Dynamic Reconfiguration of Wireless Sensor Networks to Support Heterogeneous Applications

Marcin Szczodrak[1] **Omprakash Gnawali**[2] **Luca Carloni**[1]

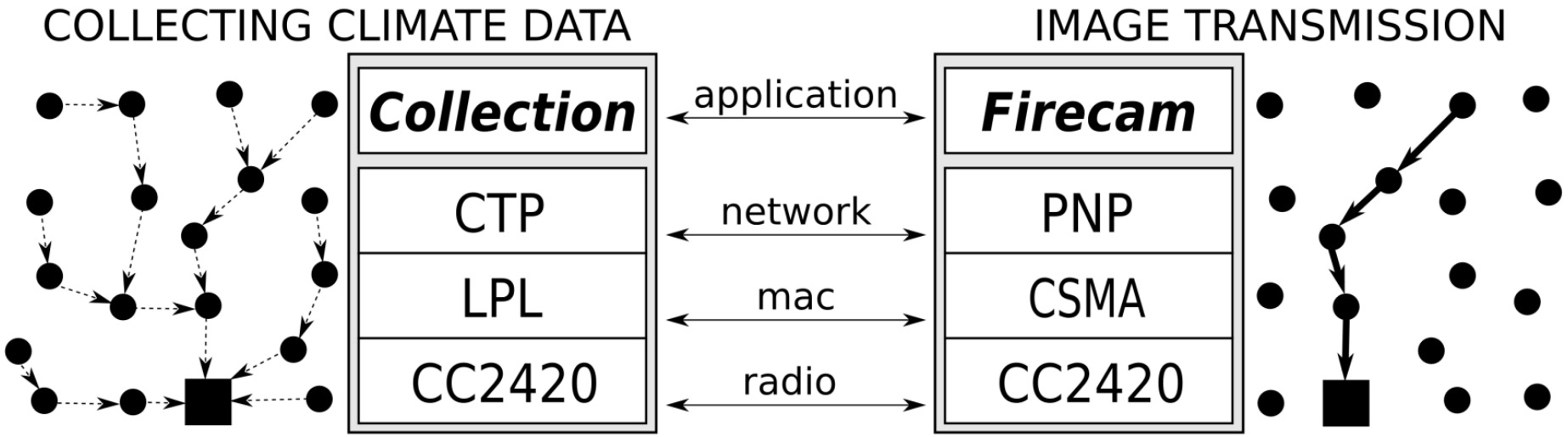
Columbia University[1] **University of Houston**[2]

DCOSS Conference · Cambridge, Massachusetts · May 21, 2013

Motivation: Run Two Applications on the Same WSN

Collection: collect building's climate data
- reliable, low-rate
- many-to-one
- energy efficient

Firecam: sends images across a building
- low-latency
- point-to-point
- bulk-data streaming

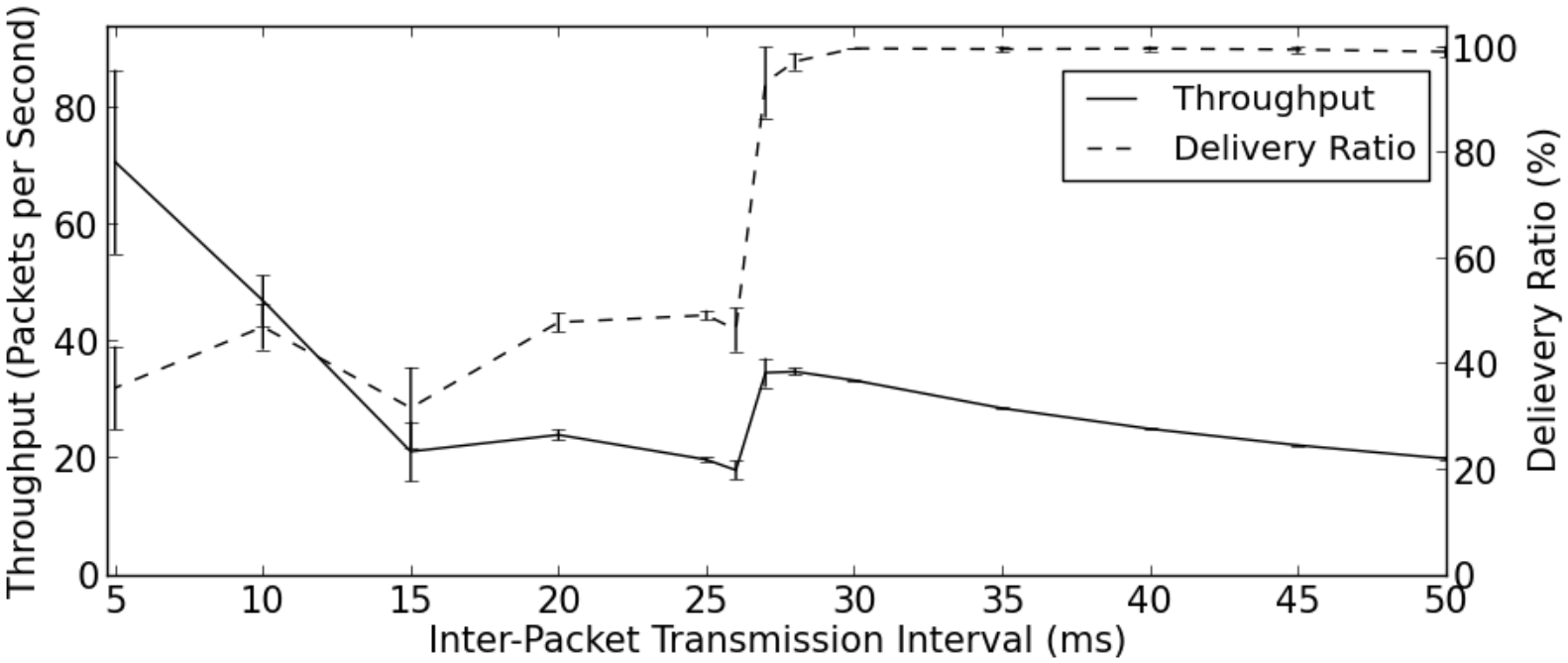


Emergency Response Requirement:

During normal operation, run **Collection**. When an emergency event occurs (e.g. a smoke sensor goes off), switch to **Firecam**.

