A Scalable Simulator for TinyOS Applications

L. Felipe Perrone  (perrone@ists.dartmouth.edu)
David M. Nicol  (nicol@ists.dartmouth.edu)

ISTS Dartmouth College
Motivation

The Smart Dust project
UC Berkeley

macro scale

micro scale
The need for simulation

Network features:
Massively parallel, large-scale, self-configurable, application diversity, wireless, dynamic, mobility, behavior dependent on environmental conditions.

Environment features: Diversity of independent and inter-dependent dynamic processes.

Difficulties: Development, testing, debugging, performance evaluation.

- Chemical sensor
- Traffic sensor
- Monitor/sensor
Wish list for a simulator

Simulates:
- The processes that drive the sensors in the motes
- The programs that run on motes
- The communication medium

Supports:
- Very large numbers of motes
- Direct-execution of programs that run on motes
- Different applications in the same environment
- Accurate radio propagation model
TinyOS

The operating system on the mote platform.

**Frames** represent the internal state of the component and are statically allocated.

**Events** are analogous to signals or hardware interrupts. They may signal other events or call commands.

**Commands** can call other commands or post tasks.

**Tasks** may be interrupted by events, but not by other tasks. They may signal events and call commands.

Within a mote, tasks are scheduled in FIFO order.
A TinyOS application description

```
include modules{
MAIN;
COUNTER;
INT_TO_LEDS;
CLOCK;
};
```

```
app.desc file
```

```
include modules{
MAIN:MAIN_SUB_INIT COUNTER:COUNTER_INIT
MAIN:MAIN_SUB_START COUNTER:COUNTER_START
COUNTER:COUNTER_CLOCK_EVENT CLOCK:CLOCK_FIRE_EVENT
COUNTER:COUNTER_SUB_CLOCK_INIT CLOCK:CLOCK_INIT
COUNTER:COUNTER_SUB_OUTPUT_INIT INT_TO_LEDS:INT_TO_LEDS_INIT
COUNTER:COUNTER_OUTPUT INT_TO_LEDS:INT_TO_LEDS_OUTPUT
};
```

```
app.desc file
```

```
include modules{
MAIN;
COUNTER;
INT_TO_LEDS;
CLOCK;
};
```
Application code for one mote: components are wired together through compilation and linking.

Directly executed on a simulator

Unnecessary replication of the same code within the simulator.

10^4
or
10^5
Frames and local variables

Simulator’s memory space:

One instance of the application code

Multiple instances of the component frames in the application

Frame declaration:
```
define TOS_FRAME_TYPE mycomp_frame
TOS_FRAME_BEGIN(mycomp_frame) {
    int x;
}
TOS_FRAME_END(mycomp_frame);
```

Frame variable reference:
```
VAR(x)=0;
```

Frame declaration:
```
struct BLINK_frame : public TOSSF_Frame {
    char state;
};
```

Frame variable reference:
```
registerFrame("BLINK", new BLINK_frame, moteld);
BLINK_frame* TOSSFptr = (BLINK_frame*)
    getFrame("BLINK", moteld);
(TOSSFptr->state)=0;
```
To each application associate an object that maps the outbound wires of a component to the inbound wires of another. This object can be initialized at run time: applications can be defined at run time from a definition file or script.

```c
char BLINK_INIT_COMMAND(long moteld) {
    registerFrame("BLINK", new BLINK_frame, moteld);
    BLINK_frame* TOSSFptr = (BLINK_frame*) getFrame("BLINK", moteld);

    (*TOSSFwiringMap("BLINK","BLINK_LEDr_off_COMMAND")) (moteld);
    (*TOSSFwiringMap("BLINK","BLINK_LEDy_off_COMMAND")) (moteld);
    (*TOSSFwiringMap("BLINK","BLINK_LEDg_off_COMMAND")) (moteld);
    (TOSSFptr->state)=0;
    (*TOSSFwiringMap("BLINK","BLINK_SUB_INIT_COMMAND")) (moteld, tick1ps);
    return 1;
}

REGISTER_COMMAND("BLINK", BLINK_INIT_COMMAND);
REGISTER_COMMAND("BLINK", BLINK_LEDr_off_COMMAND);
REGISTER_COMMAND("BLINK", BLINK_LEDy_off_COMMAND);
REGISTER_COMMAND("BLINK", BLINK_LEDg_off_COMMAND);
```
The simulation substrate

- Environmental processes
  - Mobility (SWAN)
  - Terrain (SWAN)
  - RF Channel (SWAN)
  - Mobile computing nodes

DaSSF
A simple TOSSF model

MODEL [
  ARENA [
    MOBILITY [
      model "mobility.stationary"
      deployment "preset"
      seed 12345
      xdim 5000 ydim 5000
    ]
    NETWORK [
      model "network.fixed-range"
      cutoff 200
    ]
  ]
  MOTE [
    ID 1
    xpos 0 ypos 0
    battery 500
    _extends .APPLICATION_TYPES.BLINK
  ]
  ...
]

APPLICATION_TYPES [
  BLINK [
    components [
      session [name "LEDS" use "system.LEDS"]
      session [name "MAIN" use "core.MAIN"]
      session [name "CLOCK" use "core.CLOCK"]
      session [name "BLINK" use "app.BLINK"]
    ]
    wiring [
      map [MAIN MAIN_SUB_INIT BLINK BLINK_INIT]
      map [MAIN MAIN_SUB_START BLINK BLINK_START]
      map [BLINK BLINK_LEDy_on LEDs YELLOW_LED_ON]
      ...
    ]
  ]
]

DML script describing the application and the simulation scenario
Limitations of TOSSF

- All interrupts are serviced after a task, command or event finishes executing.
- Commands and event handlers execute in zero simulation time units.
- No preemption.
Scalability

- The complete SWAN code occupies 1.5M bytes of memory.
- A workstation with 256M bytes memory can hold roughly 32,500 motes.
- The memory overhead associated with each application type definition is that of a wiring map definition.
- The processing overhead involves table lookups for every variable reference and every function call (command or event). The cost incurred is application dependent.
- The model can be broken up for parallel simulation in SWAN: we’ll be able to experiment with very large network.
Future work on TOSSF

- Mote platforms got a lot more powerful: memory has increased from 8K to 128K. One can code up a single executable containing different applications to be deployed in all motes.

- A new generation of motes slated to be released soon will use different radio technology.

- With the release of TinyOS 1.0, applications are described in a different way in a dialect of C: nesC. All the source-to-source translation in TOSSF needs to be rethought.

- The nesC language is said to be a transient solution: a more powerful programming language are a work in progress.