Programming Abstractions for
Sensor Networks

Jun Yang
CPS 296.1, Spring 2007
Sensor Data Processing
With contents from M. Welsh and D. Chu

Announcements (Apr. 12)
♦ Project milestone 2
  ▪ Need to know your progress by April 20!
  ▪ Please schedule a group meeting with me by then
♦ Project presentation/report
  ▪ Monday April 30, 2-5pm (final exam slot)
♦ Reading for next Tuesday
  ▪ PRESTO: sensornet with in-network storage

Programming abstractions
♦ Matt Welsh and Geoff Mainland. "Programming
  Sensor Networks Using Abstract Regions."
  Symposium on Networked Systems Design and
  Implementation, 2004
♦ Also, briefly:
  ▪ David Chu, Kaishen Lin, Alexandre Linares, Ky Giang
    Nguyen, and Joseph M. Hellerstein. "sdlib: A Sensor
    Network Data and Communications Library for Rapid
    and Robust Application Development." International
    Conference on Information Processing in Sensor Networks, 2006

Challenges in programming sensors
♦ Development is notoriously difficult
  ▪ Bandwidth and energy limitations force in-network
    processing
  ▪ Requires complex, distributed algorithms
♦ Observation: applications often involve coordination
  within local regions of the network, e.g.:
  ▪ Coordinated detection of local phenomena
  ▪ In-network aggregation for bandwidth reduction

Example applications
♦ Moving vehicle tracking and pursuit
  ▪ Sensors take magnetometer readings
  ▪ and locate centroid of readings
♦ Habitat monitoring on GDI
  ▪ Collect readings from petrel nests;
    determine occupancy of nests to
    understand breeding/migration
    behavior
♦ Tracking frontier of phenomenon of interest
  ▪ Sensors communicate locally to detect contour

Accuracy/cost tradeoff
♦ Extremely resource-constrained devices
  ▪ Mica2 motes running TinyOS: 7.3MHz CPU, 4KB
    RAM, 128KB ROM, 38.4 Kbps radio, 2 AA batteries
♦ Inherent tradeoff between resource consumption
  and accuracy
  ▪ More messages → increased energy and bandwidth
    consumption → greater precision
♦ Nodes have limited energy ⇒ apps must deal with
  lossy communication, imperfect results
  ▪ How do apps explore this tradeoff?
Macro-programming

- An aggregation programming model for sensor networks
  - Current programming models are node-centric and low-level
  - Scientists don’t want to think about radios, timers, etc.
- Provide powerful primitives to reduce programming efforts
  - Focus on spatial computation within local neighborhoods
  - Abstract low-level details of local coordination
  - Neighborhood maintenance, routing, collective communication
- Allow applications to control accuracy/cost tradeoff
  - Expose parameter settings in lower layers to apps
  - Provide feedbacks to apps, e.g., timeouts on communication, accuracy and completeness of collective operations
  - Feedback used to adapt to changing network conditions

Abstract regions

- Group of nodes with some geographic or topological relationship, e.g.:
  - All nodes within distance \( d \) from node \( k \)
  - Neighbors of \( k \) in a planar mesh (based on radio connectivity)
  - Spanning tree rooted at \( k \)
- Regions capture common idioms in sensor network programming
  - Flexible addressing of “local” nodes
  - Sharing state across nodes in a group
  - Efficient data aggregation within a region

Region operations

- Neighbor discovery identifies nodes in a region
  - Continuous background process; nodes notified of changes in region membership (e.g., nodes moving, joining, or leaving)
- Shared variables support inter-node coordination
  - \( \text{get}(\text{var}, n) \) retrieves value of \( \text{var} \) from node \( n \)
  - \( \text{put}(\text{var}, \text{val}) \) stores value \( \text{val} \) in variable \( \text{var} \) at this node
  - Implementation may be broadcast, pull-requested data, or gossip
- Reductions support in-network aggregation
  - \( \text{reduce}(\text{op}, \text{var}, d) \) aggregates values of \( \text{var} \) within the region using operator \( \text{op} \) (e.g., max, average, sum, etc.) and stores the result in \( d \)

Radio and geographic neighborhoods

- Nodes within \( n \) hops, \( n \) nearest neighbors, etc.
- Node discovery implementation
  - Nodes periodically advertise id and location, and filter received ads to determine neighbors
- Shared variable implementation
  - \( \text{put} \) stores \( (\text{var}, \text{val}) \) pair in local hash table, and \( \text{get} \) fetches from remote node
  - Alternatively, \( \text{put} \) broadcasts, and \( \text{get} \) is local
- Reduction implementation
  - Broadcast request for \( \text{var} \), collect replies, and aggregate

