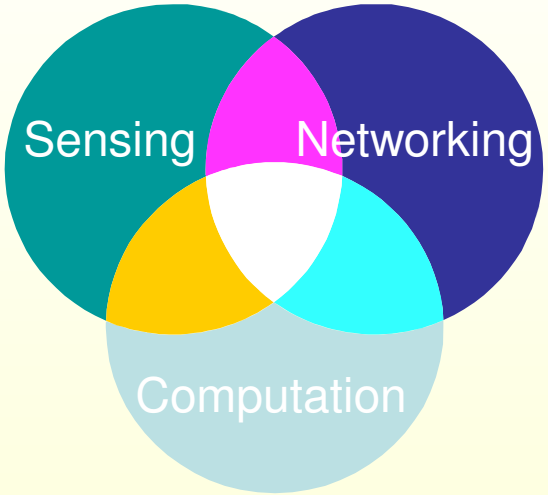
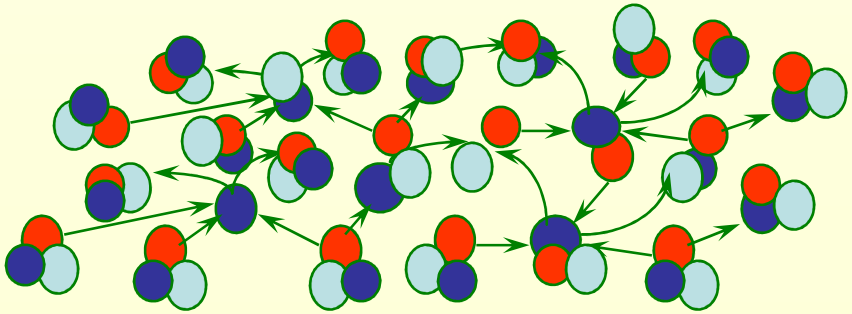
CS321: Double Rulings, Location Services, Q-Digests



Leonidas Guibas
Computer Science Dept.
Stanford University



Information Brokerage
Push-Pull Schemes

Push vs. Pull Information Brokerage

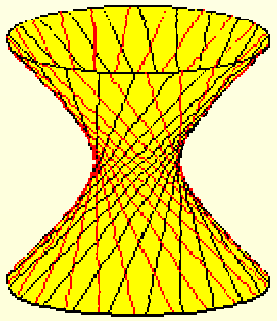
- Directed diffusion lets consumers do all the work, or producers do all the work
- GHTs are more symmetric, but do not establish streams between a producer and a consumer
- Is there another way?

Producer

Consumer

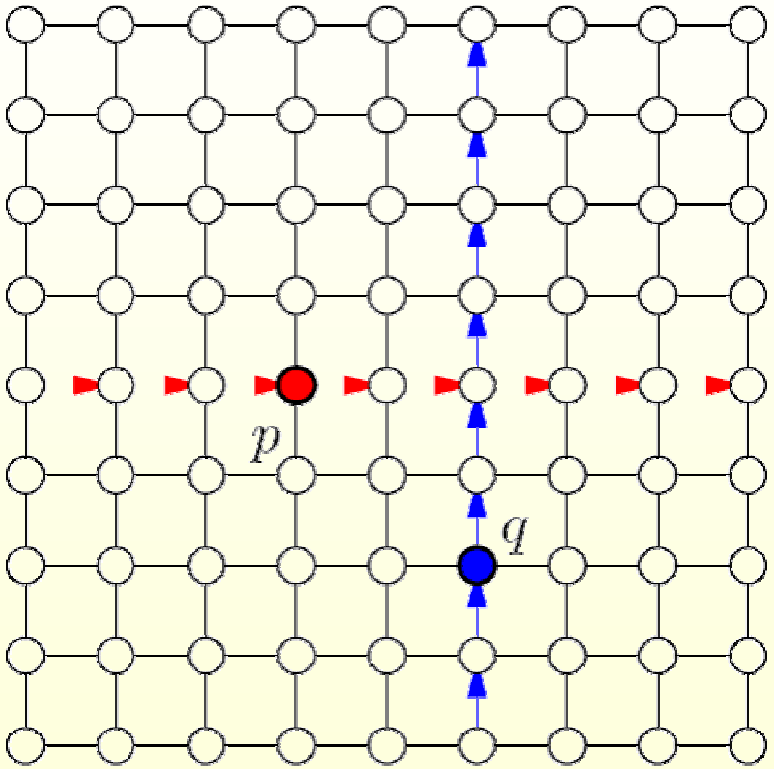
Double Rulings

- Hash data to a 1-d curve, instead of a 0-d point
- Retrieve data by following another 1-d curve
- Motivations for generalization
  - Data delivery uses multi-hop routing
    - Why not leave information along route at no extra cost?
  - More efficient data retrieval
    - No flood is required, either on the producer, or the consumer side



Simple Double Ruling

- Rectilinear Double Ruling
  - Producer stores data on horizontal lines
  - Consumer searches along vertical lines
  - Correctness : Every horizontal line intersects every vertical line
  - Distance-sensitive: q finds p in time $O(d)$, where $d=|pq|$
  - Lines are functions of the producer or consumer locations alone (not the data in question)



References: [Liu Huang Zhang 04], Rumor routing [Braginsky-Estrin 02], Quorum-based routing [Stojmenovic99].

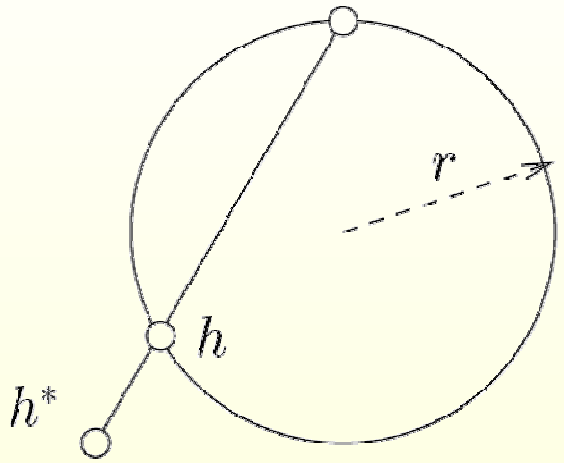
Spherical Double Rulings

- Producer follows a circle to the hashed location
  - Includes GHTs as a special case
  - Allows a large variety of retrieval mechanisms
- Improves on GHTs
  - Load balancing for popular data types
  - Distance sensitivity
  - Flexible data retrieval schemes improve system robustness

[Sarkar, Zhu, Gao,'06]

Double Rulings on a Sphere

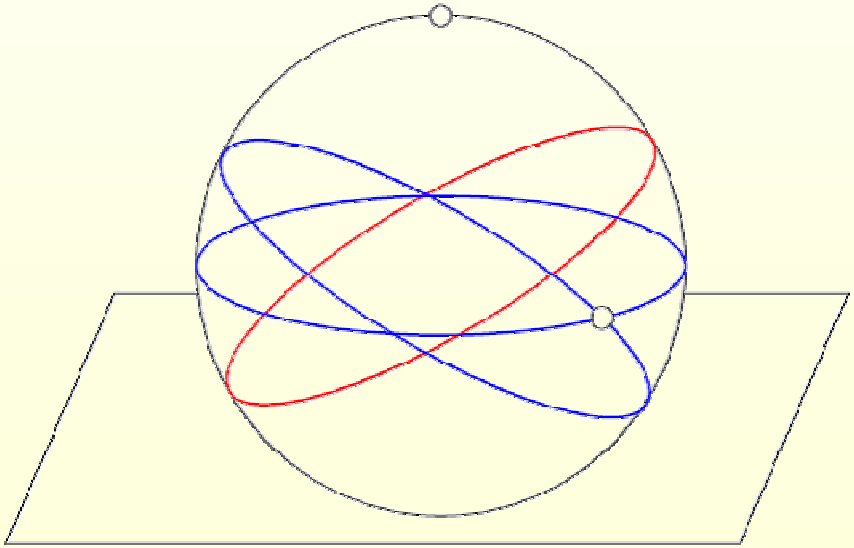
- Stereographic projection maps a projective plane to a sphere
  - Circles map to circles
  - May incur distortion

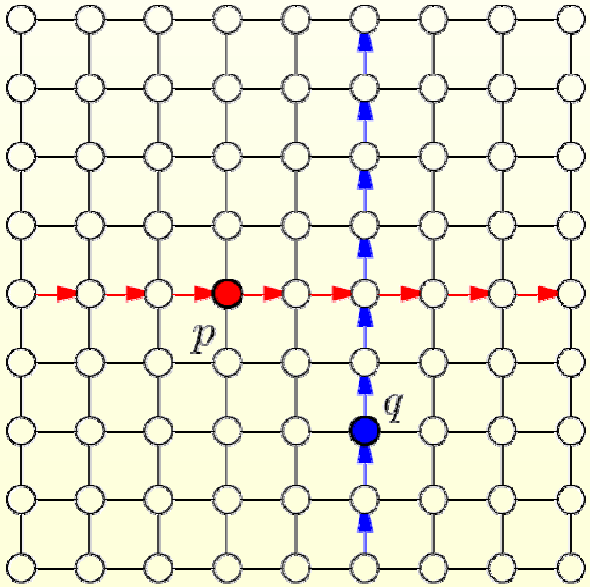


- For a finite sensor field
  - Can choose location and size of sphere such that distance distortion is bounded by 1+ε
  - Geographic routing can be used to follow specific curves: routing on a curve

Spherical Double Rulings

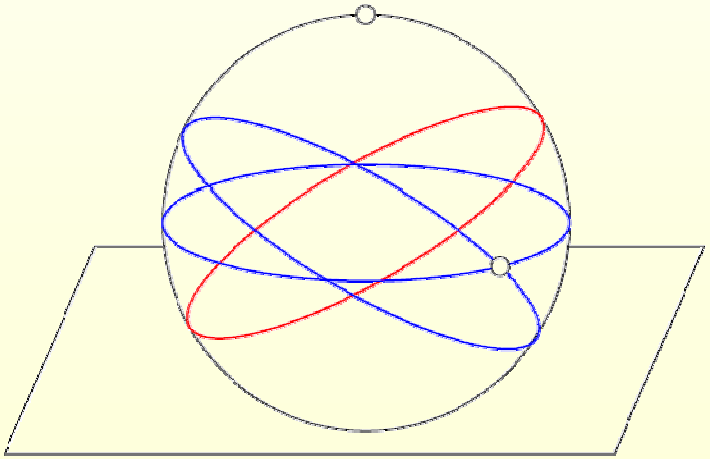
- Any two great circles intersect
  - Use great circles in place of vertical/horizontal lines

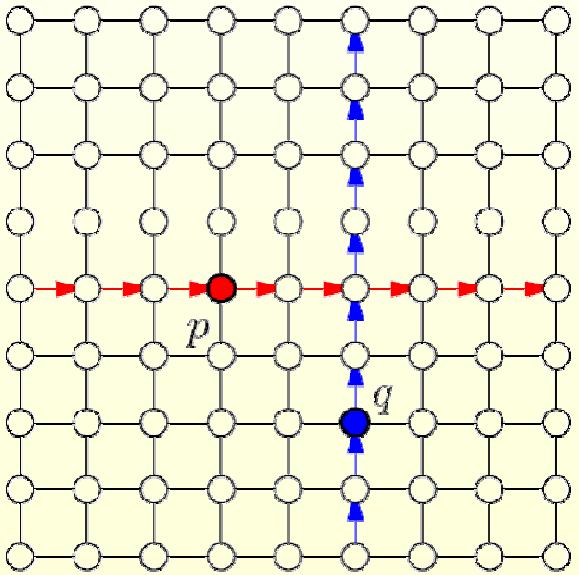




Spherical Double Rulings

- One major difference with rectilinear double rulings:
  - Infinitely many great circles pass through a point
  - A lot more flexibility and pat diversity

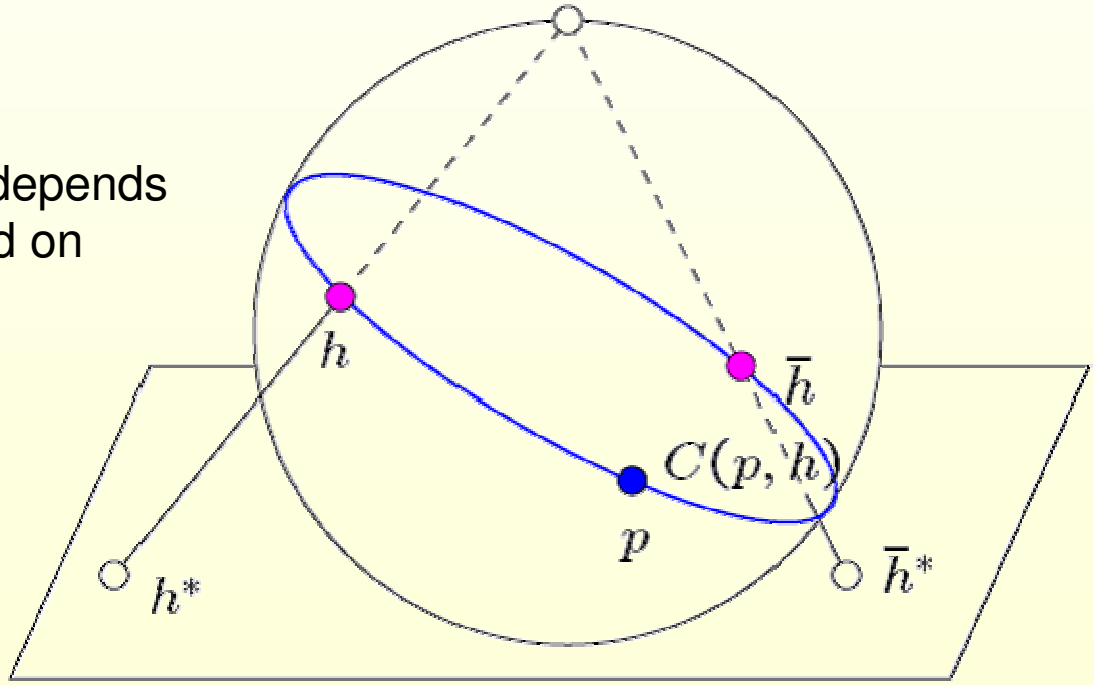




Data Replication

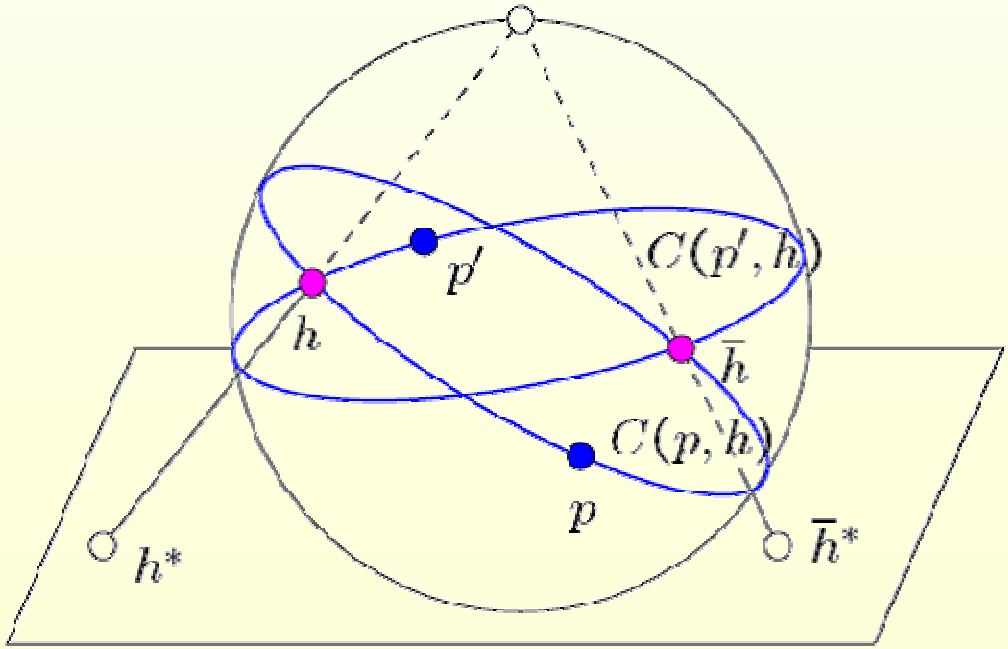
- Data centric hash function $h(T_i)=h_i$.
- Producer *p* replicates data along the great circle $C(p, h_i)$

Replication line depends both on hash and on source location

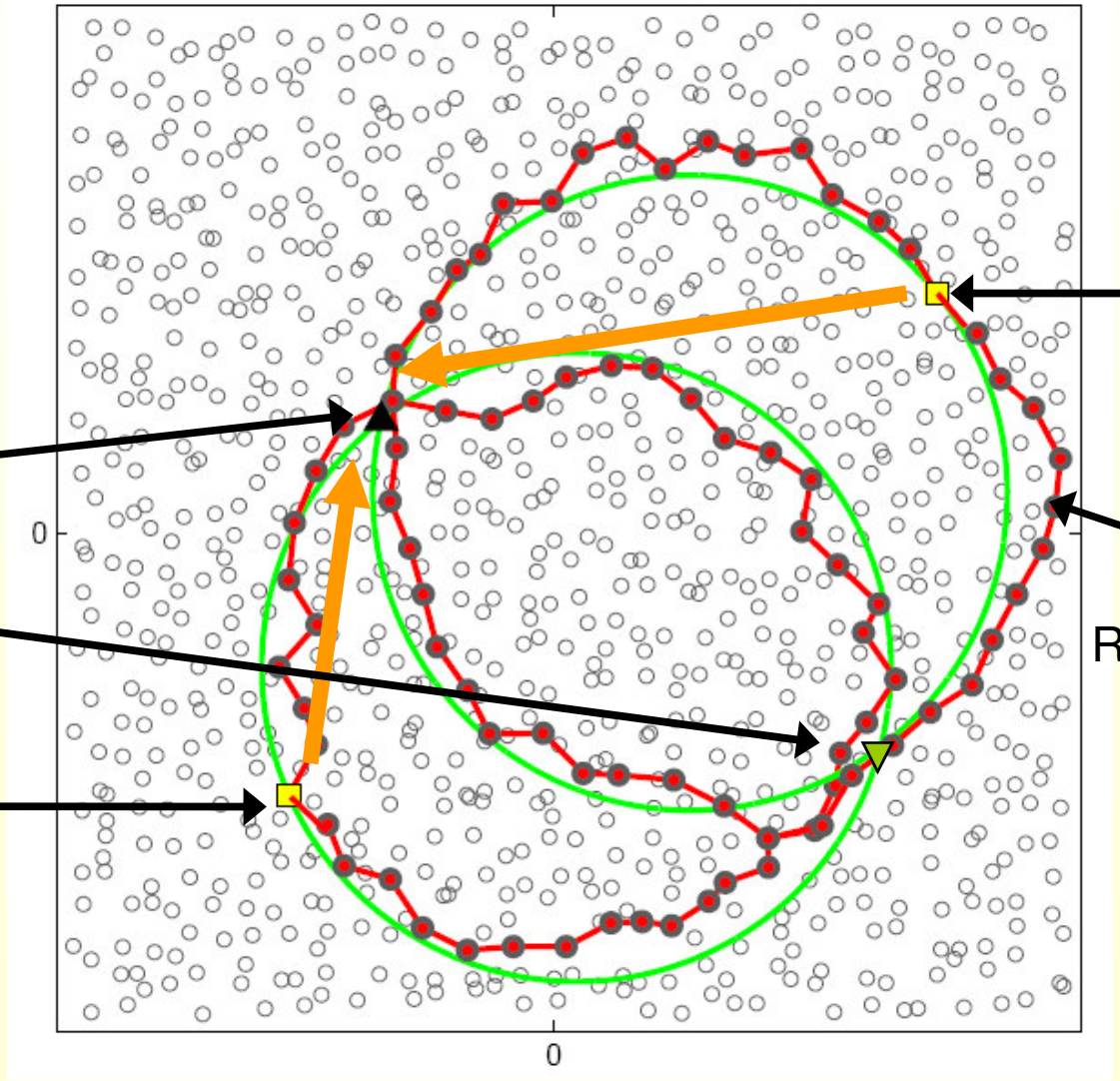


Data Replication

- Different producers with the same data type hash to different great circles, all passing through h, and its antipodal point $\bar{h}$
  - Allows information aggregation.



Replication Curve Examples



Data Retrieval

- Flexible retrieval rules
  1. GHT-Style Retrieval
  2. Distance Sensitive Retrieval
  3. Aggregated Data Retrieval
  4. Entire Data Retrieval

1. GHT Style Retrieval

- GHT still works
- Consumer q wants data T_i

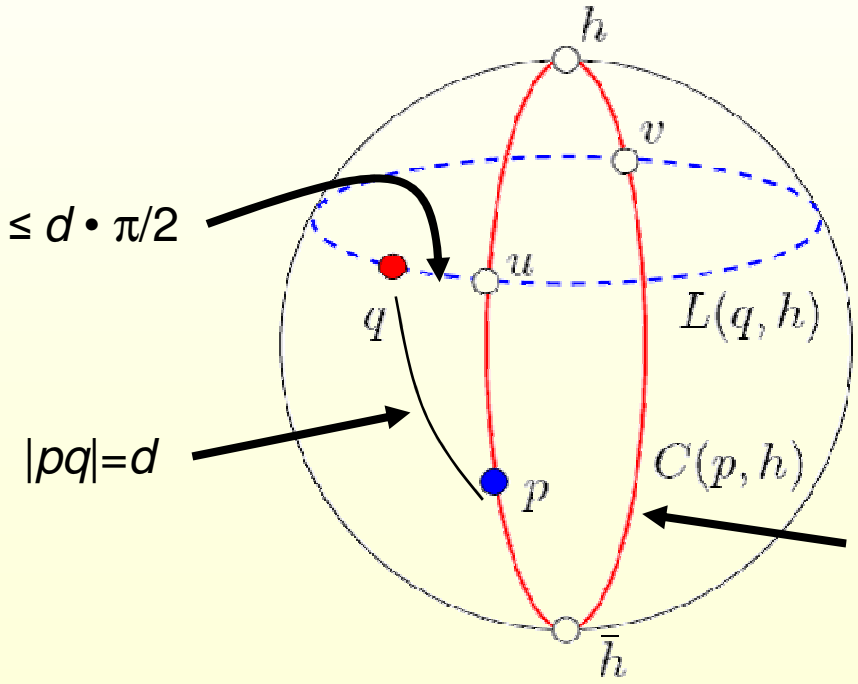
Consumer goes to hashed node h, or its antipode, whichever is closer.

2. Distance Sensitive Retrieval

- Distance Sensitive : If producer *p* is at distance *d* from *q*, consumer can find data at cost *O*(*d*)
  - Consumes use less network resources
  - Users are likely to be more interested in data in their immediate vicinity
  - Lower delay --- important in emergency response, or real-time action scenarios

2. Distance Sensitive Retrieval

- Rotate the sphere so that hashed node is at the north pole.



Retrieval along the latitude curve

Replication along the longitude curve

If *q* is *d* away from *p*, the distance from *q* along latitude curve is ≤ *d* • π/2.

