

---

# Data-Centric Query in Sensor Networks

Jie Gao

Computer Science Department  
Stony Brook University



# Papers

---

- Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin, [Directed diffusion: A scalable and robust communication paradigm for sensor networks](#), In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00), August 2000, Boston, Massachusetts.
- Sylvia Ratnasamy, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, Brad Karp, Scott Shenker, [GHT: A Geographic Hash Table for Data-Centric Storage](#), In First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2002.
- Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger and Robert Morris, [A scalable location service for geographic ad hoc routing](#), MobiCom'00.

# Scenario I: tourists and animals

---

- A sensor network in a zoo.
- A tourist asks: where is the elephant (or giraffe, or zebra)?
- So which sensor has the data about the elephant (or giraffe, or zebra)?



## Scenario II: location service

---

- A missing part of routing with geographical or virtual coordinates: how does the source know the location (or virtual coordinates) of the destination?
- Location service: a brokerage service that answers queries such as: where is the node with ID 23?
- Geographical routing:
  - The **source asks for the location of destination**;
  - The source routes by using geographical routing.
- Notice: chicken and egg problem.

# Data-centric

---

- Traditional networks: routing is based on network ID (e.g., IP addresses).
- Communication abstractions are based on **data** rather than node network addresses.
- Data-centric routing
  - Route to the node with the data the user wants.
- Data-centric storage
  - Store all the data with the general name (elephant) at the same node.

# Abstraction of data-centric routing

---

- Information producer/consumer game.
- Information producer.
  - Can be anywhere in the network.
  - Dynamic, mobile.
  - Multiple producers generating data about the same data type.
- Users = information consumer.
  - Can be anywhere in the network.
  - Concurrent multiple consumers.

# Challenges

---

- Information producers/consumers have no idea about each other.
- Yet we want them to find each other quickly.
- Main approaches:
  - **Push-based**: producers do most of the work.
  - **Pull-based**: consumers actively search.
  - **Push-pull**: both producers/consumers search to find each other.

# This class

---

- Directed diffusion
  - Push-based
- Geographical hash table
  - Push-pull
  - In-network storage
- Location service (hierarchical hashing)
  - Structured hashing for naming services

# Directed diffusion


---

- Data is named by **attribute-value pairs**.

```
type = four-legged animal // type of animal seen
instance = elephant        // instance of this type
location = [125, 220]     // node location
intensity = 0.6           // signal amplitude measure
confidence = 0.85         // confidence in the match
timestamp = 01:20:40      // event generation time
```

- Query is represented by **interest**.

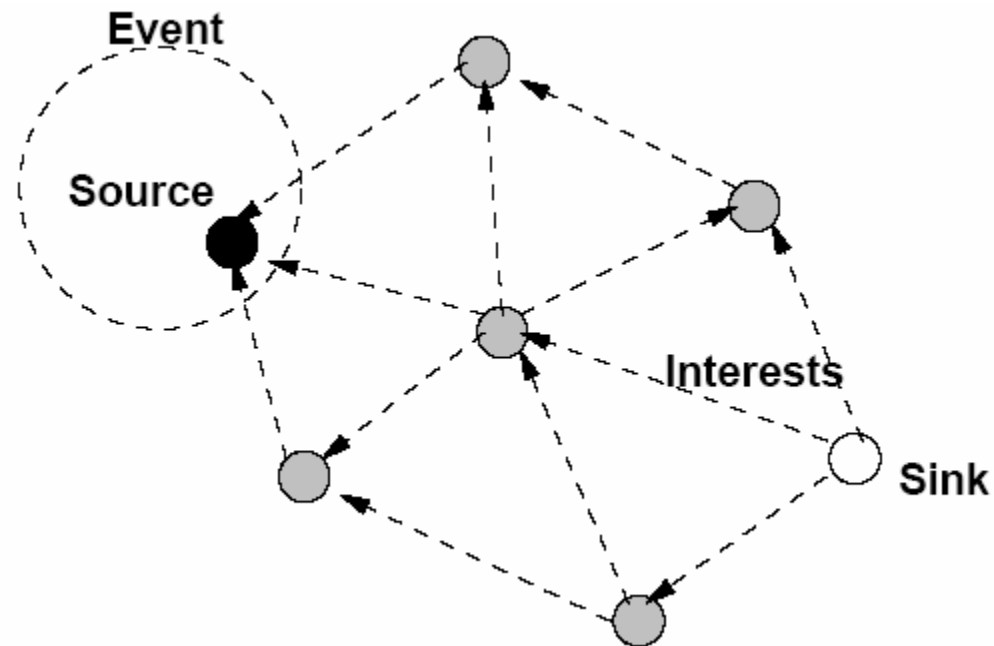
```
type = four-legged animal // detect animal location
interval = 20 ms          // send back events every 20 ms
duration = 10 seconds     // .. for the next 10 seconds
rect = [-100, 100, 200, 400] // from sensors within rectangle
```



