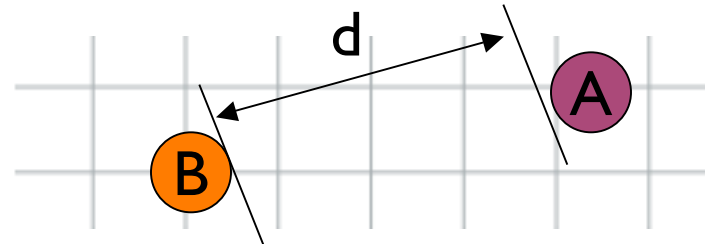
Current W_c



Update Filters: *Local Relative Distance* Heuristic (4)

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s



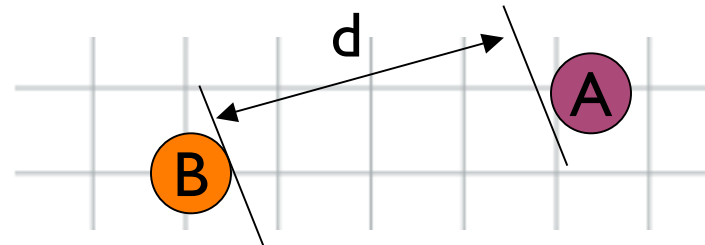
Current W_c



Update Filters: *Local Relative Distance* Heuristic (5)

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s



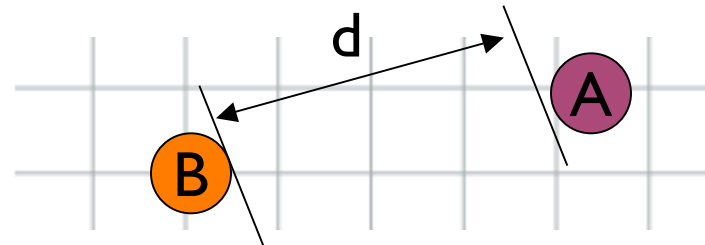
Current W_c



Update Filters: *Local Relative Distance* Heuristic (6)

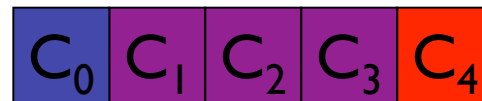
Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

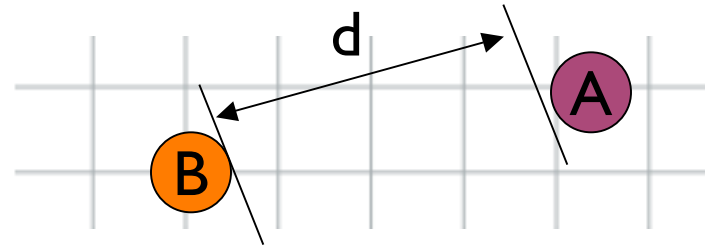
Start W_s Current W_c



Update Filters: *Local Relative Distance* Heuristic (7)

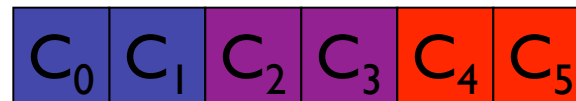
Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s Current W_c



3. Compare centroids of windows

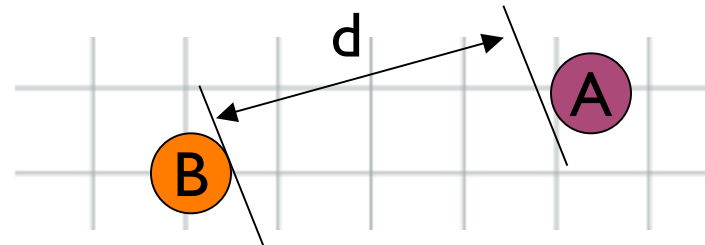
If $\text{Centroid}(W_s) - \text{Centroid}(W_c) > d \times \epsilon$



Update Filters: *Local Relative Distance* Heuristic (8)

Base update on distance moved relative to nearest neighbor

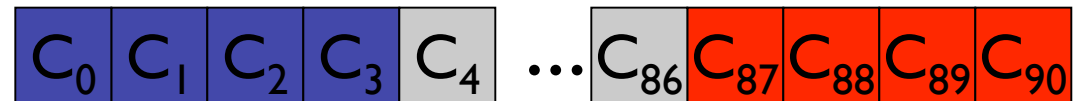
1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s

