Trickle

A self-regulating algorithm for code propagation and maintenance in wireless sensor networks

Philip Levis, Neil Patel, David Culler, Scott Shenker

presented by Łukasz Chodarcewicz
Distributed Systems, 2011

Agenda

- Abstract
- Introduction
- Methodology
- Trickle's concept
- Scalability
- Code propagation
- Related work and algorithms
- Conclusions

Abstract

- Trickle – algorithm for propagating and maintaining code updates in wireless sensor networks
- Uses techniques from the epidemic/gossip, scalable multicast and wireless broadcast literature

Introduction – sensor network

- A wireless sensor network is network composed from spacially distributed autonomous computing nodes („motes”)
- Tipically nodes are small devices like sensors used to monitor physical or environmental conditions like temperature, sound, vibration, pressure etc.
- For an example TinyOS mica2 mote has 916 Mhz radio, 128KB program memory, 4KB RAM, 7MHz 8-bit microcontroller

Sensor network

- May be composed of large number of motes
- Often must uperate unattended for months or years
- Need of ability to introduce new code to retask network
- Nodes of the network are resource constrained
- Motes may come and go, due to temporary disconnections, failure and network repopulation

Code propagation

- Propagating code is costly.
- Learning when to propagate is even more so.
- May transmit metadata to determine when the code is needed

Propagation determination algorithm

- Low maintenance
- Rapid propagation
- Scalability
- Trickle is an algorithm designed to meet all the above criteria

Methodology

- To investigate and evaluate the algorithm, three different platforms were used:
  - High level abstract algorithmic simulator.
  - TOSSIM – bit-level mote simulator for TinyOS (sensor network operating system)
  - TinyOS mica-2 motes for empirical studies
- Same implementation of Trickle on all of them.

Trickle – basic concept

- Periodically, every node in sensor network transmits code metadata if it has not heard a few other motes transmit the same thing.

Trickle

- Uses „polite gossip” to exchange code metadata
- Each mote maintains counter c, threshold k (small number)
- Algorithm breaks time into intervals of length T
- At the beginning of every interval we choose randomply t of range [0, T].
- At time t we broadcast our metadata if c < k

Trickle

- When we hear someone's metada that is same as ours, we increment c
- When we hear someone has old code, we send our code update
- When we hear someone has newer code, we send our metadata

