Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS Programming

Mehrzad Lavassani

Distributed Systems I – DT017A, Fall 2017

Mittuniversitetet
MID SWEDEN UNIVERSITY

Introduction

- Purpose:
  - To provide a better understanding about distributed systems and programming
- Goals:
  - Getting familiar and program wireless sensor nodes (motes).
  - Simulating communication in small WSNs.
  - Demonstrating knowledge and understanding about distributed communication and algorithms
    - group communication
    - Mutual Exclusion
    - Ordering

Mittuniversitetet
MID SWEDEN UNIVERSITY

Introduction (cont.)

- Lab work is preformed in pairs
- Two approaches
  - HIGHLY recommended
    - TinyOS, nesC and TOSSIM
    - Three lab assignments
    - Group communication, ME, and ISIS algorithm
  - If you insist!
    - Java
    - One project
    - Group communication, total and causal order

Mittuniversitetet
MID SWEDEN UNIVERSITY

Introduction (cont.)

- Deadlines and lab assignments

Sep 18	Introduction to TinyOS/nesC/TOSSIM, Assignment I
Oct 5	Assignment II, delivering assignment I
Oct 10	Assignment II/III, delivering assignment II
Oct 17	Assignment II/III, delivering assignment II
Oct 27	Delivering assignment III

- Deadline for final report 20th November 2017

Mittuniversitetet
MID SWEDEN UNIVERSITY

Introduction to TinyOS and nesC

Mittuniversitetet
MID SWEDEN UNIVERSITY

What is TinyOS?

- Open-source operating system for wireless embedded sensor networks
- Event-driven and component-based architecture
- A set of software **components** that can be **wired** together into a single binary that runs on the motes
- TinyOS and its applications are written in nesC (network embedded system C), a component-based C dialect

Mittuniversitetet
MID SWEDEN UNIVERSITY

Why TinyOS?

- Lightweight OS
- Aggressive system and mechanism for saving power and memory
- Makes our lives easier!
  - Set of services and abstractions
  - Concurrent execution model

  ➔ build applications out of reusable services and components

- Generic platforms support, and easy to port to new platforms

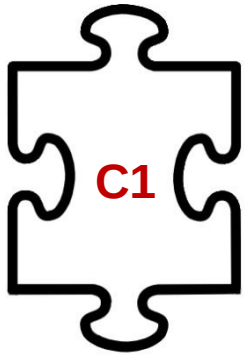
Mittuniversitetet
MID SWEDEN UNIVERSITY

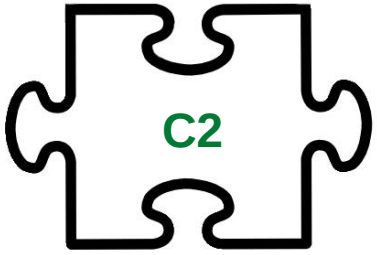
Terminology

- **Application** – one or more **components** wired together
- **Components** – Basic building blocks for nesC applications; **modules** and **configurations**
- **Module** – components that implements one or more **interfaces**
  - (implements application code)
- **Configuration** – components that wires other components together
- **Interface** – provides an abstract definition of the interaction between two components (sets of commands, bidirectional)

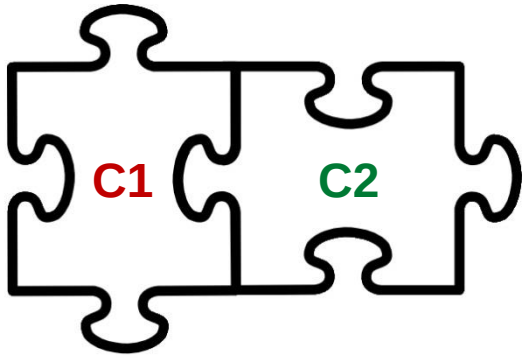
TinyOS – nesC components

```
module C1{
   provides {
      interface  ... ;
  } uses {
      interface ... ; }
}
implementation { ... };
module C2{
   provides {
      interface  ... ;
  } uses {
      interface ... ; }
}
implementation { ... };
```






```
configuration app {}
implementation {
    components C1, C2;
    C1 -> C2;
}
```