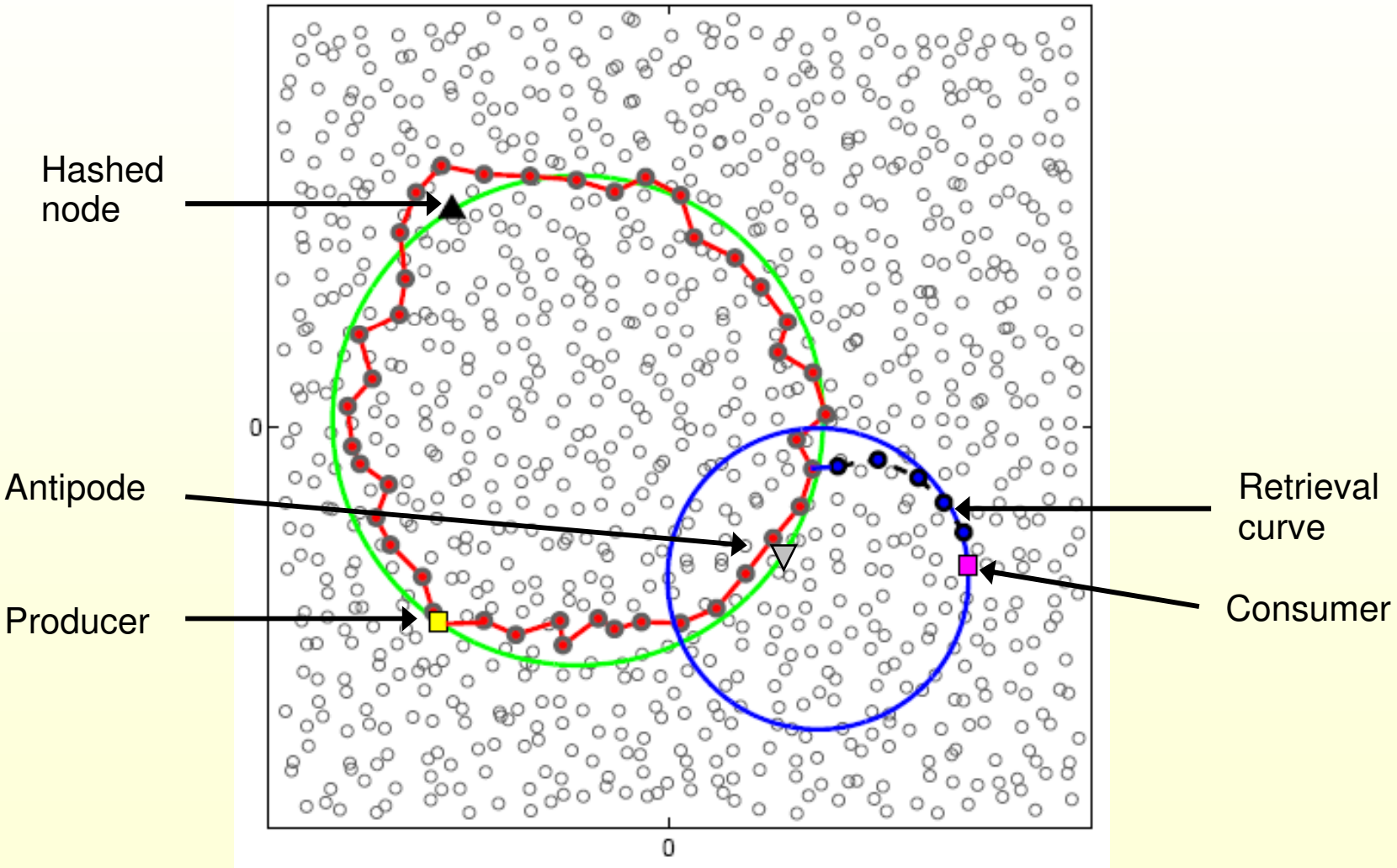
2. Distance Sensitive Retrieval

- Distance-Sensitive : If producer is at distance d from q, consumer can find data at cost $O(d)$.

  Consumer q follows the circle with fixed distance to the hashed location.

- Wrong direction?
  - Handled using a doubling technique
  - A random choice of direction works well in practice (used in the simulations).

2. Distance Sensitive Retrieval



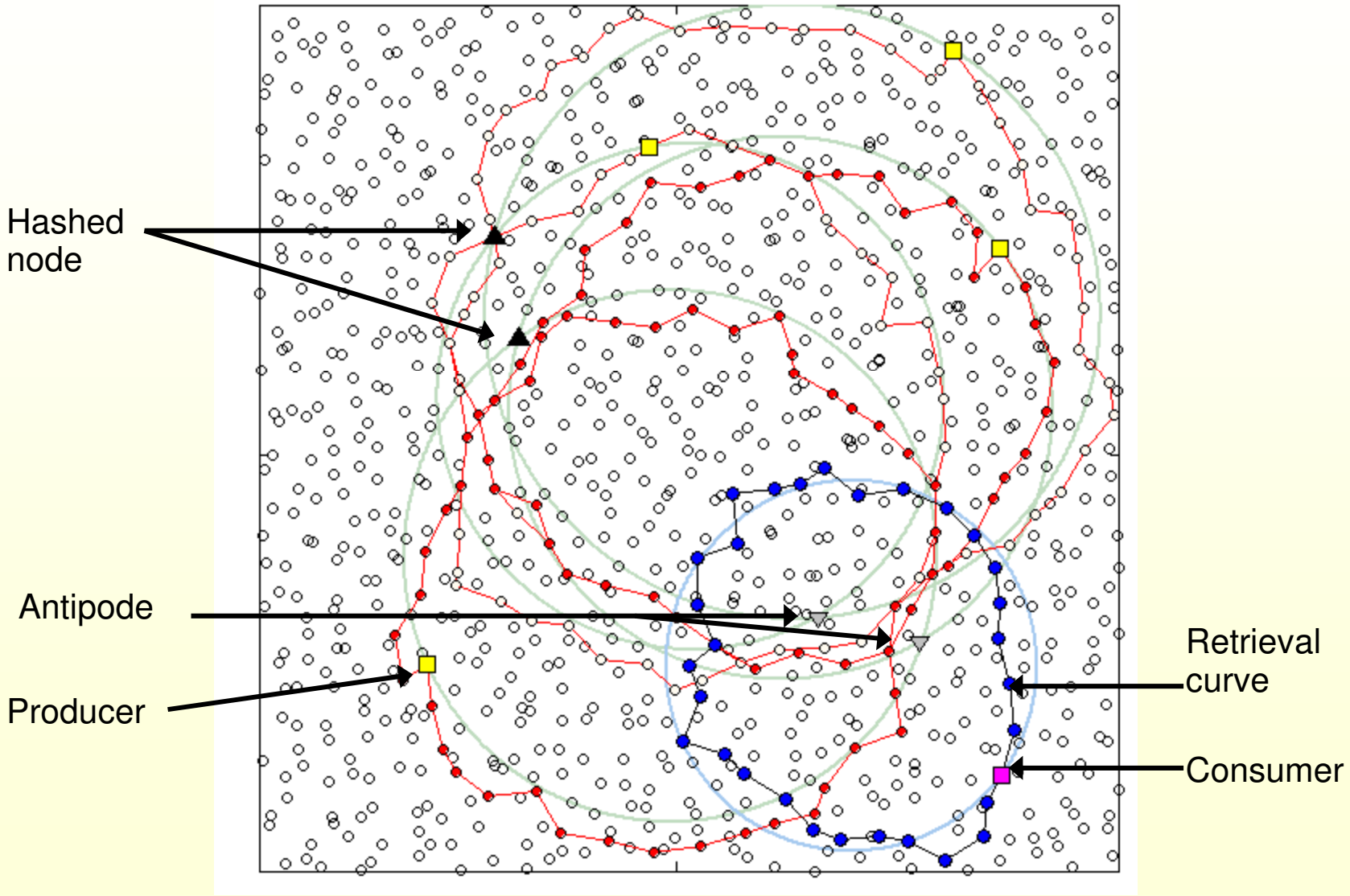
3. Aggregated Data Retrieval

- Consumer wants data of several data types $\{T_i\}$
  - E.g., monkey and elephant detections

> Follow a closed curve that separates h_i and its antipodal point, for each data type T_i

  - Correctness: Any closed cycle that separates h_i from its antipodal intersects the producer curve
  - Many such retrieval curves! therefore more freedom for consumers and better load balancing

3. Aggregated Data Retrieval



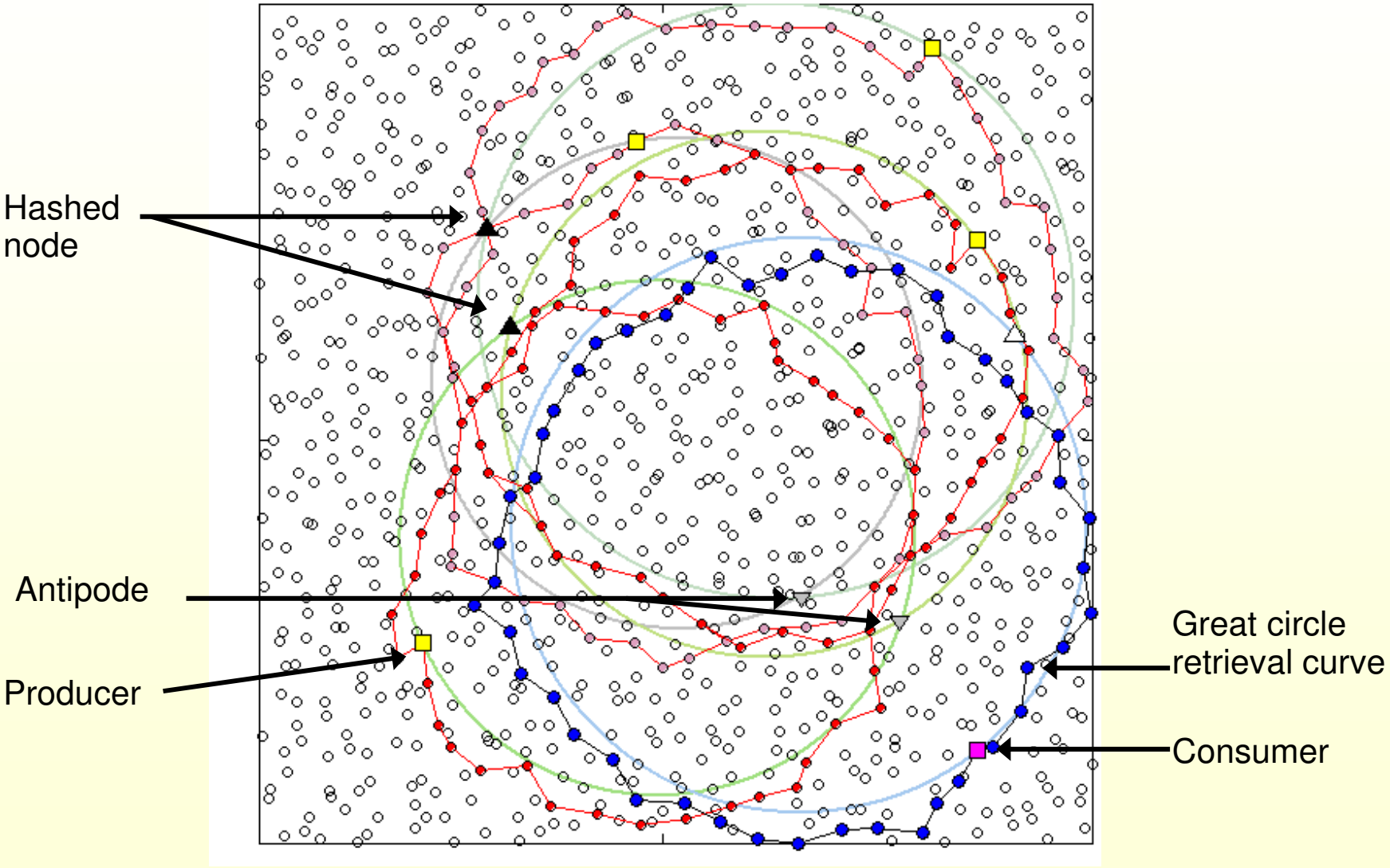
4. Entire Data Retrieval

- Consumer wants all the data in the network

> Follow a (any) great circle, retrieve all data.

  - Correctness : Any two great circles intersect
  - Many such great circles – path diversity

4. Full Power Data Retrieval

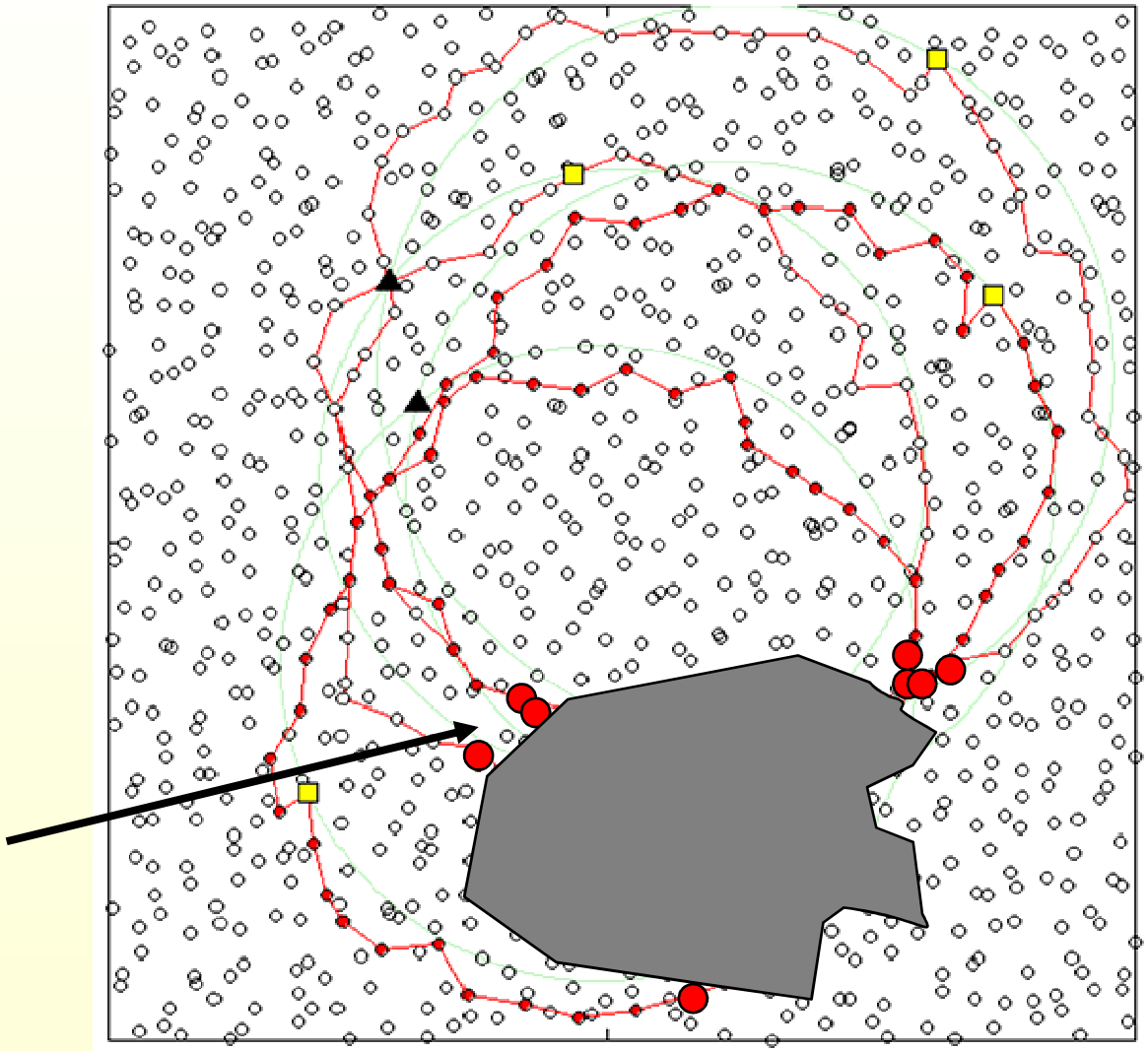


Local Data Recovery Upon Node Failures

- When a group of nodes are destroyed,

> All the data on those nodes are available on the boundary of destroyed region.

Local Data Recovery Upon Node Failures

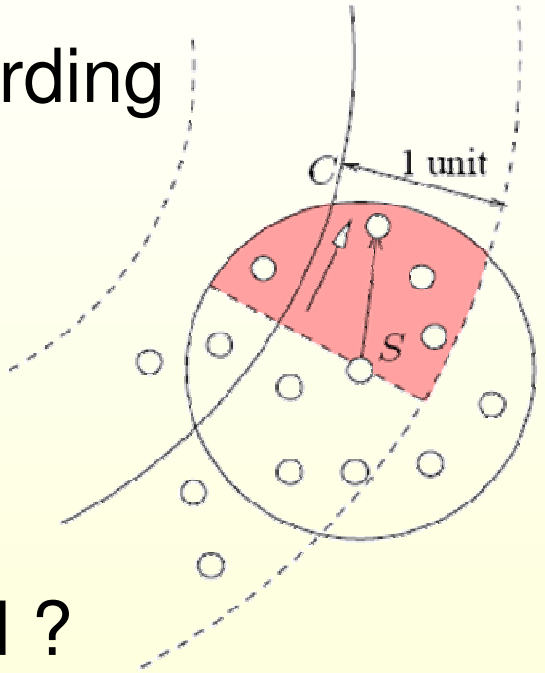


Survived data replicas on the boundary

Implementation

- How to forward data on a virtual curve ?
  - Use “geographic greedy forwarding on a curve”

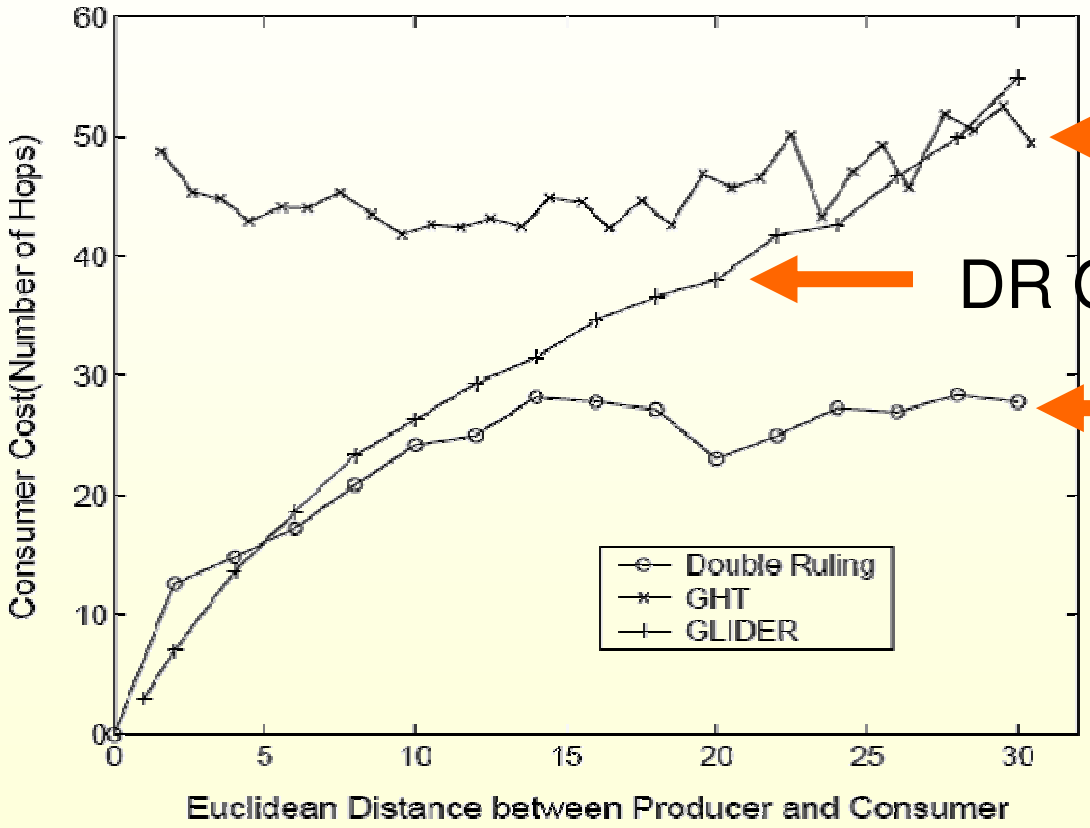
Badri Nath and D. Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 2003.



- A question of density:
  - Is it always possible to forward ?
  - Simulation : A suitable 2-hop neighbour exists, with high probability, for networks with avg. degree ≥ 5.

Simulation: Distance Sensitivity

4200 nodes with average degree 8 per node.



GHT

DR GLIDER scheme

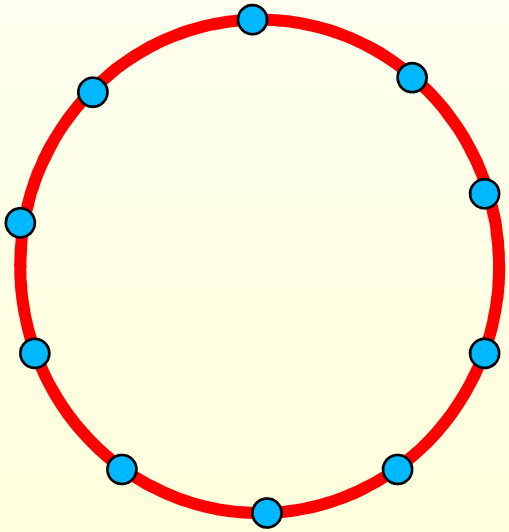
Spherical Double Ruling

Distance sensitivity of queries

GLIDER based scheme : Q. Fang et. al. Landmark-based information storage and retrieval in sensor networks. *INFOCOM* 2006.

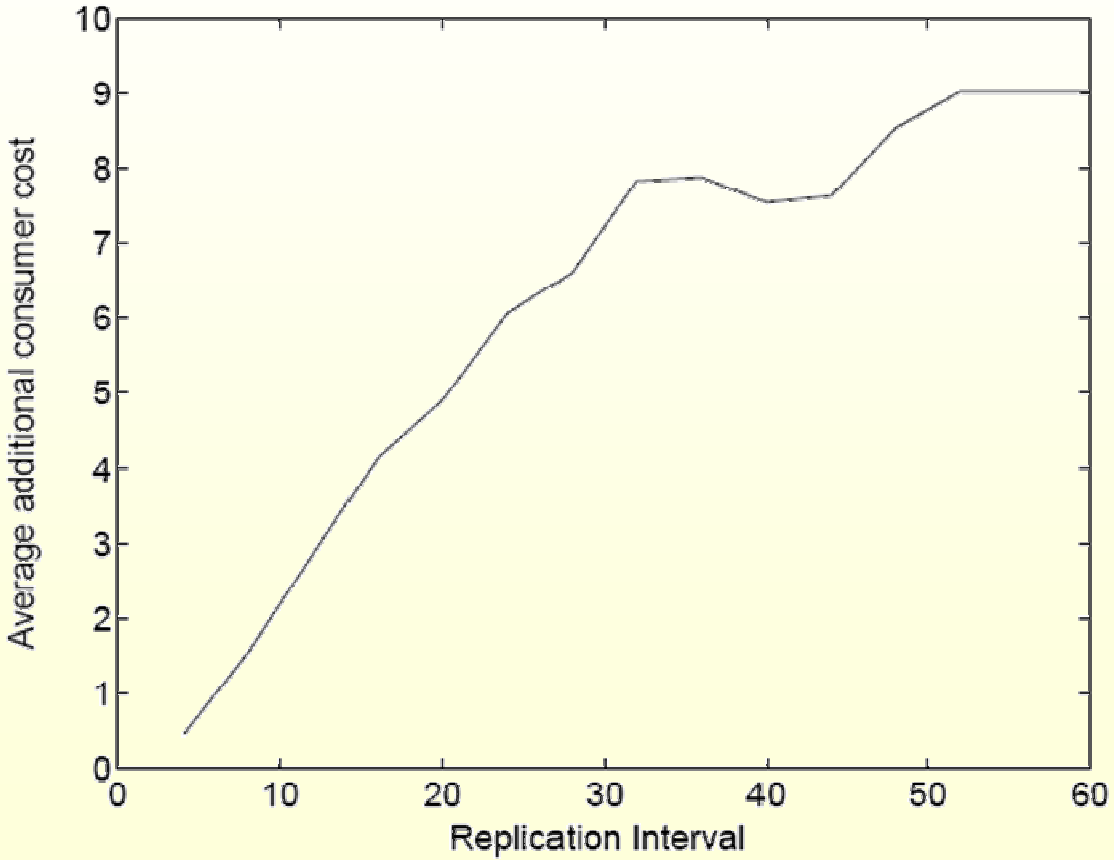
Simulation: Storage/Retrieval Trade-off

Nodes on replication curve can store the data or a pointer to the actual data.