Trickle - intervals

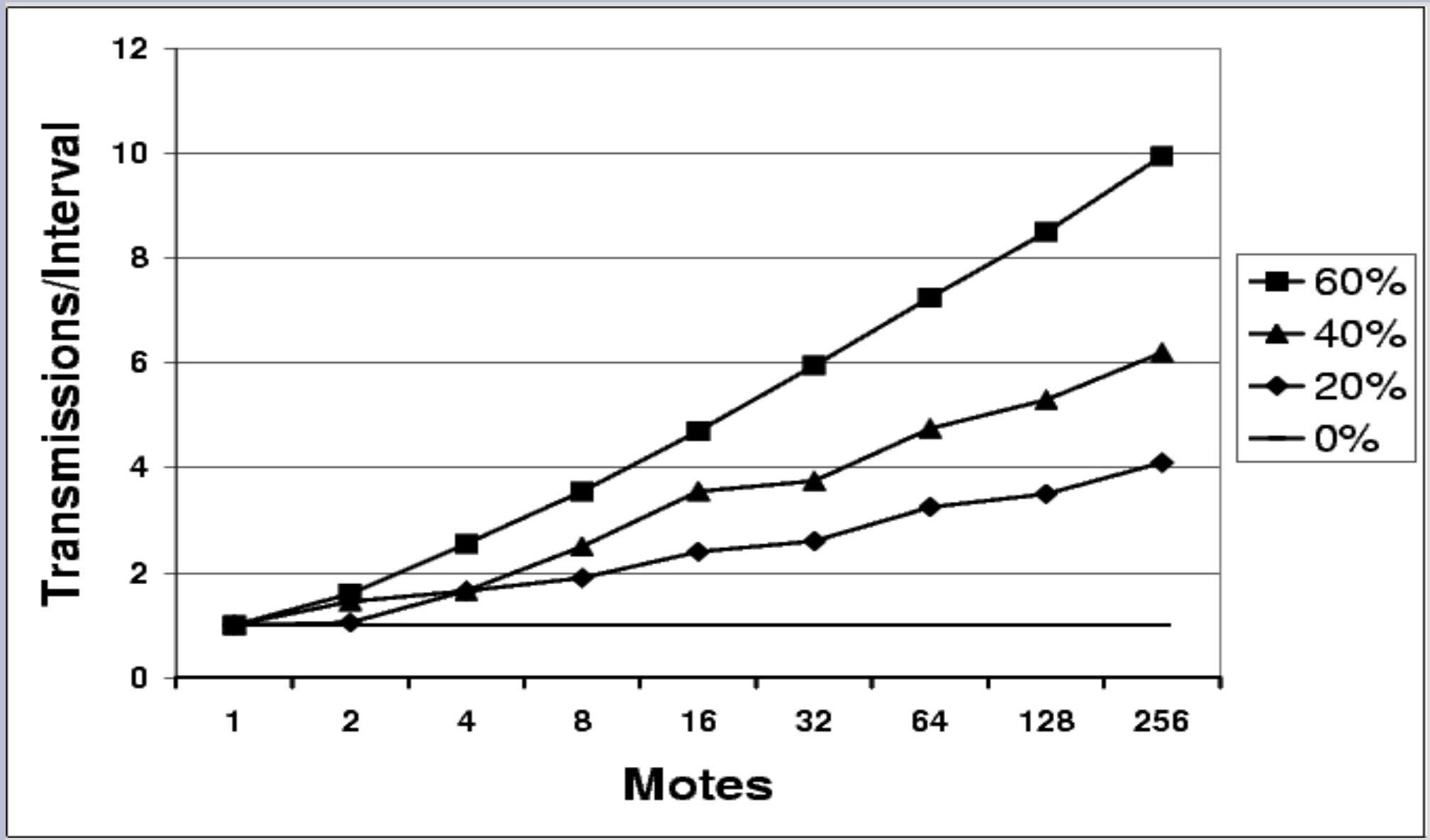
Intervals and maintenance

- In a lossless, single hop network with perfect sinchronization there will be exactly k broadcasts in every interval. No matter how many nodes there are.
- Random selection of t uniformly distributes the choice of who broadcasts in a given interval.

Maintenance with packet loss

- We assumed that every transmission is received by every node in network without any errors or distruptions.
- When testing Trickle with packet loss, the number of transmissions grows with density at O(log(n))