Current W_c



3. Compare centroids of windows

If $\text{Centroid}(W_s) - \text{Centroid}(W_c) > d \times \epsilon$

4. Update app-level coordinate

App coord = $\text{Centroid}(W_c)$



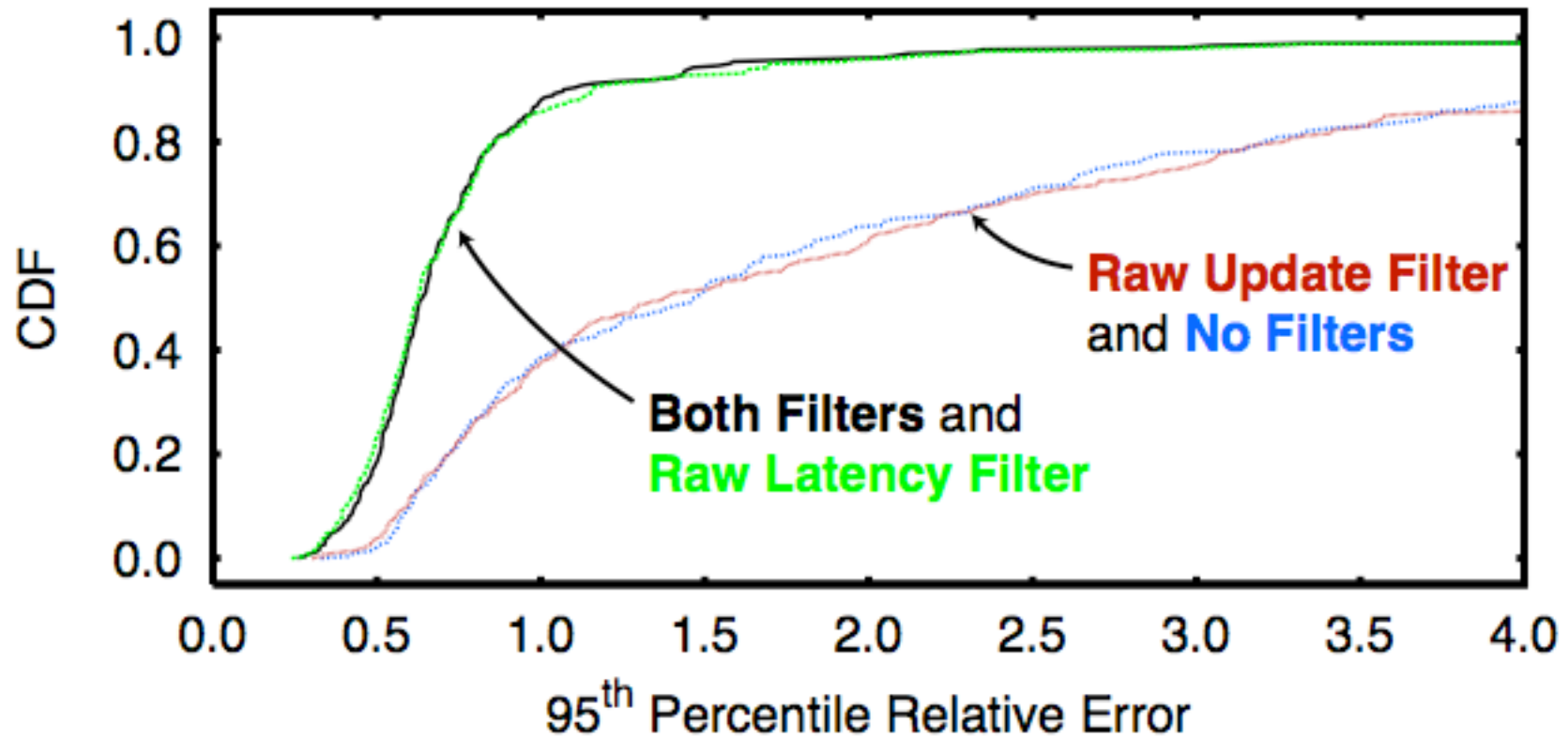
WORLD'S '05

Video Comparison

- Both Filters Combine for a much more stable set of coordinates.
- Same 10 minute segment as before.



Update Filters: Greater Stability, Same Accuracy



270 PlanetLab nodes; last two of four hour trial.
Graph shows relative error experienced by each node.



Other Results

- Application-oriented metrics: Relative Rank Loss (Lua et al.), two new metrics
- Effect of neighbor set size
- Impact of triangle inequality violations on stability

- Using Network Coordinates for Distributed Query Operator Placement (IPTPS '05, NetDB '05, ICDE '06)



Conclusion: They work, go use them

NCs work in the “real world” with the addition of two techniques:

1. Latency filters - Summarize current link latency
2. Update filters - Tell app only when necessary

NC service available on January 1, 2006

- Proxy coords for non-PL nodes soon after

Go use ‘em!

Questions?

