Distributed Quad-Tree for Spatial Querying in Wireless Sensor Networks

Murat Demirbas and Xuming Lu

ICC 2007 proceedings

Contents

- Introduction
- Model
- DQT Structure and Construction
- Querying in DQT
- Fault Tolerance
- Simulation
- Conclusions

In-network querying in WSNs

In contrast to traditional WSN applications that perform only data collection, new generation of WSN applications require in-network information querying.

- Battlefield applications
- Tracking

Where is the nearest enemy tank?

A soldier queries the WSN via a palm device. Energy efficiency and latency suffers drastically if queries are always routed to a centralized Base-station over many hops.

In-network querying in WSNs

In-network querying should satisfy

- **Distance sensitivity**
  Cost of answering a query for an event is at most a constant factor "s" of the distance "d" to the nearest event in the network.

- **Efficient information storage**
  Cost of advertising an event information is at most a constant factor of the diameter "D" of the network.
Contributions of this paper

This paper presented an in-network querying infrastructure that satisfies all these requirements for event querying:

- Distance sensitivity: the cost is at most $\sqrt{F}$ (stretch factor) times the distance $d$ to the nearest event
- Efficient information storage
- Low-cost maintenance: bottom-up construction is avoided
- Graceful resilience: single mote failures are masked and performance degrades proportionately with the severity of holes (failures of motes in a region)

Contents

- Introduction
- Model
- DQT Structure and Construction
- Querying in DQT
- Fault Tolerance
- Simulation
- Conclusions

Model

Assumptions:

- WSN motes (physical WSN nodes) sit on a 2D plane and their coordinates are made available to themselves.
- Assume a connected network and availability of geographic routing (GPSR or CLDP)
- The network is divided into grid cells (DQT nodes) and all motes in each node are within one hop distance.
- Cost of querying an event is measured as the number of hops traveled from the querying mote to a mote that holds an advertisement about the event.

Contents

- Introduction
- Model
- DQT Structure and Construction
- Querying in DQT
- Fault Tolerance
- Simulation
- Conclusions

DQT Structure and Construction

DQT Structure and Construction
DQT Structure and Construction

- To achieve low-cost construction, we exploit location info at the nodes:
  - Nodes know the boundary coordinates of deployment, and calculate which DQT node id they fall into using their coordinates.
  - We choose the cluster-heads closer to the base station to avoid backward links during querying and data collection.

- Mapping from localization to DQT addressing:
  - \((x_s, y_s)\) at NW and \((x_e, y_e)\) at SE be the two endpoints of the area. Assume DQT have \(i\) levels. The area of each level 1 box of partition is \(w^i\), where:
    - width : \(w = (y_e - y_s)^{1/2}\)
    - length : \(l = (x_e - x_s)^{1/2}\)
  - DQT address of a node \((x, y)\) can be calculated as:
    \[
    \text{DQT addr} = \left\lfloor \frac{x - x_s}{w} \right\rfloor \cdot 2^i + \left\lfloor \frac{y - y_s}{l} \right\rfloor \cdot 2^i
    \]

DQT Structure and Construction

- DQT Local Construction:
  - Find the cluster-head at level \(i\):
    
    **Procedure Cluster-head Validate (node \(p\), level \(i\))**
    
    ```
    switch (p.address(h)) {
    case 3: //p in SE region
      if p.address(i*) = 0 then return true
      else return false
    case 2: //p in SW region
      if p.address(i*) = 1 then return true
      else return false
    case 1: //p in NE region
      if p.address(i*) = 2 then return true
      else return false
    case 0: // p in NW region
      if p.address(i*) = 3 then return true
      else return false
    }
    ```

DQT Structure and Construction

- DQT Local Construction:
  - Find the neighbors at level \(i\)
    - Using \(p\)’s location information and increase its coordinates \(x, y\) value by a level \(i\) box lateral length to find a neighbor node in each direction.
    - For each of these nodes, we find their level \(i\) cluster-heads.
    - These cluster-heads are node \(p\)’s level \(i\) neighbors.