Maintenance with packet loss



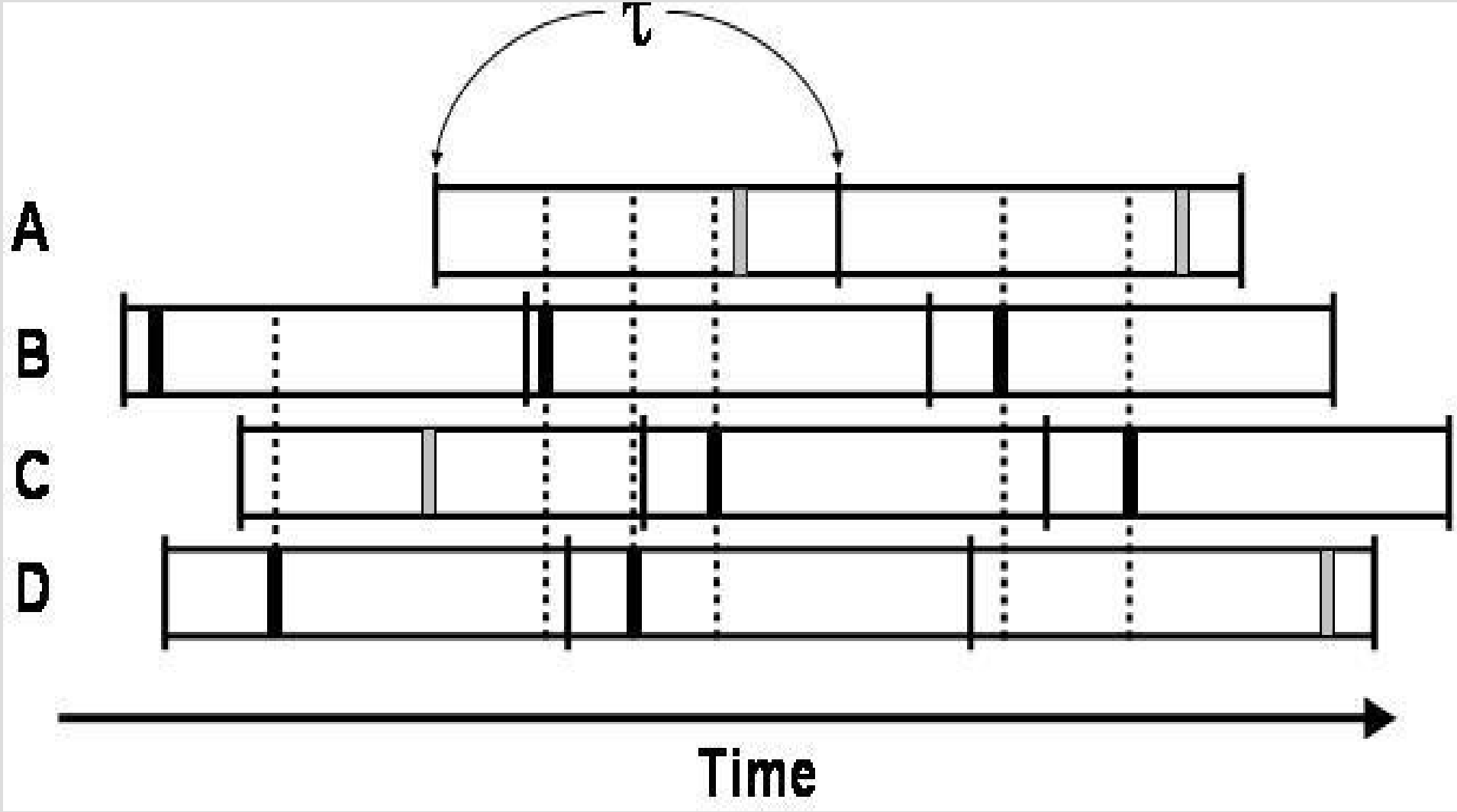
Maintenance without synchronization

- All prior results assume, that all nodes in network have synchronized intervals.
- But time synchronization imposes a communication, and therefore energy, overhead...
- Unfortunately without synchronization, Trickle can suffer from the short-listen problem.