# Interest dissemination

---

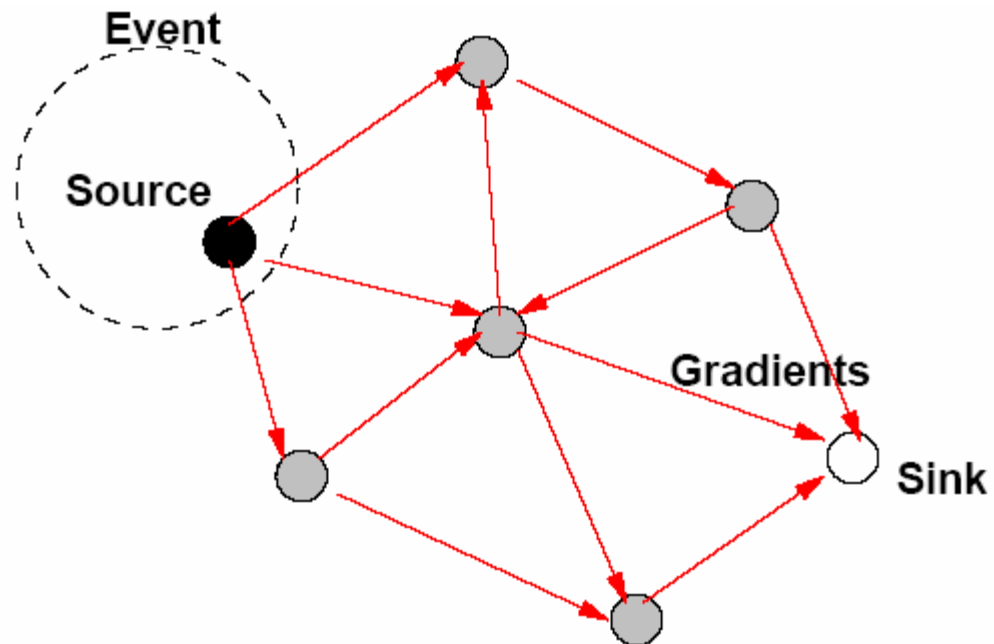
- A sensing task is disseminated in the network as an interest for named data.
- Interest is refreshed for robustness.