Replication Interval

Increasing Consumer cost

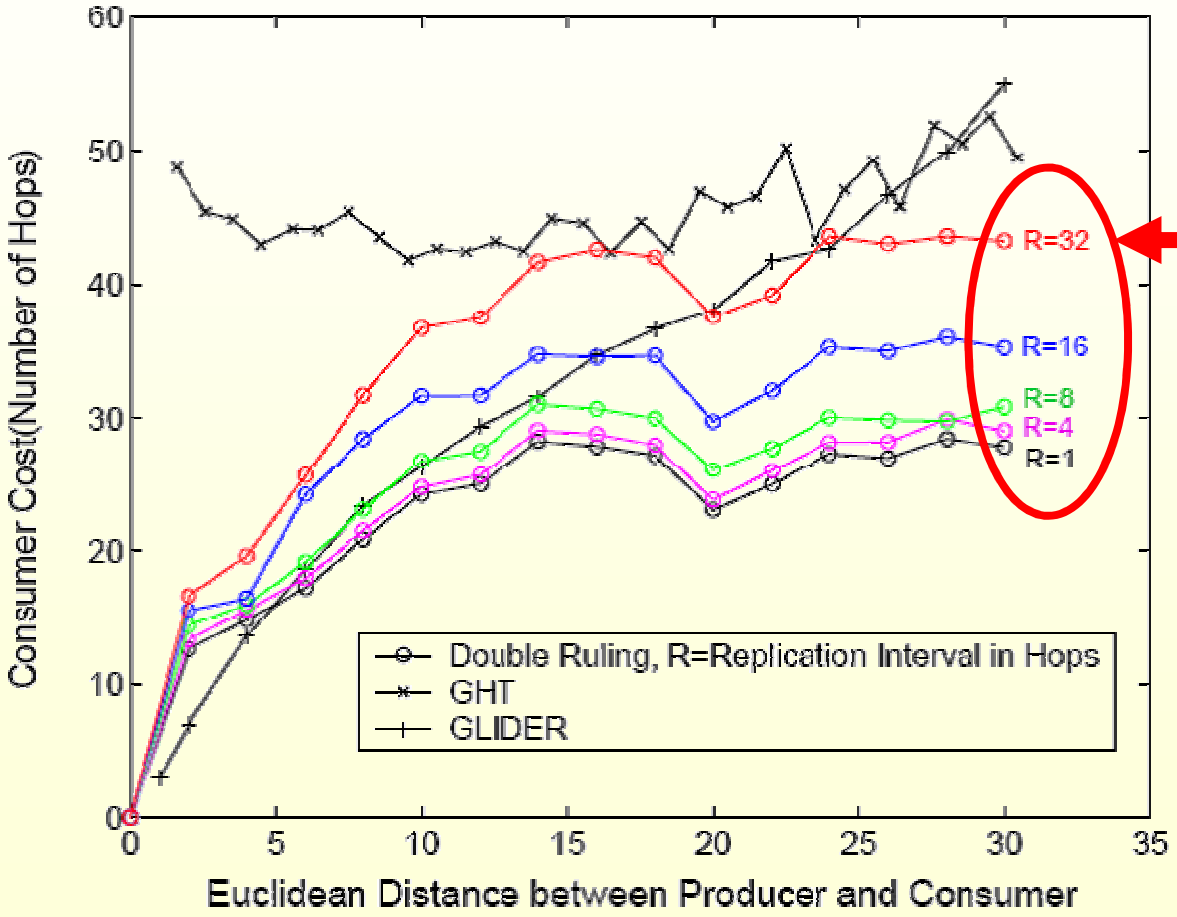


Larger Replication Interval

Decreasing Storage Cost

Simulation: Storage/Retrieval Trade-off

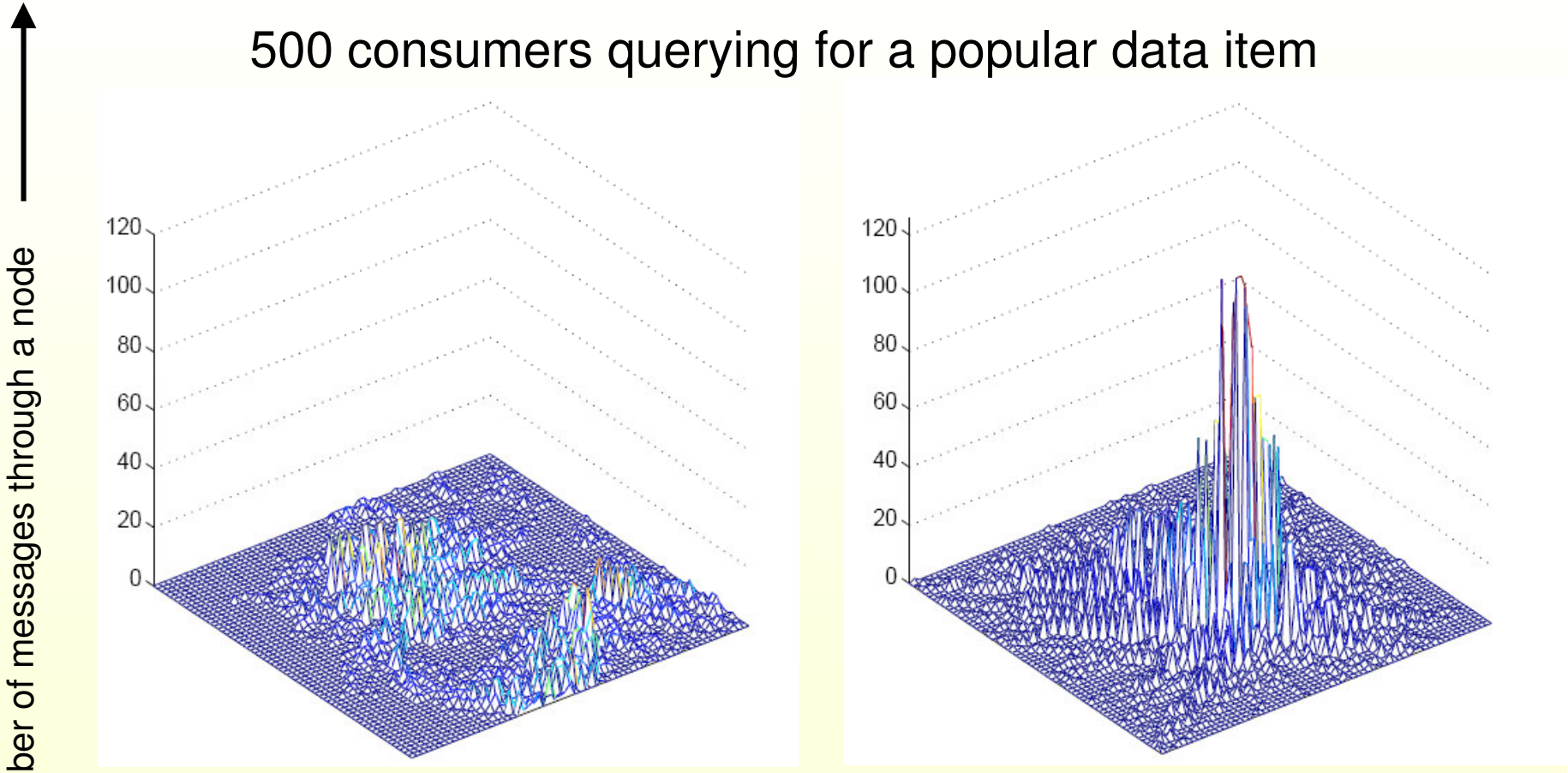
More storage, lower retrieval cost.



Replication only on the hashed node and antipode

Simulation: Load Balancing

500 consumers querying for a popular data item



Double Ruling GHT

Load Distribution

Discussion

- Data collection by mobile data mules
  - Physically move along any retrieval curve
- Advanced hashing schemes
  - E.g., similar data types are placed nearby
- Networks with holes
  - Require special care

When Layout is not Regular…

- Double rulings on an irregular shape
  - Shape parameterization
- Integrate double rulings with other approaches

Summary

- Double rulings: hashing on a curve.
- Must ensure any consumer curve hits any producer curve
- Major advantages: improved location-sensitivity; improved load balancing and data robustness
- In practice, we can use a combination of GHTs and double rulings

Geographic Location Services

Possible Designs for a Location Service

- Flood to get a node's location (LAR, DREAM).
  - excessive flooding messages
- Central static location server
  - not fault tolerant
  - too much load on central server and nearby nodes
  - the server might be far away for nearby nodes or inaccessible due to network partition
- Every node acts as server for a few others
  - good for spreading load and tolerating failures

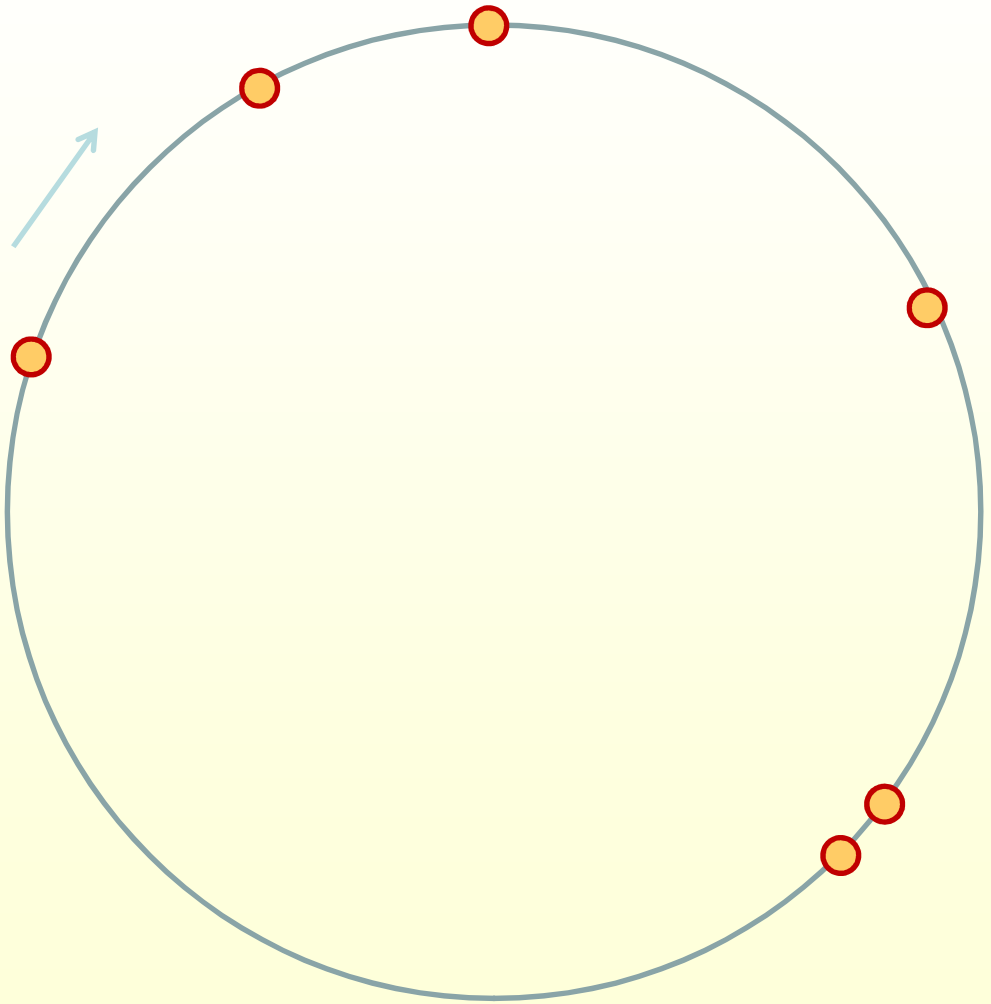
Desirable Properties of a Distributed Location Service

- Spread load evenly over all nodes
- Degrade gracefully as nodes fail
- Queries for nearby nodes stay local
- Per-node storage and communication costs grow slowly as the network size grows

The Grid Location Service (GLS)

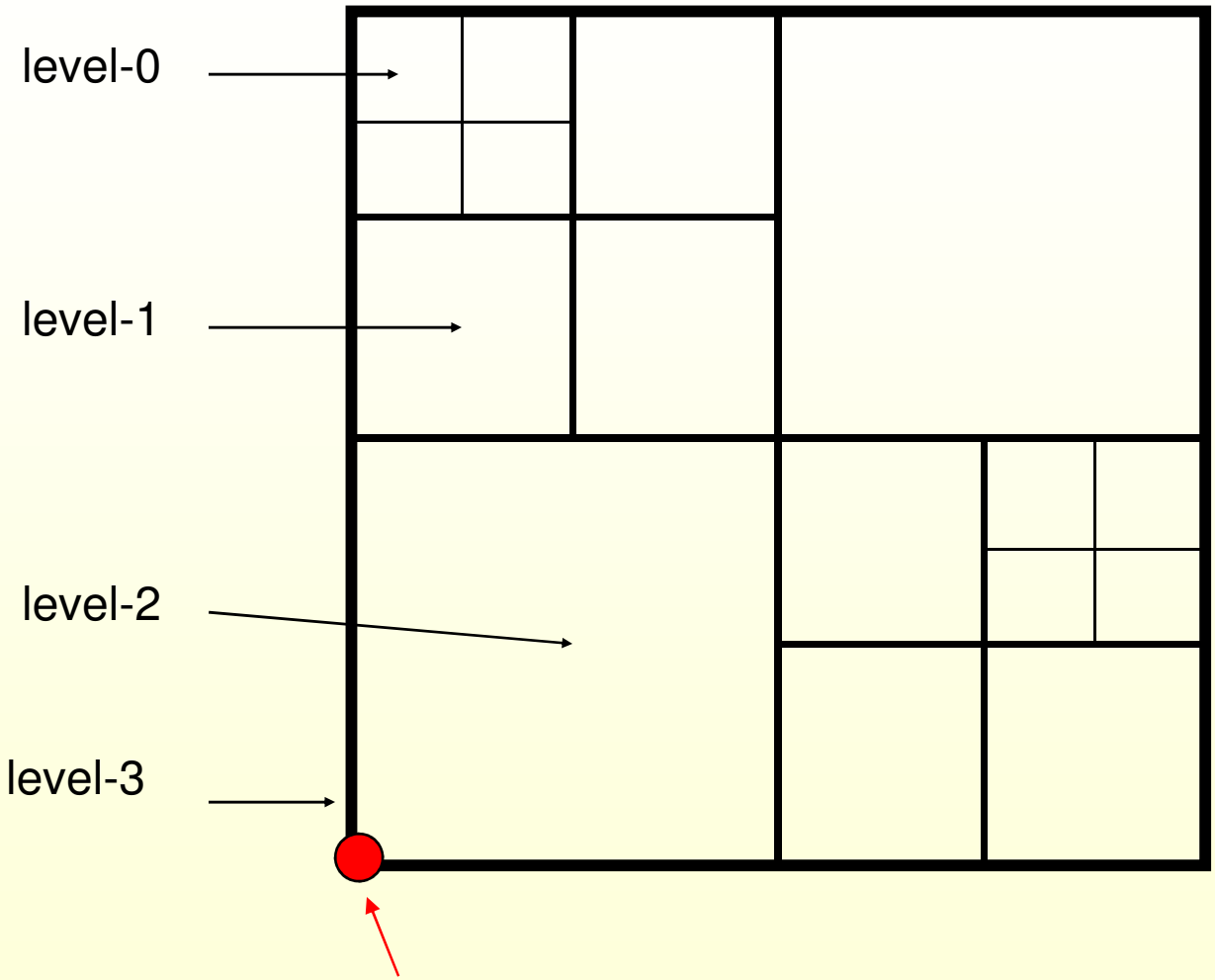
[Li, Jannotti, De Couto, Karger, Morris, A Scalable Location Service for Geographic Ad Hoc Routing, ACM Mobicom 2000, Boston, Massachusetts]

Circular Node Numbering



Based on unique node IDs

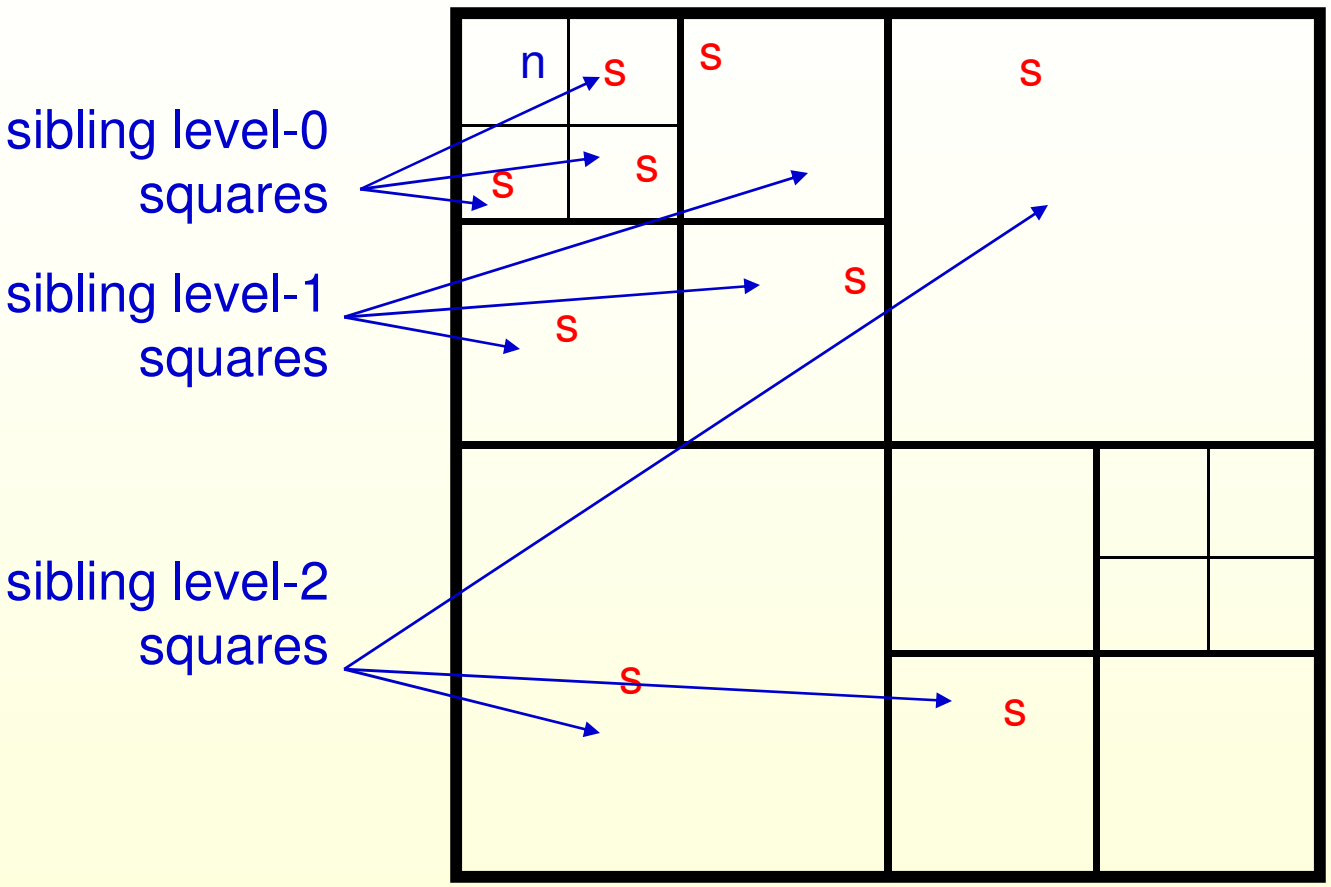
GLS's spatial hierarchy



All nodes know their geographic locations

All nodes agree on the global origin of the grid hierarchy

3 Servers Per Node Per Level



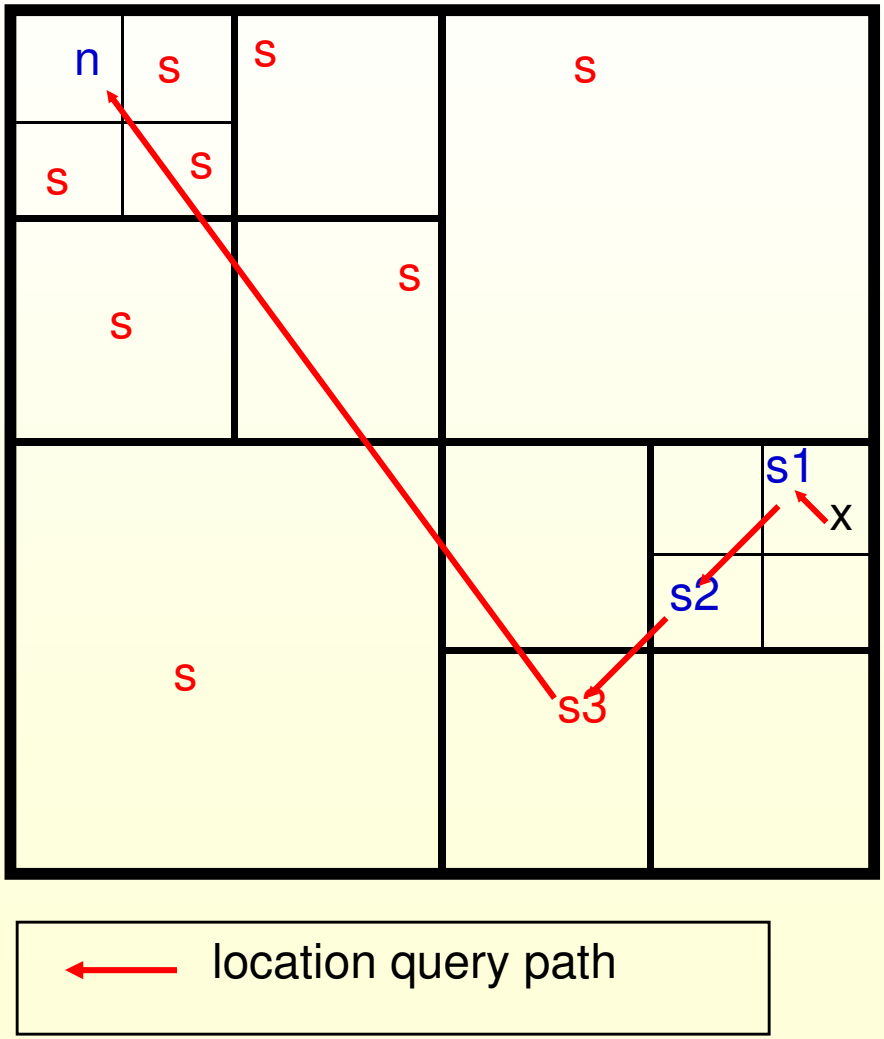
Nodes who know *n*'s location

- *s* is *n*'s successor in that square.

