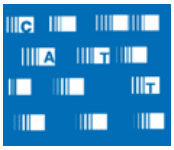


Sensor Networks

Part 3: TinyOS

CATT Short Course, March 11, 2005

Mark Coates

Mike Rabbat

1

Operating Systems 101

operating system ***(äp'ər āt'ing sis'təm)*** ***n.*** 1 software that controls the operation of a computer and directs the processing of programs by assigning storage space in memory and controlling input and output functions [miriam-webster.com]

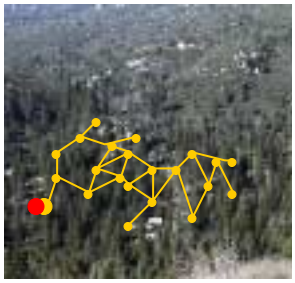
Key Ingredients:

1. Hide low-level machine operations
2. Provide common services
3. Manage tasks
4. Maintain system integrity

Part 3: 2

Sensor Network Characteristics

1. Small, simple, cheap
2. Concurrent operation
3. Flexible, efficient usage
4. Robust design



Part 3: 3

Something Completely Different

Existing embedded RTOSes

1. Too bulky or too constrained
2. Control-centric
3. Task-based

The answer…

Part 3: 4

Design Goals

1. Take account of current and likely designs for nodes and networks
2. Be flexible to allow for a diverse set of implementations and applications and for varying mixes of hardware and software
3. Address WSN-specific challenges

The TinyOS Approach:

- Event-driven concurrency model
- Efficient modular framework

Outline

- Background
- The TinyOS Architecture
  - Architecture overview
  - Modules and configurations
  - Managing concurrency
  - Clustering example
- Communication Services

TinyOS Overview

- Event-driven concurrency model
  - Efficient decentralized processing for networked systems
  - "Micro-threaded" architecture without hardware parallelism
- Efficient modular framework
  - Easily abstract hardware
  - Reuse common software modules by maintaining consistent interfaces
- The TinyOS Philosophy
  - No user/system boundaries
  - No specific set of system services to drag around
  - Just defines a framework for establishing boundaries
  - Provides a list of services apps can choose from in addition to providing facilities for designing/building new services
  - Let the list of "commonly used services" evolve on its own
- Core components
  - Processor/hardware initializaiton
  - Scheduler to manage tasks
  - RunTime to facilitate tasks & event interrupts, provide robustness
  - Total: less than 500 bytes of default code and storage!

The Berkeley Mote

2000

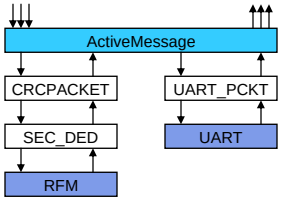
- 4MHz 8 bit MCU
- 512 bytes RAM, 8K ROM
- 900 MHz radio (10-100ft.)
- Temp. & light sensors
- LED and serial port

2005

- 8MHz 8 bit MCU
- 128K + 512K RAM, 4K ROM
- ZigBee radio
- Temp, light, barometric, bio, pressure,… sensors
- 3 LEDs, serial, extension

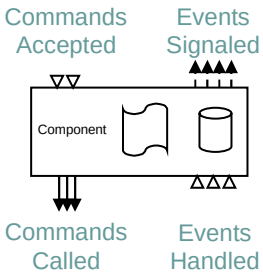
TinyOS Architecture

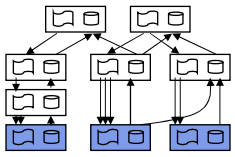
- Event-Based Scheduling
  - Events triggered by hardware
  - Tasks posted to queue
- Component Model
  - Self-containing code unit
  - Interfaces
  - Tasks
  - Storage



An Efficient Modular Framework

- Component modules
  - Self-containing units
  - Reusable code
  - Abstract hardware
- Configuration
  - "Wiring" modules together
  - Specifies a solution