# Gradient establishment

---

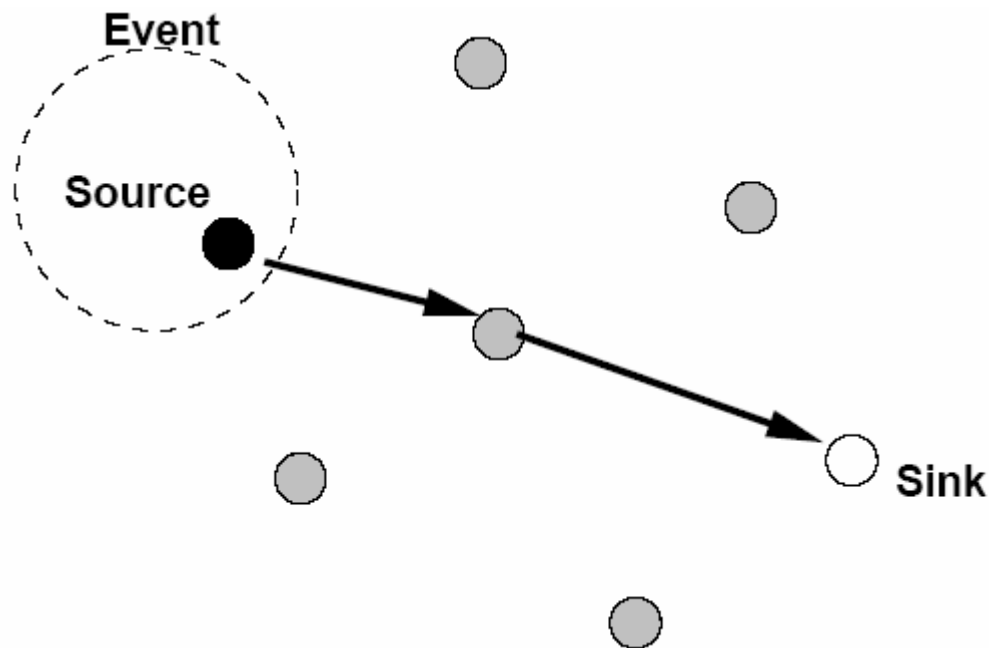
- Each node caches a **gradient** for interest: which specifies the data rate and duration.



# Data transmission

---

- Data is transmitted back to sink.
- Multi-path can be adopted.
- Good paths (low delay, more reliable ones) are reinforced.



## Pros and Cons

---

- The earliest proposal for data-centric routing.
- Pull-based approach.
- Similar to TinyDB.
- Ok for streaming data type.
- Flooding is expensive for infrequent queries, or queries that only involve a small set of nodes.

# This class

---

- Directed diffusion
  - Push-based
- Geographical hash table
  - Push-pull
  - In-network storage
- Location service (hierarchical hashing)
  - Structured hashing for naming services

# Distributed hash table (DHT)

---

- For Bob and Alice to find each other.
- “Lost and found”.
- Basic idea: data-dependent rendezvous.
- Use a content-based hash function  
 $h(\text{elephant}) = \text{sensor \#10}$ .
- All the sensors with elephants info send to #10.
- All the tourists interested in elephants go to #10 to fetch the information.

# Distributed hash table (DHT)

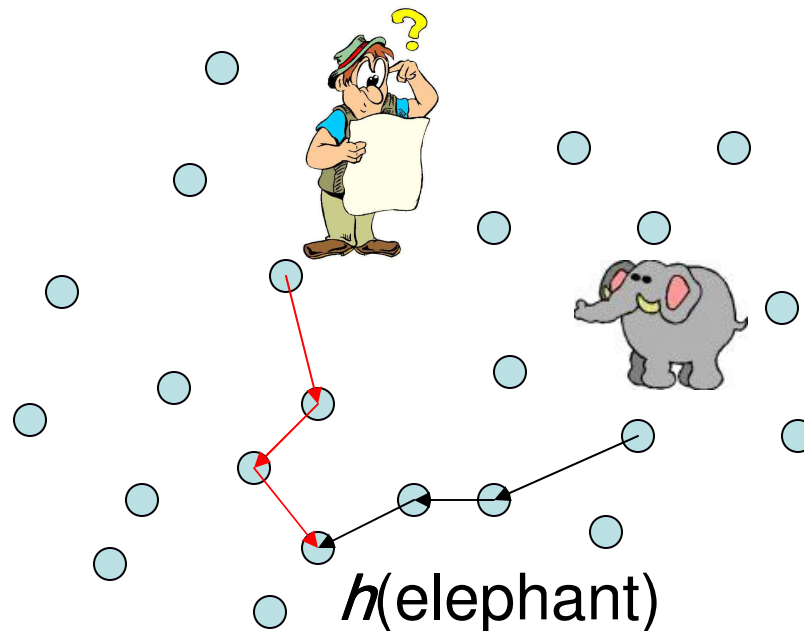
---

- Originally proposed for Peer-to-Peer routing on the Internet.
  - E.g, Chord, Pastry, Tapastry, etc.
- A data object is given a key.
- Each node saves a set of keys.
- A routing algorithm allows any node to locate the one with an arbitrary key.