TinyOS – Hardware

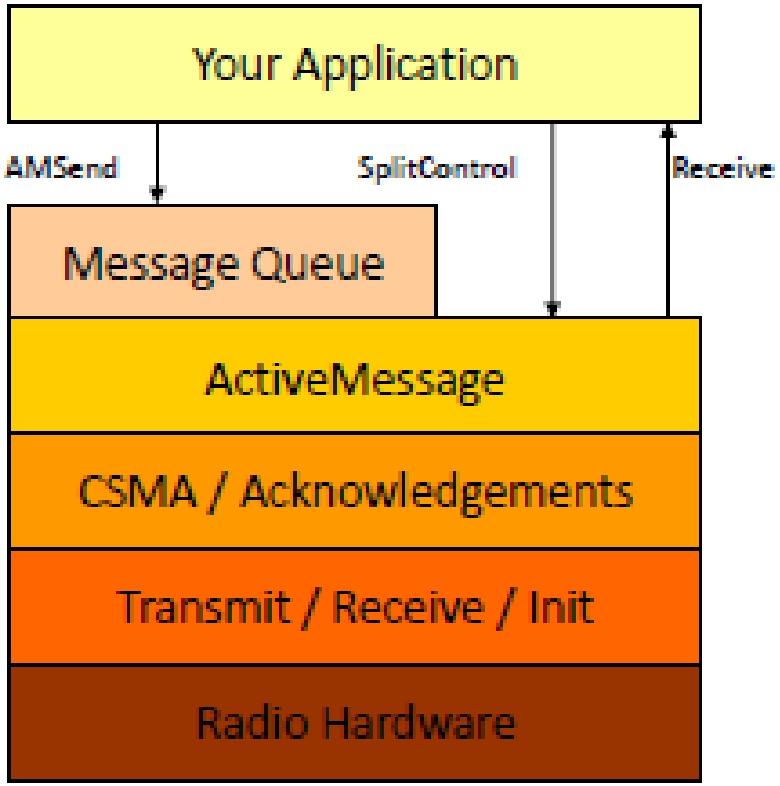
- MicaZ

- TelosB

2400-2483.5 MHz
IEEE 802.15.4 Radio Transceiver

TinyOS – Radio Stack

- Many layers sitting between the application and hardware.
- Highest level: data and header modification in each packet.
- Lowest level: actual send and receive behavior determination.
- The stack can be extended or condensed to meet application requirements.



Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS – Single-hop radio communication

- **Active Messages (AM)**
  - The lowest networking layer
  - Implemented indirectly over a mote's radio
  - Provides unreliable, single-hop packet transmission and reception
  - High level protocols can be built over AM.
  - 8-bit integer identifies the packet type.

Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS – Single-hop radio communication (cont.)

- **Main radio interfaces**
  - SplitControl
    - Provided by **ActiveMessageC**
    - Used to start and stop the communication stack.
  - AMSend
    - Provided by **AMSenderC**
    - The virtualized AM send abstraction
  - Receive
    - Provided by **AMReceiverC**
    - The virtualized AM reception abstraction

Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS – Single-hop radio communication (cont.)

- SplitControl

```
event void boot.booted(){
        call SplitControl.start();
}

event void SplitControl.startDone(error_t error) {
        post sendMsg();
}

event void SplitControl.stopDone(error_t error){}
```

Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS – Single-hop radio communication (cont.)

- AMSend

```
message_t myMsg;

task void sendMsg(){
        if(call AMSend.send(AM_BROADCAST_ADDR,
        &myMsg, 0) != SUCCESS) {
                post sendMsg();
        }
}
event void AMSend.sendDone(message_t *msg,
error_t error) {
        post sendMsg;
}
```

Mittuniversitetet
MID SWEDEN UNIVERSITY

TinyOS – Single-hop radio communication (cont.)

- Receive

```
event message_t *Receive.receive(message_t *msg,
   void*payload, uint8_t length) {
       call Leds.led0Toggle();
       return msg;
}
```

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM Simulator

Mittuniversitetet
MID SWEDEN UNIVERSITY

What is TOSSIM?

- TinyOS discrete event simulator (emulator)
- Simulates MicaZ mote (CC2420)
- Low-level simulator that incorporates a realistic signal propagation and noise model derived from real world (CPM)
- Simulation library use programming interfaces to write a program that configures a simulation and runs it
  - Python and C++
- Advantage:
  - Runs the same code written in nesC for sensor hardware.
  - Easy transitions between simulation and real network

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM – Building process

- `make micaz sim`

1. Generate an XML schema — *app.xml*
2. Compile the application — *sim.o*
3. Compile the Python support — *pytossim.o*, *tossim.o*, *c-support.o*
4. Build a share object — *_TOSSIMmodule.o*
5. Copying the Python support — *TOSSIM.py*

```
$ ./sim.py
```

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM – Useful TOSSIM Functions

.getNode() → TOSSIM.Mote

.radio() → TOSSIM.Radio

.newPacket() → TOSSIM.Packet

.mac() → TOSSIM.Mac

.runNextEvent()

.ticksPerSecond()

.time()

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM – Useful functions

- Debug messages can be added in the nesC code,
  - Example: dbg(“App”, “Output string %d \n”, arg);

• char*	sim_time_string()
• sim_time_t	sim_time()
• int	sim_random()
• sim_time_t	sim_ticks_per_sec()

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM – Radio Model

- Closest-fit Pattern Matching (CPM)
- Low-level simulator that incorporates a realistic signal propagation and noise model derived from real world
- RF noise and interference from other nodes and outside sources

- Important radio functions
  - add(source, destination, gain)
  - .connected(source, destination) → True/False
  - .gain(source, destination)

Mittuniversitetet
MID SWEDEN UNIVERSITY

TOSSIM – Node functions

- .bootAtTime(time)
  - Time is expressed in ticks
- .addNoiseTraceReading(noise)
  - Feed a particular noise trace to the model (at least 100 samples)
- .createNoiseModel()
  - Instructs CPM to finalize noise model creation

- Nodes should always start at different times.

Mittuniversitetet
MID SWEDEN UNIVERSITY

References and useful links

- http://www.tinyos.net/dist-2.0.0/tinyos-2.0.2/doc/nesdoc/micaz/
- https://github.com/tinyos/tinyos-main/blob/master/doc/pdf/tinyos-programming.pdf
- http://docs.tinyos.net/index.php/TOSSIM
- Improving Wireless Simulation Through Noise Modeling
  - H. Lee, A. Cerpa, and P. Levis (IPSN 2007)