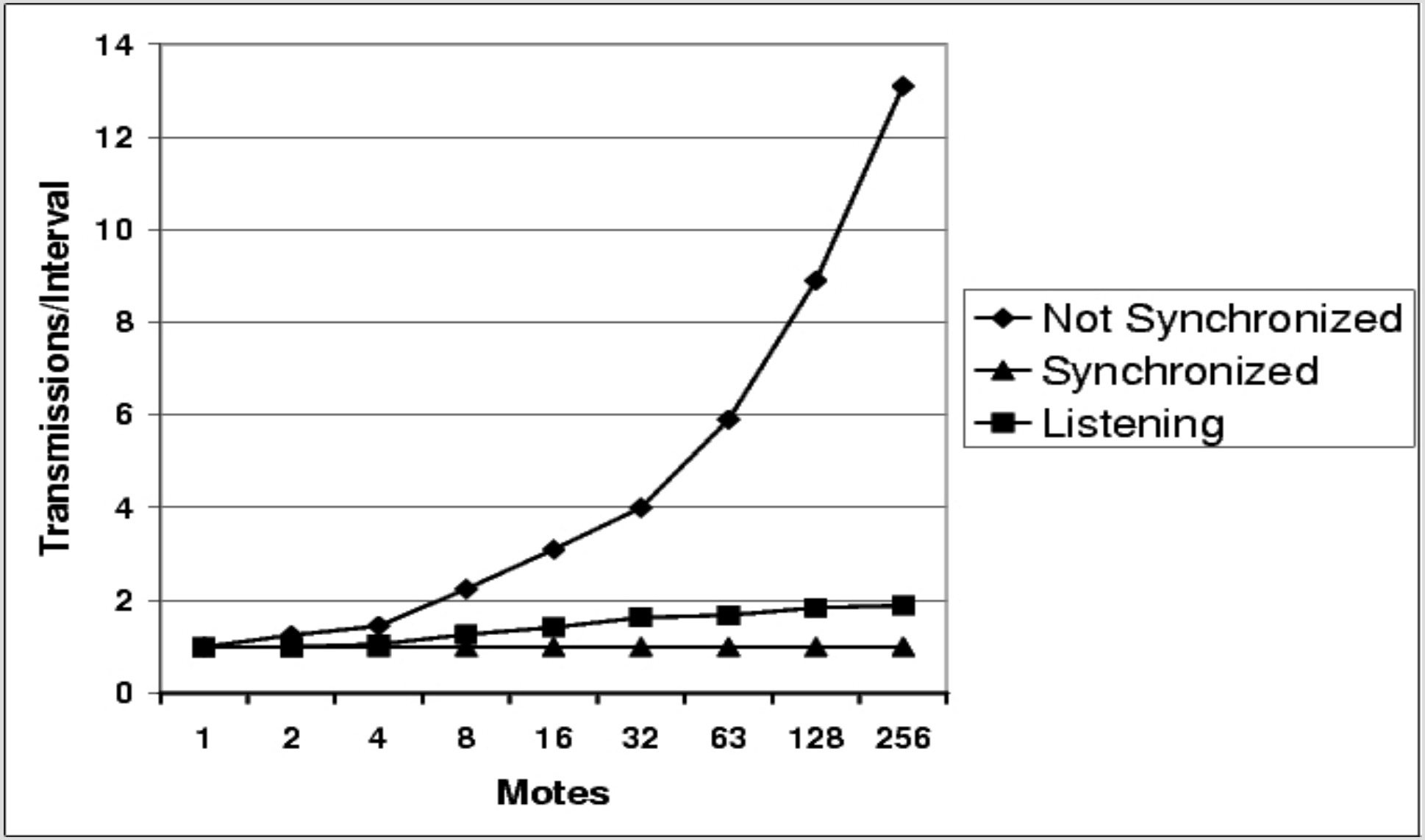
Maintenance without synchronization

- Short listen problem is when some subset of motes gossip soon after the beginning of their interval, listening for only a short time before anyone else has a chance to speak up.
- Unfortunatly overlaping intervals and broadcast time drawn unsuccesfully badly result with redundant transmissions.