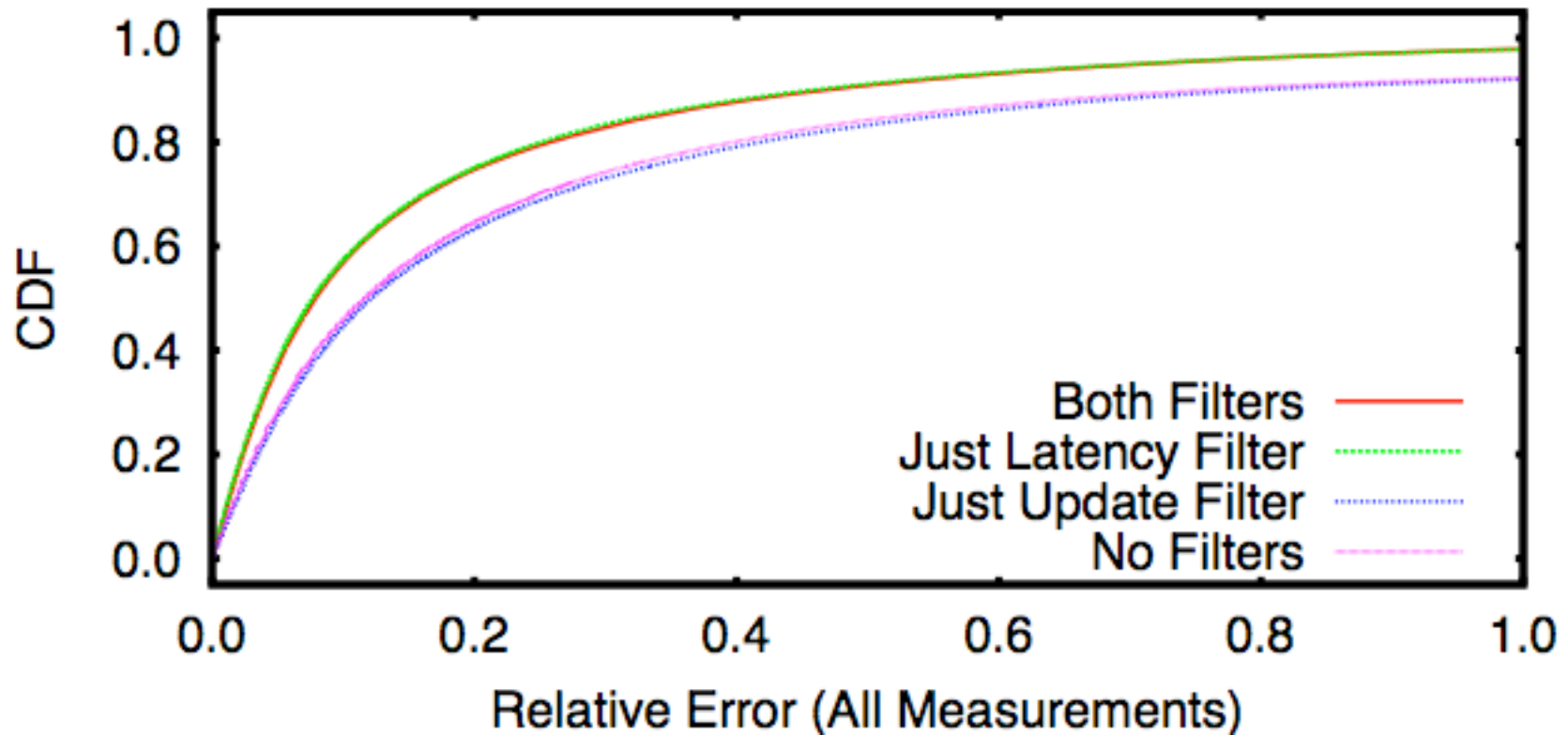
Hourglass Project at Harvard

<http://www.eecs.harvard.edu/~syrah/hourglass>



WORLDS '05

Relative Error on PlanetLab



Second half of four hour run; June 24, 2005;
270 nodes; Medians: 8% and 12%



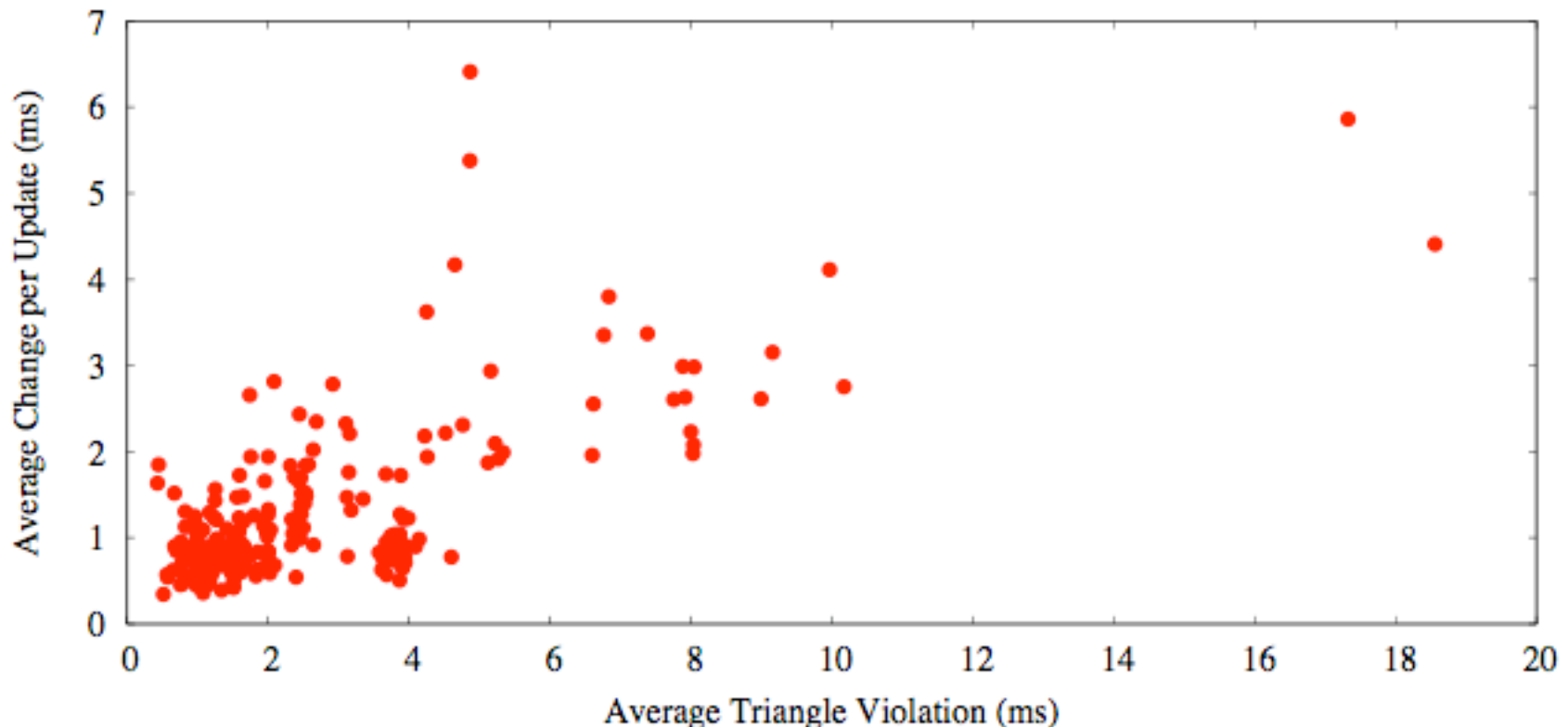
Network Coordinate Service

- **API:**
 - double estimateLatency (double[] remoteNC)
 - double[] getNC ()
 - double getConfidence ()
 - double getRelativeError ()
 - double forceUpdate (IP remoteNode)
- **Available January 1, 2006**
- **XML/RPC interface**