Two Applications, Different Communication Requirements

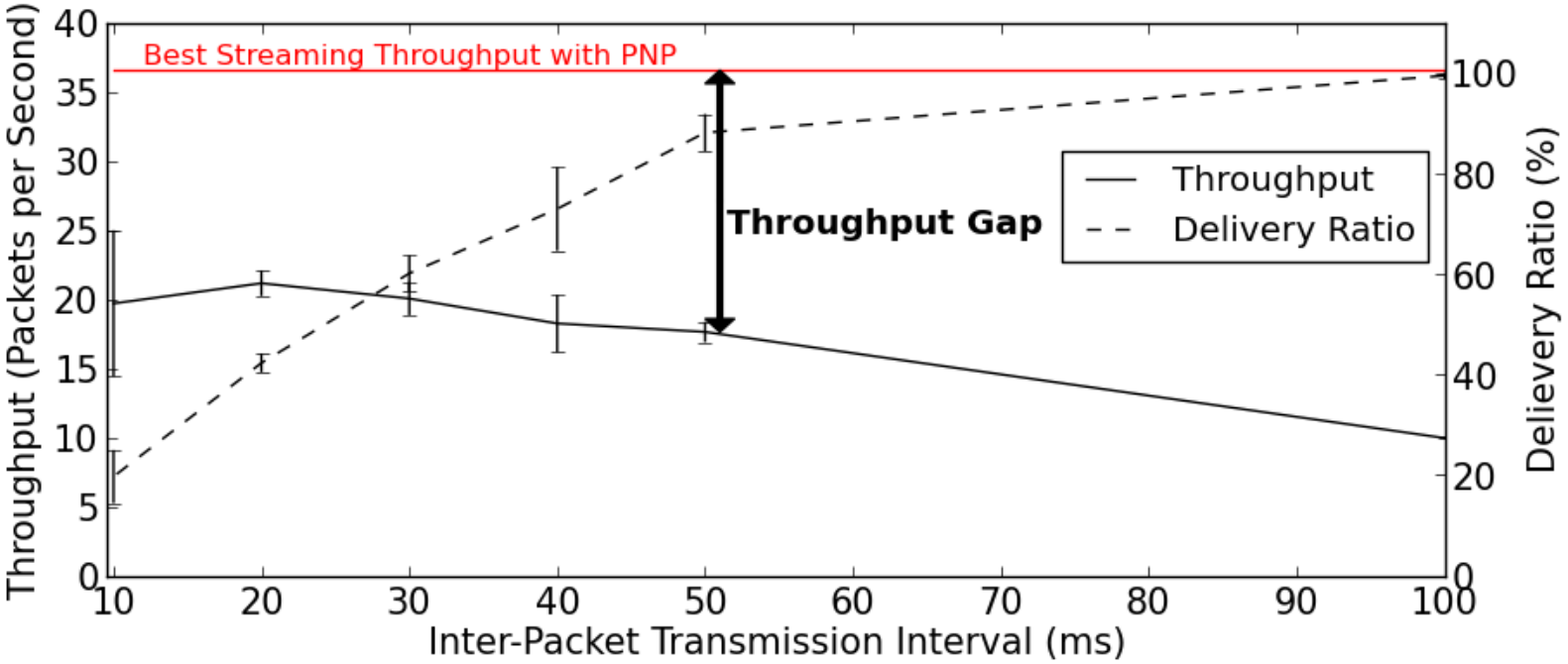
- setup: Indriya testbed, a three-floor building in NUS with 139 TelosB motes
- Transmitting 75 KBytes across the building, over 7-9 hop path
- 76800 bytes takes a 240x320 picture with each pixel encoded as a byte



- Almost 100% of packets delivered when transmission delay was 30ms or more
- For 28ms delay, PNP's delivery ratio was 97.4%
- PNP transmits 36.5 packets per second
- Sending a 75KByte picture takes ~ **21 seconds**

Two Applications, Different Communication Requirements

- Transmitting 75 KBytes across the building, over 7-9 hop path



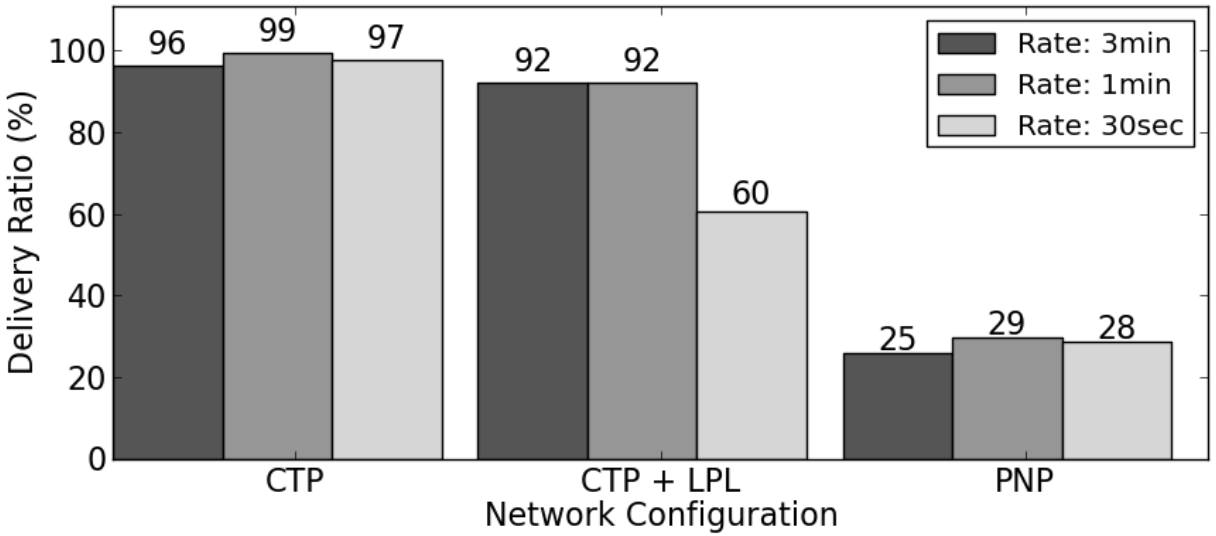
- For 50ms delay CTP delivery ratio is 88%
- This delivers 17.61 packets per second
- Sending 75KBytes picture takes ~ **44 seconds**

Experimental Observation:

CTP does not support high throughput on a point-to-point path

Two Applications Need Different Protocol Stacks

- Collecting data from the whole network at one sink node
- Each node periodically sends 4 bytes of data



- Low-Power Listening at 100ms
- CTP performs well with radio always ON or duty-cycled
- PNP drops more than 70% of packets

Experimental Observation:

PNP cannot support the same many-to-one delivery ratio as CTP

The Need for Dynamic Protocol Stack Reconfiguration

Each of the protocol stacks supports well the corresponding application, for which it has been optimized, while poorly supporting the other application

Fennec Fox Framework

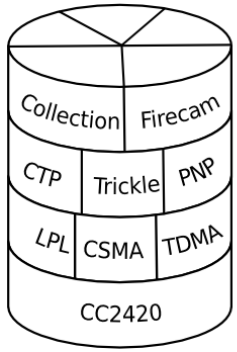
1. Programming Language – Swift Fox

- defines applications and supporting communication infrastructure
- specifies when to context-switch applications

```
configuration Monitoring { collection(1024, NODE, 107)
                           ctp(107) lpl (100, 10), cc2420(1)}
configuration Emergency {firecam(28), pnp(),
                                  csma(0,0),cc2420(0)}
event fire {smoke = ON }
event check {timer = 30 sec}
from Monitoring goto Emergency when fire
from Emergency goto Monitoring when check
start Monitoring
```