# Geographical hash table (GHT)

---

- Assume nodes know their locations and do geo-routing.
- The content-based hash function outputs a **geographical location**:  $h(\text{elephant}) = (14, 22)$ .
- Use GPSR for information producers/consumers to route to the rendezvous.



# Geographical hash table (GHT)

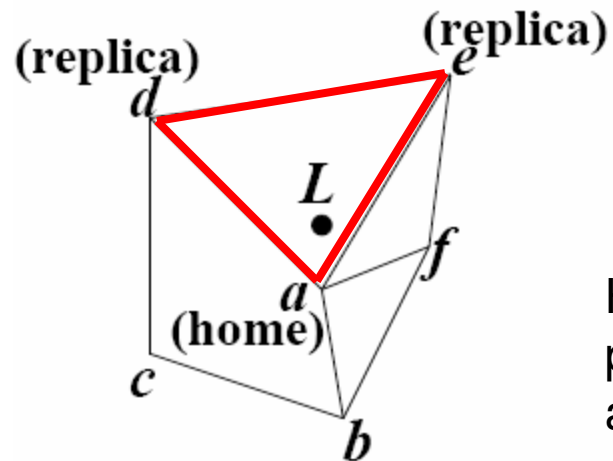
---

- The content-based hash function  
 $h(\text{elephant}) =$  a geographical location (14, 22).
- Use geographical routing for information producers/consumers to route to the reservoir.
- Two questions:
  - What if there is no sensor at location (14, 22)?
  - What if geographical routing gets stuck?

# Geographical hash table (GHT)

---

- We route to location  $L=(14, 22)$  and GPSR finds out there is no way to  $(14, 22)$  by touring along a perimeter of a face and get back to where it started.



Home node: the one that is geographically closest to  $L$ .

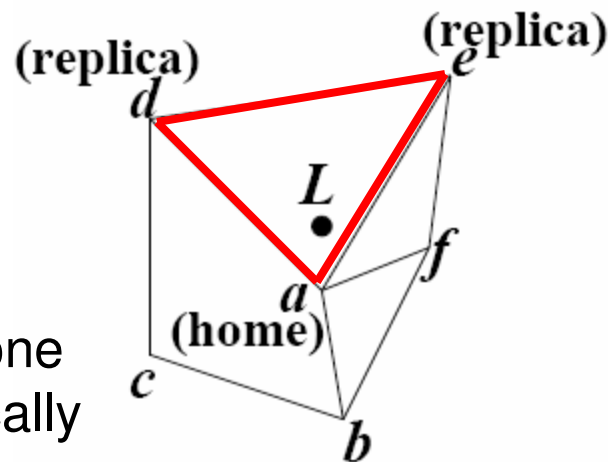
Home perimeter: the perimeter that GPSR tours around.

# Geographical hash table (GHT)

---

- We replicate elephant information on all the nodes on the perimeter.
- The query follows the same home perimeter and retrieve the message.

Home node: the one that is geographically closest to  $L$ .

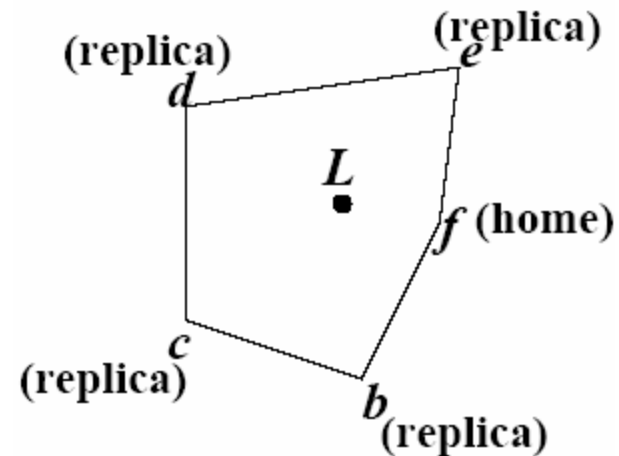
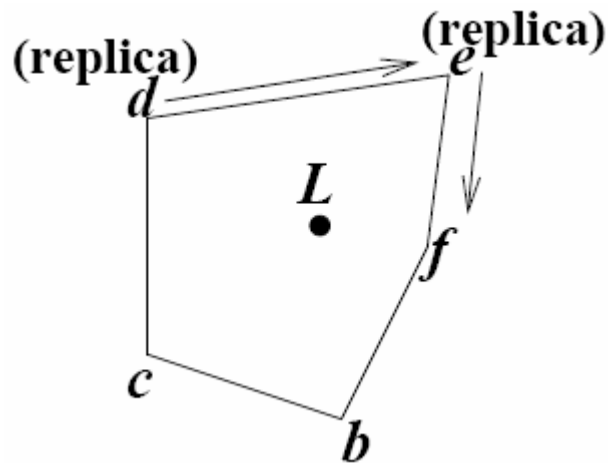