Example Interfaces

```
interface StdControl { // booting & power management
    command result_t init();
    command result_t start();
    command result_t stop();
}

interface ADC { // data collection
    command result_t getData();
    command result_t getContinuousData();
    event result_t dataReady(uint16_t data);
}
```

Part 3: 11

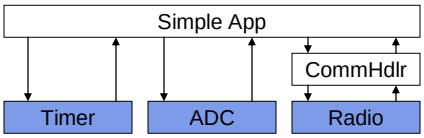
Event-Driven Concurrency

- Events
  - Triggered by hardware
  - Run to completion, never preempted
- Tasks
  - Posted to a queue
  - Run to completion, may be preempted by events
  - Can sleep when queue is empty

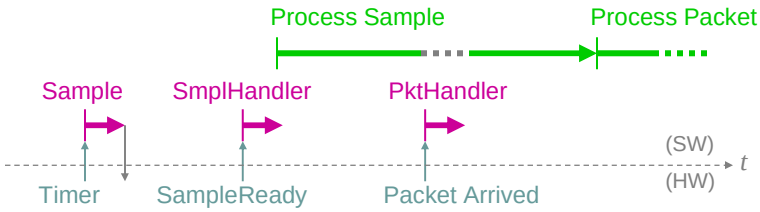
Part 3: 12

Concurrency Example

Example Config:

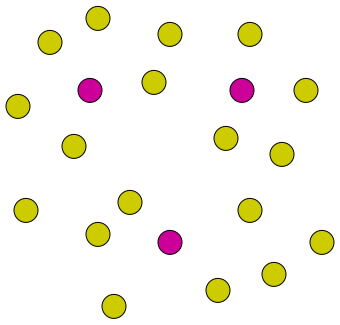


Event Cycle:



Forming Networked Clusters

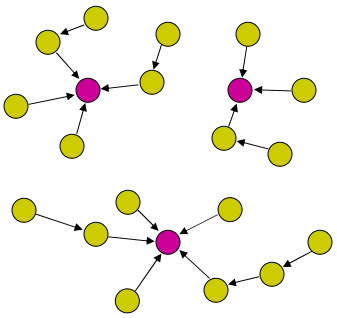
- Commonly assumed networking setup
- Randomly chosen cluster heads broadcast
- Nodes join group of nearest cluster head



(Thanks to Frederic Thouin)

Forming Networked Clusters

- Commonly assumed networking setup
- Randomly chosen cluster heads broadcast
- Nodes join group of nearest cluster head

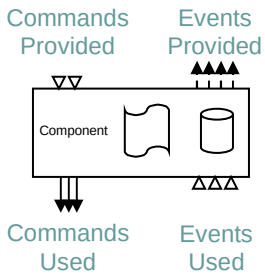


(Thanks to Frederic Thouin)

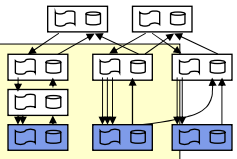
Clustering Module

```
module ClusteringM {
    provides {
            interface StdControl;
    }
    uses {
            interface Timer as TimerA;
            interface Timer as TimerB;
            interface Timer as TimerC;
            interface ReceiveMsg;
            interface SendMsg;
            interface Random;
    }
} implementation {
    // State space allocation
    ...

    // Command implementations
    task void advertisement() {
            ...
            call SendMsg.send(...);
    }
    ...
}
```



Clustering Configuration



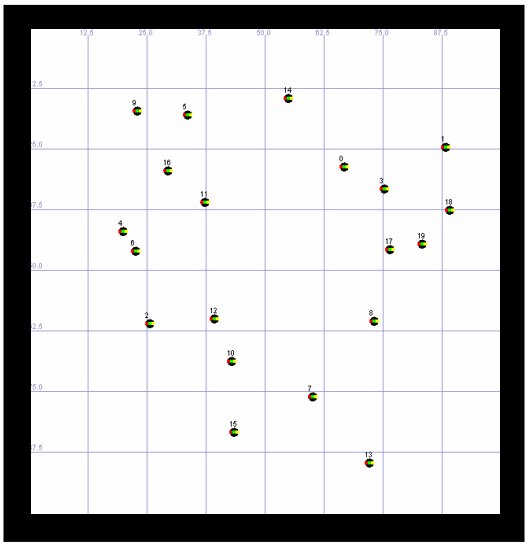
```
configuration Clustering {
} implementation {
   components Main, ClusteringM, TimerC,
        GenericComm as Comm,
        RandomLFSR as Random;

   Main.StdControl -> Comm;
   Main.StdControl -> TimerC;
   Main.StdControl -> ClusteringM;

   ClusteringM.TimerA -> TimerC.Timer[unique("Timer")];
   ...
}
```