Future Work: Triangle Violations and Instability

- Do these filters matter on a larger network? -- Azureus
- Strong correlation between extent of triangle violations and stability



Application-Oriented Metrics

Example of Incorrect Ranking: $RTT(X,A) = 50$, $RTT(X,B) = 60$

RRL:

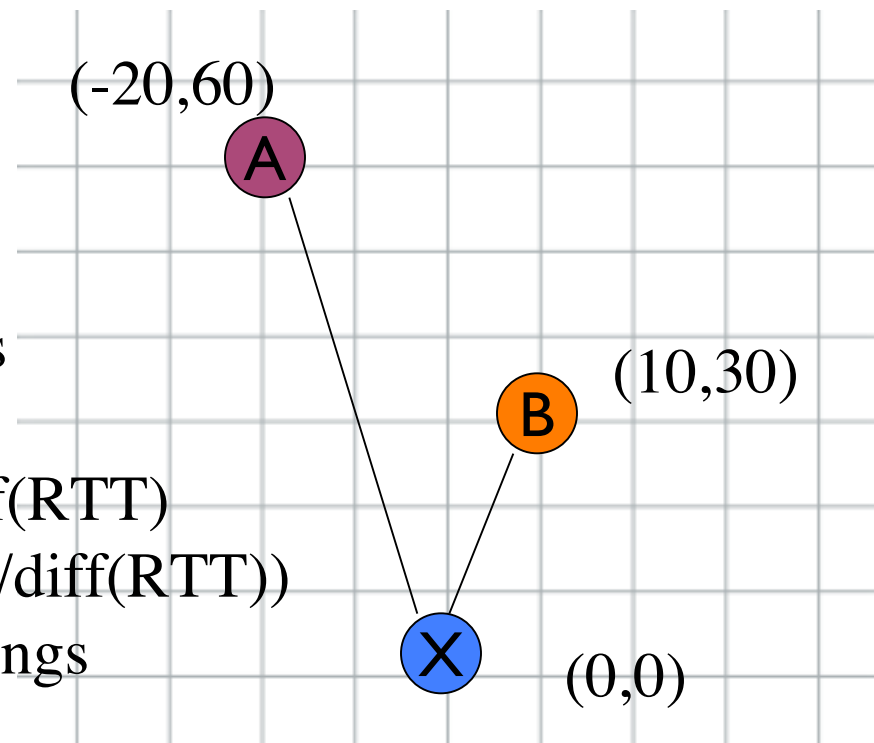
- incorrect: add 1
- divide by number of pairs
- percent of incorrect rankings

WRRL:

- incorrect: add $\text{diff}(RTT)/\text{diff}(RTT)$
- divide by sum of $(\text{diff}(RTT)/\text{diff}(RTT))$
- magnitude of incorrect rankings