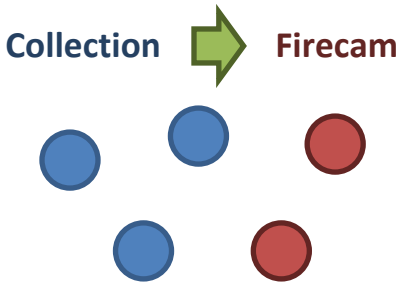
2. Protocol Stack

- layered system design of application and communication protocols
- starts and stops modules running across the layers



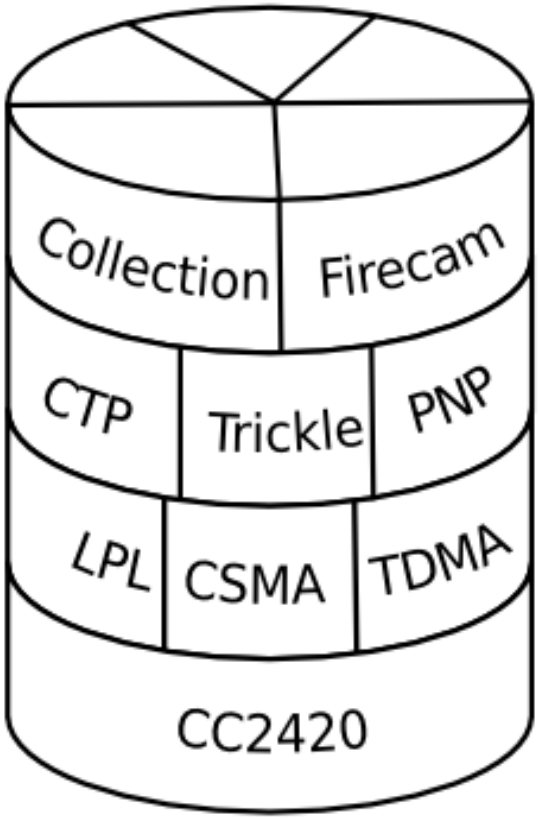
3. Network Reconfiguration

- executes application context switch together with communication protocols
- ensures that all nodes run the same modules across the layers of the protocol stack



Fennec Fox Protocol Stack

- four layer protocol stack
- application, network, MAC, and radio layers
- each layer contains multiple modules implementing its functionality
- Fennec Fox schedules the execution of the modules across the stack

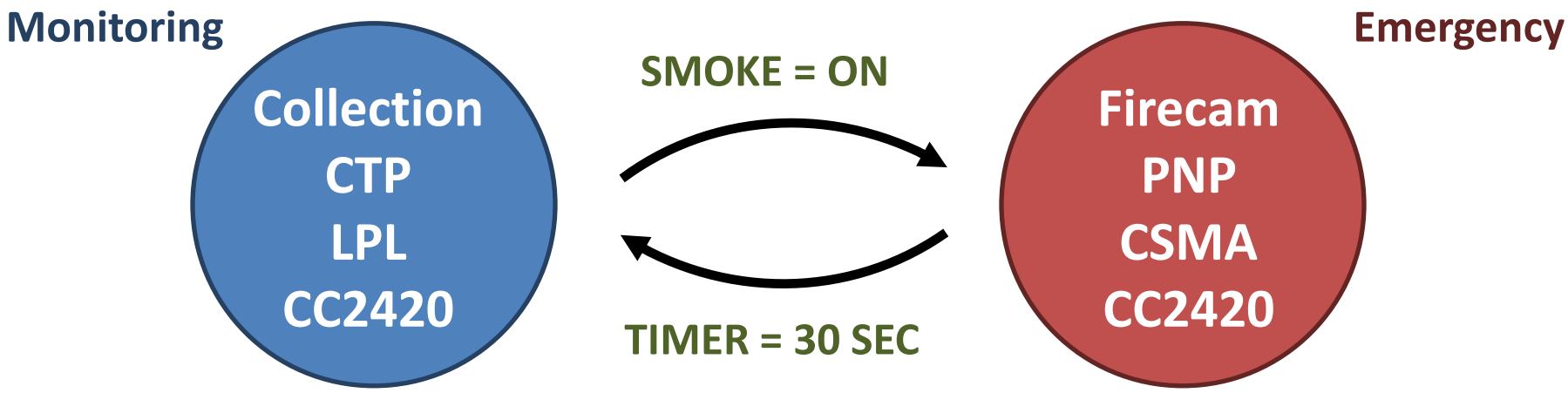


- builds upon existing research work
- runs on top of TinyOS, implemented in nesC
- modules must obey inter-layer API
- reuse and combine already existing protocols
- modular design simplifies programming and validation

Protocol Stack Configuration

Set of four modules executing on the four-layer stack, one module for each layer. Each configuration has globally unique **id**

