
Philip Levis, Neil Patel, Scott Shenker and David Culler
EECS Department. University of California, Berkeley

Presented by:
Simrat Wason
MS-ECE, Duke University
ECE 299.02 Wireless Networking & Mobile Computing
Addressing the Problem:

Wireless Sensor Networks

Consist of large no. of small, resource-constrained computing nodes, Operate unattended for long durations, scale and embedded nature

(e.g.- collared on roving herds of zebras or buried in bird burrows)

REQUIRE

CODE to be PROPAGATED over the Network.
→ Codes need to be propagated QUICKLY and EFFICIENTLY

→ Maintenance of a consistent code image throughout the Network

+ Low power radios (high loss, transient disconnection)
Similar Example:

Maté, a tiny virtual machine designed for Sensor Networks:
- Maintenance and Propagation: Code fragmentation transmissions
- Limitations that prevent it from being feasibly deployable.
- Easily saturates a network.

TRICKLE

addresses this problem.
Prior research:

Although Network Characteristics are different, these techniques can be borrowed to solve the problem.

1. **Controlled density** aware research
   (for wireless and multicast networks)

2. **Epidemic and gossiping** algorithms
   (for maintaining data consistency in wired, distributed systems)
AN ‘EFFECTIVE’ ALGORITHM:

- Must **adjust to local network density** as **controlled floods** do,

BUT

- Continually **maintain consistency** in a manner similar to **epidemic algorithm**.
TRICKLE:

To exchange code metadata
→ “Polite Gossip” policy

- If a node hears gossip with same metadata, it keeps quiet.
- If it hears an old gossip, it broadcasts a code update.
Instead of FLOODING

→ Algorithm CONTROLS send rate
→ Each node hears a small TRICKLE of packets (just enough to stay up to date!)
GOALS:

TRICKLE:

- Propagation (RAPID)
- Maintenance (LOW)
→ Nodes adjust the length of their gossiping attention spans, communicating more often when there is new code.

i.e. it self-regulates using a local “polite gossip” to exchange code metadata.
Experimental Methodology:

Three Different Platforms:

1. **Abstract Simulation**
   - High level abstract algorithm simulator: Trickle-specific, quickly evaluates change in algo behavior

2. **TOSSIM**
   - A bit-level simulator for TinyOS, a sensor network operating system

3. **Tiny OS nodes**
   - Used for empirical studies
MAINTENANCE: “polite gossip” to exchange code metadata with nearby network neighbors

Each node maintains a counter c, threshold k, timer t in range [0, T] where T = time constant

- When a node hears metadata identical to its own, it increments c.
- At time t, the node broadcasts a summary of its program if c < k.
- When the interval of size T completes, c is reset to zero and t is reset to a new value in the range [0, T].
Maintenance Algorithm

- Maintenance with Loss
- Maintenance without Synchronization
- Maintenance with Multiple Cells
Maintenance with Loss:

The logarithmic behavior represents the probability that a single node misses a no. of transmissions.

Figure 4: Number of Transmissions as Cell Density Increases for Different Packet Loss Rates
Maintenance without synchronization:

Trickle can suffer from SHORT-LISTEN problem

Figure 5: The Short Listen Problem. Dark bars represent transmissions, light bars suppressed transmissions, and dashed lines are receptions. Note that node B transmits in all three intervals.
Modification to remove short-listen effect:

Figure 7: Trickle Maintenance with a $k$ of 1 and a Listen-Only Period. Dark boxes are transmissions, grey boxes are suppressed transmissions, and dotted lines are heard transmissions.
The ‘listening line’ shows the no. of transmissions in a cell with no synchronization when Trickle uses this listening period.

Figure 6: The Short Listen Problem’s Effect on Scalability, \( k = 1 \). Without synchronization, Trickle scales with \( O(\sqrt{n}) \). A listening period restores this to asymptotically bounded by a constant.
Maintenance with Multiple Cells:

(a) Total Transmissions per Interval  
(b) Receptions per Transmission  
(c) Redundancy

Figure 8: Simulated Trickle Scalability for Multiple Cells with Increasing Density. Nodes were uniformly distributed in a 50’x50’ square area.
Load Distribution and Empirical Study

- The communication rate across the network is fairly uniform.
- The maintenance scales logarithmically. (as the simulation results indicate)
PROPAGATION:

→ Two goals, i.e., Rapid propagation, and Low overhead require:
  Comm. To be frequent/ Comm. to be infrequent

→ DYNAMICALLY scaling T, use the maintenance algorithm to rapidly propagate updates with a very small cost.
TRICKLE Pseudo code:

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$ Expires</td>
<td>Double $\tau$, up to $\tau_h$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>$t$ Expires</td>
<td>If $c &lt; k$, transmit.</td>
</tr>
<tr>
<td>Receive same metadata</td>
<td>Increment $c$.</td>
</tr>
<tr>
<td>Receive newer metadata</td>
<td>Set $\tau$ to $\tau_l$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>Receive newer code</td>
<td>Set $\tau$ to $\tau_l$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>Receive older metadata</td>
<td>Send updates.</td>
</tr>
</tbody>
</table>

$t$ is picked from the range $\left[\frac{\tau}{2}, \tau\right]$
PROPAGATION:

SNLR (Sensor Network Language Runtime), a Trickle Implementation

- Small static set of code routines, each with a version no.
- Runtime keeps the most recent version no.
- Nodes broadcast version summaries INSTEAD of sending entire routines.

A user can update a network’s program by replacing these routines.
Simulation and Empirical study:

Used TOSSIM to observe the behavior of Trickle during a propagation event.

Little trade-off between

→ the maintenance overhead of Trickle, and
→ its effectiveness in the face of a propagation event.
SNLR implementation requires-

- Few system resources (approx. 7 bytes of RAM)
- Trickle requires only 11 bytes for Counters, remaining RAM for coordinating pending and initialization flags.

HENCE,

The Algorithm requires

- very few CPU cycles,

and

- can operate at a very low duty cycle.
To conclude:

- Quickly propagate new code
- Very small overhead.
- Simple mechanism.
- Requires very little state
- Scaling logarithmically with cell size, it can be used effectively in very dense networks.
- **LIMITATION**: Assumes nodes are always ON.
Discussion:

- Trade-off btw ENERGY OVERHEAD and REPROGRAMMING RATE
- Acts as a flood control protocol, w/c is inverse of protocols such as SPIN, w/c transmit metadata freely but controls data transmission
- Can go beyond code propagation( by adding predicates to summaries)
Questions?