Part 3: 17

Clustering In Action



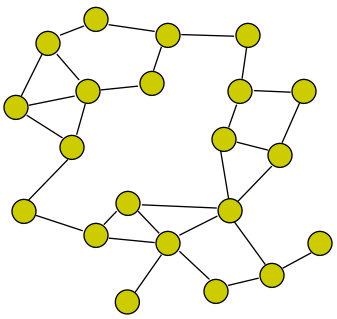
Part 3: 18

Outline

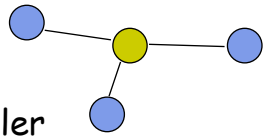
- Background
- The TinyOS Architecture
- Communication Services
  - Neighbor-to-neighbor
  - Multi-hop
  - Synchronization
  - Power savings strategies

Communication Overview

- Single hop communication
  - Active Messages
  - Implementation issues
- Multi-hop communication
  - Tree-based routing
  - Epidemic protocols
- Related services
  - Synchronization
  - Saving energy



Active Messages



- Integrating communication and computation
- Message contains data and handler
- Ideal for data-centric applications

```
interface SendMsg{
    command result_t send(uint16_t addr, uint8_t len, TOS_MsgPtr msg);
    event result_t sendDone(TOS_MsgPtr msg, result_t success);
}

interface ReceiveMsg{
    event TOS_MsgPtr receive(TOS_MsgPtr m);
}
```

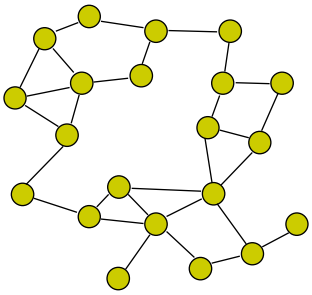
Part 3: 21

Active Message Implementations

- Variations with new HW generations
- More functionality in HW
  - Free microprocessor cycles
  - Hooks
- S-MAC adds RTS/CTS
- ZigBee, the stabilizer?

Part 3: 22

Multi-Hop Routing



- "The network is the sensor"
- Key elements
  - Link quality estimation
  - Neighborhood discovery
- Tree-based routing
- Broadcast/Epidemic Protocols
- Point-to-point communication not common

Link Quality Estimation

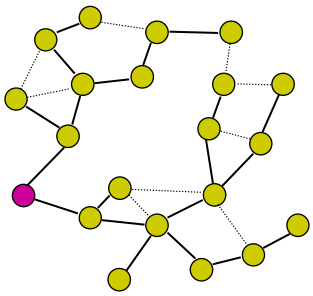
- Track observed/unobserved packets using *ACKs*
- Received signal strength varies wildly over time, environmental conditions, bad indicator
- Broadcast nature of transmission allows "snooping"
- Listening uses energy ➔ Low-power listening
- No implicit loss indicator (e.g., sequence number)
- Instead, assume average transmission rate
- Smooth using Exponentially Weighted Moving Average

Neighborhood Management

- Discovery
  - Passive "snooping"
  - Actively probing (e.g., with beacons)
- Maintenance
  - Use link quality estimates to identify good candidates
  - Challenge: Deciding which entries to maintain in a dense or mobile network
- Simple example policy
  - Insert new neighbors based on received signal strength
  - Periodically down-sample at random
  - Reinforce "good" neighbors