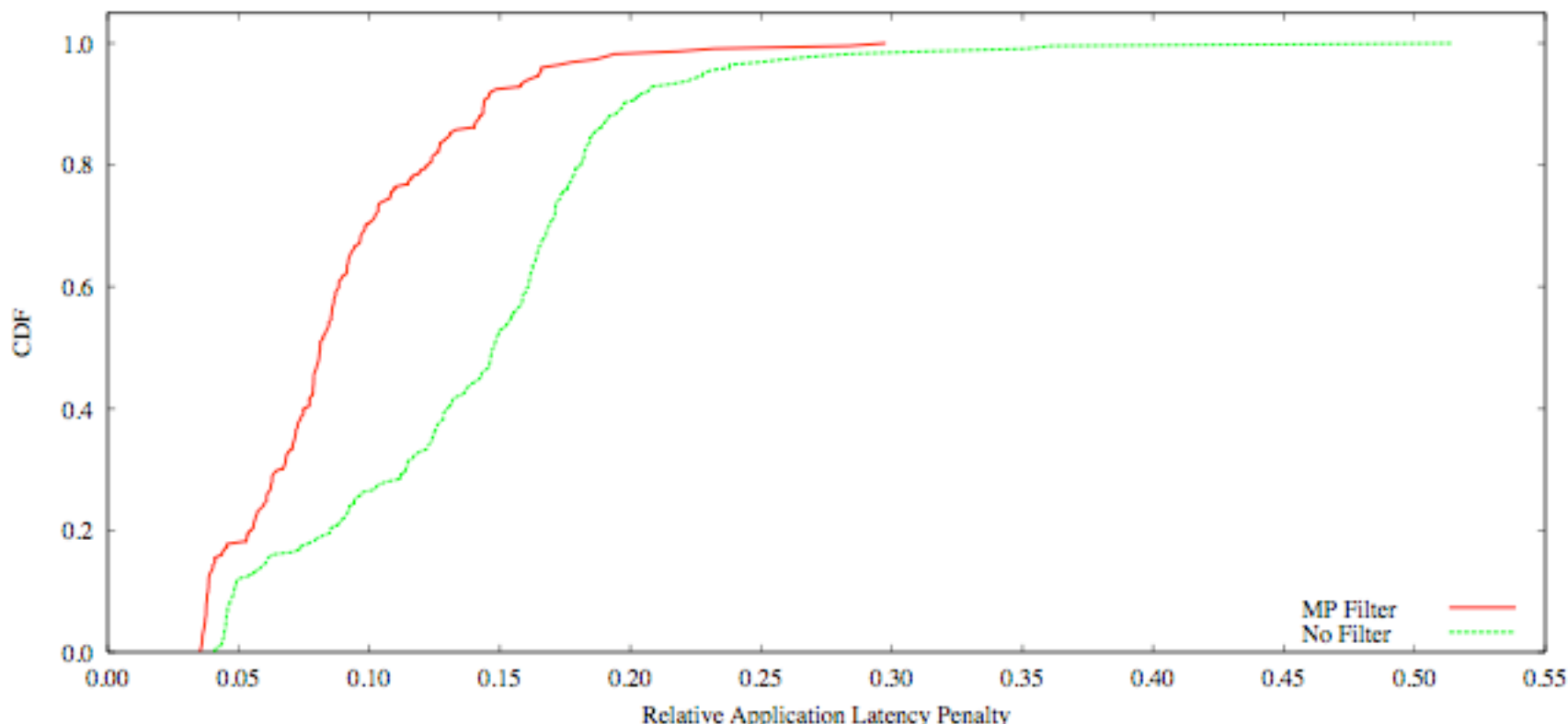
RALP:

- incorrect: add pct loss e.g. $(RTT(X,B)-RTT(X,A))/RTT(X,A)$
- I.e. percentage slow-down for incorrect coordinates
- reflects app-perceived penalty for using coordinates