Swift Fox Programming Language

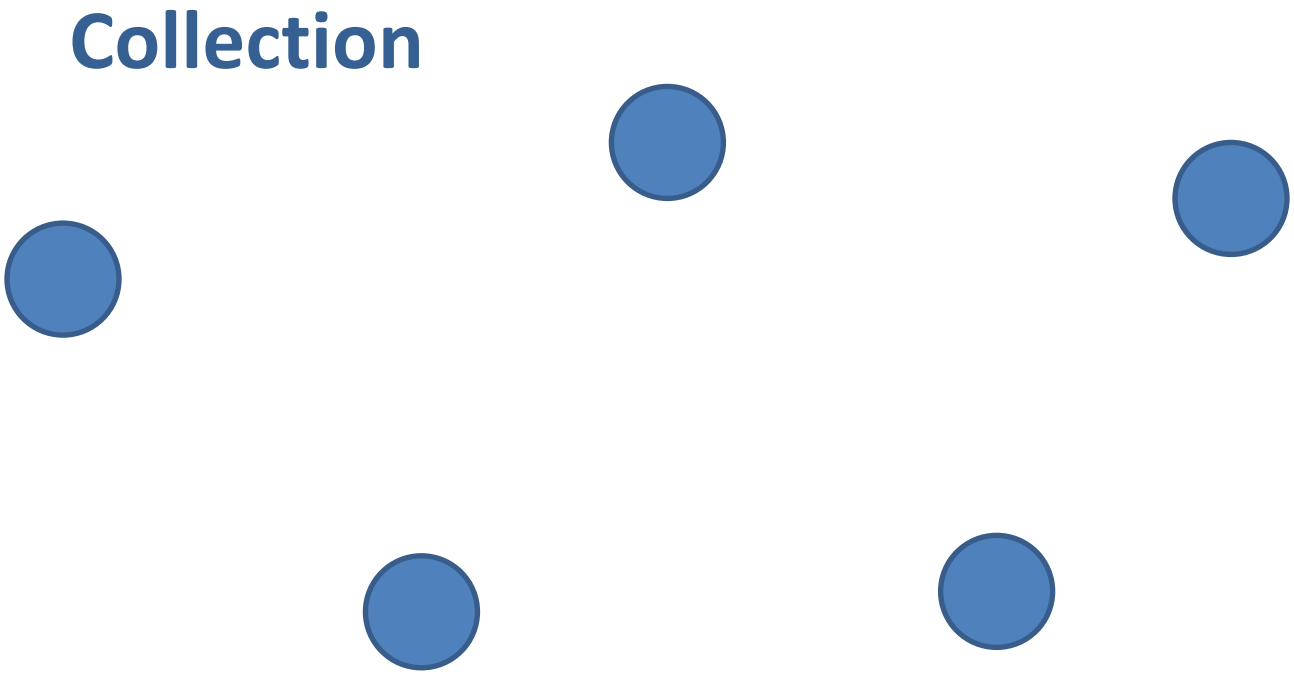


- decouple the network reconfiguration from the application implementation
- model network behavior as **Finite State Machine**
- at *design time* a programmer specifies the FSM of the network reconfiguration logic
- programs are compiled to a system that **at *run-time* automatically reconfigures the network**

```
configuration Monitoring { collection(1024, NODE, 107)
                           ctp(107)  lpl (100, 10)   cc2420(1)}
configuration Emergency {firecam(28)   pnp()
                         csma(0,0)   cc2420(0)}
event fire {smoke = ON }
event check {timer = 30 sec}
from Monitoring goto Emergency when fire
from Emergency goto Monitoring when check
start Monitoring
```

Network Reconfiguration

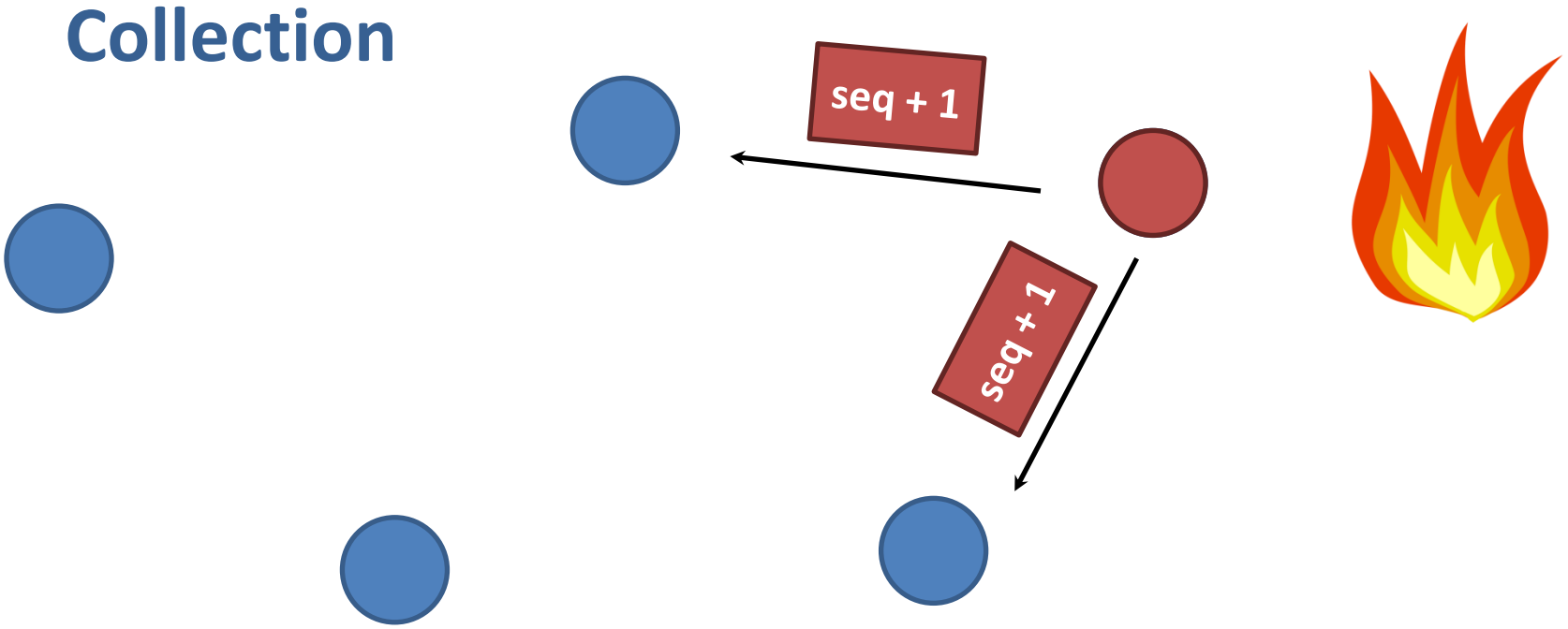
- network is reconfigured by disseminating **control messages**
- each message contains a **sequence number** and the **configuration id**
- control messages are disseminated by following modified Trickle protocol
- new sequence number triggers reconfiguration, but also checks payload consistency



- network is **unsynchronized** when there exist control messages with the same sequence number but different configuration ids
- detects network state inconsistency and synchronizes all nodes

