Starfish: Policy Driven Self-Management in Wireless Sensor Networks

THEMIS BOURDENAS, MORRIS SLOMAN
Agenda

• Wireless Sensor Networks

• Fault Management

• Starfish platform

• Conclusions
Wireless Sensor Networks

- feedback component in pervasive computing
- broad range of application requirements
  - *setup* – small (body) to vast (landscape) networks
  - *accessibility* – easily to impossible to retrieve nodes
  - *power* – usually battery powered
  - *capabilities* – very limited CPU/memory

![Diagram of wireless sensor network](image)

- Sensor Node
- Gateway
- Sensor Node
Motivation

• faults in WSNs are not exceptional cases
  • inaccurate sensors
  • conflicting / missing readings

• causes
  • sensitive / inexpensive electronics
  • exposed to physical world / users
  • low battery levels
  • circuitry fouling

• impractical management even for small networks
Motivation

- fault examples
  - spikes, noise, drift, missing data
WSN Architecture

- high level architecture
  - functional roles
  - deployment agnostic
- attaching self-healing services
  - in different levels and scope
Self-healing Framework

detection algorithms + recovery techniques

algorithms + framework
Starfish Platform Overview

• Finger2 embedded policy system
  • mote middleware

• Starfish Module Library
  • common programming facilities

• Starfish Editor
  • authoring and management tool
Roles, Missions and Modules
Finger2 Policies

def <authpolicy>[+/-]
    subject <role>
    target <role>
    if <condition>
    action <name>

def <obligpolicy>
    on <event>
    if <condition>
    do <action>
Finger2 Policies

```python
def allow_nurse_policy_install +
    subject nurse
    target patient
    if power.Level() > .2
        and nurse.type is staff_nurse
    action policy.Install

def ECG_request
    on gui.RequestUpdate(patient, type)
    if network.IsAvail(patient) and type is ECG
    do patient.policy.Install(ECG_update)

def ECG_update
    on sensor.Reading(type, value)
    if type is ECG
    do network.Send(nurse, value, timer.Now())
```

```python
def <authpolicy>[+/-]
    subject <role>
    target <role>
    if <condition>
    action <name>

def <obligpolicy>
    on <event>
    if <condition>
    do <action>
```
Finger2 Architecture
Starfish Module Library

- sensing
  - sync / async reads from sensors
- buffering
  - storage on motes
- feature extraction
  - statistical functions data preprocessing
- timers
  - periodic scheduling, notifications
- communication
  - network and serial port communication primitives
- policies / missions / roles
  - installation and dynamic management of roles and policies
interface EventSourceI {
  event void evt(event_t eid, argument_t* args);
}

interface PredicateI {
  command int evaluate(argument_t* args);
}

interface ActionI {
  command int perform(argument_t* args);
}

def allow_nurse_policy_install +
  subject nurse
  target patient
  if power.Level() and nurse.type is staff_nurse
  action policy.Install

def ECG_request
  on gui.RequestUpdate(patient, type)
  if network.IsAvail(patient) and type is ECG
  do patient.policy.Install(ECG_update)
Policy Authoring Tool
Fault Detection

- **OK**: normal local variance
- **IRREG**: high variance ratio
- **FAULTY**: normal variance ratio
- **high local variance**

Diagram shows transitions between states based on variance criteria.
Fault Detection

```python
def CalcVariance
    on sensor.Reading(type, value)
    if type in [TMPR, HUMIDITY]
    do feature.Var (type, buffer.Get(type))

def OkIrregTransition
    on feature.VarRatioRet(type, variance)
    if variance > VAR_TH
    do mission.Swap(Irreg, Ok),
       feature.VarRatio(type)

def IrregFaultyTransition
    on feature.VarRatioRet(type, variance)
    if variance < VAR_TH
    do mission.Swap(Faulty, Irreg)

def FaultyOkTransition
    on feature.VarRatioRet(type, variance)
    if variance < VAR_TH
    do mission.Swap(Ok, Faulty)
```

**Diagram:**
- OK to IRREG
- IRREG to FAULTY
- FAULTY to OK
- OK to FAULTY
- FAULTY to IRREG

-高局部变异（high local variance）
-正常局部变异（normal local variance）
-高变异比率（high variance ratio）
-正常变异比率（normal variance ratio）
Finger2 Overheads

• memory
  • ROM: 12.23KB
  • RAM: 0.72KB
    • policies min 24bytes, expected average 50bytes
  • typical motes: ROM: 48-128KB, RAM: 4-16KB

• processing
  • 69μsec overhead on average for policy execution
    • does not account for application code execution (i.e. predicates and action evaluation)
Future Work

• specialize in self-healing strategies / constructs
  • Bayesian Classifiers for fault detection
  • translate FSM to policies

• integration with ponder2 and Self-Managed Cell
  • building self-healing services on the SMC architecture
  • define interactions between collaborating SMCs
    • low level SMC (cluster of sensor nodes)
    • high level SMC (PDA/smartphone based)
Questions