Home perimeter: the perimeter that GPSR tours around.

# GHT: maintenance

---

- Home node periodically refresh replication by sending a packet to the hashed location  $L$ .
- If the timer of the replica times out, then a replica node initiates a refresh.



# Geographical hash table (GHT)

---

- Advantages:
  - simple.
  - load balancing in storage.
- Disadvantages:
  - Not locality-sensitive. Consumer may travel far to fetch data even if the producer is close.
  - Fault tolerance?
  - Overload nodes on the boundary.
  - Nodes with popular data become bottleneck.

# This class

---

- Directed diffusion
  - Push-based
- Geographical hash table
  - Push-pull
  - In-network storage
- Location service (hierarchical hashing)
  - Structured hashing for naming services

# Location service

---

- Geographical routing requires obtaining the location of the destination.
- What if the sensors move? How to update the location information?
- Internet: domain name server (DNS) translates user-friendly domain name ([www.cnn.com](http://www.cnn.com)) to machine-friendly IP address.

# Centralized v.s. distributed location service

---

- Location server stores the mapping between locations and node IDs.
  - Centralized approach, single point of failure.
  - Communication bottleneck.
  - Location server might be far away.
- Distributed location servers: every node participates and acts as location servers for others.

# Challenges

---

- Problem 1: each node need to know the location server of any node.
  - To update its own location info upon movement.
  - Query for the location of any other node.
- Problem 2: how to get to the location server?
  - We need a routing algorithm, say geographical routing.
- Problem 3: geographical routing requires the knowledge of destinations.
  - How to get the location of the location server?
  - Every node can be moving.
- Chicken and egg problem?