DQT Structure and Construction

- Querying in DQT

  **Indexing of event information**
  - A query from node 011 to node 100 may route to higher level cluster-heads such as node 013 and node 033.
  - The solution is to use sibling links to nearby intermediate nodes.
  - A sibling link is the link between a node and its neighbors in each direction so each node at most have 8 sibling neighbors.
  - The sibling links only exist between nodes on the same level in the structure.
Querying in DQT

Indexing of event information

- A node at level i maintains the event information of its cluster, as well as the event information of its neighbors.
- When an event is detected at a level 1 node p, p contacts its immediate parent node at level 1. The parent node updates its record for that child. Node p also contacts its sibling nodes to update their records accordingly. Recursively, the update operation is executed till the top level.

Lemma 1. A DQT node at level i stores O(i) information.

Proof: A node at level i is cluster-head from level 1 to i along the path. The number of neighbor nodes at each level is less than or equal to eight. Therefore the node needs $9^i$ (including one record for its sub-tree) space and stores O(i) information.

Theorem 1. The total space needed for the construction of distributed quad-tree is less than 12*b, where b is the total number of level 1 nodes.

Proof: According to Lemma 1, level 1 nodes use up $9^b$ space. Similarly, all level 2 nodes total to a $9^b/4$ space usage. Thus, the total space needed for constructing the distributed quad-tree is:

$$9 \left( b + \frac{b}{4} + \frac{b}{4^2} + \ldots + \frac{b}{4^i} \right) = 12 \cdot b \left( 1 - \frac{1}{4^i} \right) < 12b$$

Lemma 2. The distance between a level i node and its neighbors is at most $2^i \sqrt{2}$ hops.

Proof: According to the partition rule of quad-tree, a level i node is the cluster-head of a $2^i \times 2^i$ area. The distance between a level i node and its neighbors is either $2^i$ (for N, S, E, W neighbors) or $2^i \sqrt{2}$ (for NE, NW, SE, SW neighbors) depending on the direction. Since the cluster-head is one of its neighbors at level i, so the distance between a level i node and its cluster-head is also less than $2^i \sqrt{2}$ hops, which is the diagonal distance of a level i partition.

Theorem 2. The distance stretch factor s for spatial query in our structure is $2\sqrt{2}$ in worst case. In another words, an event d hops away can be achieved by the querying node within $d \cdot 2\sqrt{2}$ hops.

In right figure, d is the distance from querying node Q to highest level of node M that the query is propagated; d is the distance from Q to P, where P is the destination node that node Q is querying. Distance stretch factor s is defined as $s = d/d_1$.

Distance stretch factor: $d_1/d$
Contents
- Introduction
- Model
- DQT Structure and Construction
- Querying in DQT
- Fault Tolerance
- Simulation
- Conclusions

Fault Tolerance
- DQT achieves resilience via:
  - its stateless nature, and
  - using GPSR for routing
- More specifically:
  - Mote failures do not often lead to failure of level 1 node
  - GPSR routes around coverage holes, & delivers the message to a boundary node if there is a hole in the path to destination node.
  - Since DQT is stateless, any node can act as a proxy node for another

Simulation
- Focus on two aspects:
  - Stretch factor
  - Fault tolerance
- Simulation setting:
  - 256 nodes.
  - 3200m*3200m
  - Distance between each node: 200m
  - Transmission range: 250m
  - DQT_height = 4

Simulation

![Graph 1](image1.png)

![Graph 2](image2.png)
Conclusions

- Distributed quad-tree (DQT) structure is suitable for use in real world WSN deployments.
- DQT satisfies distance-sensitive querying as well as efficient information storage in network.
- DQT construction is local and does not require any communication.
- Due to its minimalist infrastructure and stateless nature, DQT shows graceful resilience to the face of node failures.

Contents

- Introduction
- Model
- DQT Structure and Construction
- Querying in DQT
- Fault Tolerance
- Simulation
- Conclusions