Application-Oriented Metric: RALP

- Relative Application Latency Penalty

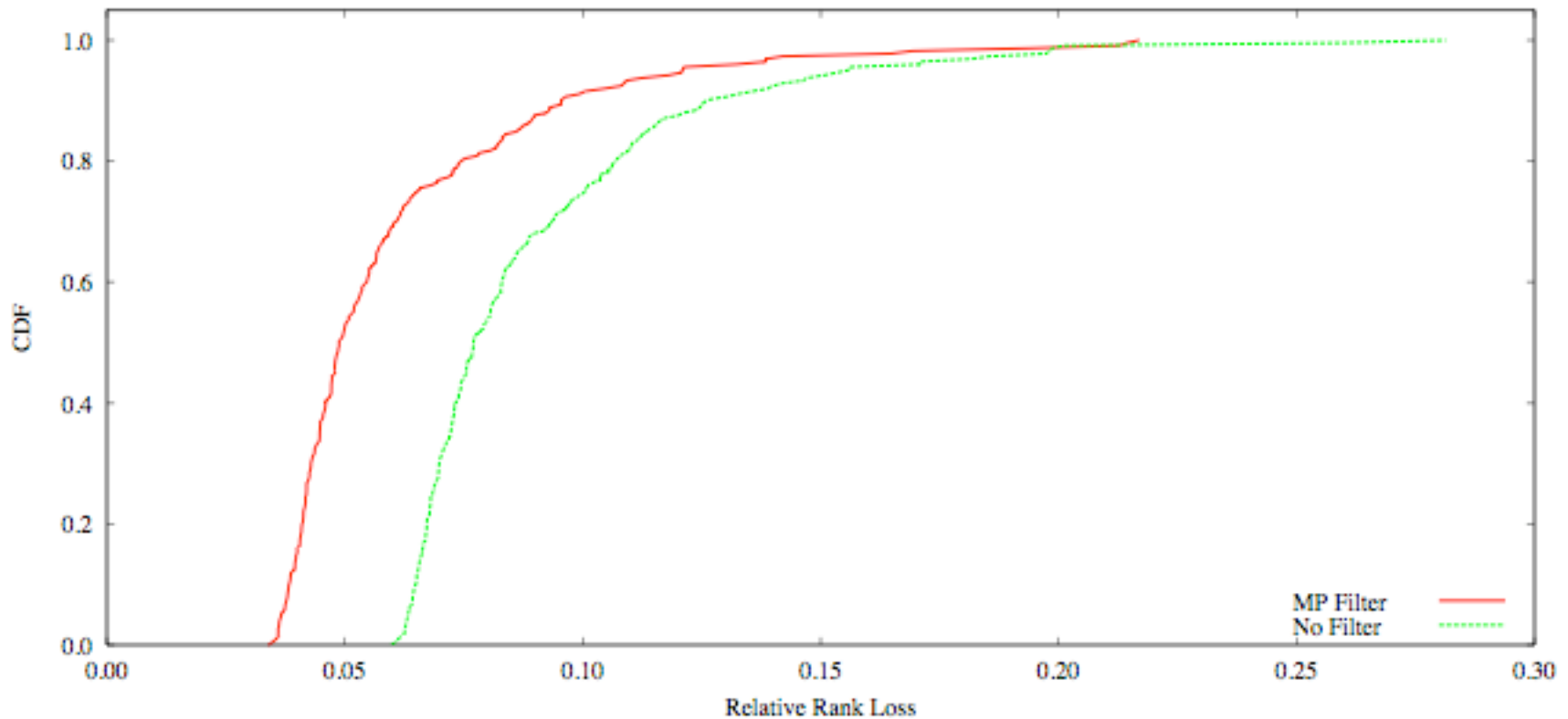


Percentage of the actual latency that you are penalized by having a mis-ordering, for all pairs of links.