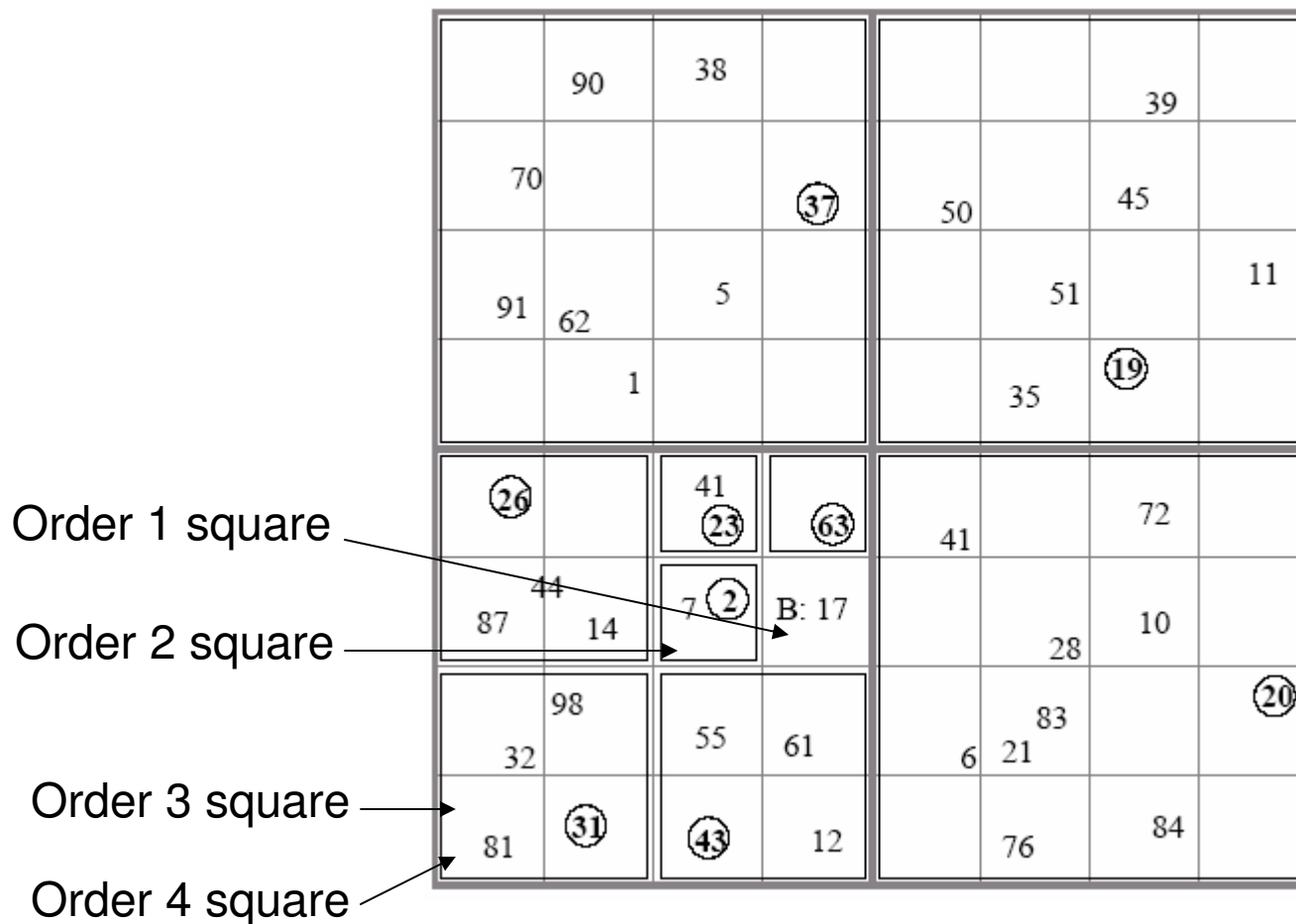
# Grid location service

---

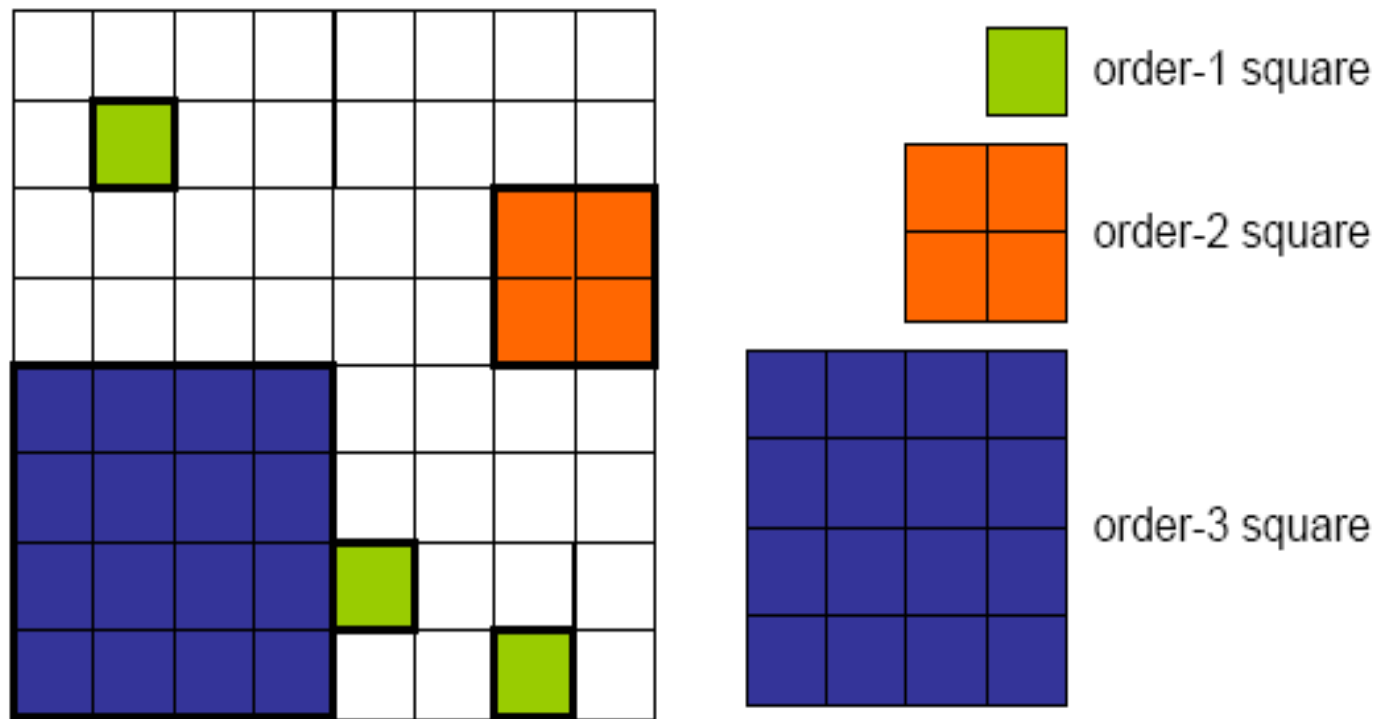
- Each node is assigned a random ID: computed by a strong hash function on physical name, e.g., MAC address.
- Each node stores/updates its location information at a set of location servers, more at nearby regions, fewer at far away regions.
- Location query uses **nothing beyond the ID**.

# Recursive partitioning

- Quad-tree partition: each node is inside a unique square on each level.



# Partitioning the world



Invariant: a node is located in exactly one square of each size (no overlapping)  
An order-x square contains always 4 order-(x-1) squares

# Location servers

- Node B's location servers: Inside each sibling square on each level, choose B's closest node.
- **Def.:** Node **closest** to B in ID space: node with **least ID greater** than B
- Circular ID space: 2 is closer to 17 than 7 is.

	90	38					
70			37			39	
91	62	5		50	45		11
	1				51		
					35	19	
26		41	23	63			
87	44	14	7	2	B: 17	41	72
	98					28	10
32		55	61				20
						6	83
81	31	43	12			21	
						76	84

# Location queries

- A queries 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.
- B even doesn't know its location servers.
- How to implement location query?

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82	
	A: 90	38				39	
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43	
70			37	50		45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50		19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51		11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98	
	1				35	19	
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70	
26		23	63	41		72	
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84	
87	14	2	B: 17		28	10	
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,76	6,10,12,14 16,17,19,84	
32	98	55	61		6	21	20
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55	2,5,6,10,43 55,61,63,81 87,98		6,10,20,41 72	20,21,28,41 72,76,81,82	
81	31	61	43	12	A: 76	84	

# Location queries

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

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82
	A: 90	38				39
1,5,16,37,62 63,90,91			16 17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43
70			37	50		45
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81 11
91	62	5			51	
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98 19
	1				35	
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70 72
26		23	63	41		
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,21 23,26,41,72 76,84	6,10,12,14 16,17,19,84
32	98	55	61		6	20
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55 61 43	2,5,6,10,43 55,61,63,81 87,98		6	21
81	31	61	12		A: 76	84