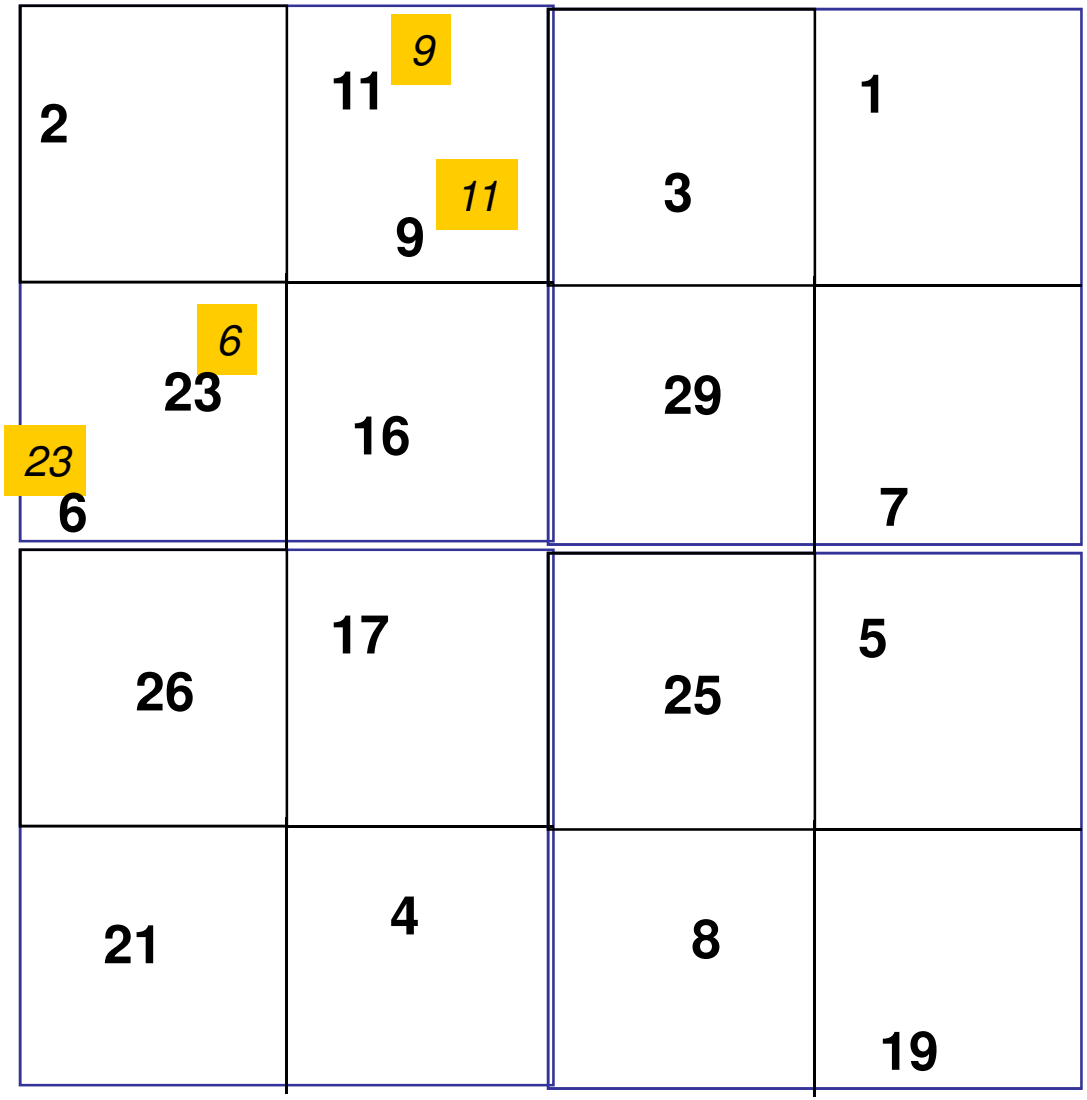
(Successor is the node with "least ID greater than" *n*)

Queries Search for Destination's Successors

Each query step:
visit *n*'s successor at each level.



GLS Update (level 0)

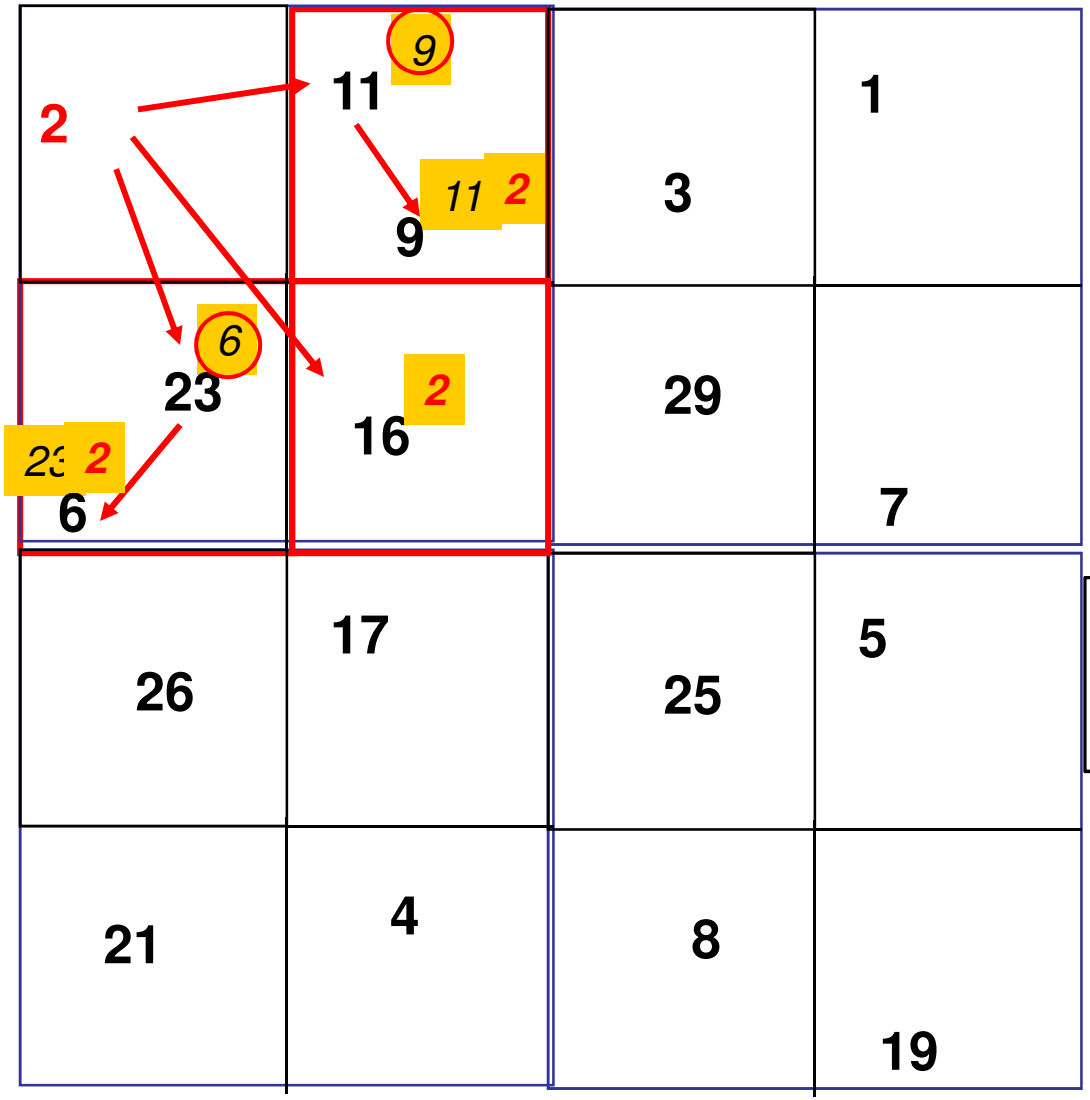


Invariant (for all levels):
For node *n* in a square, *n*'s successor in each sibling square "knows" about *n*.

Base case:
Each node in a level-0 square "knows" about all other nodes in the same square.

location table content

GLS Update (level 1)



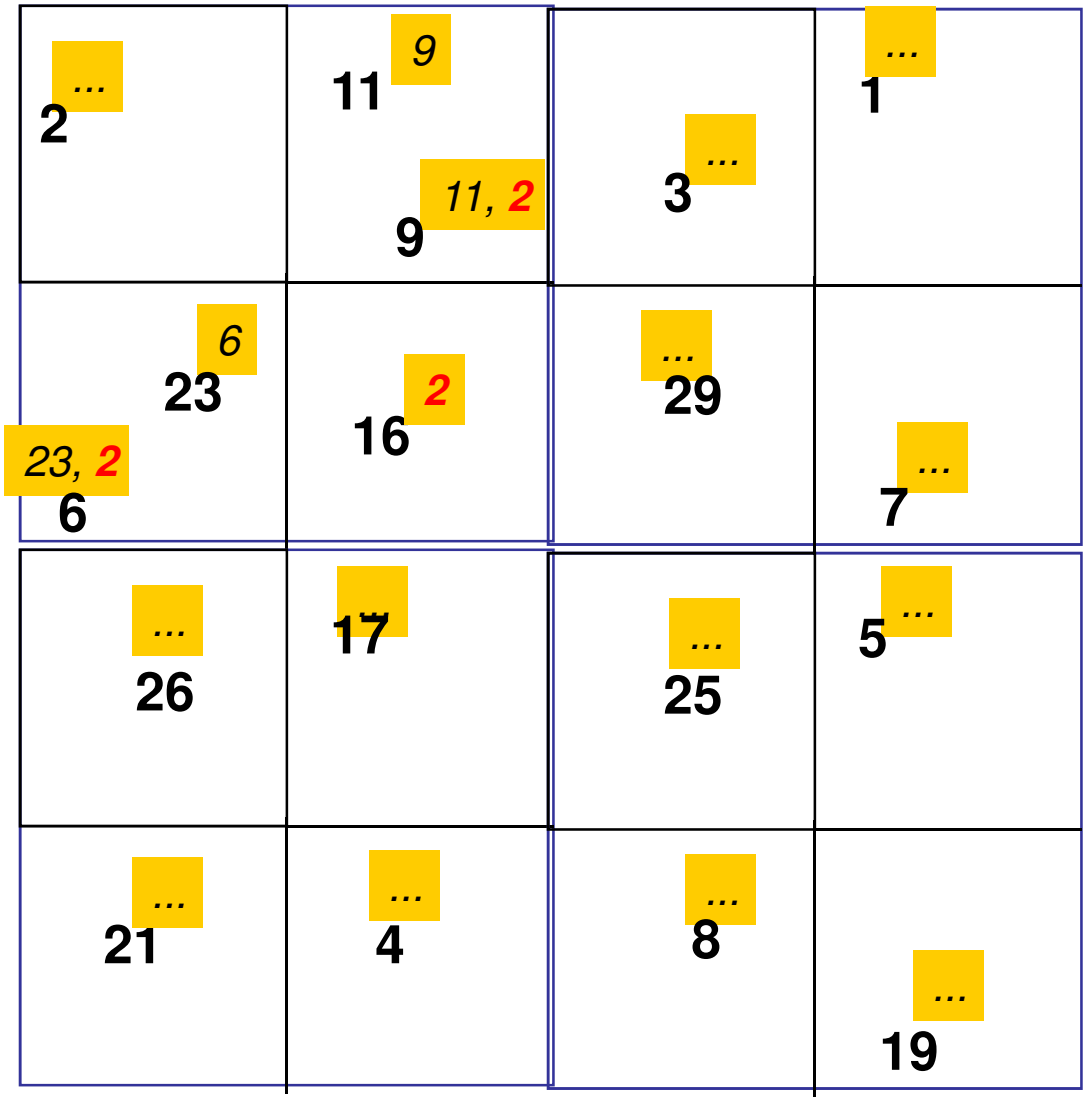
Invariant (for all levels):
For node *n* in a square, *n*'s successor in each sibling square "knows" about *n*.

Geographic forwarding can be used to send information to a tile

location table content

location update

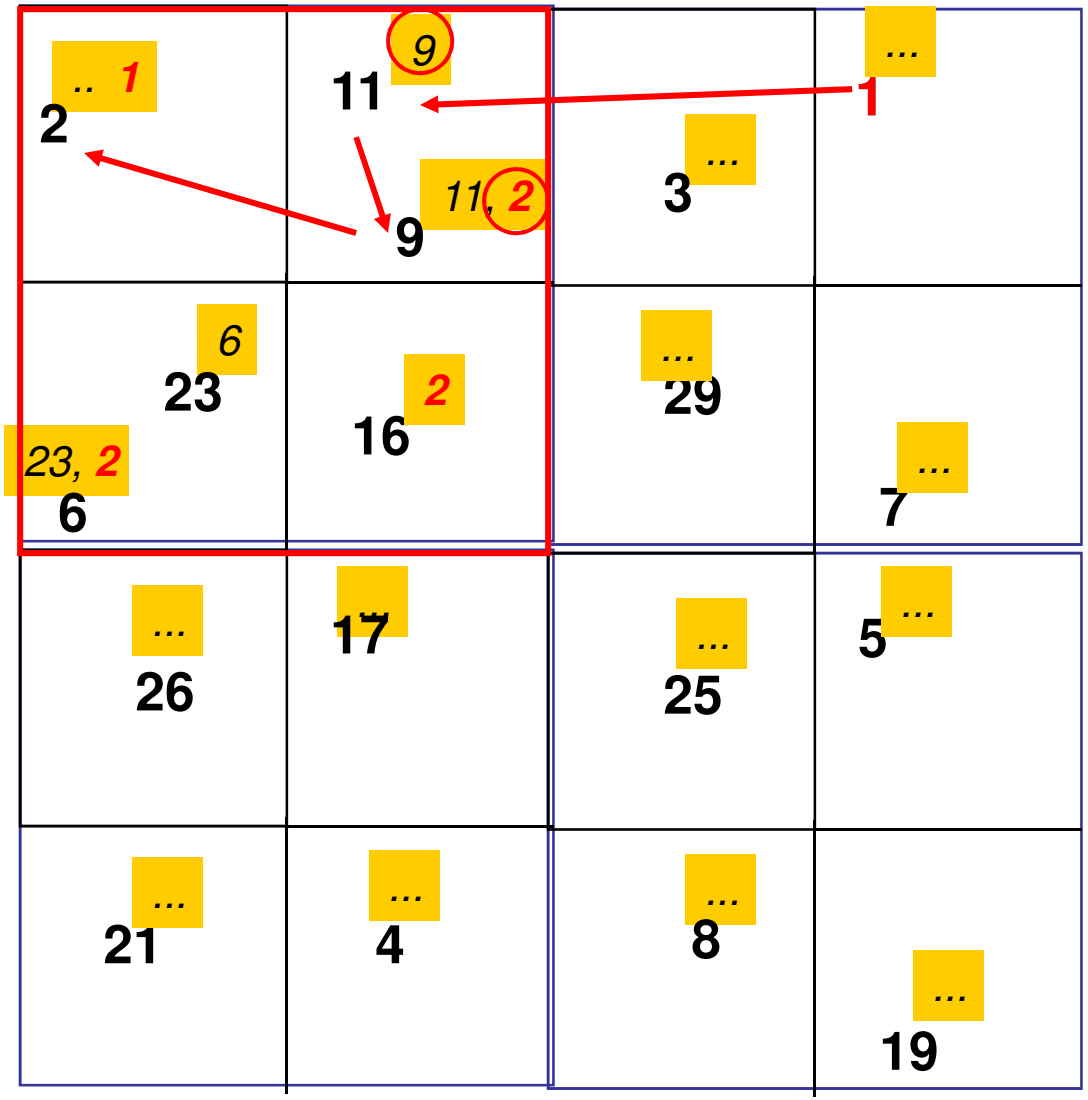
GLS Update (level 1)



Invariant (for all levels):
For node *n* in a square, *n*'s successor in each sibling square "knows" about *n*.

location table content

GLS Update (level 2)

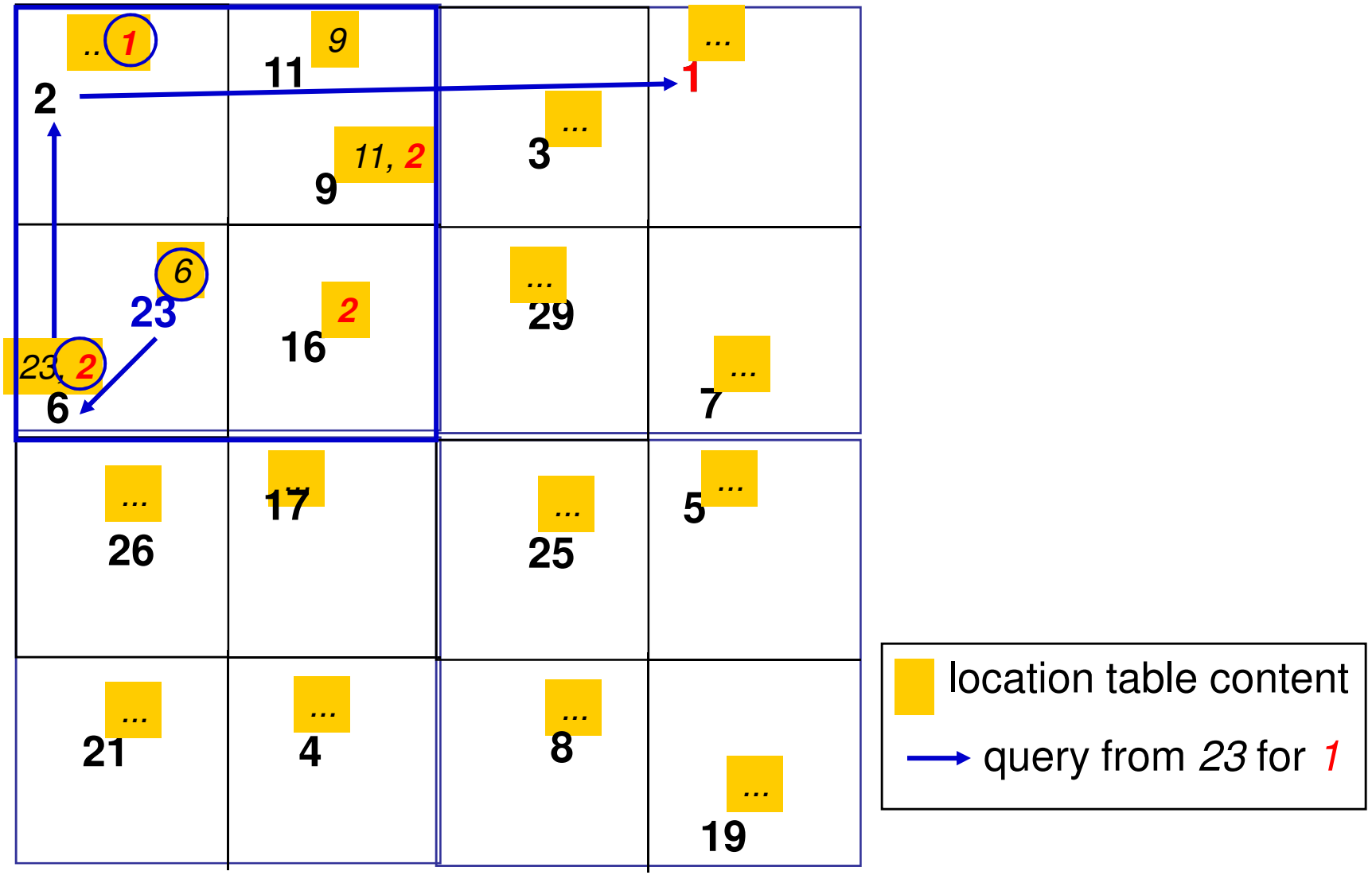


Invariant (for all levels):
For node *n* in a square, *n*'s successor in each sibling square "knows" about *n*.

location table content

location update

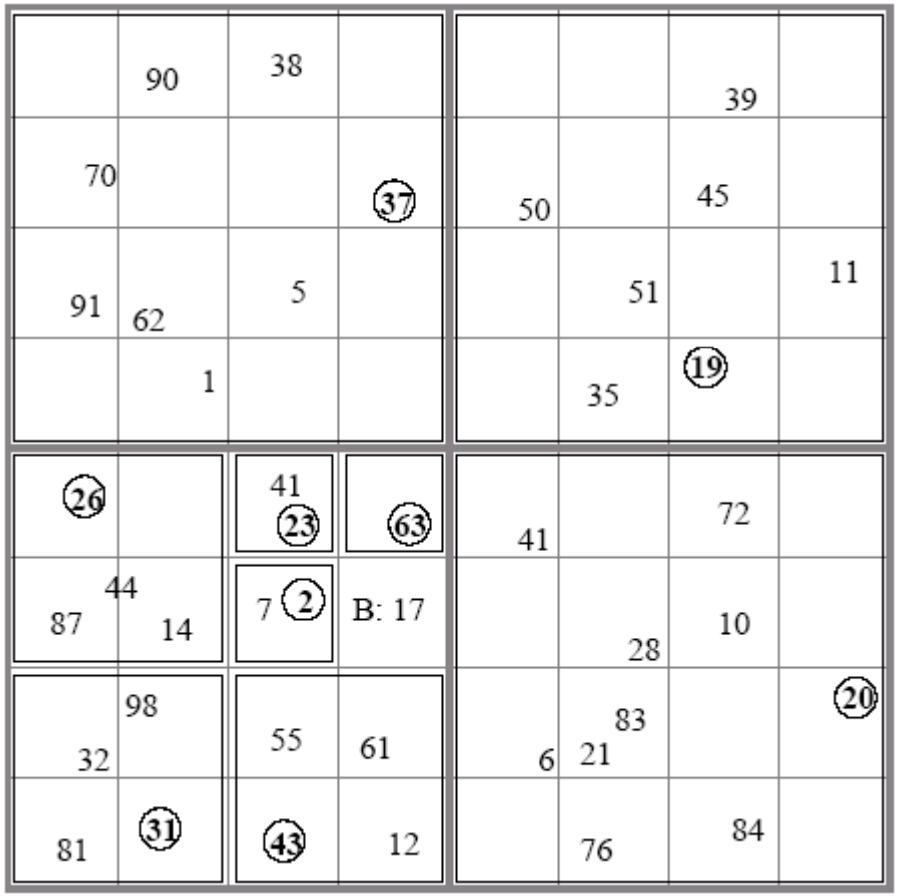
GLS Query


..
1
9
11
2
1
...
3
...
11, 2
9
6
23
2
16
...
29
23, 2
6
...
7
...
26
17
...
25
5
...
...
21
...
4
...
8
...
19
location table content
query from 23 for 1

A more complex scenario ...

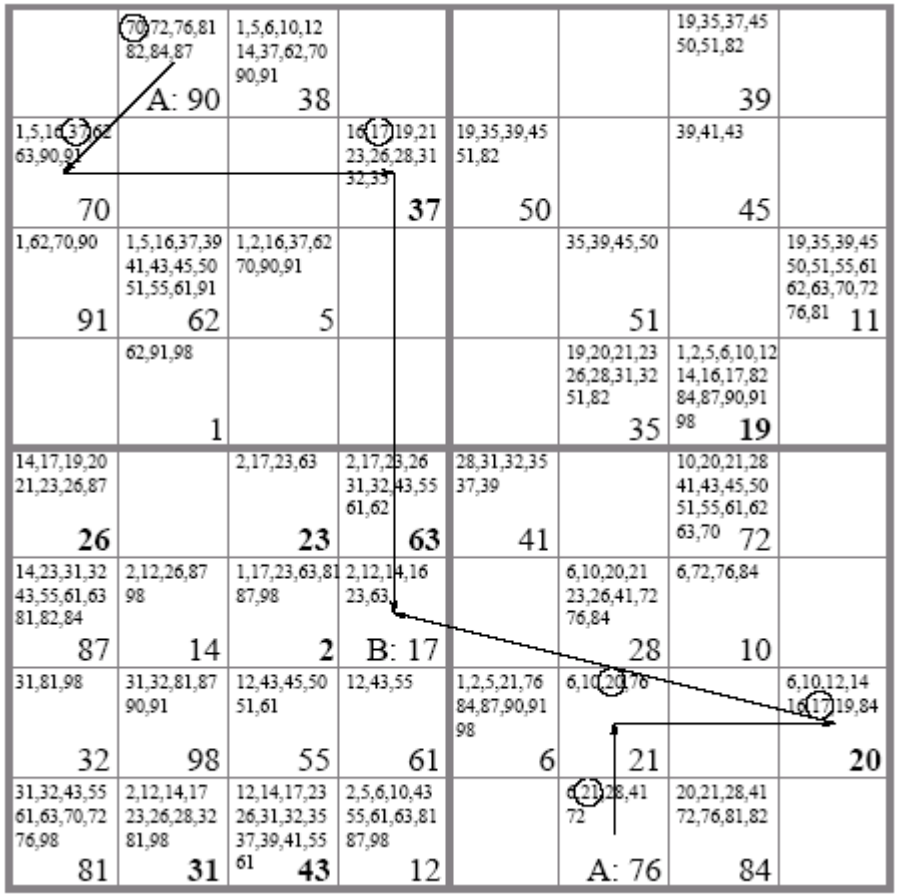
Location Servers

- Node B's location servers: Inside each sibling square on each level, choose B's successor node
- *Successor:* node with least ID greater than B
- Circular ID space



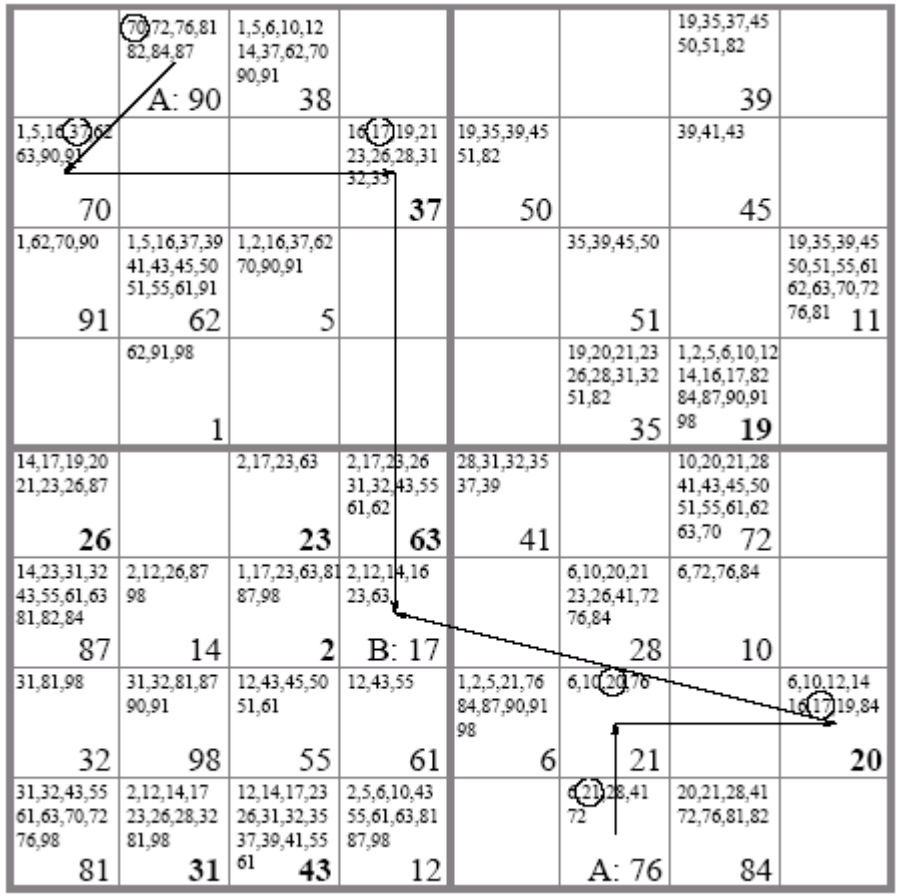
Location Queries

- A queries for the location of B:
- A's only information about B is the ID of B
- A does not know who are B's location servers.
- Even B doesn't know its own location servers
- How to implement the location query?