Approx. planar mesh neighborhood

- Divide space into non-overlapping cells
  - Recall face routing?
  - Different planarizations possible:
    - True planarity is difficult in practice due to incomplete/inaccurate location/distance info
    - Strive for approximate planarity (allow some crossing edges)
    - Number of crossing edges measures the inaccuracy in planarization

Adaptive spanning tree

- Useful for aggregating data to a single sink
  - Nodes continually evaluate link quality to neighbors and select ideal parents
  - Topology responds rapidly to changes in network conditions
  - Implemented on top of radio neighborhood (layering!)
- Shared variable and reduction semantics
  - \( \text{put} \) request issued at root floods data to all nodes
  - \( \text{get} \) request issued at root fetches from specific descendent
  - \( \text{reduce} \) always stores result at root
Quality feedback and tuning

- Region operations are inherently unreliable
  - Collective operations report yield: fraction of nodes responded to a request
  - Each operation also provides a timeout
- Programmer can tune many parameters
  - Max. number of and delay between retransmissions
  - Max. number of neighbors to consider in region formation
  - Frequency and number of ads
  - Threshold for removing neighbor from region
  - Timeout for various region operations
- Supports adaptivity

App: object tracking using regions

Nodes near the vehicle detect high magnetometer readings

Store local sensor reading as a shared variable (each node forms k-nearest neighbor region)

Node with the highest reading calculates centroid of neighbors’ readings (reduce)

Code: object tracking using regions

```c
location = get_location();
region = k_nearest_region(centroid);
while (true) {
  reading = get_sensor_reading();
  // Store local data as shared variables
  region.put(reading_key, reading);
  region.put(reading + location.x);
  // Process region readings
  if (reading > threshold) {
    // If I am the leader node ...
    if (leader_id == my_id) {
      region_reduce(centroids, reading);
      centroid_x += region_reduce(centroids, reading); // Process region readings
      centroid_y += region_reduce(centroids, reading);
      centroid = centroid_x / num_nodes;
      centroid = centroid_y / num_nodes;
    }
  } // Sleep(periodic_delay);
}
```

Evaluation environment

- TOSSIM with realistic radio model
  - 10 x 10 grid over 20ft x 20ft area
  - Radio model derived from trace of Mica motes outdoors

Object tracking accuracy and cost

- Object moves in circles
- Tuning knob: size of neighborhood
  - Increases both accuracy and message overhead

App: contour finding

- Construct approximate planar mesh of nodes
- Nodes above threshold compare readings with neighbors
- Contour = midpoints of edges crossing threshold
Contour detection accuracy and cost

- Tuning knob: number of broadcasts (ads) used in approximate planar mesh construction
  - Increases quality of planar mesh (fewer crossed edges) and reduces error in contour points

Other apps

- Recall GPSR
  - Default: greedy forwarding to the neighbor closest to destination
  - When stuck, switch to face routing on planar graph
  - Easy to implement with radio and planar mesh regions

- Recall directed diffusion
  - Sink floods interests to nodes
  - Nodes with matching data send results back to sink
  - Easy to implement on top of spanning tree regions
  - 188 LOC for directed diffusion (on top of 917 LOC for spanning tree)

Discussion

- Program the “network” rather than the “individuals”
  - Requires appropriate programming models and communication primitives
- Abstract regions provide simple data sharing and aggregation while hiding low-level communication details and complexity of creating and maintaining regions
- Expose the accuracy/cost tradeoff to apps
  - Claim: lots of knobs are better than no knobs (true?)
  - Do knobs allow you control accuracy/cost directly?
  - Do we even know what the accuracy is (let alone how to reach the desired level)?

sdlib: sensor data library

- Motivation
  - Many apps are written from scratch
  - Need to share processing of semantically similar services
  - Can we find a standard abstraction to encourage code reuse and shared processing?
- Observation
  - Sensor networks are about data
  - Producers, consumers, and data flows are fundamental to sensor networks
  - Library and runtime support for constructing data flows in sensor networks

Example data flows

- Raw data collection: Video → Collection
- Processed data collection: Video → Feature Detector → General Store → Collection
- … with archival:
  - General Store

sdlib supports a simple syntax for specifying and changing these data flows on the fly

Evaluation

- Four mature apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Implementation</th>
<th>sdlit Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td>164</td>
<td>158</td>
</tr>
<tr>
<td>GGB</td>
<td>342</td>
<td>158</td>
</tr>
<tr>
<td>Mate</td>
<td>669</td>
<td>73</td>
</tr>
<tr>
<td>Deluge</td>
<td>410</td>
<td>60</td>
</tr>
</tbody>
</table>

- Reasonable resource usage and performance
Discussion: abstraction ideas

- TinyDB
  - Low-level per-node code → high-level queries over the entire network
- Directed diffusion
  - Node-centric addressing → data-centric addressing
- Abstract regions
  - Node-centric programming → neighborhood-centric programming
- sdb
  - Function calls → data flows