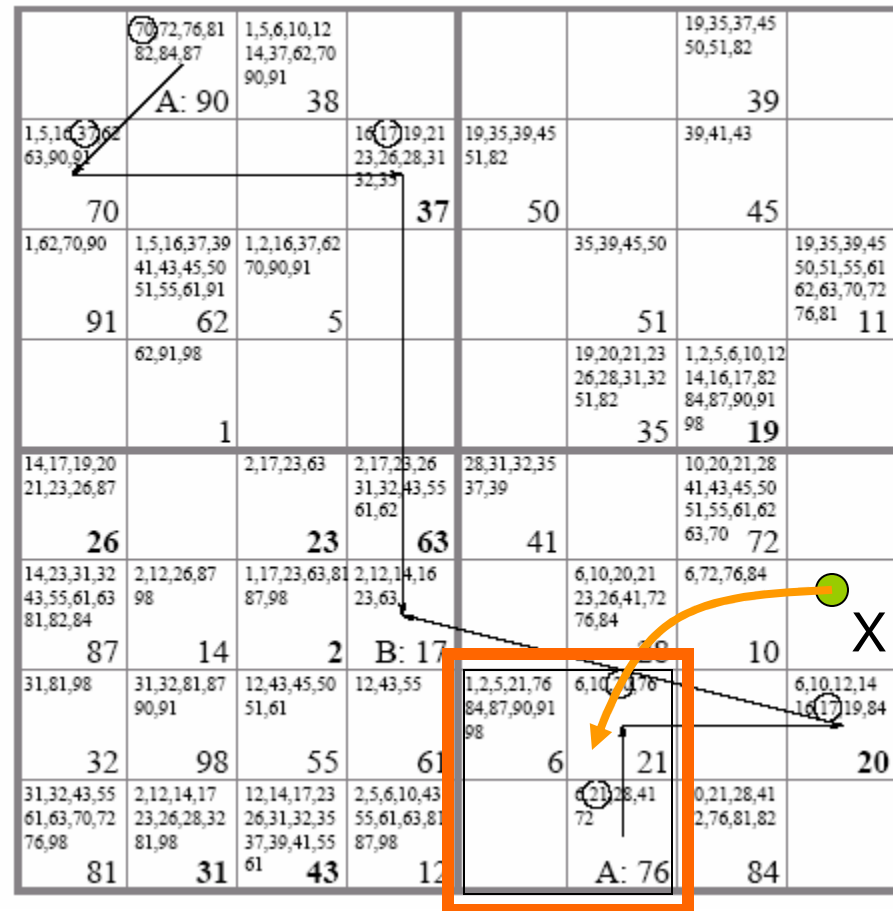
# Location queries

- 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.

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82
	A: 90	38				39
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43
70			37	50		45
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81 11
91	62	5			51	
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98 19
	1				35	
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70 72
26		23	63	41		
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,21,76	6,10,12,14 16,17,19,84
32	98	55	61		6	
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55 61 43	2,5,6,10,43 55,61,63,81 87,98		21	20
81	31		12		A: 76	84

# 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 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 identify.
- A routes to a square with geographical routing.
- The first node in the square S performs a location query of 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!

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91			19,35,37,45 50,51,82	
	A: 90	38			39	
1,5,16,37,62 63,90,91			16,17,19,21 23,26,28,31	19,35,39,45 51,82	39,41,43	
70			37	50	45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51	11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98
	1				35	19
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39	10,20,21,28 41,43,45,50 51,55,61,62 63,70	72
26		23	63	41		
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,76	6,10,12,14 16,17,19,84
32	98	55	61		6	21
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55	2,5,6,10,43 55,61,63,81 87,98		6,10,20,41 72	20,21,28,41 72,76,81,82
81	31	61	43	12	A: 76	84

# The bootstrapping

- When the entire system is turned on, order-1 squares exchange their information with 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.

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82	
	<b>A: 90</b>	38				39	
1,5,16,37,62 63,90,91			16 17,19,21 23,26,28,31	19,35,39,45 51,82		39,41,43	
70			32,33	37	50	45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50		19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51		11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98	19
	1				35		
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70	72
<b>26</b>		<b>23</b>	<b>63</b>	41			
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84	
87	14	2	<b>B: 17</b>		28	10	
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,76		6,10,12,14 16,17,19,84
32	98	55	61		6	21	<b>20</b>
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55	2,5,6,10,43 55,61,63,81 87,98		6,21,28,41 72	20,21,28,41 72,76,81,82	
81	31	61	<b>43</b>	12	<b>A: 76</b>	84	

## Take a rest and enjoy the beauty of this algorithm

---

- It solves location service problem by using geographical 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?