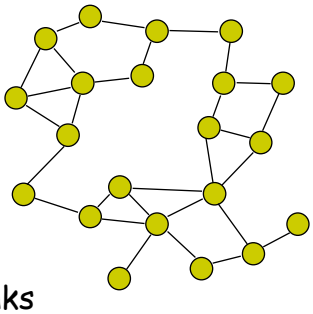
Tree-Based Routing

- Prevelant applications
  - Habitat monitoring
  - TinyDB
  - Aggregation



- Broadcast to form tree
- Maintain list of neighbors
- Choose best candidate as parent

Epidemic Protocols



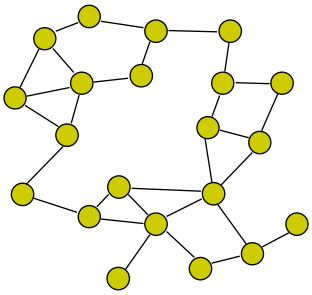
- Dissemination via local broadcasts
- Single *flood* reaches many nodes quickly
- Problem: Collisions and lossy links prevent all nodes from receiving the message
- Solution: Random local broadcasts

Trees –vs- Epidemics

- Application requirements
- Complexity (maintenance, especially)
- Scaling with network size
- Bottlenecks
  - Communication (capacity)
  - Security
- Energy usage distribution
- Other emerging multi-hop paradigms

Point-To-Point Approaches

- Major difference between WSNs and the Internet



- Directed Diffusion
  - Grow and then prune a tree
  - Difficult to maintain
- Geographic approaches
  - Nodes addressed based on geography
  - Requires GPS or localization
  - Perimeter problems (holes in the network)

More Recent Trends

- Snooping
  - Build up info about neighbors, link quality based on observed packets, not necessarily destined for you
  - Used to weed out uni-directional links
- Send Queues
  - Newer hardware has more memory
  - Enables more sophisticated multi-hop communication
- Secure Communication

Time Synchronization

- Problems with NTP in the Internet
  - Beacons synchronized using GPS
  - Adjust clock rate based on beacon signals
  - Result: Very irregular, unreliable behavior
- Hardware hooks for fast retransmission
- Cross-layer design
  - Synchronization integrated with applications
  - E.g., Modify sleep time to compensate
- Goal: Reliable, robust coarse (15-30 μs) synchronization

Power-Savings

- When awake, listening eats up the most power
- Typical application, 1000ms cycle
  - Transmit 1 packet for 50 ms → 600 μJ
  - Receive 1 packet for 50 ms → 250 μJ
  - Listen for 900ms → 4500 μJ!!
- Low-power listening
- Coordinated sleep-wake cycles

Low-Power Listening

- Sleeping saves energy, but might miss something
- Periodic listening:
  - Sleep 90 out of 100ms → 90% energy savings
  - Transmitter must broadcast for 100ms to be sure it reaches an awake receiver
- Low-power listening
  - Sleep 30 out of 300μs → 90% energy savings
  - Transmitter only need to broadcast for 300μs
  - Much less overhead
- Enabled by direct processor control of radio

Part 3: 33

HW/SW Boundary Tradeoffs

Trend: Move more complexity into hardware

More cycles for processing
-vs-
Slower control of hardware devices

Coordinated Sleep/Wake Cycles

- New radios don't transition as fast
- But hooks allow for more precise synchronization
- Synchronize sleep/wake cycles
  - Forward packets when other nodes are awake
  - Efficient TDMA-style system reduces interference
  - Requires robust, reliable clock synchronization
- Main application: Habitat Monitoring
  - Regularly schedule sampling
  - Not latency-sensitive

In Summary

- Modular, event-based ➔ Flexible, expandable
- Active Messages ➔ Integrated comm. and computation
- Variety of other comm-related services evolving

- Additional facilities:
  - TOSSIM
  - Matlab interaction

- Other platforms:
  - MANTIS
  - EmStar

Trickle & Maté

- For propagating code updates reliably through a WSN

Part 3: 37