Mobile and Sensor Systems

Lecture 6: Sensor Network
Reprogramming and Mobile Sensors
Dr Sarfraz Nawaz
In this lecture

• We will describe techniques to reprogram a sensor network while deployed.
• We describe briefly mobile sensor networks and mobile sensor network reprogramming.
Sensor Network Programming/Reprogramming

• Long Lifetime requires retasking the sensors.
• However programming each node separately may not be feasible.
• What is reprogramming?
  – Send function parameters ("wake up every X seconds").
  – Sending binaries or code to compile.
• Checking that each node has the right code can be quite costly too.
Idea

• The first step is to detect when nodes need updates (continuous process).

• When there is no new code:
  – **Maintenance** cost should approach zero

• When there is new code.
  – **Propagation** should be rapid.
Trickle

• Simple, “polite gossip” algorithm.

• “Every once in a while, broadcast what code you have, unless you’ve heard some other nodes broadcast the same thing, in which case, stay silent for a while.”
Trickle

• Within a node time period:
  – If a node hears older metadata, it broadcasts the new data.
  – If a node hears newer metadata, it broadcasts its own metadata (which will cause other nodes to send the new code).
  – If a node hears the same metadata, it increases a counter: If a threshold is reached, the node does not transmit its metadata. Otherwise, it transmits its metadata.
Trickle – Main Parameters

• Counter $c$: Count how many times identical metadata has been heard

• $k$: threshold to determine how many times identical metadata must be heard before suppressing transmission of a node’s metadata

• $t$: the time at which a node will transmit its metadata. $t$ is in the range of $[0, \tau]$
Example Trickle Execution

\[ k = 1 \]

\[ t = \tau \]

transmission

suppressed transmission

reception
Example Trickle Execution

\[ k = 1 \]

transmission

suppressed transmission

reception
Example Trickle Execution

\[ c \]

\[ k = 1 \]

\[ t_{1a} \]

\[ \tau \]

transmission
suppressed transmission
reception
Example Trickle Execution
Example Trickle Execution

\[ t_{1a} \]

\[ t_{3a} \]
Example Trickle Execution

\[
\begin{align*}
&\text{transmission} & \text{suppressed transmission} & \text{reception} \\
time & & & \\
& 1 & & \\
& 2 & & \\
& 3 & & \\
& \tau & & \\
\end{align*}
\]
Example Trickle Execution

\[ k = 1 \]

\[ t_{1a}, t_{2a}, t_{3a} \]

transmission
suppressed transmission
reception
Example Trickle Execution

\[ t_{1a}, t_{2a}, t_{2b}, t_{3a} \]

transmission  suppressed transmission  reception

\[ k=1 \]

\[ c \]

\[ k=1 \]
Example Trickle Execution

\[ c = 1 \]

\[ k = 1 \]

\[ \tau \]

transmission

suppressed transmission

reception
Example Trickle Execution

C

k=1

1

transmission

suppressed transmission

reception

τ

time

transmission

suppressed transmission

reception
Assumptions

• Precise node synchronization
• No packet Loss

• Impact of these assumption?
Trickle: Impact of Packet Loss

Figure 4: Number of Transmissions as Density Increases for Different Packet Loss Rates.
Trickle Maintenance without Synchronization – Short Listen Problem

Figure 5: The Short Listen Problem For Motes A, B, C, and D. Dark bars represent transmissions, light bars suppressed transmissions, and dashed lines are receptions. Tick marks indicate interval boundaries. Mote B transmits in all three intervals.

- Mote B selects a small $t$ on each of its three intervals:
  - Although other motes transmit, mote B’s transmissions are never suppressed.
- The number of transmissions per intervals increases significantly.
Trickle – Impact of Short Listen Problem

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.
Solution to Short Listen Problem

• Instead of picking a $t$ in the range $[0, T]$, $t$ is selected in the range $[T/2, T]$
Propagation

• Tradeoff between different values of $T$
  – A large $T$
    • Low communication overhead
    • Slowly propagates information
  – A small $T$
    • High communication overhead
    • Propagate more quickly

• How to improve?
  – Dynamically adjust $T$
    • Lower Bound $T_l$
    • Upper Bound $T_h$
Figure 10: Communication topography of a simulated 400 mote network in a 20x20 grid with 5 foot spacing (95’x95’), running for twenty minutes with a duration of one minute. The x and y axes represent space, with motes being at line intersections. Color denotes the number of transmissions or receptions at a given mote.

b) Contributions in Figure 1, a five foot spacing forms a six hop network from grid corner to corner. This simulation was run with a duration of one minute, and ran for twenty minutes of virtual time. The topology shows that some motes send more than others, in a mostly random pattern. Given that the predominant range is one, two, or three packets, this non-uniformity is easily attributed to statistical variation. A few motes show markedly more transmissions, for example, six. This is the result of some motes being poor receivers. If many of their incoming links have high loss rates (drawn from the distribution in Figure 1), they will have a small observed density, as they receive few packets.