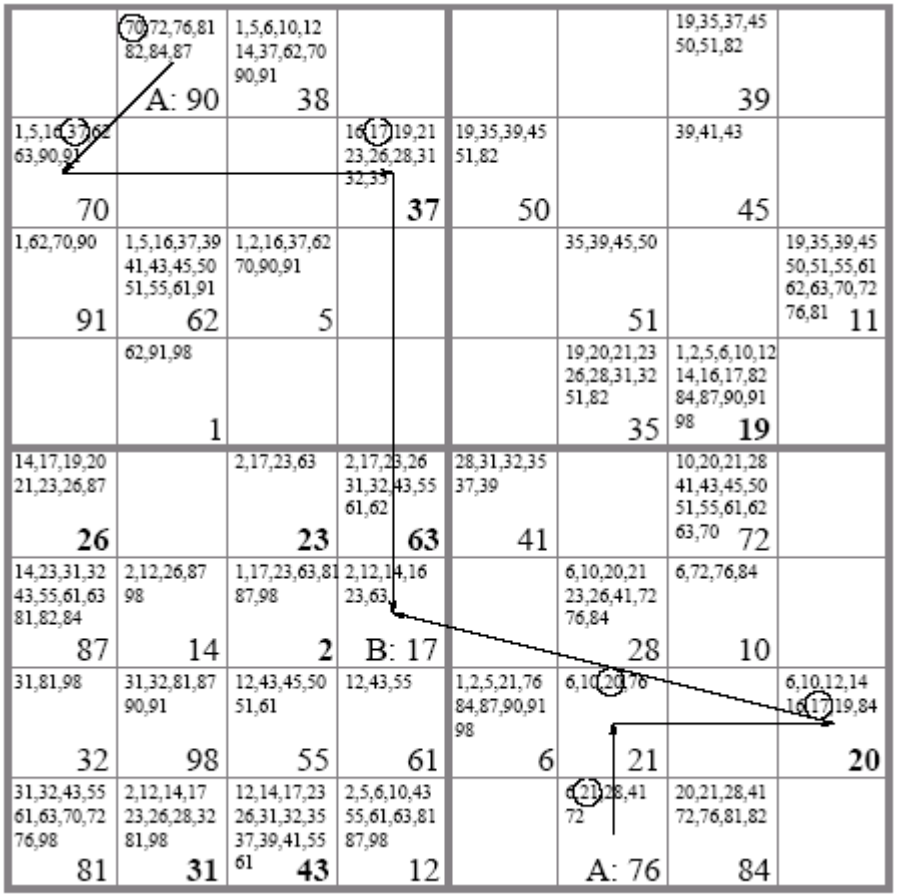
Location Queries

- A queries location of B:
- A stores location information for some other nodes.
- A sends the request to the one that is closest to B, among those about which A has location information.
- Continue until we hit one of B's location servers.
- This works! Why?



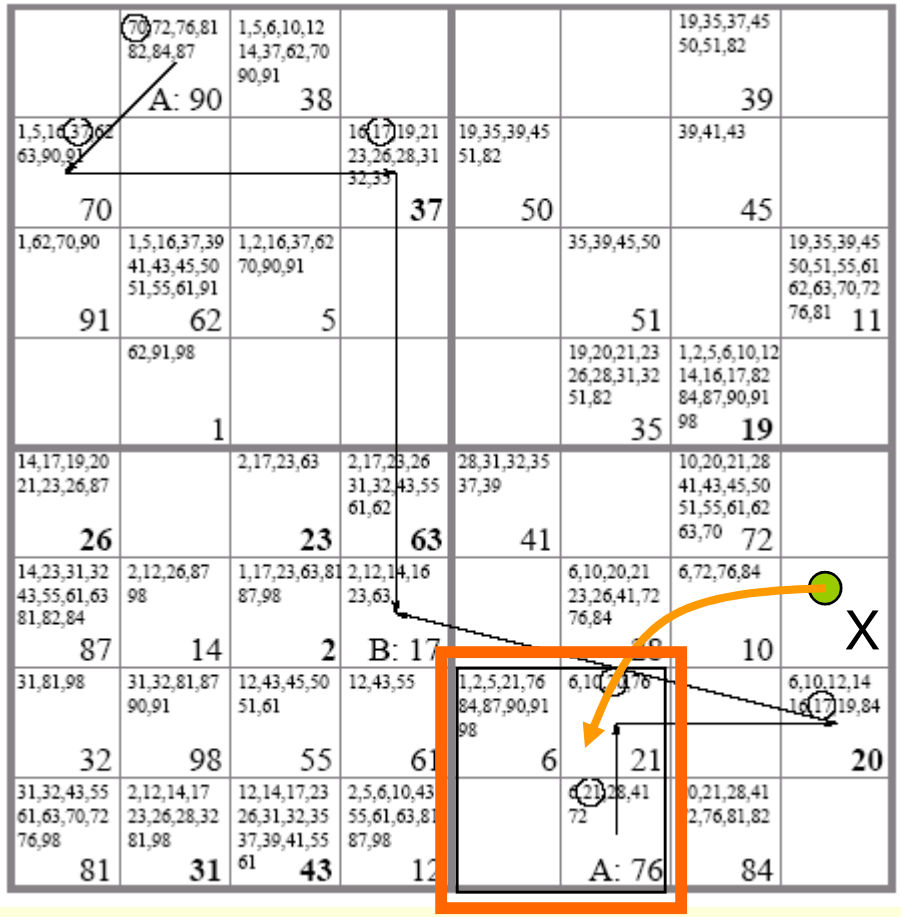
Location Queries

- Claim: the query visits the node closest to B in A's order-i square.
- The query always goes to B's closest node, as the covering scope increases.
- The correctness of the alg: when A's order-i square contains B, the closest node is B itself.
- Proof by induction. It's obvious for order-1 square.



Location Queries

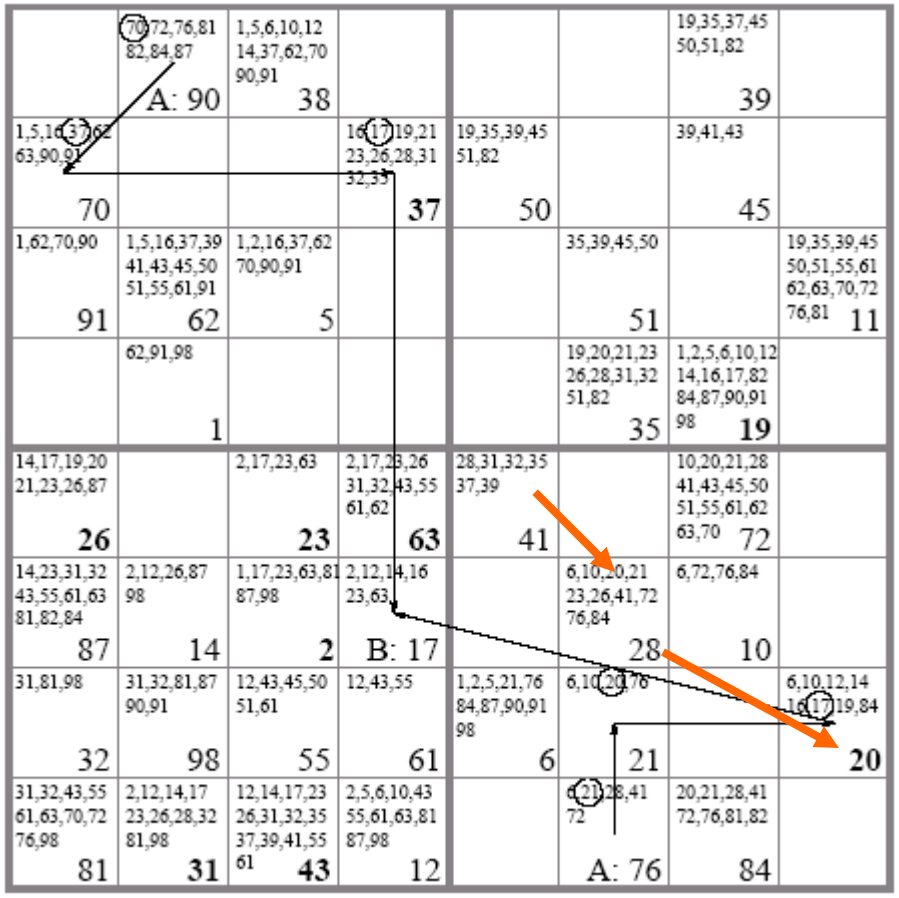
- Assume 21 is B's closest node in A's order-2 square
  ➔ no node is between 17 and 21 in order-1 square
- Suppose a node X in A's order-2 sibling square is between 17 and 21. By the replication rule, X picks 21 as its location server
- 21 stores the location of all the nodes between 17 and 21 in its sibling order-2 square, obviously the one closest to 17



Inform/Update Location Servers

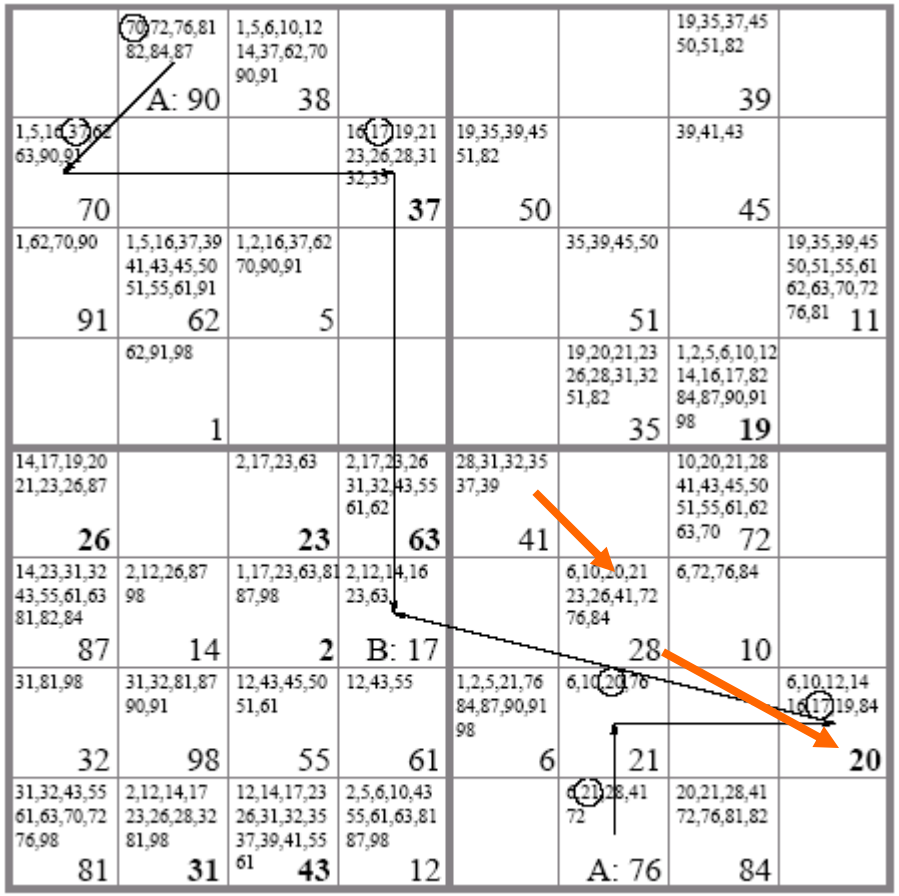
- A can update its location server inside a square S without knowing its identity.
- A routes to a square with geographical routing.
- The first node in the square S performs a location query for A.
- The query ends up at a node closest to A, who is A's location server!

Hidden assumption: the nodes in S have distributed their locations inside S!



Bootstrapping

- When the entire system is turned on, order-1 squares exchange their information with a local protocol, then nodes recruit their order-2 location servers and so on.
- No flooding needed. The location service is constructed by geographical unicast routing only.



Challenges for GLS in a Mobile Network

- Out-of-date location information in servers
- Tradeoff between maintaining accurate location data and minimizing periodic location update messages
  - Adapt location update rate to node speed
  - Update distant servers less frequently than nearby servers
  - Leave forwarding pointers until updates catch up

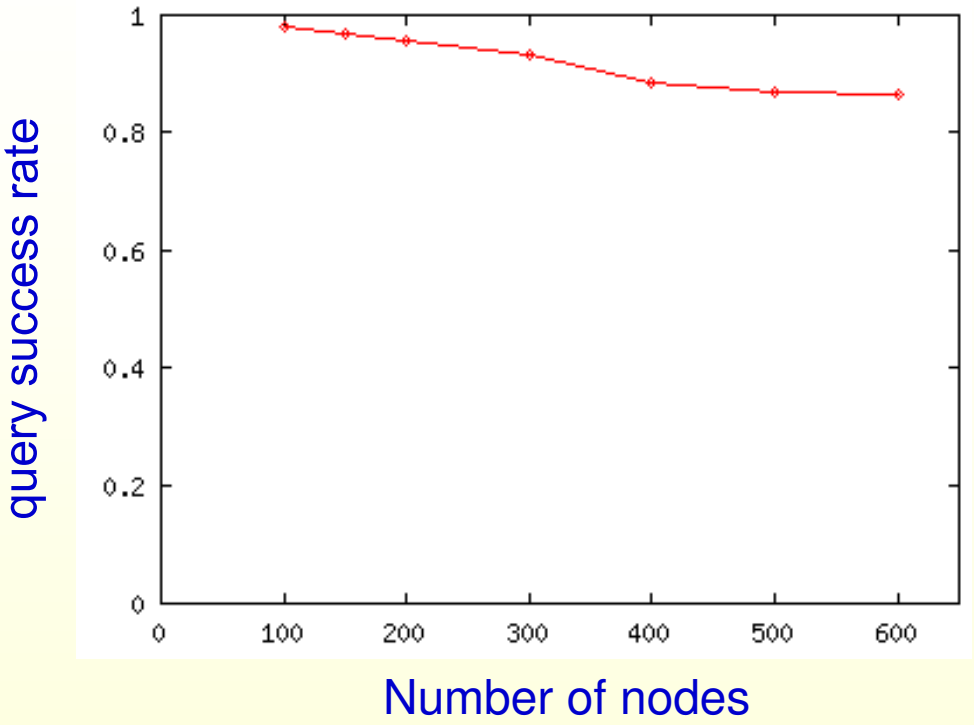
Performance Analysis

- How well does GLS cope with mobility?
- How scalable is GLS?
- How well does GLS handle node failures?
- How local are the queries for nearby nodes?

Simulation Environment

- Simulations using ns with CMU's wireless extension (IEEE 802.11)
- Mobility Model:
  - random way-point with speed 0-10 m/s (22 mph)
- Area of square universe grows with the number of nodes in the network.
    - Achieve spatial reuse of the spectrum
- GLS level-0 square is 250m x 250m
- 300 seconds per simulation

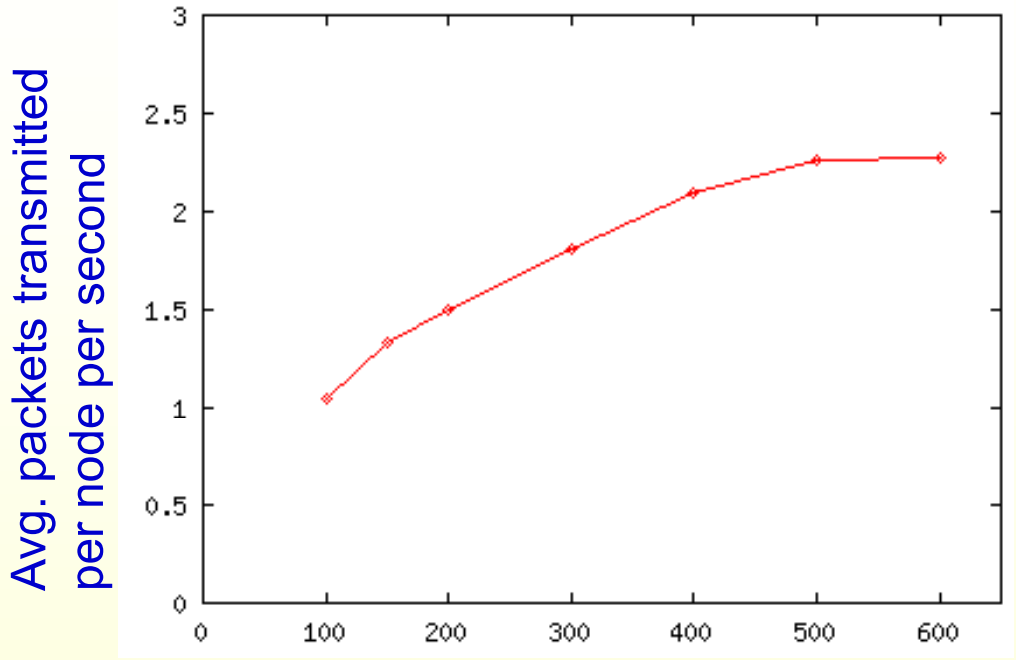
GLS Finds Nodes in Big Mobile Networks



Biggest network simulated:
600 nodes, 2900x2900m
(4-level grid hierarchy)

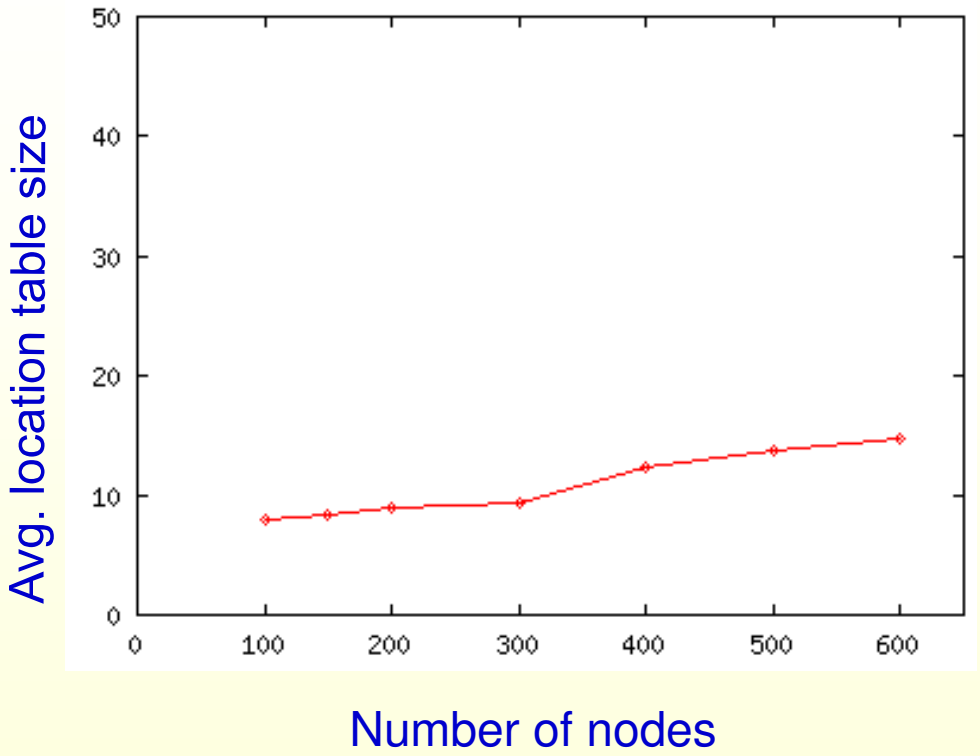
- Failed queries are not retransmitted in this simulation
- Queries fail because of out-of-date information for destination nodes or intermediate servers

GLS Protocol Overhead Grows Slowly



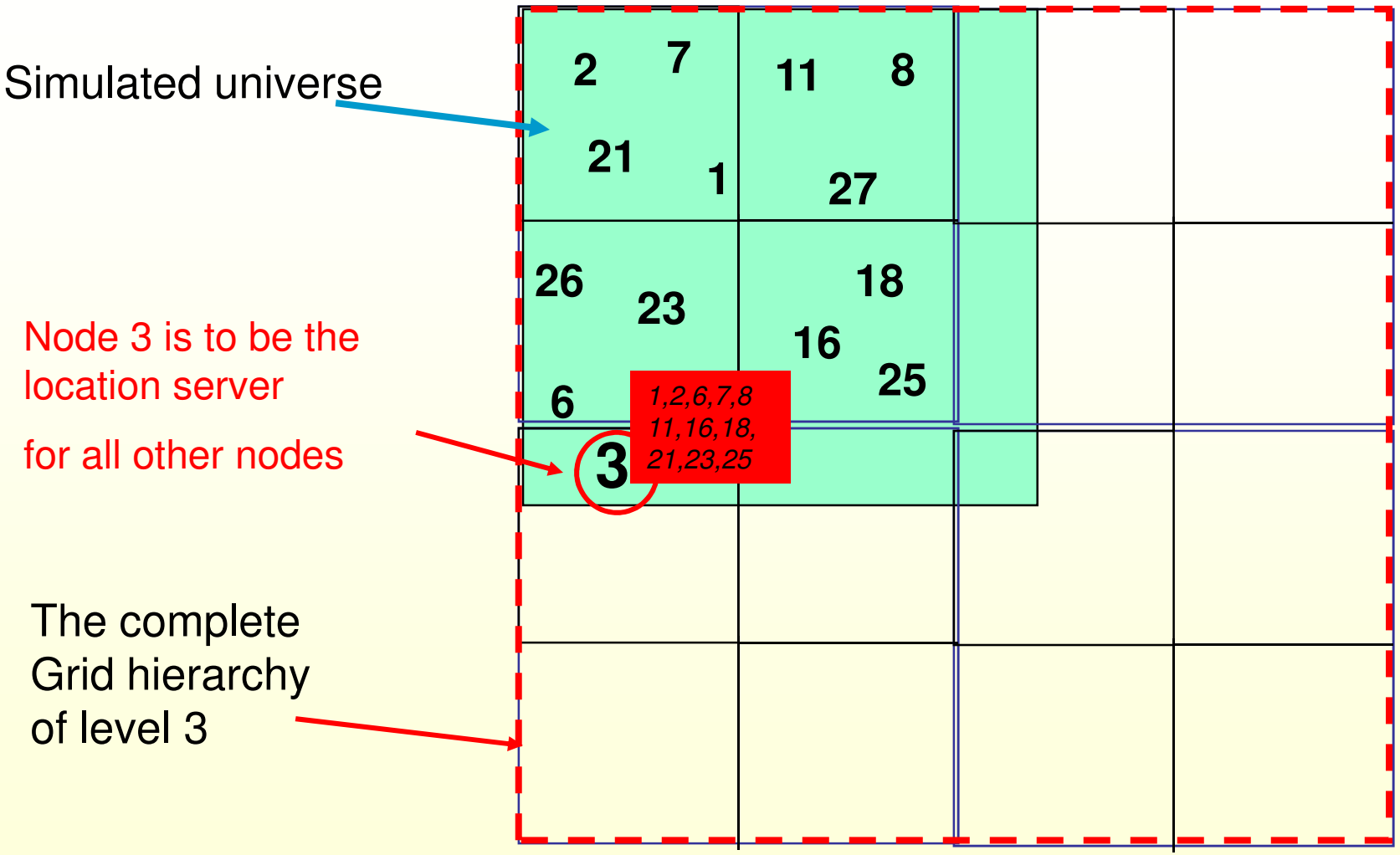
- Protocol packets include: GLS update, GLS query/reply

