Temporal Monitors for TinyOS
Application area: wireless sensor/actuator systems running TinyOS, a best-effort, asynchronous embedded OS

In summary:
Application area: wireless sensor/actuator systems running TinyOS, a best-effort, asynchronous embedded OS

Specifications: qualitative temporal (LTL) with:

atomic propositions = \{ boolean expressions over variables; checkpoints, including interrupts \}
Application area: wireless sensor/actuator systems running TinyOS, a best-effort, asynchronous embedded OS

Specifications: qualitative temporal (LTL) with:
- atomic propositions = \{ boolean expressions over variables; checkpoints, including interrupts \}

Verifier: native TinyOS software component
- quantified overhead
- emulation: absolute RAM and CPU overhead per monitor is negligible; ROM overhead for a basic LTL pattern is \(\sim 5.5\%\) of that available on a Telos revision B platform.

In summary:

[2000] Berkeley designs a platform; Crossbow, Inc., mass produces the hardware.
TinyOS: A network embedded OS


[2000] Berkeley designs a platform; Crossbow, Inc., mass produces the hardware.

[2002] Network embedded system C (nesC) is developed, Intel Research / UC Berkeley.

[2000] Berkeley designs a platform; Crossbow, Inc., mass produces the hardware.

[2002] Network embedded system C (nesC) is developed, Intel Research / UC Berkeley.
Doina Bucur, RV’12

TinyOS: A network embedded OS

[lifeunderyourfeet.org / Microsoft Research Sensor Map]
nesC is modular
nesC is modular
...with interrupt-based concurrency

Event handlers execute with priority.

A single interrupt level. Events may cascade on some platforms.

...with sequential event handlers, plus deferred computational tasks.
Native verifier in TinyOS

The verifier is a new OS component (offline, generated).
NesC instrumentation (offline, manual):

—[ events rooted in hardware interrupts, software conditions or checkpoints; tested with 60+ through the OS

——[ nesC events signaled to verifier; signaling time as per system load
Instrumentation

NesC instrumentation (offline, manual):

- events rooted in hardware interrupts, software conditions or checkpoints; tested with 60+ through the OS

- nesC events signaled to verifier; signaling time as per system load

```plaintext
generic module HplMsp430GeneralIOP([..]) {
    [..]
    uses async event void notify(uint16_t ap, bool val);
}
implementation
{
    [..]
    async command void IO.set() {
        [..]
        signal pax_notify((PORTx*10+pin), TRUE);
    }
```
Low visibility of system faults in deployments.

There exists a broad but incomplete set of system specifications, fault reports

node-local / distributed
System specifications

- Low visibility of system faults in deployments.
- There exists a broad but incomplete set of system specifications, fault reports
  - node-local / distributed

- Buffer and stack overflows, deadlocked/livelocked software, data races.
System specifications

- Low visibility of system faults in deployments.
- There exists a broad but incomplete set of system specifications, fault reports
  - node-local / distributed

- Buffer and stack overflows, deadlocked/livelocked software, data races.
- No next-hop destination for routing in a multihop network; unexpected value, outlier or gradient in sensed data or battery level; incorrect order of use of the OS kernel’s API.
LTL patterns (basic)

# Universality. p is true:
# (F01) Globally
[](p)
# (F02) Before r
<>r -> (p U r)
# (F03) After q
[](q -> [](p))
# (F04) Between q and r
[]((q & !r & <>r) -> (p U r))
# (F05) After q until r
[](q & !r -> (p W r))

# Precedence. s precedes p:
# (F16) Globally
!p W s
# (F17) Before r
<>r -> (!p U (s | r))
# (F19) Between q and r
[]((!p & !r & <>r) -> (!p U (s | r)))
# (F20) After q until r
[](q & r -> (!p W (s | r)))

“Between an timer alarm and the next one there must be new sensor data.”

“Every time a LED is on, it must have been preceded by the sending of a packet.”
LTL patterns (composed)

\[ \bigvee_{i=1}^{k} G p_i \]
and a generic event-sequence chain

\[ p_1 U (p_2 U (\ldots U p_k)) \]

In both cases, randomly generated formulas.
Monitor generation

\[ Fr \rightarrow (\neg p \ U (s \mid r)) \]
(Precedence pattern, Before scope)

Generated with a LTL-to-TGBA translator (part of SPOT).

Deterministic, all transitions accepting.
Monitor generation

\[ Fr \rightarrow (\neg p U (s \mid r)) \]
(Precedence pattern, Before scope)

Generated with a LTL-to-TGBA translator (part of SPOT).
Deterrministic, all transitions accepting.

```c
void step() {
    atomic {
        // calculate new state with (ap, val)
        current_checking_steps++;
        current = next; next = -1;

        if (current == 1) {
            if ((call stateBV.get(r)) ||
                (call stateBV.get(s)))
                next = 2;
            else if ((call stateBV.get(p)) &&
                !(call stateBV.get(r)) &&
                !(call stateBV.get(s)))
                next = 3;
            else if (!(call stateBV.get(p)) &&
                !(call stateBV.get(r)) &&
                !(call stateBV.get(s)))
                next = 1;
        } else if (current == 2) {
            next = 2;
        } else if (current == 3) {
            if (!(call stateBV.get(r)))
                next = 3;
        }
    }
    finished_checking = (next == -1);
}
```

Doina Bucur, RV’12
Tool chain

AP set
LTL: $[\Box(q \rightarrow \Box(p))$

SPOT

1

!p & !r & !s

r | s

p & !r & !s

2

1

3

!r

language translator

TinyOS application

AP set

TinyOS system components

Runtime path verifier

Hardware

AP set

ncc

binary

emulator

don't understand this one

notification and trace

Doina Bucur, RV'12
Evaluation

For **CPU** and **stack** overhead: emulator (MSPsim for TelosB).
For (other) **memory** overhead: compilation to executable suffices (there is no dynamic allocation).
Monitor size

Automaton size (no. states x no. transitions)

ROM overhead (TelosB)
RAM overhead (TelosB)
Accurate to the clock tick (advantage of emulating), modulo sampling rate.
ROM overhead

Doina Bucur, RV'12
RAM overhead