Maintenance without synchronization, short-listen problem



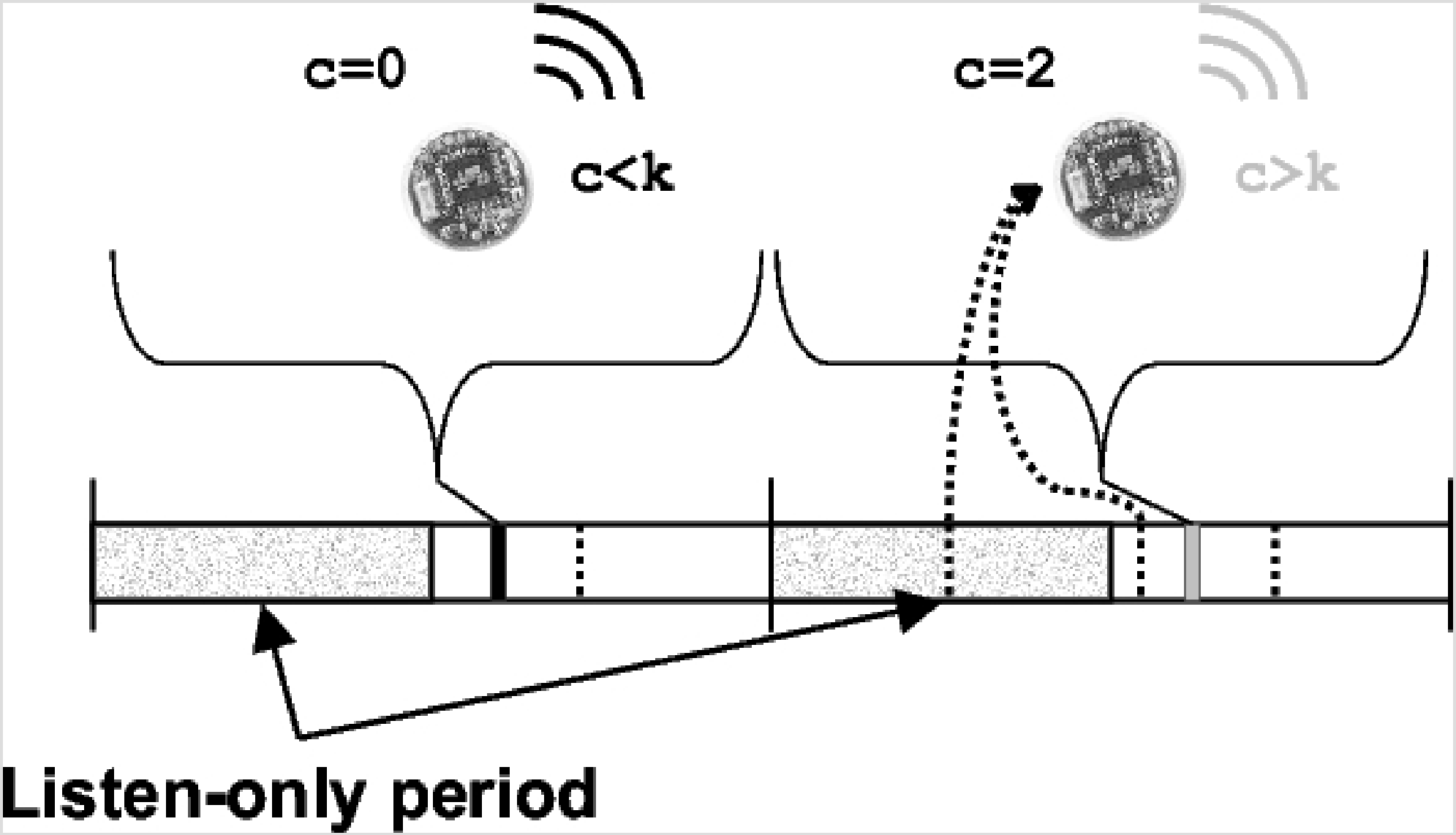
Maintenance without synchronization



Solution

- To remove the short-listen problem effect, Trickle is modified slightly:
  - Instead of picking t in range of [0, T], it is selected in [T/2, T].
  - This defines a „listen-only” period of the first half of the interval.
- This bounds total sends in a lossless single-hop network to be 2k per interval, and with loss it scales as 2k * log(n) witch is still O(log(n))