Figure 10(b) shows the reception distribution. Unlike the transmission distribution, this shows clear patterns. Motes toward the edges and corners of the grid receive fewer packets than those in the center. This is due to the non-uniform network density; a mote at a corner has one quarter the neighbors as one in the center. Additionally, a mote in the center has many more neighbors that cannot hear one another; so that a transmission in one will not suppress a transmission in another. In contrast, almost all of the neighbors of a corner mote can hear one another. Although the transmission topology is quite noisy, the reception topography is smooth. The number of transmissions is very small compared to the number of receptions: the communication rate across the network is fairly uniform.

4.5 Empirical Study
To evaluate Trickle’s scalability in a real network, we recreated, as best we could, the experiments shown in Figures 6 and 8. We placed motes on a small table, with their transmission signal strength set very low, making the table a small multi-hop network. With a duration of one minute, we measured Trickle redundancy over a twenty minute period for increasing numbers of motes. Figure 11 shows the results. They show similar scaling to the results from TOSSIM-bit. For example, the TOSSIM-bit results in Figure 8(c) show a 64 mote network having an redundancy of 1.1; the empirical results show 1.35. The empirical results show that maintenance scales as the simulation results indicate it should: logarithmically.

Evaluating propagation requires an implementation; among other things, there must be code to propagate. In the next section, we present an implementation of Trickle, evaluating it in simulation and empirically.

5. PROPAGATION
A large (gossiping interval) has a low communication overhead, but slowly propagates information. Conversely, a small has a higher communication overhead, but propagates more quickly. These two goals, rapid propagation and low overhead, are fundamentally at odds: the former requires communication to be frequent, while the latter requires it to be infrequent.

By dynamically scaling, Trickle can use its maintenance algorithm to rapidly propagate updates with a very small cost. has a lower bound, and an upper bound . When expires, it doubles, up to . When a mote hears a summary with newer data than it has, it resets to be . When a mote hears a summary with older code than it has, it sends the code, to bring the other mote up to date. When a mote installs new code, it resets to , to make sure that it spreads quickly. This is necessary.

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

Figure 12: Trickle Pseudocode.
Mobile Sensor Networks

- We have considered fixed sensor networks.
- There are however examples in which the sensor networks are mobile, i.e., the nodes of the networks do not have a fixed position.
- Example of this are when sensors are moved through controlled movement (E.g. a sensor robot) or when sensors are attached to moving entities and the mobility is independent from the sensing activity (E.g. animals or vehicles or humans).
Impact of Mobility

• MAC Layer protocols:
  – Mobility impacts the protocol of duty cycling as the neighbours of the nodes are not the same all the time.
  – Adaptation of low power listening protocols are reasonably suitable.
  – Alternatively, approaches which keep into account periodic encounter patterns.
Impact of Mobility (2)

• Routing protocols:
  – All of a sudden establishing a tree structure does not make sense any longer.
  – Delay tolerant routing protocols are applicable (on top of duty cycling approaches).
Impact of Mobility(3)

• Reprogramming:
  – Existing solutions target connected fixed networks.
  – Delay tolerant solutions could be applied however some attention to targeted set of nodes should be applied (eg reprogram only nodes which go to certain areas) and attention to avoid useless code broadcasts should be paid.
Mobile WSN Reprogramming

I want to retask my network

Network of animals wearing sensors on their collars
What’s the best way to distribute the update?

- Flooding? No too expensive.
- These animals are social!
- These social groups tend to be stable over time, and more importantly, they spend a lot of time together, regularly.
Social Dissemination

- Dissemination:
  - use social characteristics of the network!

- Selective update:
  - use the network to figure out whom to update.
Social Dissemination

• Instead of flooding the network, let us try to use the social characteristics: social groups, social links between nodes, as well as group leaders;

• Groups tend to stay connected - perfect for maintenance!

• Animals do not behave all in the same way - some are more active than others:
  – group leaders: identify leaders, and spread code among them. Identify clusters: wait until they come together let leaders disseminate code using smart broadcasts to their group.
References