Average Location Table Size is Small



- Average location table size grows extremely slowly with the size of the network

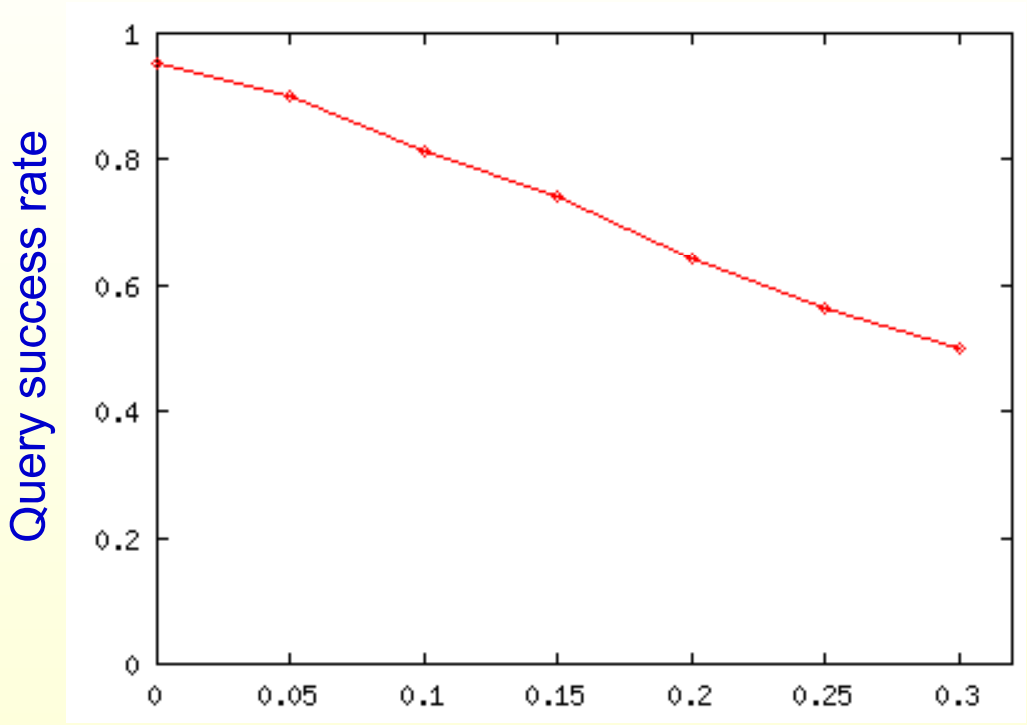
Non-uniform Location Table Size



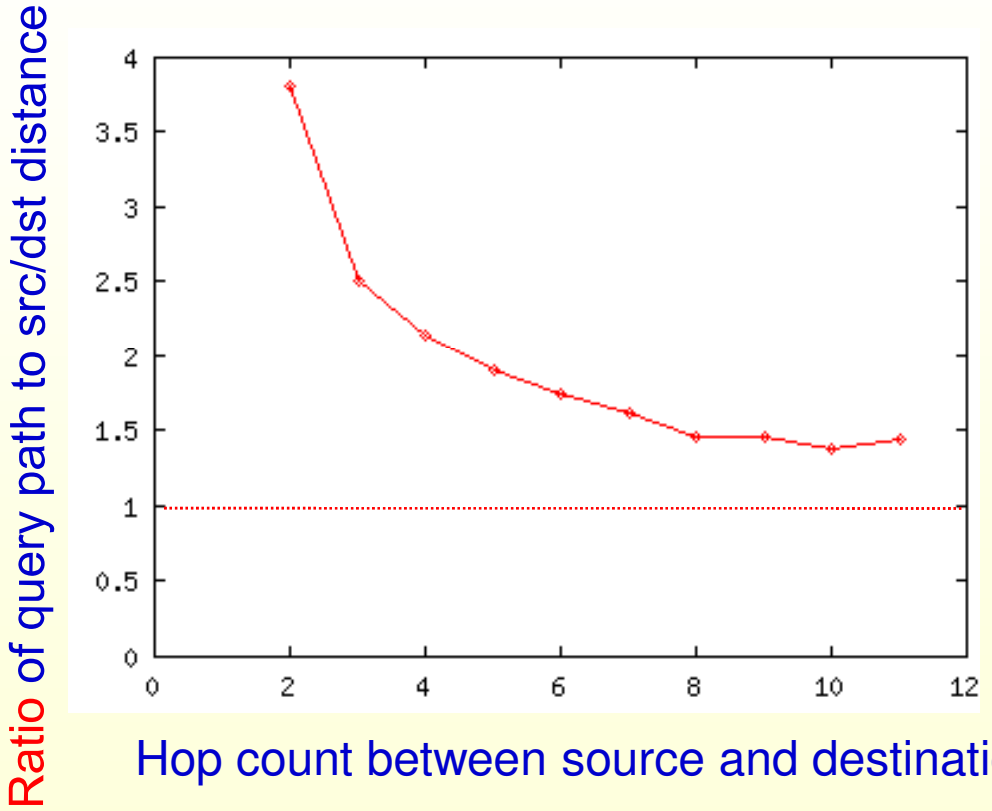
Possible solution: dynamically adjust square boundaries

GLS is Fault Tolerant

• Measured query performance immediately after a number of nodes crash simultaneously.
(200-node-networks)



Query Path Length is Proportional to the Distance between Source and Destination

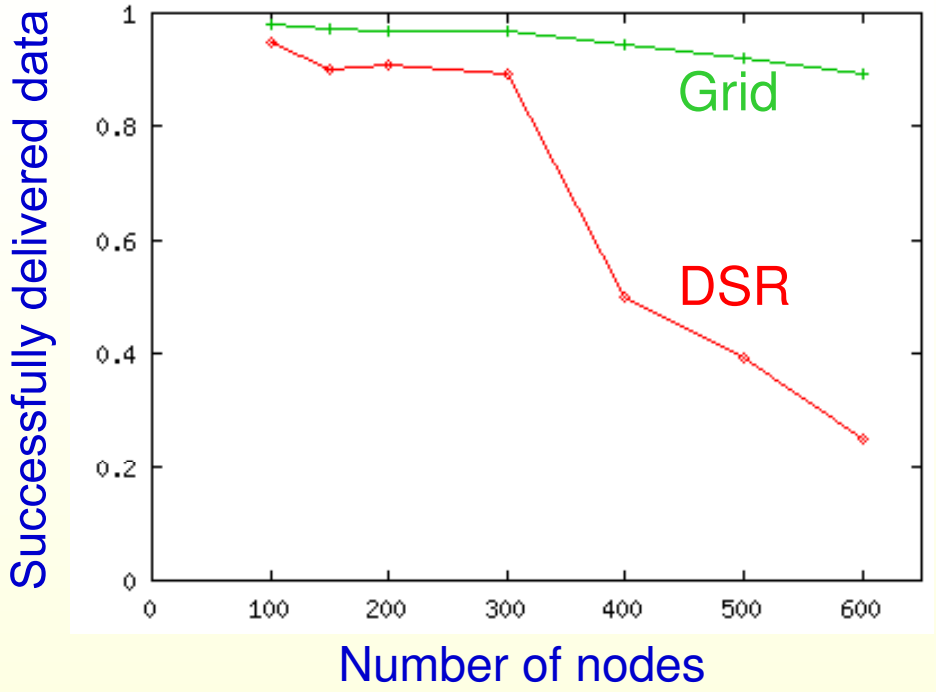


But only on the average ...

Performance Comparison between Grid and DSR

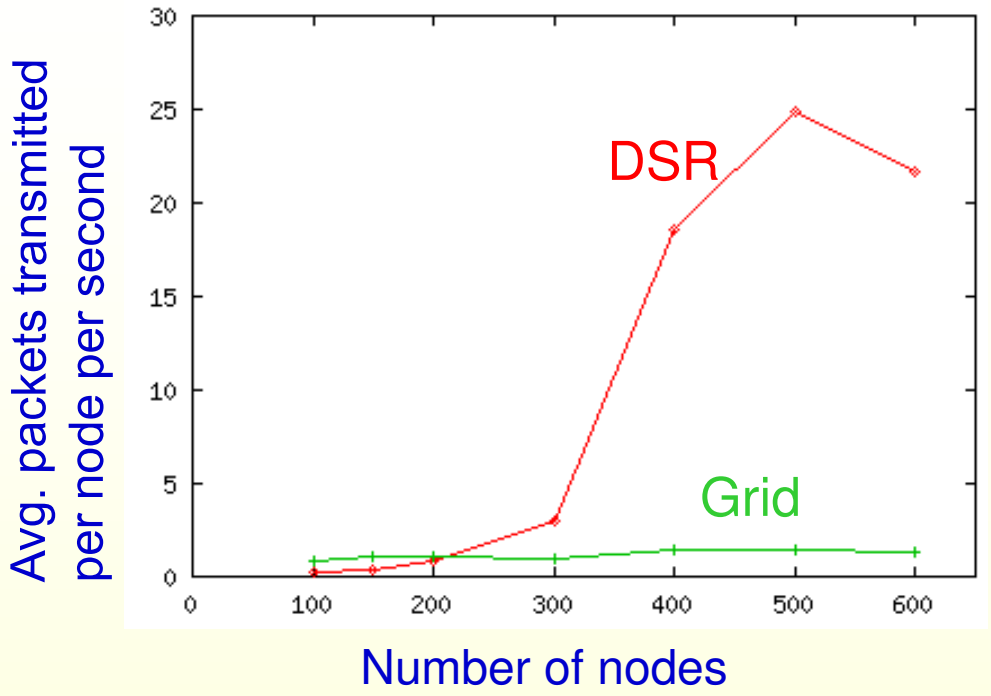
- DSR (Dynamic Source Routing)
  - Source floods route request to find the destination.
  - Query reply includes source route to destination.
  - Source uses source route to send data packets.
- Simulation scenario:
  - 2Mbps radio bandwidth
  - CBR (constant bit rate) sources, 4 128-byte packets/second for 20 seconds.
  - 50% of nodes initiate over 300-second life of simulation.

Fraction of Data Packets Delivered



- Geographic forwarding is less fragile than source routing.
- Why does DSR have trouble with > 300 nodes?

Protocol Packet Overhead



- DSR prone to congestion in big networks:
  - Sources must re-flood queries to fix broken source routes
  - These queries cause congestion
- Grid's queries cause less network load.
  - Queries are unicast, not flooded.
  - Un-routable packets are discarded at source when query fails.

Conclusion

- GLS enables routing using geographic forwarding.
- GLS preserves the scalability of geographic forwarding.
- Current work:
  - Implementation of Grid in Linux

http://pdos.lcs.mit.edu/grid

GLS Summary

- Grid solves location service problem by using geographic routing
- More locality sensitive: a node acquires the location from a nearby server
- Load balancing: location servers are spatially distributed
- Simple rule, simple construction and maintenance
- Worst-case query behavior is not bounded, however ☹

Open Issues on Location Service

- Make use of node mobility?
  - When two nodes pass by, they keep each other's info.
- Security issue with location service?

In-Network Information Aggregation

Overview

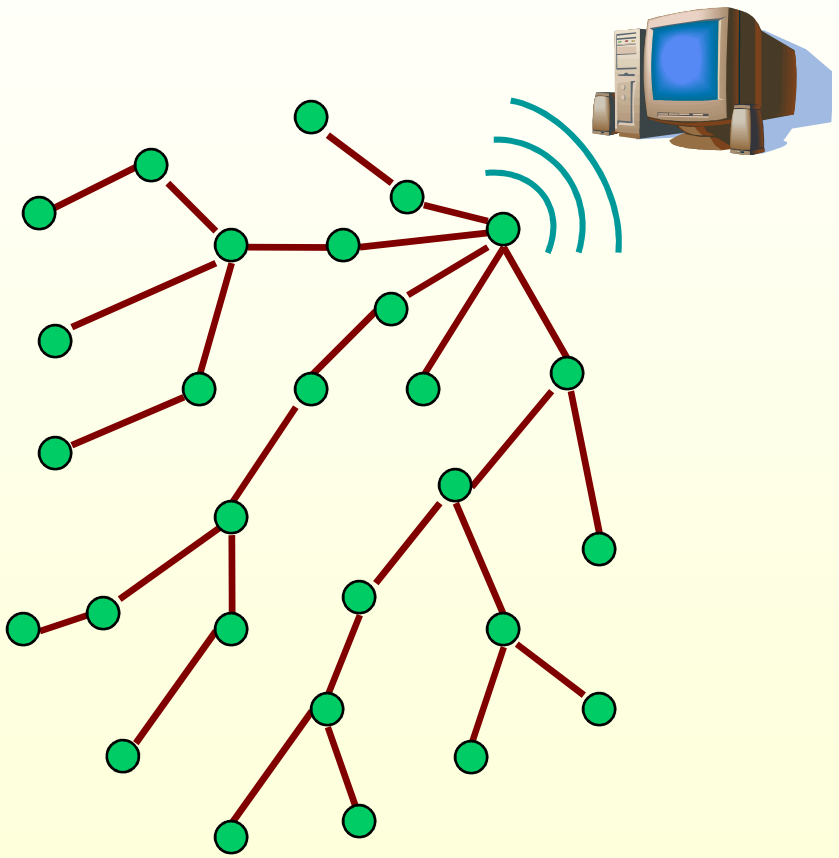
- Aggregates on sensor data
  - Simple aggregates:
    - MIN, COUNT, AVG
    - In-network Aggregation [tinyDB]
  - Sophisticated aggregates:
    - Quantile/Percentile (Ex: MEDIAN)
    - Range-query, Histogram, Frequent items

- q-digest: Novel technique to compute sophisticated aggregates

[Shrivastava, Buragohain, Agrawal, Suri, ‘04]

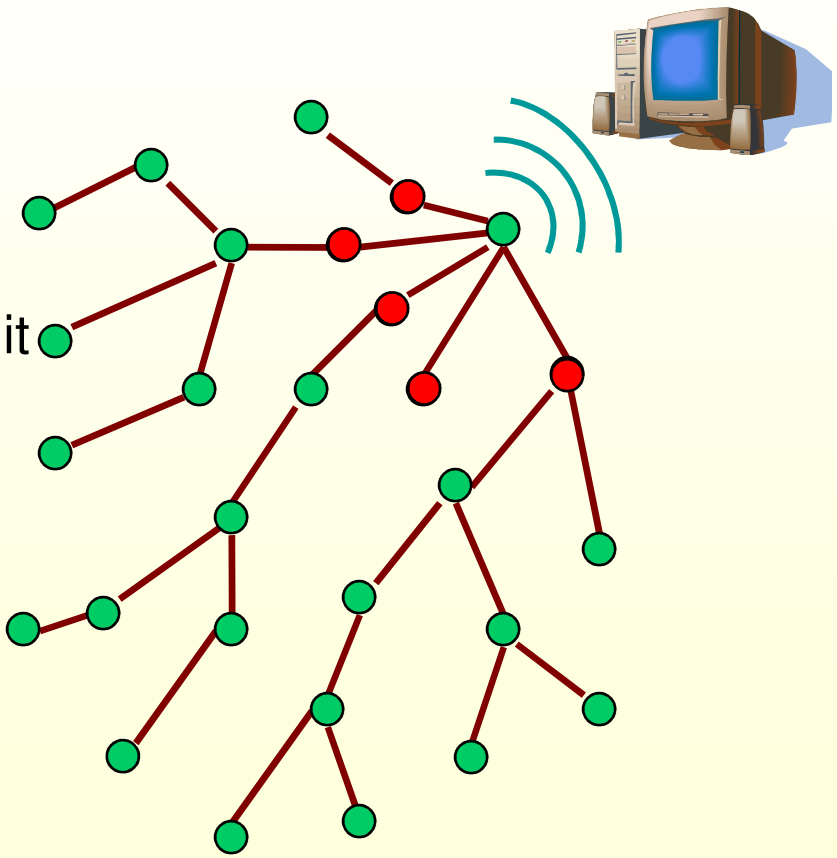
Sensor Networks

- Base station connected to the outside world
- Routing tree rooted at base station
- Query issued at base station
  - query propagates in the network
  - results are collected back at the base station



Processing in Sensor-Nets

- Naïve approach:
  - Send all values to base station
  - Compute result
- Not power efficient:
  - Nodes near base station transmit $O(n)$ bytes
- In-network aggregation:
  - Each node computes partial results for data in its subtree
  - Sends only this partial result (or a *summary*)



Aggregation

- Simple aggregates: MIN, MAX, AVG
- AVG: At each node
  - Receive summaries (<SUM, COUNT>) from its children
  - Merge summaries; transmit it to the parent
- Constant sized summary transmitted per node

What about more complex, but important queries:
MEDIAN, Quantiles, Range queries?

Key Query: Quantiles

ϕ-quantile: Value with rank = $\phi\, n$

(in sorted sequence of n values)

- MEDIAN is .5-quantile
- Computing MEDIAN exactly is expensive
  ($O(n)$ bytes transmitted per node)

ε-approx. quantile: Value with rank = $(\phi \pm \varepsilon)n$

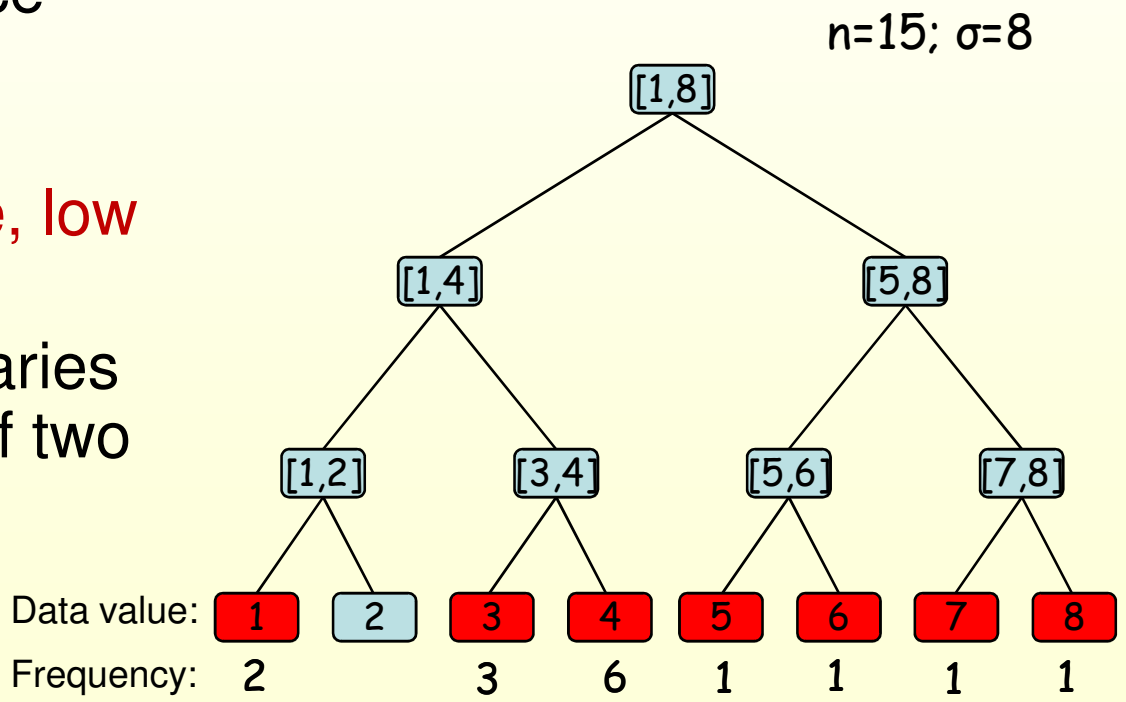
Quantile Summary: q-Digest

- Answers all quantile queries with ε-precision
- Communicated data size depends on ε, independent of total item size n !
- Distributes power consumption evenly
- Independent of routing topology
- Generic scheme: single summary answers variety of queries

Structure of q-digest

Given: *n* values from Range [1…σ]

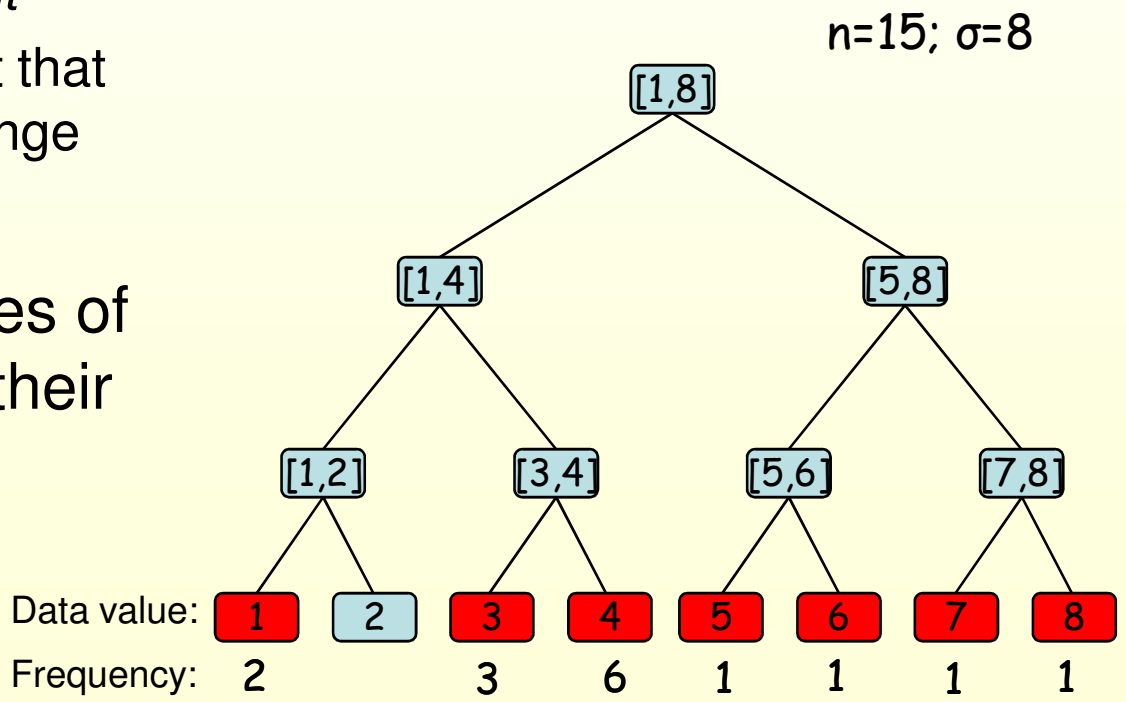
- Given σ buckets, exact quantiles are possible
- Merge buckets: reduce memory (but lose information)
- Combine consecutive, low frequency buckets
- Larger bucket boundaries aligned to multiples of two
- Hierarchy of buckets



Structure of q-digest

- Logical view: binary tree
  - Each node has a range [*min, max*] and a *count*
  - Each node is a bucket that counts values in its range

q-digest: subset of nodes of this logical tree, with their counts.



Structure of q-digest

Which buckets/when to merge?

