Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks
What is the problem?

<table>
<thead>
<tr>
<th>Sensor Nets</th>
<th>Challenges</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;#, small, resource constrained nodes Long, remote, unattended operation</td>
<td>Transmission Expensive ➔ Metadata not Code ➔ How often? Nodes unreliable ➔ Topology not static ➔ Continuous propagation effort</td>
<td>Low Maintenance Rapid Propagation Scalability</td>
</tr>
</tbody>
</table>

Need

Introduce new tasks ➔ CODE PROPAGATION
What is trickle?

- An algorithm for wireless sensor nets:
  - Propagate & maintain code updates

- Basic Idea:
  - Metadata “Polite Gossiping”
  - “Talk more when something new”
  - Dynamic gossiping adjustment

- Initial Assumptions:
  - Lossless network
  - Synchronisation
  - Single-Hop
Methodology

• Custom algorithmic simulator
• 2 TOSSIM simulator versions
  – Bit-based (original)
  – Packet-based
• TinyOS motes - 2 experiments - Table/Office
  – Mica2
  – 7 MHz 8-bit CPU
  – 916 MHz Radio ~ 19.2 Kb/s ~ 40 TinyOS pckt/s
  – 128 KB Prog Memory
  – 4 KB RAM
  – 2xAA batteries
Trickle algorithm

- Time is divided up into intervals of length $T$
- For every transmission in $T$, it keeps a counter $c$
- Nodes transmit at random time, $t$ in $[0, T]$
- If at $t$, $c < k$ then node transmits meta-data, otherwise it remains quiet
- If a node hears old meta-data it transmits an update
- Conversely, if a node hears newer meta-data it transmits its old meta-data to provoke another node to send an update...
Introducing **Loss**

- **Ideal Case**
  - Lossless; only 1 transmission per T
  - $O(1)$
- **Worst Case**
  - Each mote is disconnected and thus always transmits
  - $O(n)$
- **Best Case**
  - $P(1 \text{ xmit lost}) = 0.1$
  - $P(2 \text{ xmit lost}) = 0.1 \times 0.1 = 0.01$
  - $P(3 \text{ xmit lost}) = 0.1 \times 0.1 \times 0.1 = 0.001$
  - $O(\log(n))$
- **This is inescapable!**
Introducing Loss

Figure 1

Figure 4
Without **Synchronization**

- **Synchronisation** is not desirable; costs energy!
- ‘Short-Listen’ problem
- **Solution:** Listen-only period, choose $t$ from the interval $[T/2, T]$

- Bounds transmissions to 2$k$ per $T$ period of time
- With loss; $2k \log(n) \Rightarrow \mathcal{O}(\log(n))$

**Figure 5**

**Figure 7**
On a Multi-Hop Net

- In the absence of collisions scales w.r.t. $\log(n)$
- TinyOS's CSMA protocol limits scaling to very dense networks
- Hidden terminal problem!
- Interfering communications affect observed density (plateaus at 75 nodes)
- Given low network utilization, it scales as expected!
On a Multi-Hop Net

Figure 8

Figure 11

Figure 10
Maté, a Trickle Implementation

- **Virtual Machine for TinyOS Sensor networks.**
  - Small static set of code routines.
  - Replacing routines a user updates a network program.
  - Each routine fits in a Tiny OS packet (30 bytes) and has a version number.

- **Motes Broadcast Version Summaries**
  - Contains Version numbers of installed routines.
  - Need for updates are determined by hearing other motes broadcast an older version.

- **Code Propagation**
  - Missing Routines broadcast every 1, 3 & 7 seconds after hearing older code.
Gossiping Intervals

• Want low communications overhead and fast propagation – at odds!
• Give gossiping interval T bounds, $T_h \leq T_l$
• Given that new code is overheard, set $T$ to $T_l$ to quicken propagation
• Given no new updates, set $T$ to $T_h$ to pull the overhead down again
Simulation

- **400 Motes**
  - Placed on 20 x 20 grid, varied spaces between motes.
  - Density Ranges from 5 – 20 ft
  - $T_l$ set to 1 second, $T_h$ set to 1 Minute.

- **Motes booted with randomized times**
  - 1st Minute, selected from uniform distribution.
  - Mote advertises a new Maté routine after 2 minutes
Simulated Code Propagation Rate for Different rates of $T_h$

- $T_h$ does not affect propagation time

Figure 13
Simulated **Time** taken to Propagate Code

**Figure 14**

- 5' Spacing, 6 hops
- 10' Spacing, 16 hops
- 15' Spacing, 32 hops
- 20' Spacing, 40 hops
**Empirical Network Propagation Time**

- $T_h$ of 1 minute, $k = 1$
- Figure 16(a)

- $T_h$ of 20 minutes, $k = 1$
- Figure 16(b)
Conclusions

• Covered / Achieved
  – Low maintenance (few packets/hour)
  – Rapid propagation
  – Simple mechanism (and very little state)
  – Load distribution (lack of addressing means node advertising code does not necessarily transmit it!)
Criticisms

• Bad assumption
  – Assumes node is always

• Not Covered
  – Exact relationship of constants like $T_h$ and $k$ is unclear
  – Policy to propagate code is ignored, concentrated on “when” to send

• Limitations
  – Does not scale up to incredibly dense networks
  – Focuses on small data units
**Related Work: Dissemination Protocols**

- Classic flooding, Gossiping
- SPIN
  - More generic, concerned with dissemination of data, not necessarily code
  - Assumes a lossless network
  - Simulated, but not implemented
Related Work: **Network Reprogramming**

- **XNP**
  - Introduced in TinyOS 1.1
  - Limited to single hops, assumes bidirectionality
  - Sends whole binary image

- **MOAP**
  - Multi-hop
  - Like XNP sends the whole image! Energy intensive, slow
Related Work: Post-trickle

• Deluge
  – Concerned with transmission of large file objects
  – Spatial multiplexing, sender selection and suppression, forward error correction, packet ordering
  – Does not support versioning
  – Energy consumption could be improved
questions