Network Reconfiguration

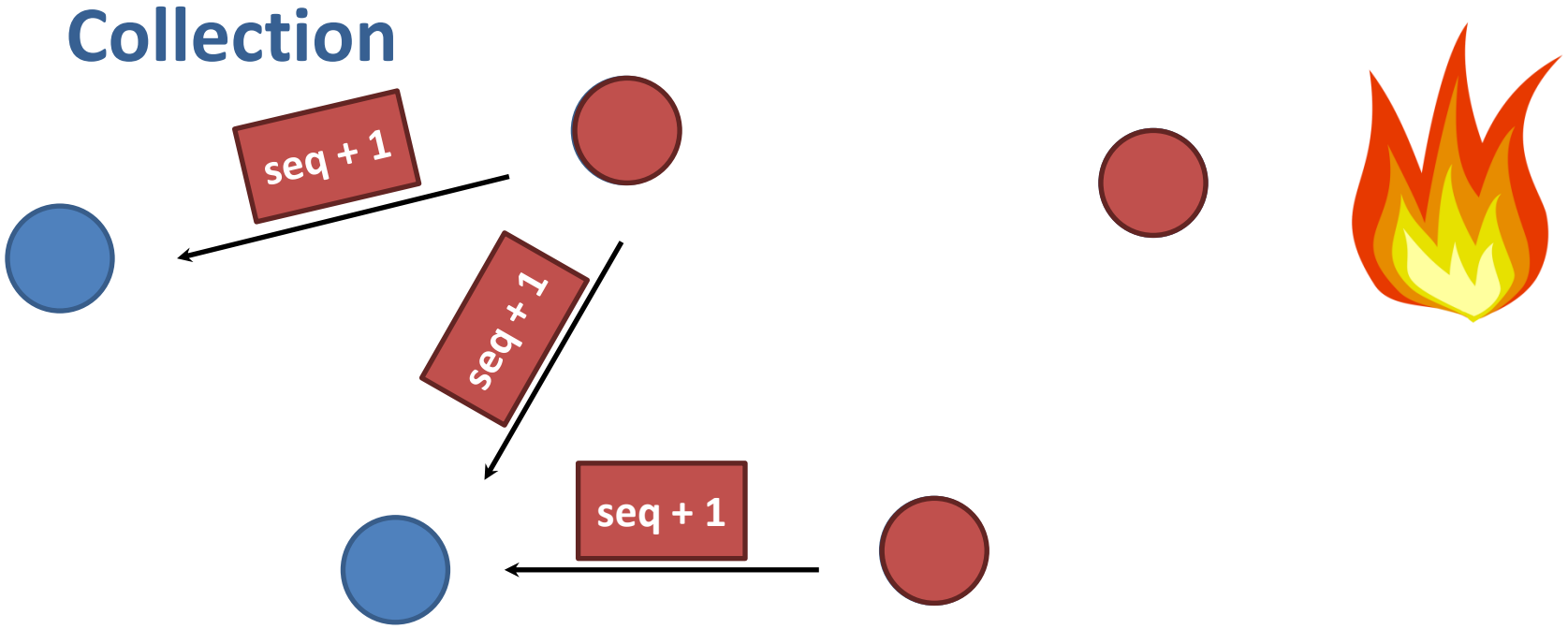
- network is reconfigured by disseminating **control messages**
- each message contains a **sequence number** and the **configuration id**
- control messages are disseminated by following modified Trickle protocol
- new sequence number triggers reconfiguration, but also checks payload consistency



- network is **unsynchronized** when there exist control messages with the same sequence number but different configuration ids
- detects network state inconsistency and synchronizes all nodes

Network Reconfiguration

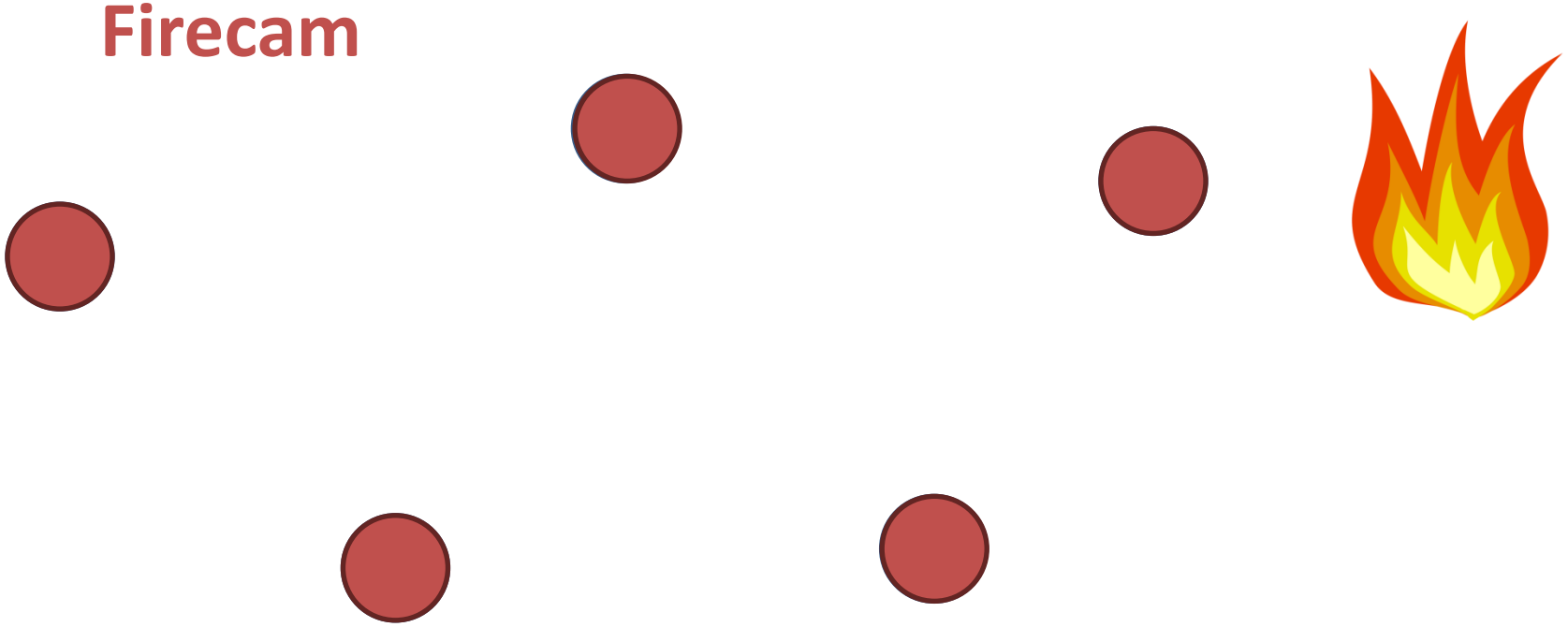
- network is reconfigured by disseminating **control messages**
- each message contains a **sequence number** and the **configuration id**
- control messages are disseminated by following modified Trickle protocol
- new sequence number triggers reconfiguration, but also checks payload consistency



- network is **unsynchronized** when there exist control messages with the same sequence number but different configuration ids
- detects network state inconsistency and synchronizes all nodes