Application-Oriented Metric: RRL

- Relative Rank Loss: probability of incorrect ranking

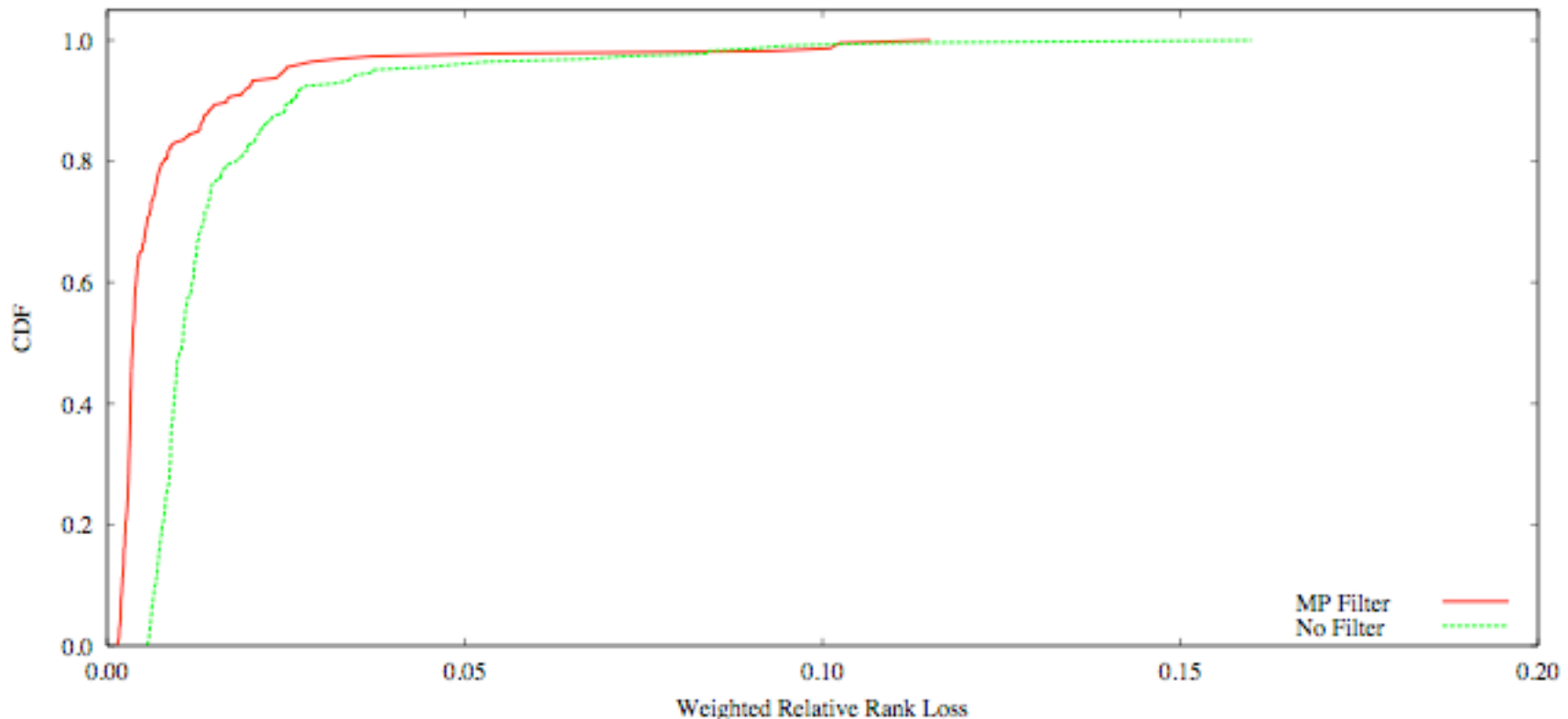


Percent of incorrect orderings of pairs of nodes.
Four hour PL trace, 226 nodes.



Application-Oriented Metric: WRRL

- Weighted Relative Rank Loss: magnitude of incorrect rankings

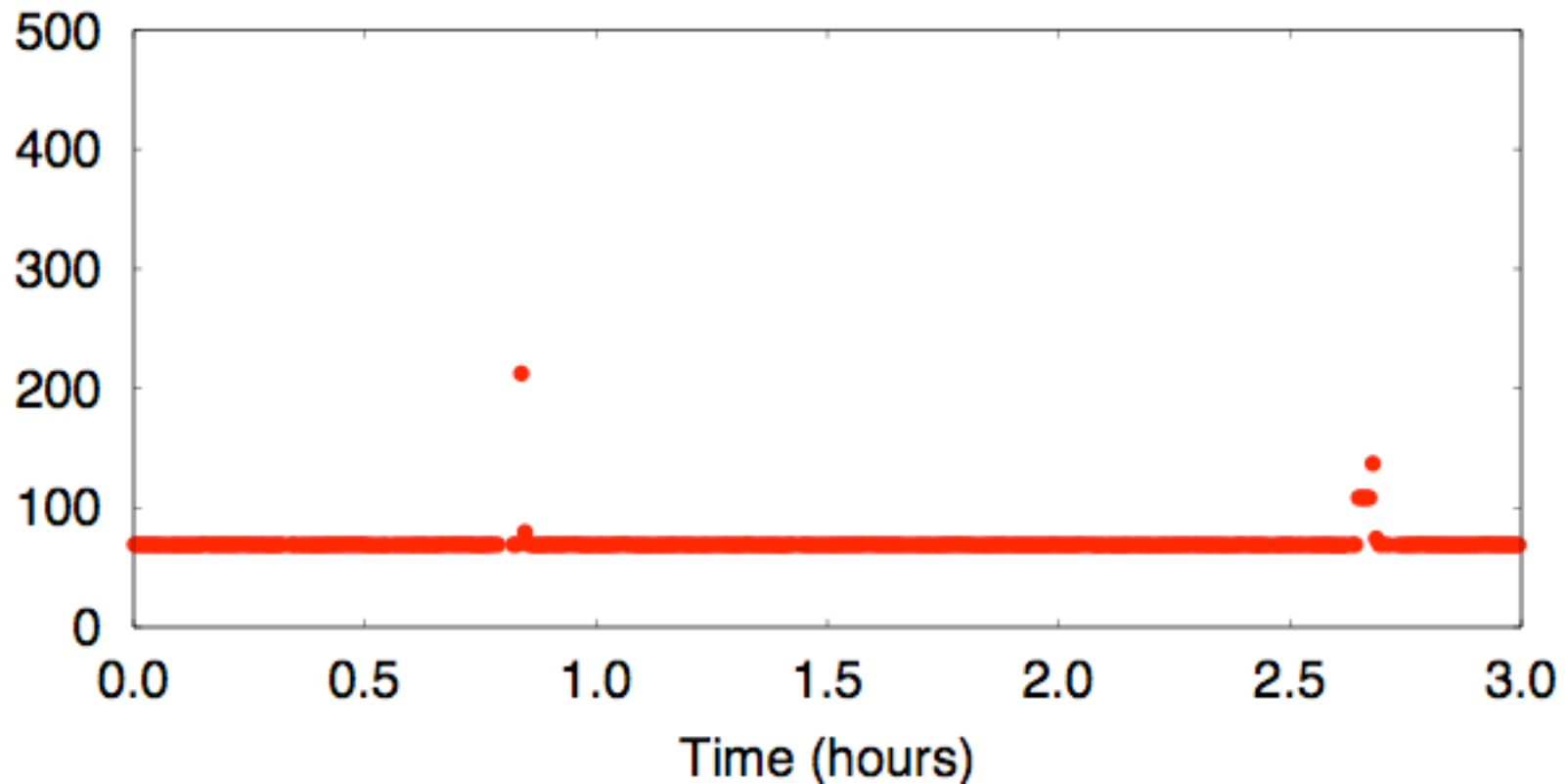


For each node, sum of magnitude of incorrect rankings,
over sum of all rankings.



