
Philip Levis, Neil Patel, David Culler, Scott Shenker
USENIX NSDI 2004

Presented by Yen-Cheng Wang
Introduction

- WSN (Wireless Sensor networks) always composed by large number of small and resource constrained computing nodes.

- Usually, WSN needs a long time of operation, as environment has changed, perhaps the application programs needs changing with environment.

- Need an efficient algorithm for motes to propagate code, in this paper propose **Trickle**, an algorithm for code propagation and maintenance in WSN.
Trickle Algorithm

- **Overview**
  - Trickle assumes that motes can succinctly describe their code with metadata.
  - Every so often, motes transmit metadata if it hasn’t heard other motes transmit the same thing.
  - Either every mote that hears the message is up to date, or a recipient detects the need for an update.
Trickle Algorithm – Maintenance

- Trickle uses polite gossip to exchange code metadata, if “mote A” heard several identical metadata from other motes in a time interval, “mote A” will stay quiet.

- Each mote selects only one random transmission point in each period for broadcasts metadata.

- Time is split into periods:
  - Each mote maintains:
    - A timer $t$ in the range $[0 \sim T]$.
    - A counter $c$ (compare with $k$).
    - A threshold $k$ (1 or 2 usually).
Trickle Algorithm – Maintenance

- Dark boxes are transmissions
- Gray boxes are suppressed transmissions
- Dotted lines are heard transmissions
- Solid lines mark interval boundaries
- Set $k=1$
Trickle Algorithm – Maintenance

Maintenance with Loss

![Graph showing the relationship between the number of Motes and the number of transmissions/interval with different loss percentages. The graph includes lines for 60%, 40%, 20%, and 0% loss, with the y-axis labeled "Transmissions/Interval" and the x-axis labeled "Motes." ]
Trickle Algorithm – Maintenance

Short-Listen problem
- If nodes are out of synchronization, some motes might transmit quickly before others had a chance to speak up.
Solution
- Enforce a listen-only period, allows transmission in the range $\frac{T}{2} \sim T$.
Trickle Algorithm – Maintenance

The short-listen problem’s effect on scalability
Trickle Algorithm – Maintenance

- Maintenance in a Multi-hop Network
- Used by TOSSIM
Trickle Algorithm – Maintenance

![Graph showing receptions/transmission vs. number of motes. The graph compares 'Hidden Terminal' and 'No Hidden Terminal' scenarios. The y-axis represents receptions/transmission with values ranging from 0 to 350, and the x-axis represents the number of motes ranging from 1 to 1024. The graph indicates a significant increase in receptions/transmission as the number of motes increases.]
Trickle Algorithm – Maintenance

- Hidden Terminal
- No Hidden Terminal
Empirical Study

- $T = \text{one minute}$
- measured redundancy over a 20 minute period for increasing numbers of motes
Trickle Algorithm – Propagation

- Large T
  - Has a low communication overhead
  - But slowly propagates information

- Small T
  - Has a higher communication overhead
  - But propagates more quickly

- Solution
  - Dynamic scaling of T
Trickle Algorithm – Propagation

Dynamic scaling of $T$

- $T$ has a lower bound, $T_l$, and an upper bound, $T_h$

<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 $[\frac{\tau}{2}, \tau]$
Trickle Algorithm - Propagation

TI = 1 seconds
Trickle Algorithm – Propagation

Motes 4, 11, 17, 18 and 19 weren’t instrumented TCP.
The others were instrumented TCP except mote 16.
$T_I = 1$ seconds.
Trickle Algorithm – Propagation

AVG : 22 seconds

\( \tau_h \) of 1 minute, \( k = 1 \)
Trickle Algorithm – Propagation

AVG: 32 seconds

$\tau_h$ of 20 minutes, $k = 1$
AVG : 29 seconds

$\tau_h$ of 20 minutes, $k = 2$
Conclusion

- Trickle can quickly propagate new code into a network, while imposing a small overhead
  - Using dynamic T values for trade-off between energy overhead and reprogramming rate
  - Using listen-only period for reduce the effect of the short-listen problem

- One limitation of Trickle is that it currently assumes motes are always on