Network Reconfiguration

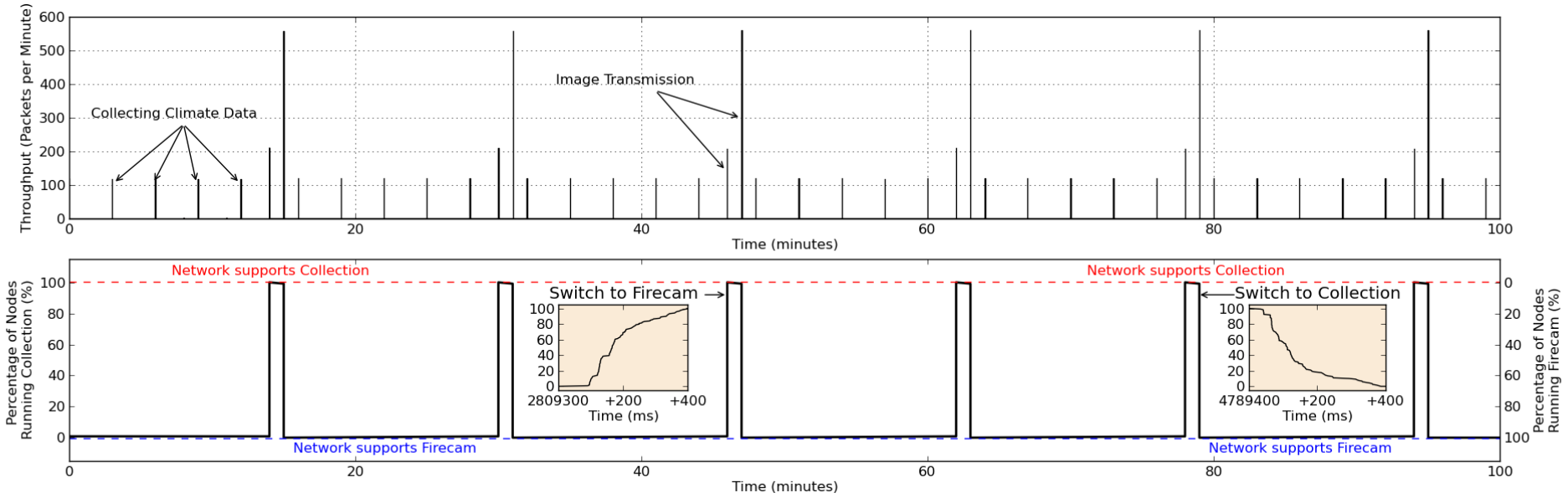
- network is reconfigured by disseminating **control messages**
- each message contains a **sequence number** and the **configuration id**
- control messages are disseminated by following modified Trickle protocol
- new sequence number triggers reconfiguration, but also checks payload consistency



- network is **unsynchronized** when there exist control messages with the same sequence number but different configuration ids
- detects network state inconsistency and synchronizes all nodes

Evaluation: Context Switch Robustness

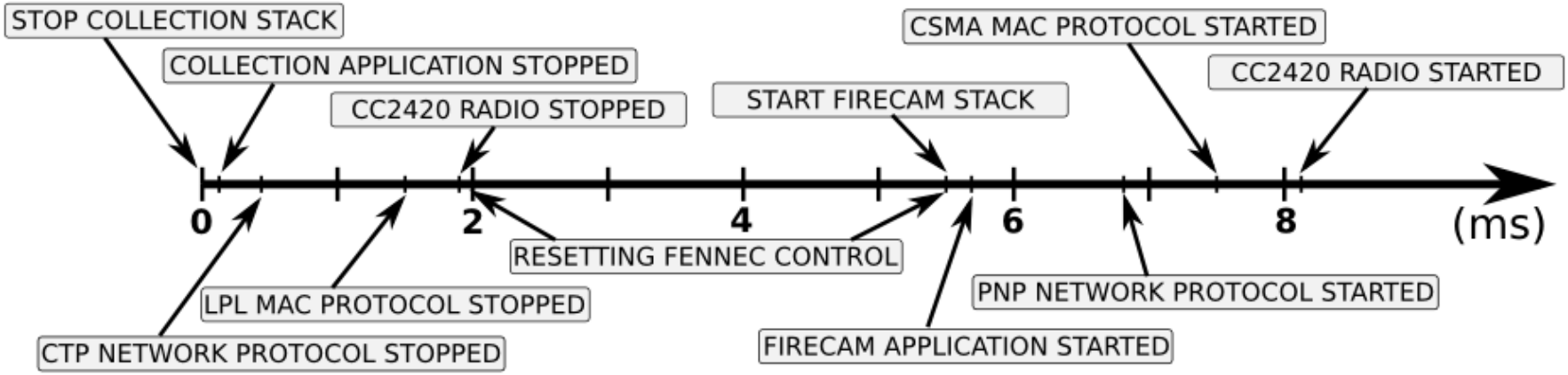
100 minute run of a network reconfiguring between the Collection and Firecam



- on average, 99.91% of network reconfigurations are successful
- 99.5% of nodes reconfigure successfully when radio is duty-cycled
- **network can reconfigure as fast as every 350ms**, with 99.68% nodes successfully following all reconfigurations

setup: Indriya testbed, a three-floor building at the NUS instrumented with 139 TelosB motes

Evaluation: Reconfiguration Overhead



Protocol stack reconfiguration from Collection to Firecam

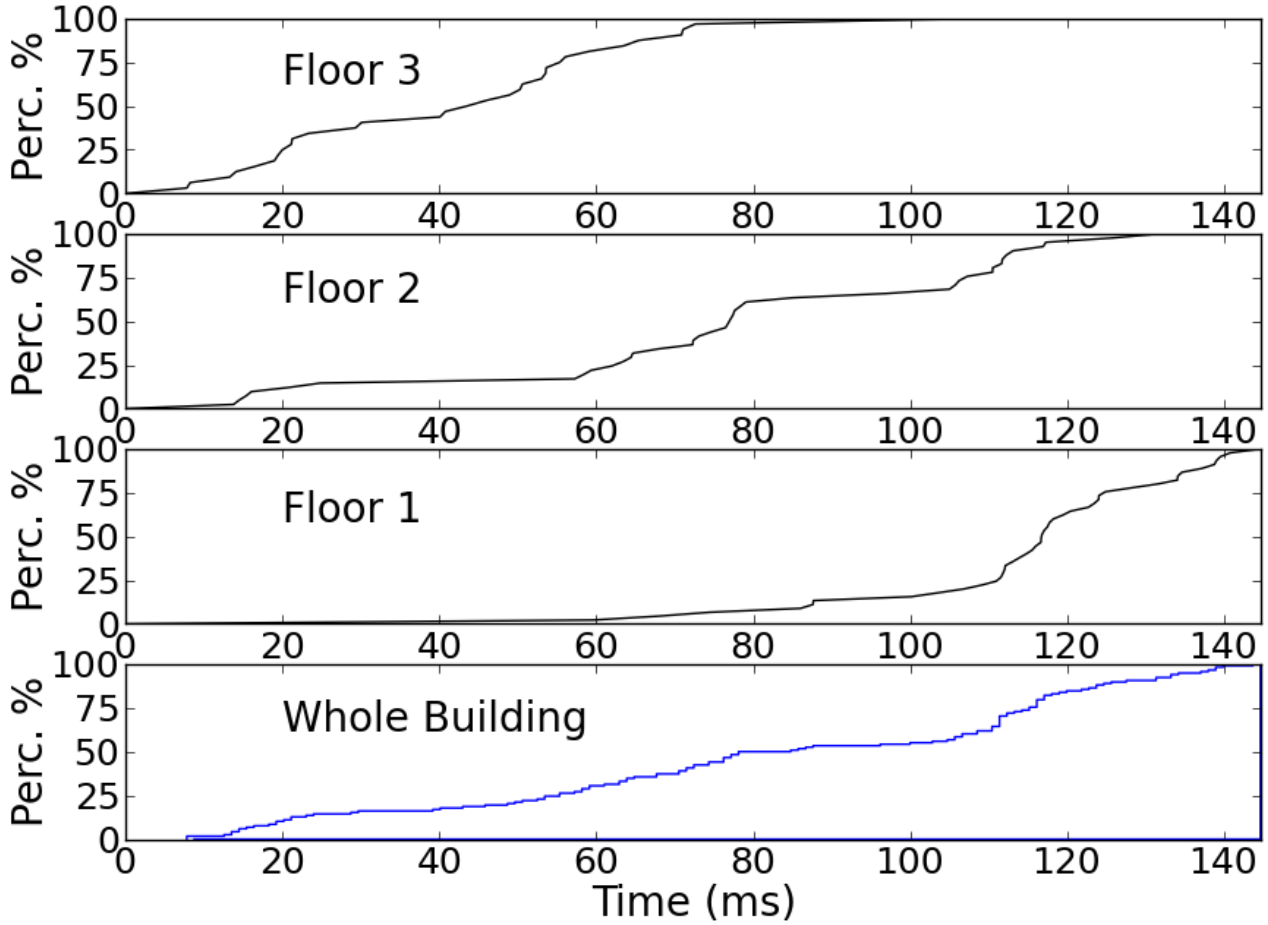
Delay:

- to switch from Collection to Firecam takes 8.125ms
- to reset the radio itself takes almost 3ms
- the whole stack reconfiguration takes between 8 and 9ms.

Memory:

- 4.7 KB of ROM and 0.2KB of RAM
- Collection and Firecam require 28.9 KB of ROM and 5.6 KB of RAM

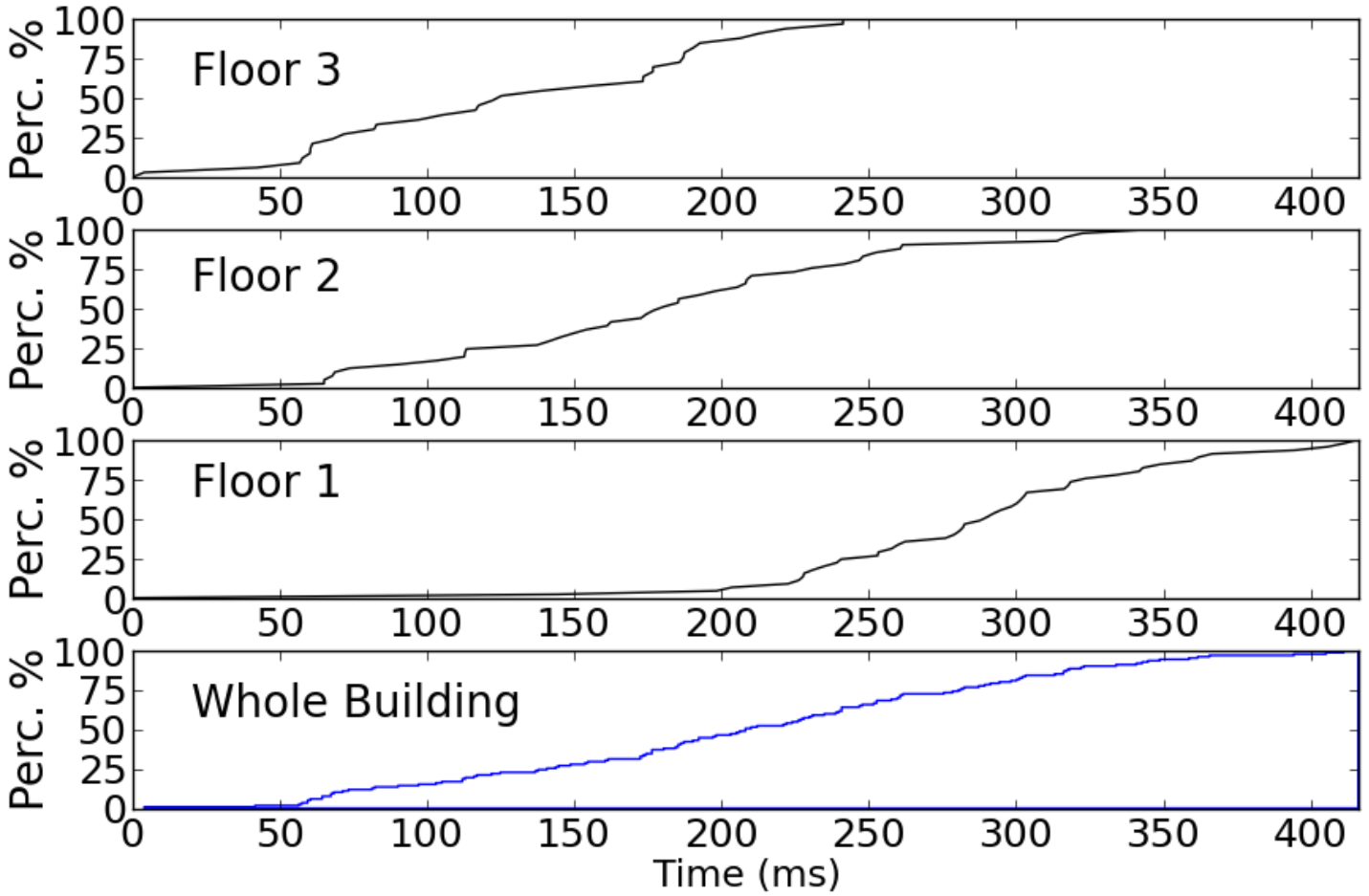
Evaluation: Reconfiguration Across a Building



Reconfiguration with radio always turned on

- reconfigurations occur in bursts
- control message broadcast quickly propagates on the same floor
- network reconfiguration is delayed on neighboring floors