Compression factor: *k*

(Make buckets of count ~ *n/k*
=> O(*k*) buckets)

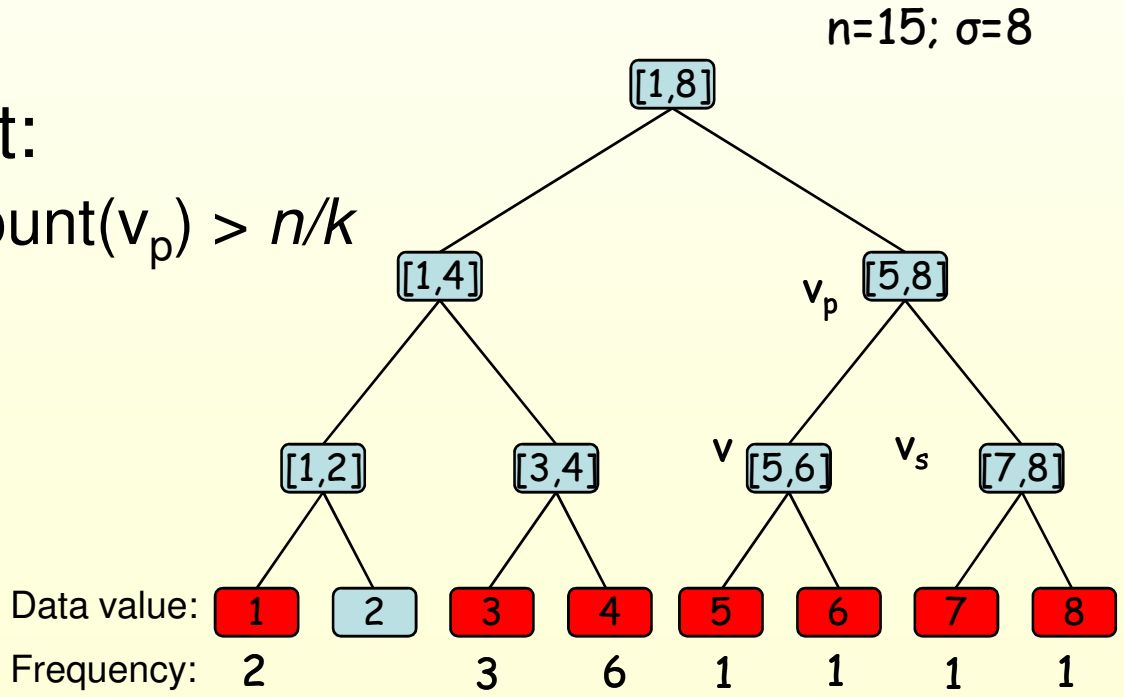
Properties of q-digest:

1) count(v)+count(v_s)+count(v_p) > *n/k*

(reduce memory)

2) count(v) ≤ *n/k*

(reduce error)

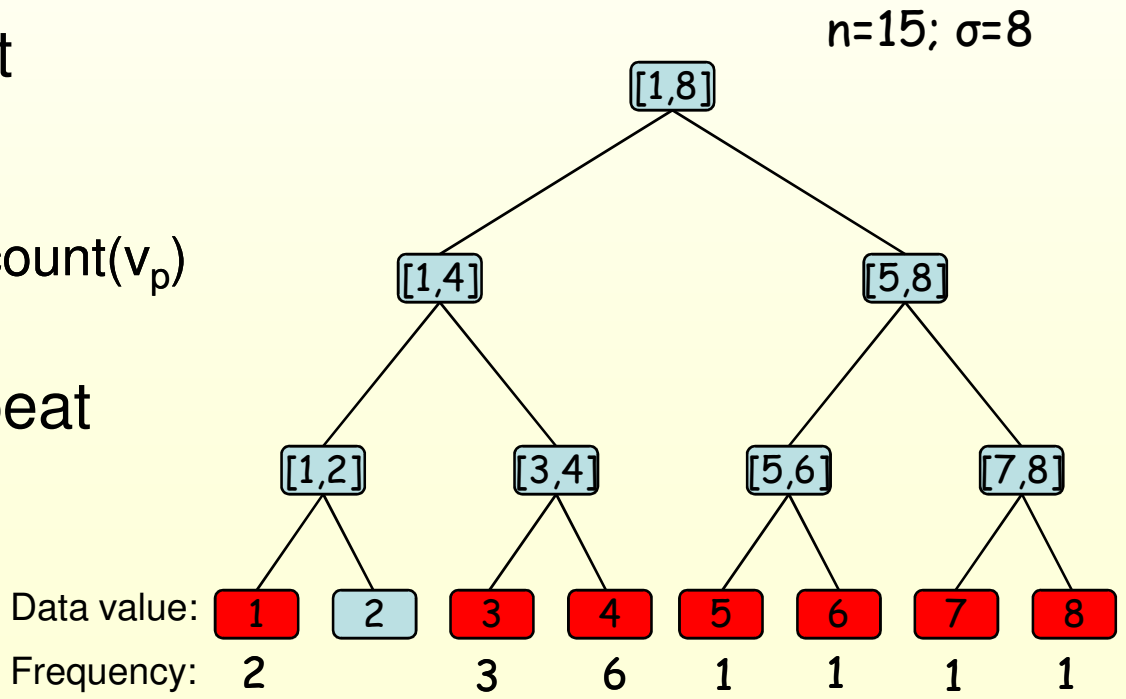


Computing q-digest

Properties of q-digest:
1) count(v)+count(v_s)+count(v_p) > *n/k*
2) count(v) ≤ *n/k*

Algorithm: COMPRESS

- Start at bottom level
- For each node v that violates Prop. 1
  - Set count(v_p) = count(v)+count(v_s)+count(v_p)
  - Delete v and v_s
- Go one level up; repeat

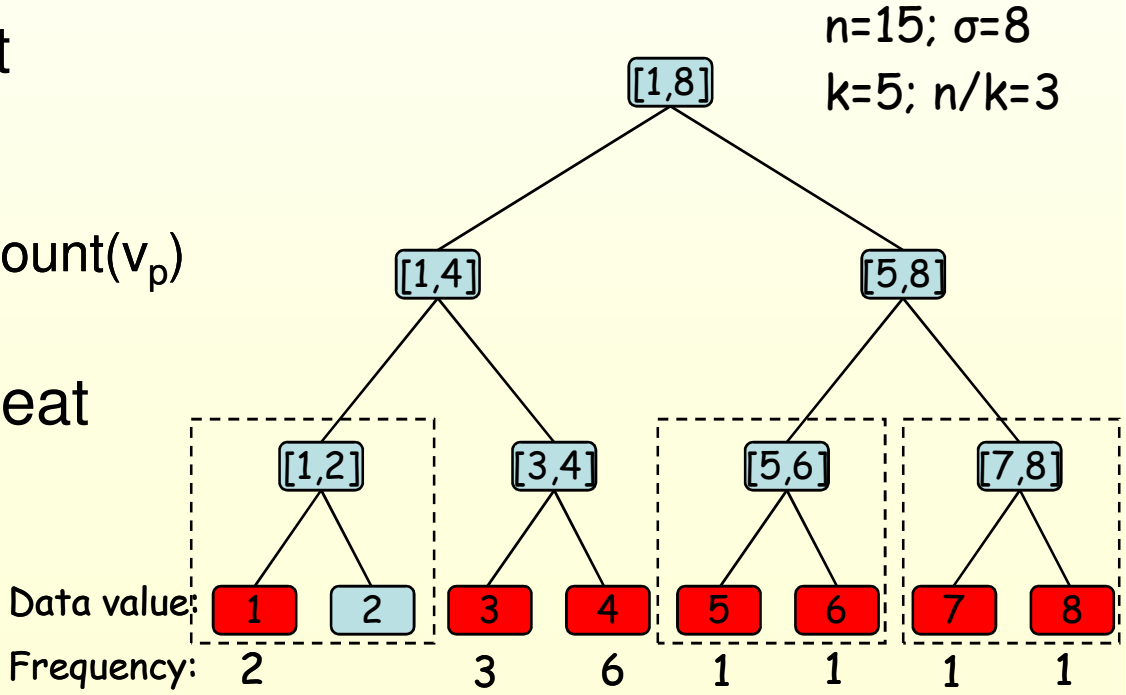


Computing q-digest

Properties of q-digest:
1) count(v)+count(v_s)+count(v_p) > *n/k*
2) count(v) ≤ *n/k*

Algorithm: COMPRESS

- Start at bottom level
- For each node v that violates Prop. 1
  - Set count(v_p) = count(v)+count(v_s)+count(v_p)
  - Delete v and v_s
- Go one level up; repeat

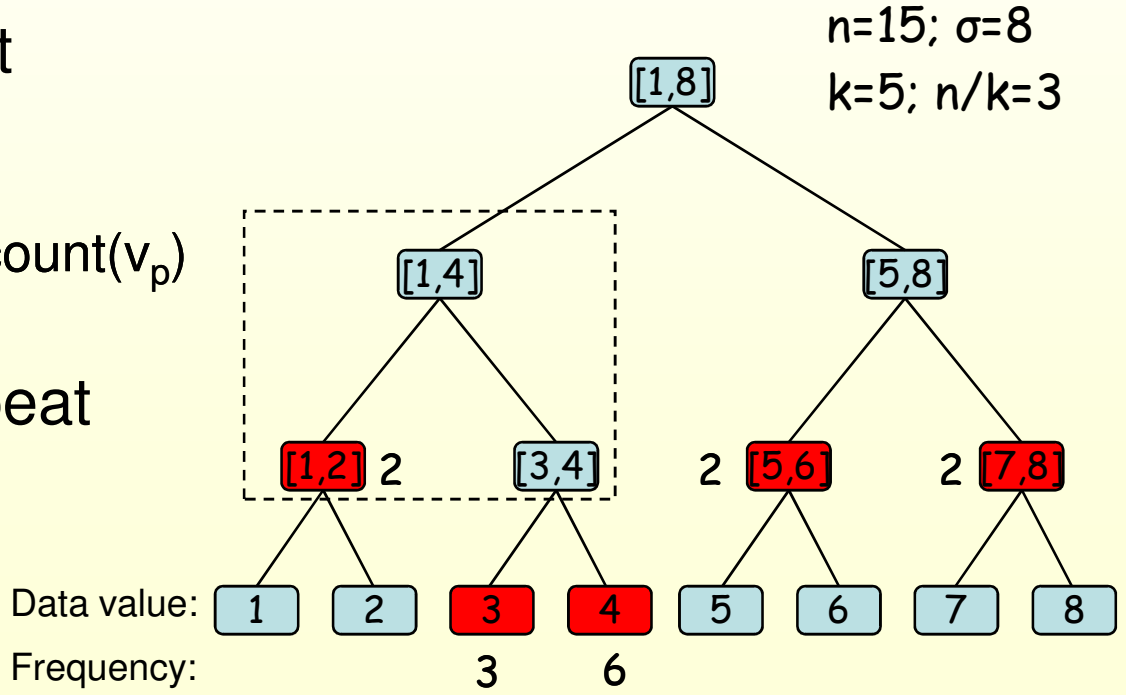


Computing q-digest

Properties of q-digest:
1) count(v)+count(v_s)+count(v_p) > *n/k*
2) count(v) ≤ *n/k*

Algorithm: COMPRESS

- Start at bottom level
- For each node v that violates Prop. 1
  - Set count(v_p) = count(v)+count(v_s)+count(v_p)
  - Delete v and v_s
- Go one level up; repeat



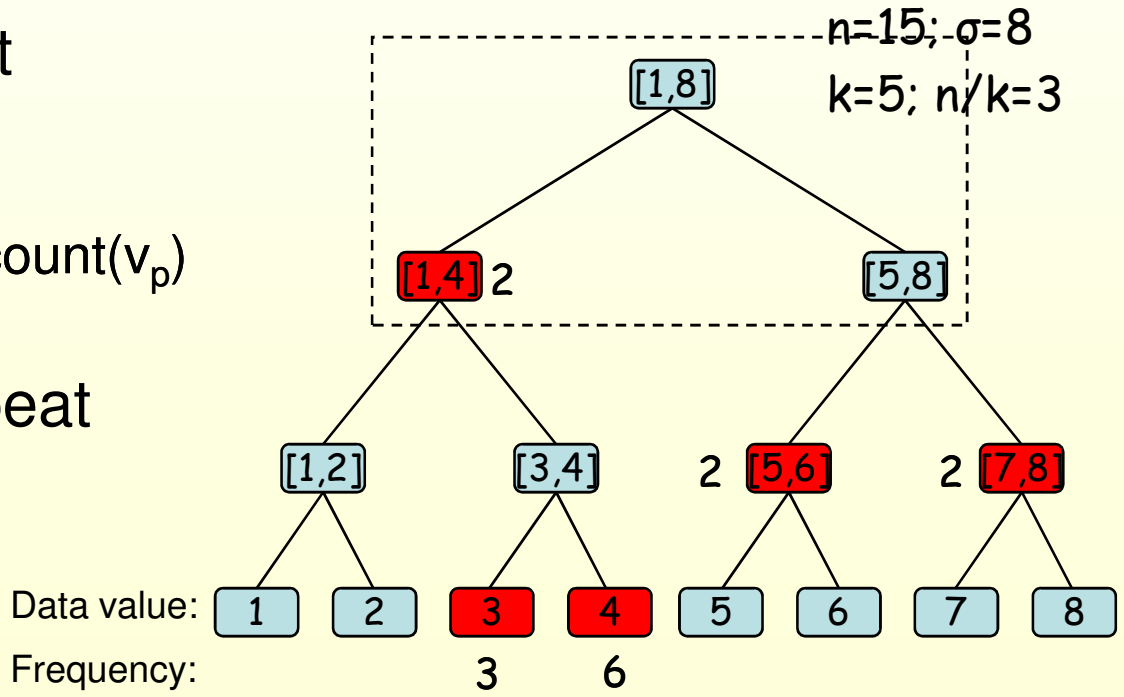
Computing q-digest

Algorithm: COMPRESS

- Start at bottom level
- For each node v that violates Prop. 1
  - Set count(v_p) = count(v)+count(v_s)+count(v_p)
  - Delete v and v_s
- Go one level up; repeat

Properties of q-digest:

1) count(v)+count(v_s)+count(v_p) > *n/k*
2) count(v) ≤ *n/k*

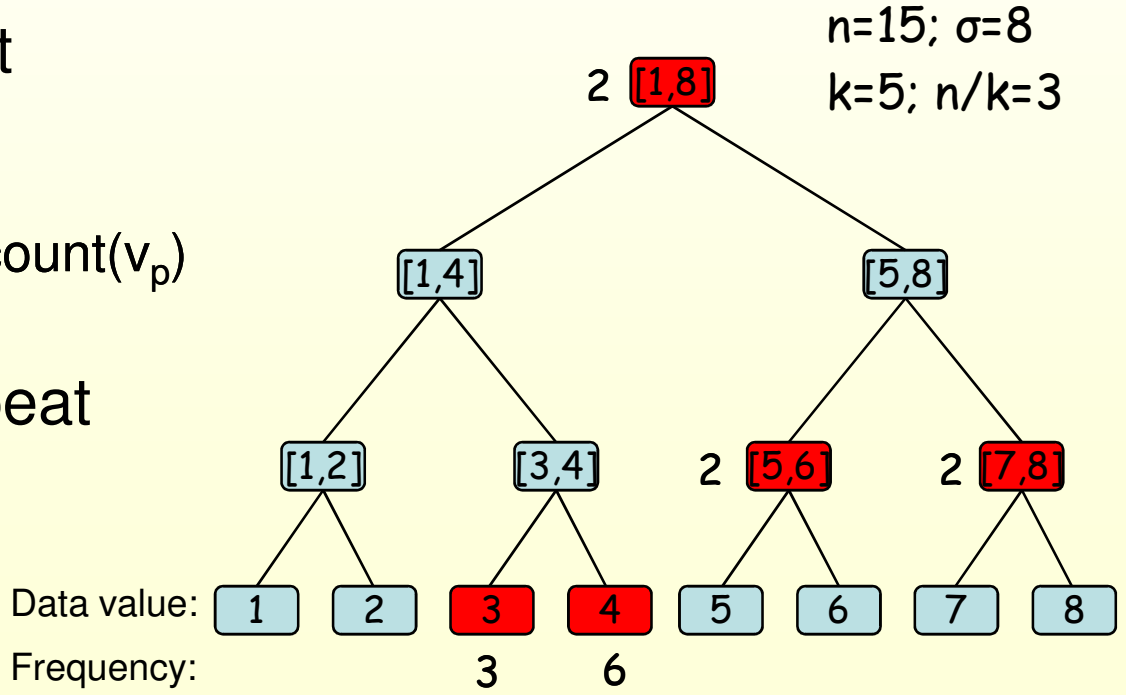


Computing q-digest

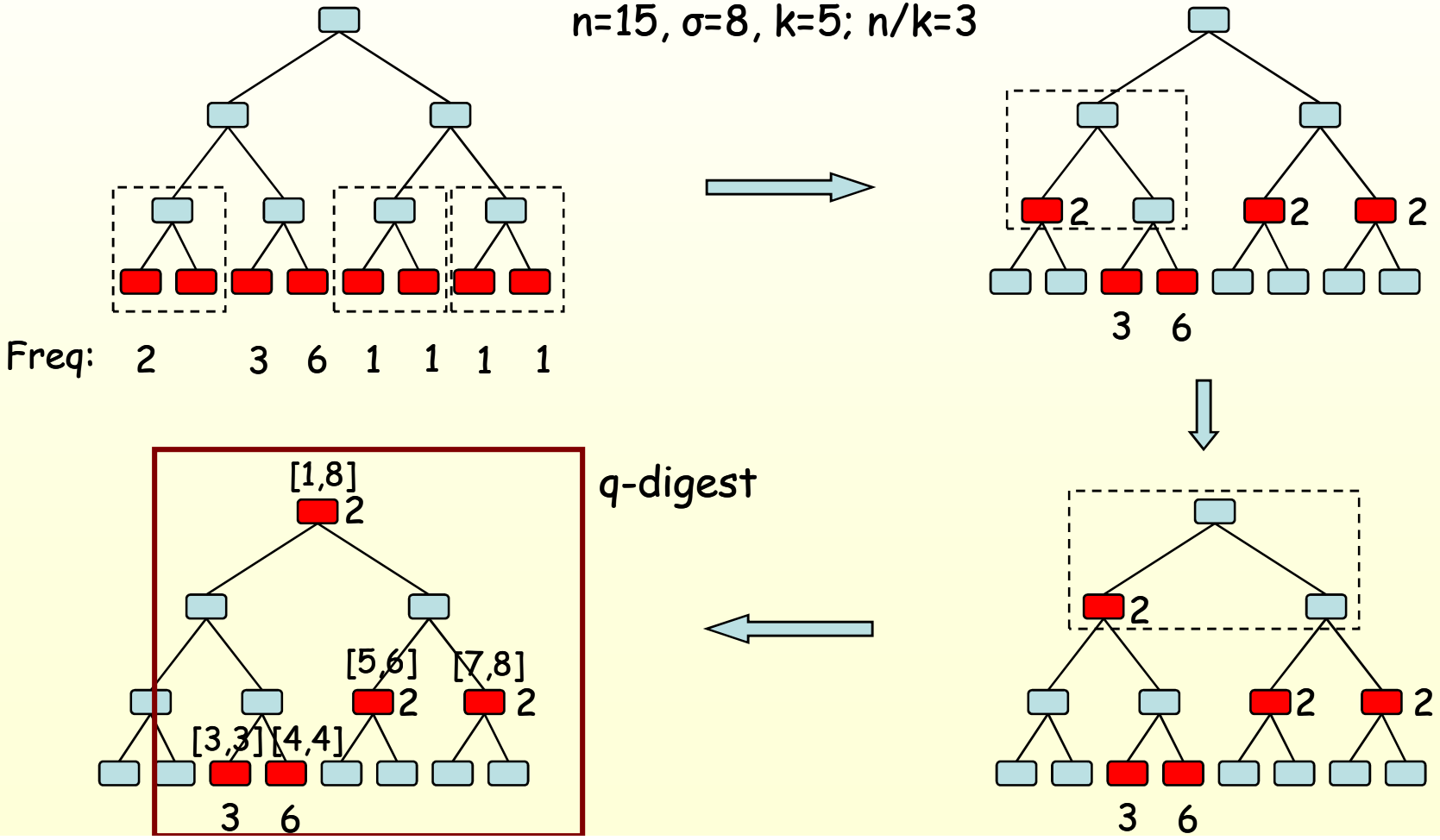
Algorithm: COMPRESS

- Start at bottom level
- For each node v that violates Prop. 1
  - Set count(v_p) = count(v)+count(v_s)+count(v_p)
  - Delete v and v_s
- Go one level up; repeat

Properties of q-digest:
1) count(v)+count(v_s)+count(v_p) > *n/k*
2) count(v) ≤ *n/k*

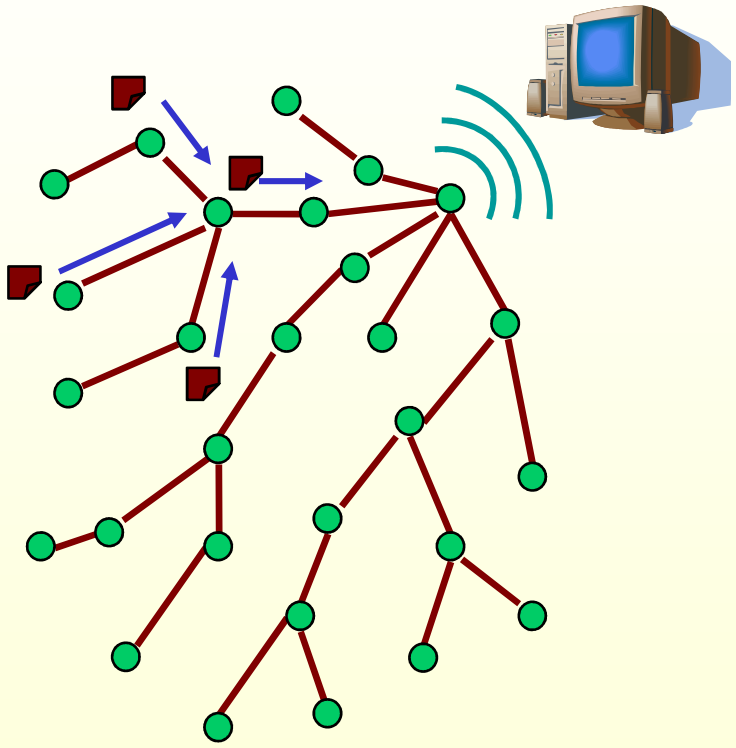


Computing q-digest: Example



Computing q-digest over a Network

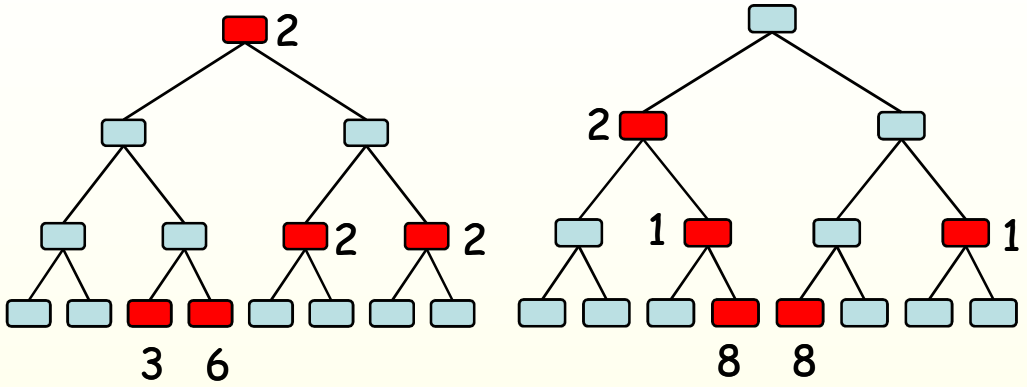
- Each sensor:
  - Receives q-digests from all its children
  - Makes a trivial q-digest containing its own (single) value
  - Merges all of them into a single q-digest
  - Pass it to its parent-sensor