Latency Filter Applied to Error-Prone Link

Same three hours of measurements from berkeley to uvic.ca

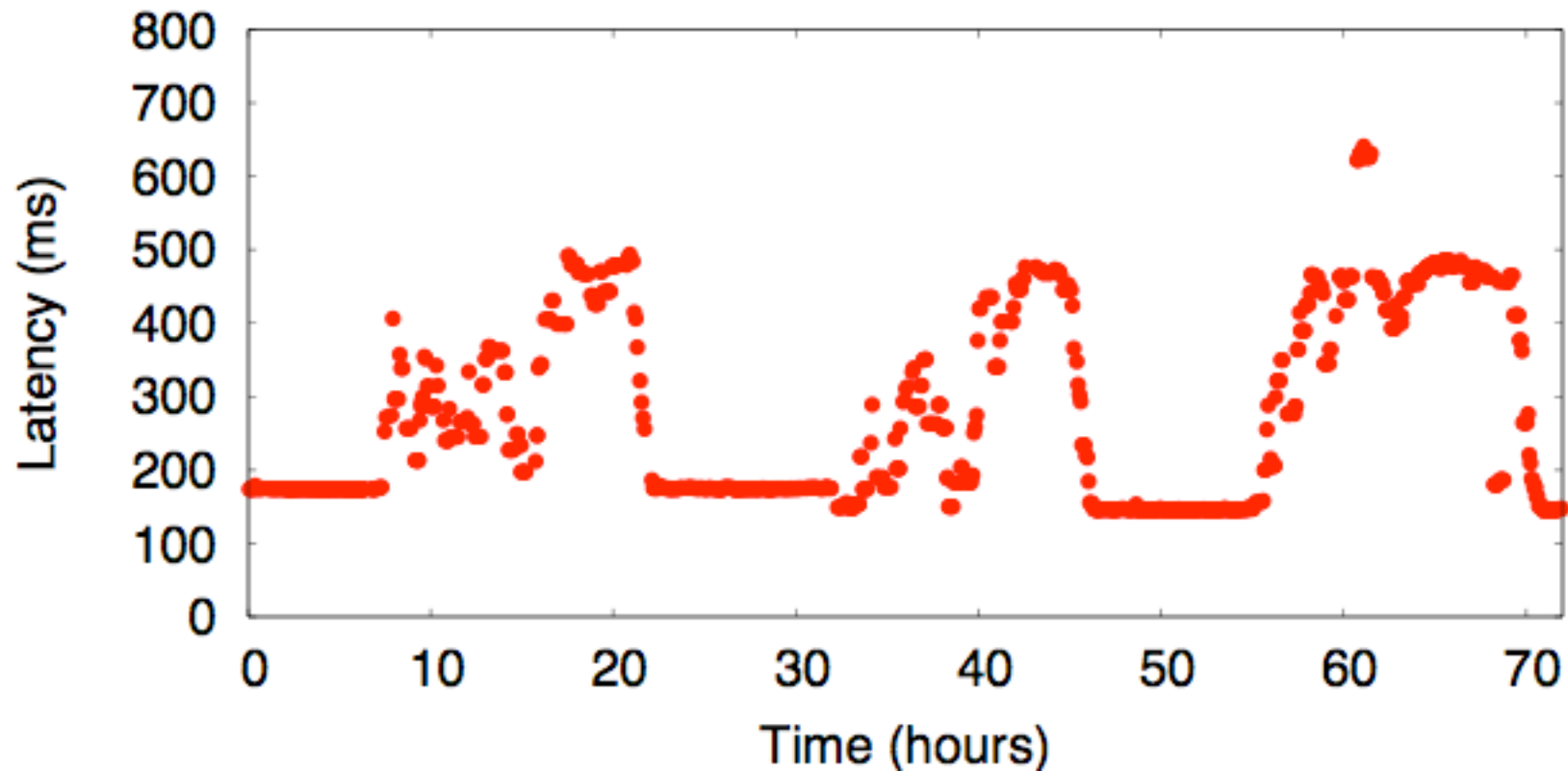


98% of output now within 1ms of median



Latency Filter Applied to Periodic Link

Three days of measurements from ntu.edu.tw to 6planetlab.edu.cn



Short filters adapt to new latency plateaus and drop outliers



Coordinate Update (II)

VIVALDI(\vec{x}_j, w_j, l_{ij})

$$1 \quad w_s = \frac{w_i}{w_i + w_j}$$

$$2 \quad \varepsilon = \frac{|\|\vec{x}_i - \vec{x}_j\| - l_{ij}|}{l_{ij}}$$

$$3 \quad \alpha = c_e \times w_s$$

$$4 \quad w_i = (\alpha \times \varepsilon) + ((1 - \alpha) \times w_i)$$

$$5 \quad \delta = c_c \times w_s$$

$$6 \quad \vec{x}_i = \vec{x}_i + \delta \times (\|\vec{x}_i - \vec{x}_j\| - l_{ij}) \times u(\vec{x}_i - \vec{x}_j)$$



Update Filters: Increase Stability

- Stability: coord change per time (ms/sec)

