Towards a Mobile Agent Framework for Sensor Networks

Leo Szumel
Jason LeBrun
John D. Owens
Overview

• Sensor networks will continue to be constrained by energy and bandwidth
  – Moore's Law gives us a “budget” for improvements
    • We can invest in increasing the capability of each node
    • OR we can invest in lowering cost and power of each node

• What makes a sensor network useful?
  – Scale, Longevity
    • We should invest in decreasing cost and increasing longevity!

• We want to decrease cost and increase longevity by:
  – Keeping node capabilities near current status quo
  – Enabling multiple users
  – Reducing cost of dynamically tasking the network
Outline

- Overview
- Our Design
- Test Applications & Results
- Discussion
Agents in Sensor Networks?
“long-lived, semi-autonomous, proactive, and adaptive” --MIT Media Lab

- Autonomous, proactive, reactive, (mobile)
- Communicate with other agents: dynamic behavior
- What are agents good for?
  - Autonomous operation
  - Fast reaction to stimuli
  - Tailored to a specific purpose (efficient use of resources)
- Our agents are tiny, they come in a range of flavors, and we love them.
  - Think of them as Skittles® for sensor networks
# Re-Tasking Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Flexibility</th>
<th>Cost</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Parameters</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Scripts/ ByteCode</td>
<td>Med</td>
<td>Med</td>
<td>Med</td>
</tr>
<tr>
<td><strong>New Binary</strong></td>
<td><strong>Very High</strong></td>
<td><strong>Very High</strong></td>
<td><strong>Very Low</strong></td>
</tr>
</tbody>
</table>

[Deluge: Reliable Network Programming (Jonathan Hui, UCB)]
# A New Re-Tasking Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Flexibility</th>
<th>Cost</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Parameters</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Scripts/ ByteCode</td>
<td>Med</td>
<td>Med</td>
<td>Med</td>
</tr>
<tr>
<td>New Binary</td>
<td>Very High</td>
<td>Very High</td>
<td>Very Low</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agent</th>
<th>Med-High</th>
<th>Lowest+</th>
<th>Med</th>
</tr>
</thead>
</table>

**Frequency**

- Very Low
- Very High
- High
- Med
A New Re-Tasking Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Flexibility</th>
<th>Cost</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Parameters</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Scripts/ ByteCode</td>
<td>Med</td>
<td>Med</td>
<td>Med</td>
</tr>
<tr>
<td><strong>New Binary</strong></td>
<td><strong>Very High</strong></td>
<td><strong>Very High</strong></td>
<td><strong>Very Low</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agent</th>
<th>Med-High</th>
<th>Lowest+</th>
<th>Med</th>
</tr>
</thead>
</table>

**Cost depends on agent propagation**
High Level Concept

**Style**  **Base (Bytes)**  **Incremental (Bytes)**

Monolithic  17082  1878

```
Fraction of Nodes to be Re-Tasked

0  20  40  60  80  100
Retask Count Over Network Lifetime
```

- Monolithic
## High Level Concept

<table>
<thead>
<tr>
<th>Style</th>
<th>Base (Bytes)</th>
<th>Incremental (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>17082</td>
<td>1878</td>
</tr>
<tr>
<td>Maté</td>
<td>40054</td>
<td>128</td>
</tr>
</tbody>
</table>

![Graph showing the fraction of nodes to be re-tasked over retask count over network lifetime]

- **X**: Monolithic
- **O**: Global VM
High Level Concept

<table>
<thead>
<tr>
<th>Style</th>
<th>Base (Bytes)</th>
<th>Incremental (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>17082</td>
<td>1878</td>
</tr>
<tr>
<td>Maté</td>
<td>40054</td>
<td>128</td>
</tr>
<tr>
<td>Agent</td>
<td>42824</td>
<td>156</td>
</tr>
</tbody>
</table>

Fraction of Nodes to be Re-Tasked

Retask Count Over Network Lifetime

- × Monolithic
- ○ Global VM
- * Agent
### High Level Concept

<table>
<thead>
<tr>
<th>Style</th>
<th>Base (Bytes)</th>
<th>Incremental (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>17082</td>
<td>1878</td>
</tr>
<tr>
<td>Maté</td>
<td>40054</td>
<td>128</td>
</tr>
<tr>
<td>Agent</td>
<td>42824</td>
<td>156</td>
</tr>
</tbody>
</table>

Agents can have lower overhead when re-tasking network subregions.
Design Overview

- Protected execution environment
- Compact, efficient code
- “Forward” ability
  - Similar to UNIX “fork”
  - PTP or broadcast
  - Lossy and ARQ modes
- System state
  - “Bread crumbs,” information tidbits at each node
  - State information that stays with each agent (agent memory)
Design: Neighbor Discovery
Design: Forward Methods

- **“Eager”**
  - Lost packet = failed transmission

- **“Reliable”**
  - Handles lost transmission and lost ACK

- **“Broadcast”**
  - Most efficient
Test Applications

Perimeter-Based Tracking

Global Data Collection

Local Data Collection
Results: Tracking

![Graph showing tracking results with different markers for Total Packets, Total Nodes Touched, Nodes Active, and Theoretical Bound.](image)
Results: Tracking

- Packets sent is proportional to event path
- Simultaneously active agents is constant
Results: Global Collection

(a) – Active nodes ratio

(b) – Alg. run time vs. Network Size

Ratio of nodes active

Percentage of algorithm complete

Time to finish

Number of nodes

5 nodes

25 nodes

50 nodes

100 nodes

7m spacing

9m spacing
Results: Global Collection

- Activity profile remains similar over $n$
- Instantaneous activity decreases with $n$
Results: Local Collection

![Graph showing the trend in fraction of total nodes touched with increasing network size.](image)
Results: Local Collection

- The fraction of nodes participating looks like $\frac{\sqrt{n}}{n}$
Architecture Directions

• Keep the agent / global program duality
  – Can cast both as “agents” where some agents are globally distributed (semantics)

• Caching of agent code

• Multiple agents per node

• Easier agent interaction

• Security mechanism to deter attacks

• Support for heterogeneous capabilities

• Dynamic linking of new opcodes
Conclusion

• Our framework is scalable
• Enables three classes of common algorithms
• Future work:
  – Revised architecture
  – Could use more comparison to monolithic apps
  – More formal energy analysis (instead of just packet analysis)
  – What HW architecture changes might we consider?
Discussion Questions

- Compelling applications?
- Fairness/Arbitration between agents?
- Is agent-agent messaging going to be useful?
- Mechanisms for detecting/handling node heterogeneity
- Agent code splitting/splicing?
Acknowledgments

• Co-Authors: Jason LeBrun and John Owens
• Philip Levis, Robert Szewczyk @ UCB
• Sandia National Labs Fellowship under Nina Berry
• UC Davis startup funds
Agent Example

- Tracker agent
- Initially deployed to perimeter of network
  - If event detected, spawn to neighbors
  - Repeat until event not detected, then die

```plaintext
if(int(light()) & 2) then     ! if event
  for break = 0 step 0 until break
    if(int(light()) & 2) then ! monitor
      fwdagentb();            ! broadcast
      wait(3);                ! sleep
    else
      break = 1;              ! expire
  end if
next break
end if
```
# Agilla Comparison

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Similar</th>
<th>Different</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>“Forward” operation</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Per-node/agent data</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Remote data access</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Agent migration decisions</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Agent “reactions”</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Geographic routing/forwarding</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Execution model</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>