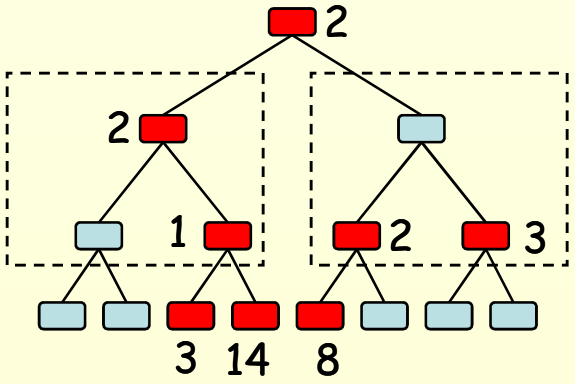
Need to merge two q-digests

Merging q-digests

$Q_1(\sigma,n_1,k) + Q_2(\sigma,n_2,k)$
$\Rightarrow Q(\sigma,n_1+n_2,k)$

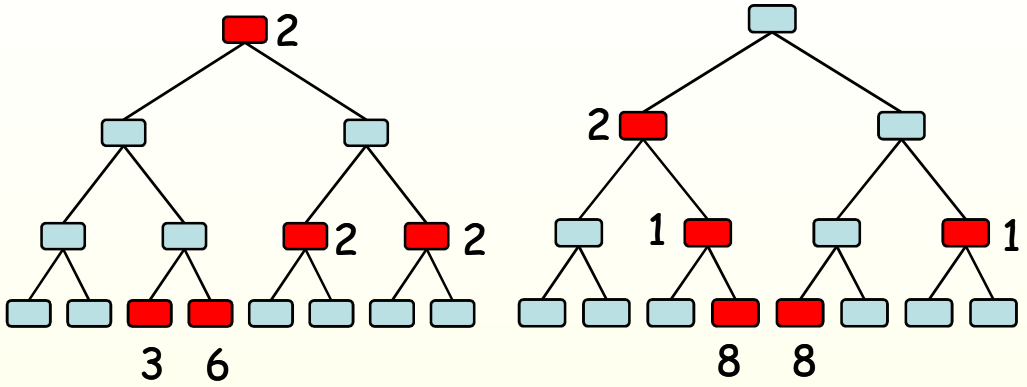


- Take union of q-digests
- Add counts of buckets with same range
- Size of q-digest may have increased
- COMPRESS



Merging q-digests

$Q_1(\sigma,n_1,k) + Q_2(\sigma,n_2,k)$
$\Rightarrow Q(\sigma,n_1+n_2,k)$



- Take union of q-digests
- Add counts of buckets with same range
- Size of q-digest may have increased
- COMPRESS

n_1=15;

n_1/k=3

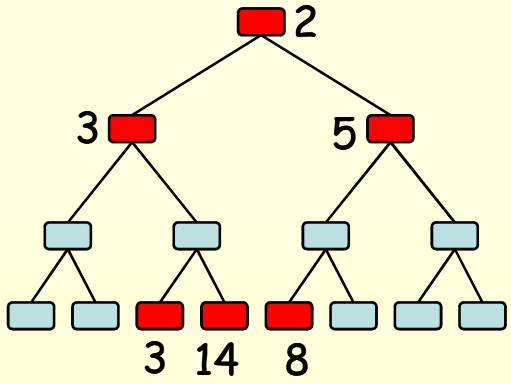
k = 5

$n=(n_1+n_2)$=35;

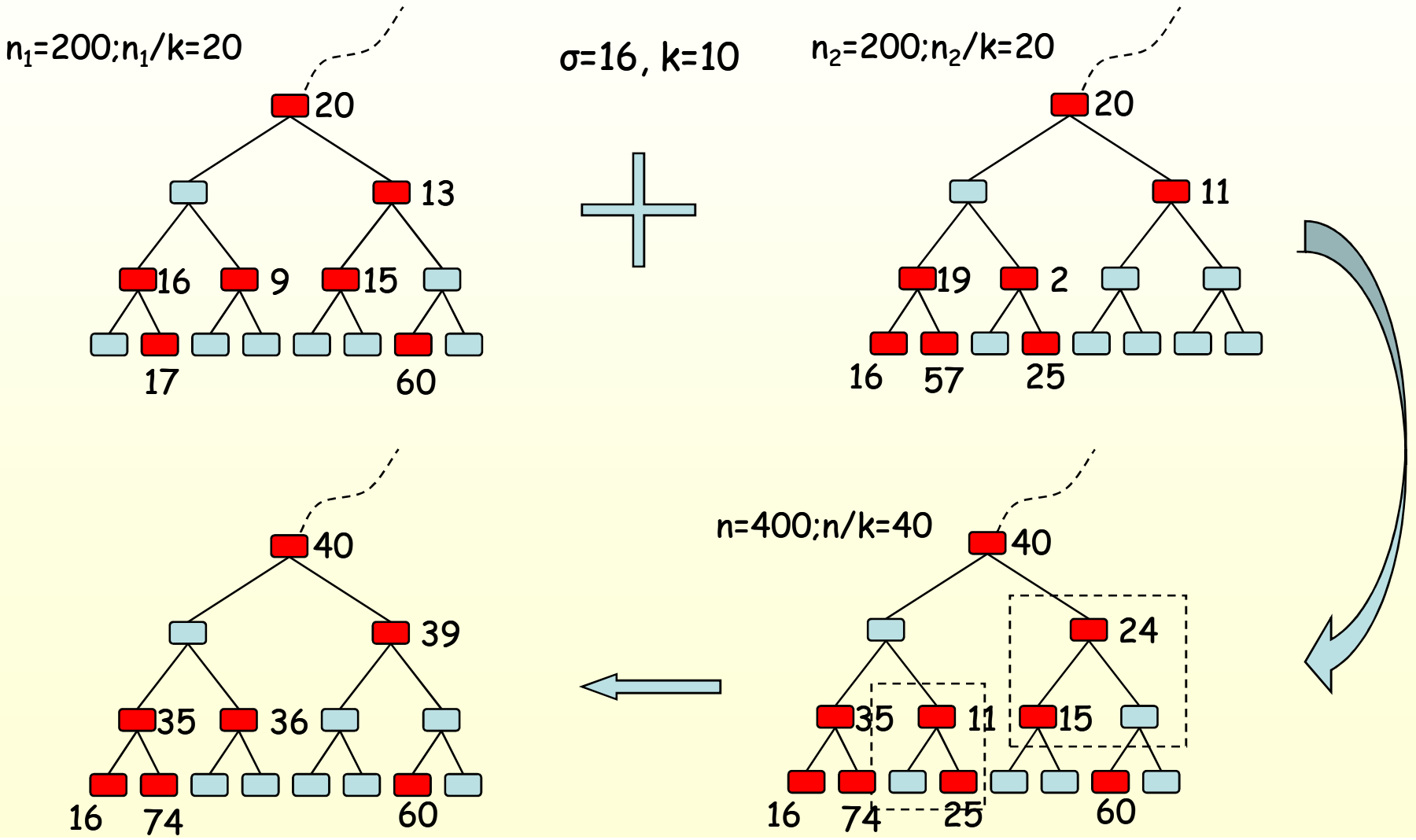
n/k=7

n_2=20;

n_2/k=4



Merging q-digests: Example



Analysis: Size of q-digest

- The merge operation conserves q-digest properties
- By Prop. 1

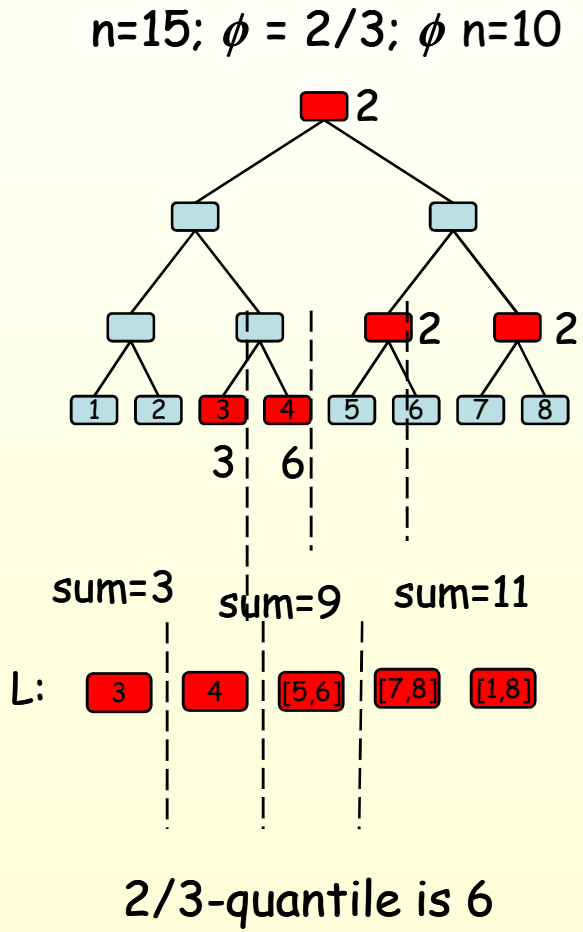
$\Sigma_{v \in Q}\{count(v)+count(v_s)+count(v_p)\} > |Q|^* n/k$

L.H.S. $< 3^*\Sigma_{v \in Q}\{count(v)\} < 3n$

Theorem: The q-digest constructed with compression factor k has size at most 3k.

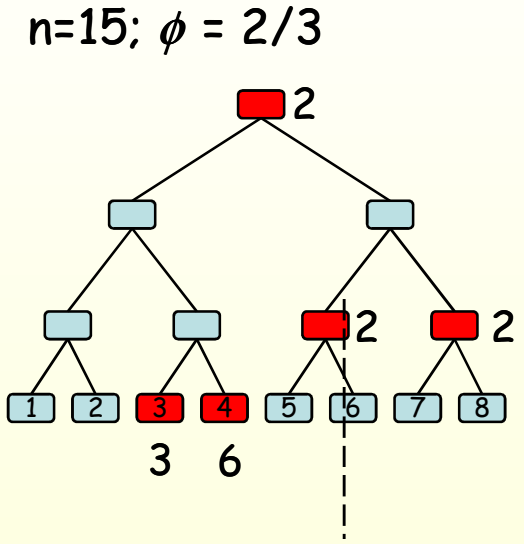
Quantile Query

- Make list L:
  Sort nodes in increasing order of right end-points; break ties by putting smaller range first (post-order traversal)
- Scan L and add the counts, until for some node v, the sum is $> \phi n$
- Report right endpoint v.max as the approx. quantile.



Quantile Query: Analysis

- Error?
  - Values less than v.max can be counted in ancestors of v
  - v can have only log(σ)-1 ancestors
  - Count of each node ≤ *n/k* (by Prop 2)
  - Rank error < (*n/k*)*log(σ)
    (ε < log(σ)/*k*)
- Combine with size-theorem



Theorem: Given memory *m* to build a q-digest, it is possible to answer any quantile query with error ε, where $\varepsilon < 3\log(\sigma)/m$

Modes of Operation

Nice size-error tradeoff:

- User specifies summary size (m):
  Ensure error $\varepsilon < 3\log(\sigma)/m$
- User specifies maximum error (ε):
  Ensure maximum summary size $m < 3\log(\sigma)/\varepsilon$
  (As compared to $O((\log n)^2/\varepsilon)$ [GK-04])

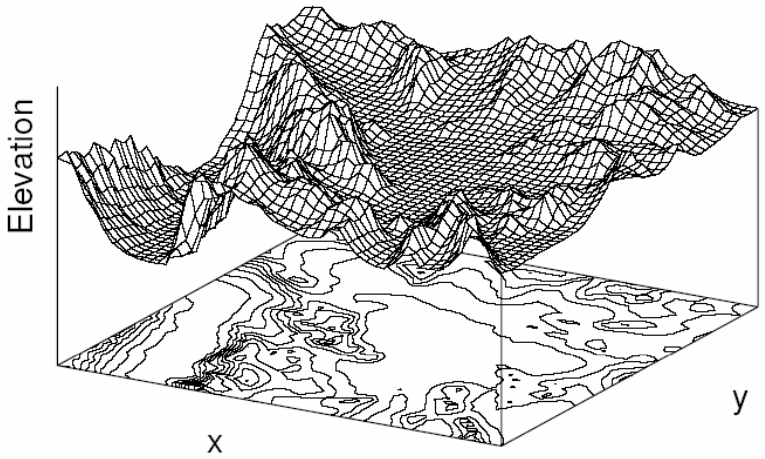
Generic Scheme

q-digest can answer several queries:

- Quantile query: Any ϕ-quantile can be answered from a single q-digest
- Rank query: Find rank of given value x
- Range query: Number of values in range [low,high]
- Build histograms
- Frequent values: Values reported by more than εn sensors

Experimental Setup

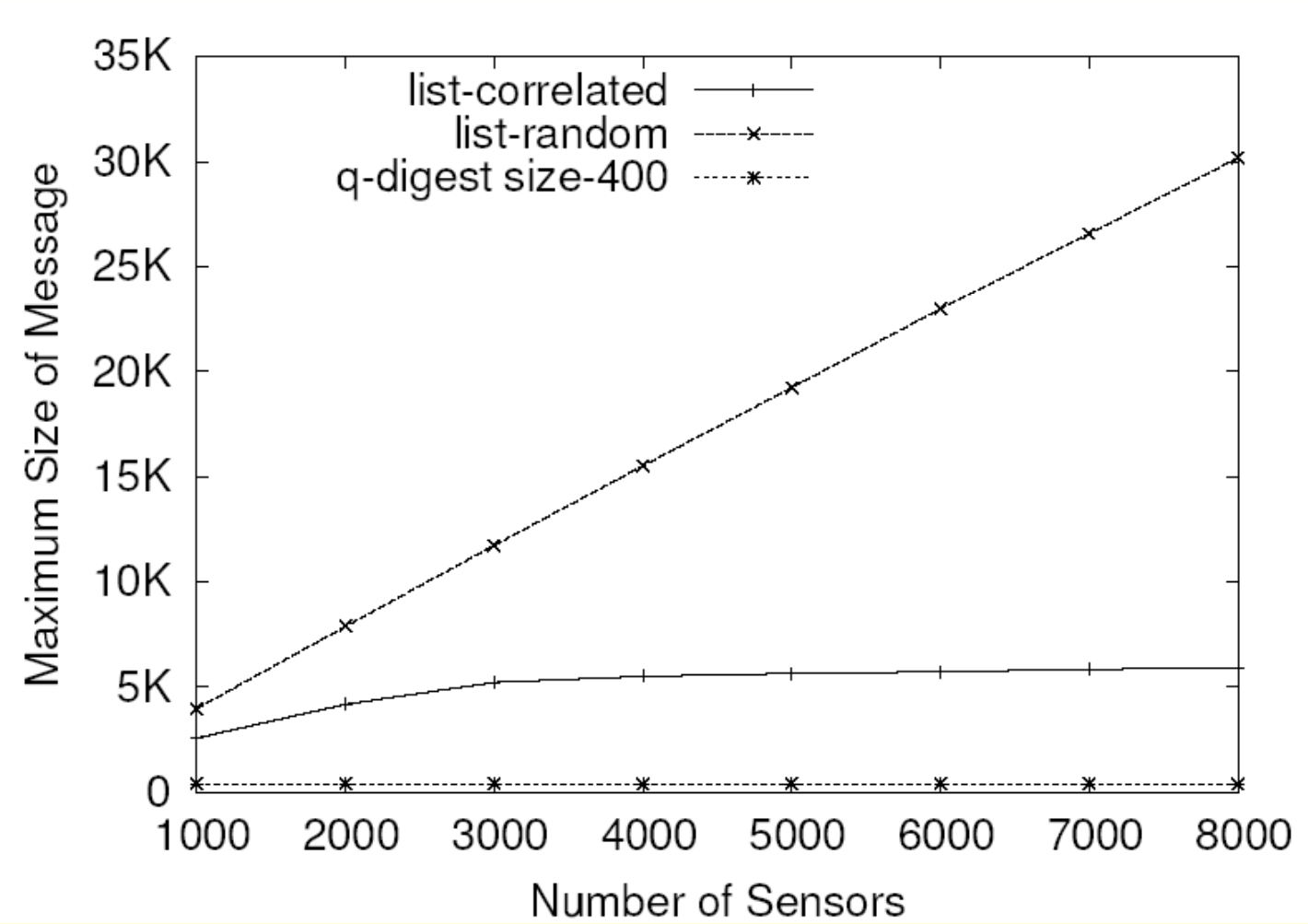
- Simulation using:
  - Network of size n=1000~8000
  - Values are 2-byte integers (σ=64,000)
- Data-sets
  - Random: Each sensor is assigned a random value
  - Correlated: Each sensor data is correlated with its neighbor
- Compare with naïve approach:
  Maintain LIST of all (non-empty) 1-size buckets and their counts



Correlated dataset

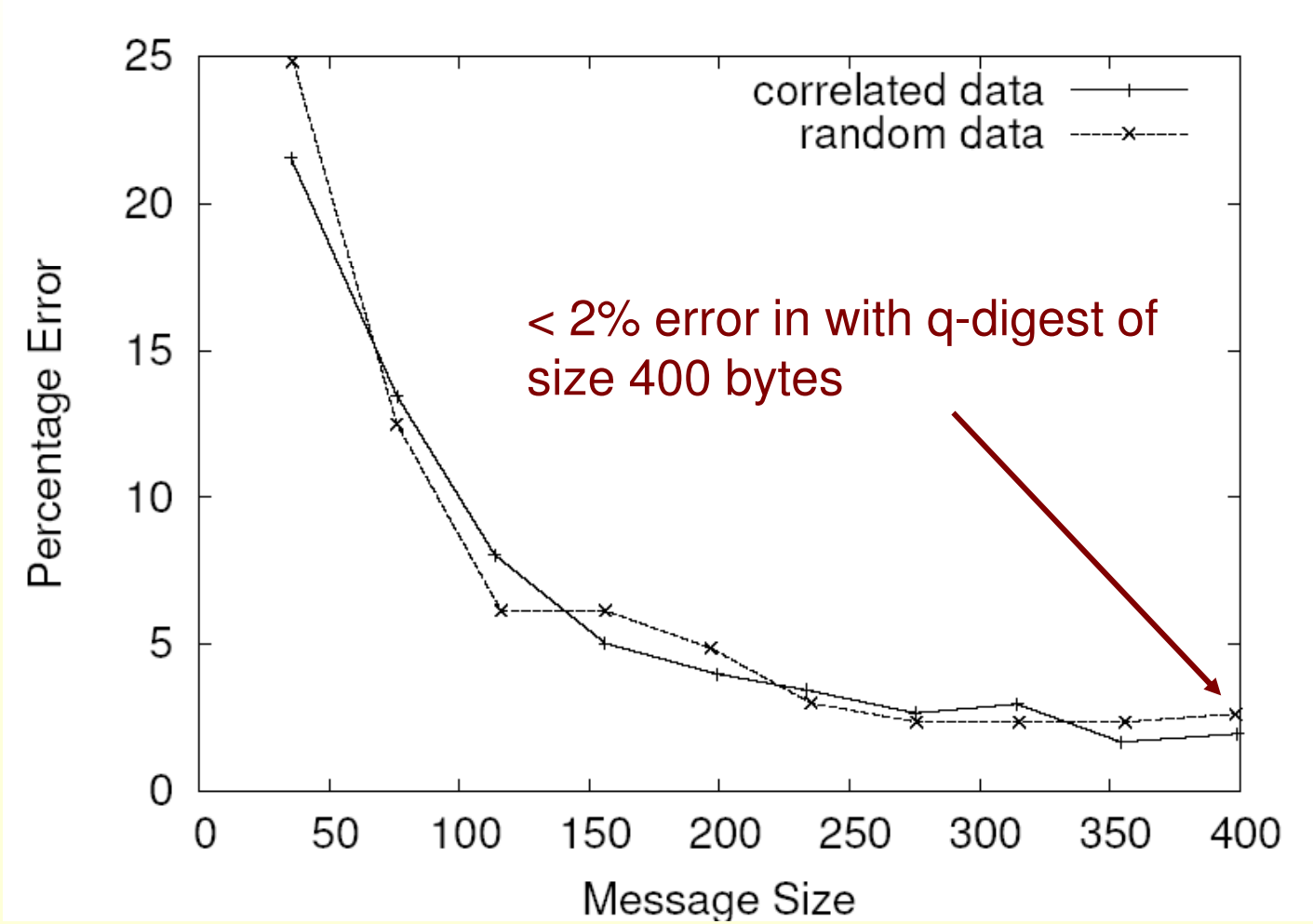
Results: Transmission Cost

q-digest of size: m=400 bytes



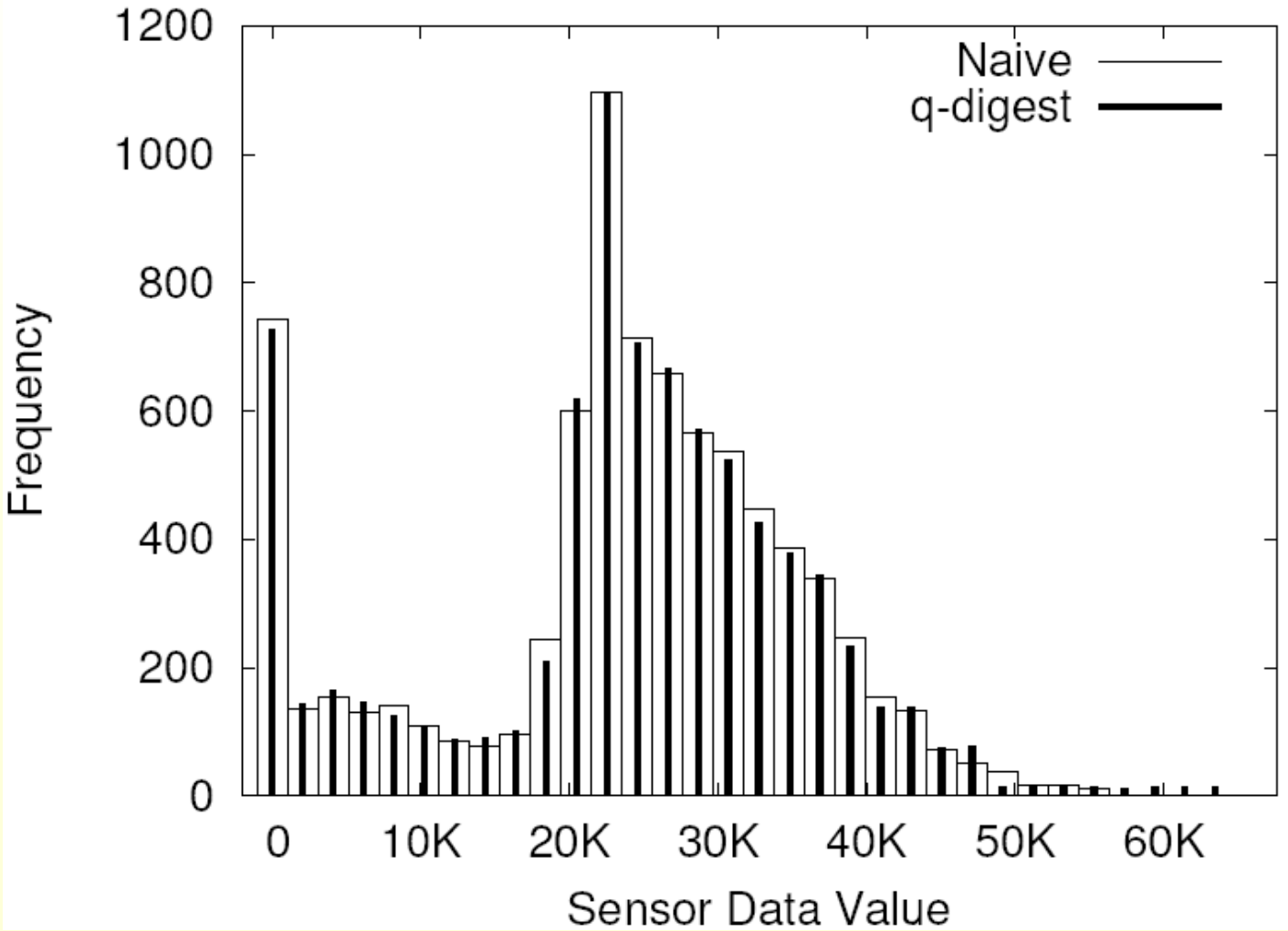
Results: Error in MEDIAN

Network size: n=8000



Results: Histogram

For Correlated data; m=400 bytes; n=8000;



Conclusion & Future Directions

- Efficient aggregation scheme
- Approximate answers to variety of queries
- Nice theoretical bounds
- Works well in practice

- Future Directions
  - Updates: re-transmit only q-digests that changed significantly
  - Extend to multi-dimensional data

The End