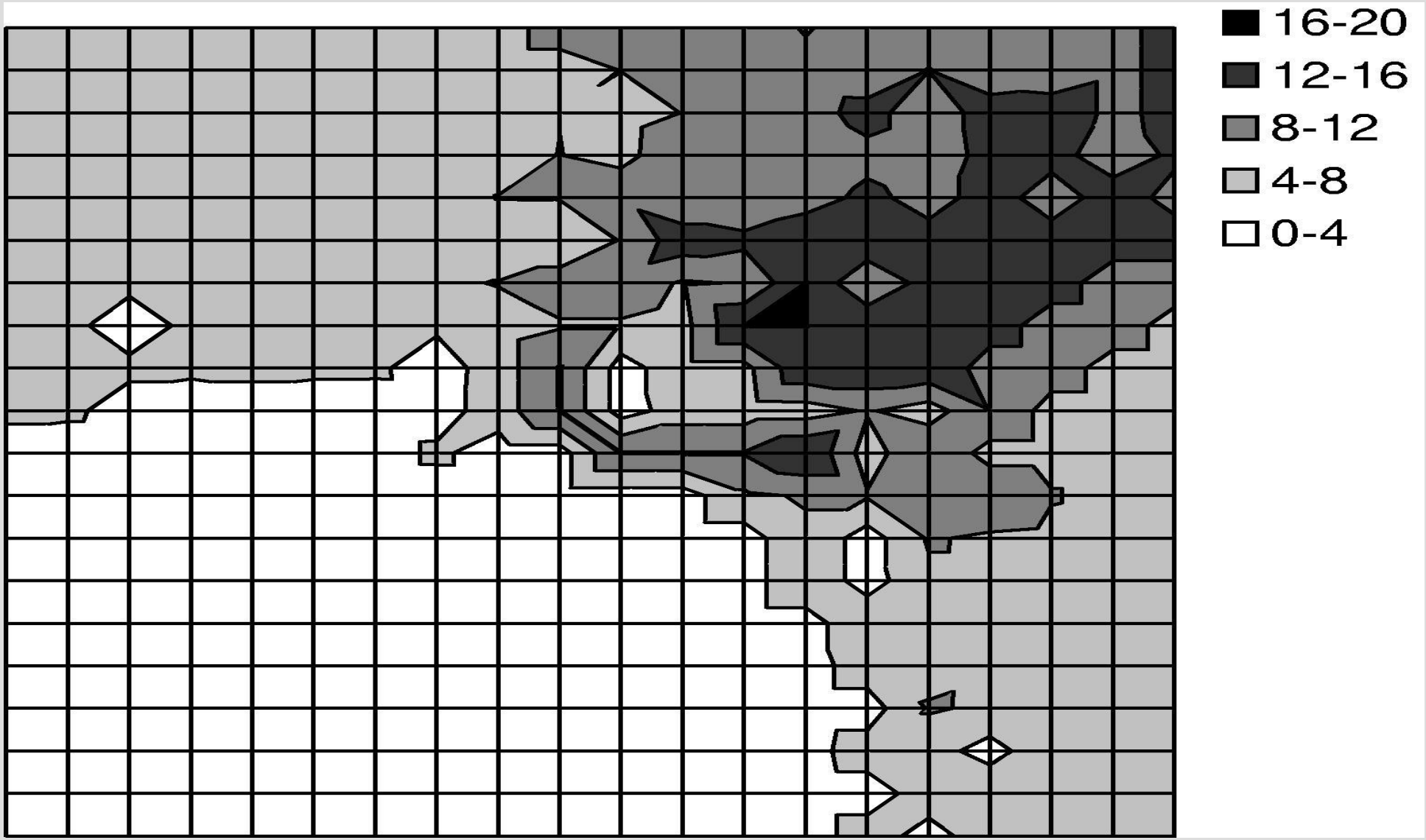
Solution



Multi-hop network

- TOSSIM simulator used to test behavior in multi-hop network
- Motes plased randomly in a 50' x 50' area with an uniform distribution, a T=1s and k=1
- Generaly the results were close to predicted.
- Interesting phenomenon when the network is very dense – hidden terminal problem

Hidden terminal problem



Code propagation

- Trickle tries to make a tradeoff between a large T (witch gives low communication overhead, but slowly propagates changes) and rapid code propagations (witch needs low T).
- To meet the above criteria, it uses dynamic scaling of T. T has a lower bound T1 and upper bound T2.
- It doesn't concentrate on how the code itself should be transmited.

Dynamic time intervals – full Trickle's algorithm

Event	Action
T expires	double T up tu T2, reset c, pick new t
t expires	if c < k transmit
receive same metadata	increment c
receive newer metadata	set T to T1, reset c, pick new t
receive new code	set T to T1, reset c, pick new t
receive older metadata	send updates

Mate, a Trickles implementation

- Small, static set of code routines.
- Each routine may be in many versions but runtime keeps the most recent one.
- Metadata – vector of versions of routines
- Uses mate to periodically broadcast version summary
- Requires few system resources:
  - 70 bytes of RAM, half of it for message buffering
  - Trickle itself uses only 11 bytes for counters.

Related work

- There are related works in the areas of flooding algorithms for wireless and multicast networks and epidemic/gossiping algorithms for maintaining data consistency in distributed systems...
- But in both cases, they assume network characteristics and priorities distinct from the ones important in Trickle algorithm.

Related work

- An interesting work in a topic related to Trickle is N. Reijers's and K. Langendoen's „Efficient code distribution in wireless sensor networks”
- Although it concentrates on how to send code update in sensor network rather than when to send it, the construction of Trickle leaves place for using separate module for sending code.

Summary

- Trickle is simple gossip algorithm with scalability, rapid propagation and low maintenance properties
- It doesn't depend on code propagation protocol

Questions?