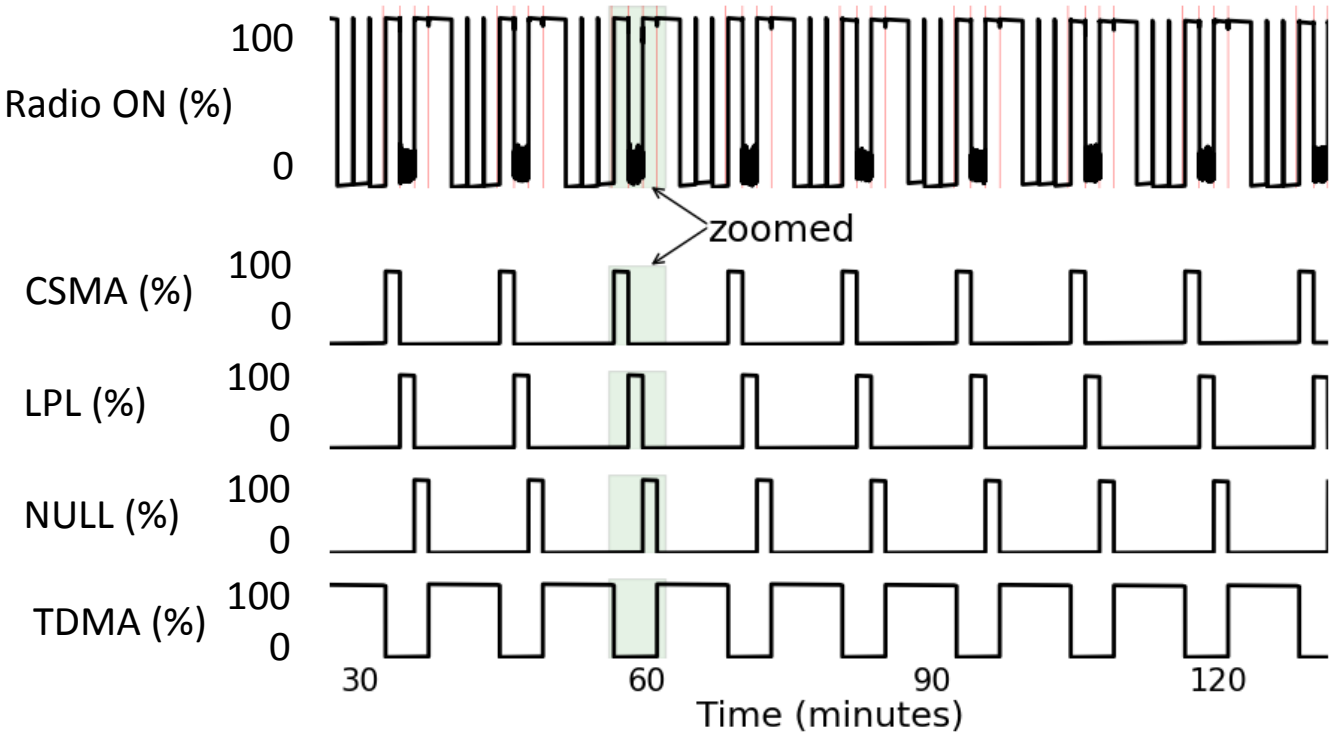
Evaluation: Reconfiguration Across a Building



Reconfiguration with radio duty-cycling

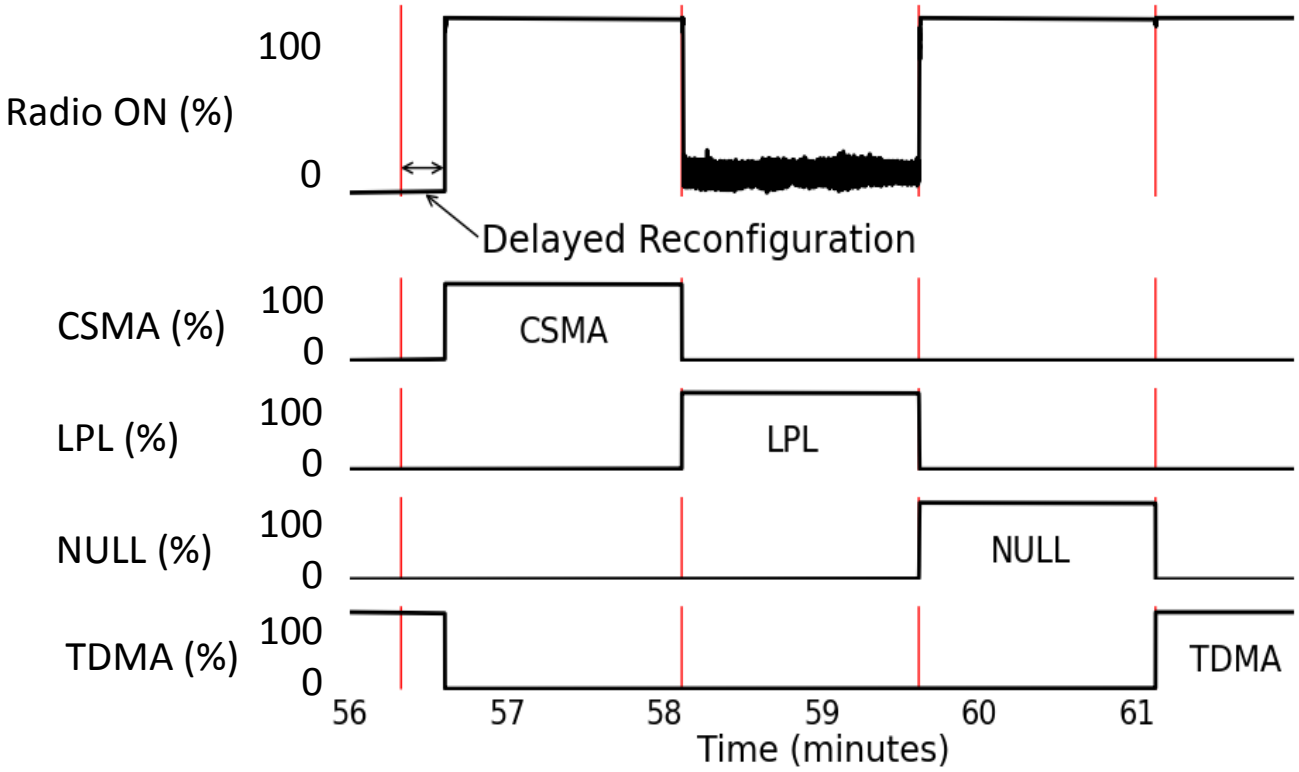
- network reconfigures progressively
- need more control message transmission attempts for smaller duty-cycle times
- 33-40% of control message transmissions attempts are suppressed

Evaluation: Reconfiguration Among Various MACs



- reconfigure a WSN among the four MAC protocols:
  - CSMA
  - LPL (unsynchronized duty-cycling)
  - Null (simple MAC that passes packet between the Network and Radio layers)
  - TDMA (synchronized duty-cycling)
- without Fennec Fox these MAC protocols cannot co-exist in the same WSN

Evaluation: Reconfiguration Among Various MACs



- Fennec Fox stack monitors the status of the radio to guess the state of the network
- when the radio is turned-off, the reconfiguration is delayed until radio is on
- to switch between two configurations with TDMA and with the same duty-cycle period, it requires to insert between these configurations an empty state with CSMA or LPL, to ensure that all motes successfully reconfigure

Conclusion

Conclusion:

- studied the problem of scheduling execution of heterogeneous applications with different communication requirements on a single WSN
- developed Fennec Fox, a runtime infrastructure build around a layered protocol stack that executes network-wide application context-switch
- showed the feasibility of Fennec Fox through experimental evaluation on a network deployed across the three-floor building

Open Source Code:

- Fennec Fox: https://github.com/mszczodrak/fennecfox
- Swift Fox: https://github.com/mszczodrak/swiftfox

Thank You,

Marcin Szczodrak - msz@cs.columbia.edu