Software Verification for TinyOS Applications

Doina Bucur and Marta Kwiatkowska
Overview
Sensor networks

- Pervasive Healthcare. Body Sensor Networks
  vip.doc.ic.ac.uk/bsn

- Sensor architecture / OS.
  TinyOS:
  - Modern OS and language in an embedded system
Software verification

- Accountable applications
- Verification vs. simulation / formal verification
- Advances in software verification
- CProver tools [cprover.org]. Counterexample-guided abstraction refinement (CEGAR)
A TinyOS application

Drivers

Platform-specific drivers

Application (C, nesC)

Kernel (nesC)
...and the software tool chain

Components (C, nesC)

Platform-specific inlined program (C + asm)

Verification, but at which stage?
Bug-Free Sensors: 
The Automatic Verification of Context-Aware TinyOS Applications

with Marta Kwiatkowska

Proceedings of the European Conference on Ambient Intelligence (AmI 2009)
Springer Lecture Notes in Computer Science (LNCS), Nov 2009
Which software?

Our tool builds on SATABS [3], a generic software verification tool for ANSI C; SATABS takes specifications written as user-specified assertions of boolean conditions inserted in the code. The verification is sound (and complete for finite-state applications): The program’s state space is exhaustively explored for violations of the specification, including e.g. behaviours triggered by unexpected, but possible, events such as scrambled incoming network packets. An execution trace is returned as a bug witness, allowing the programmer to correct the fault before deploying the application.

We (i) add native support for the C TosThreads API to SATABS, (ii) implement a SATABS-readable C model of the TinyOS system calls to stand in for the OS kernel, and finally (iii) verify application and kernel model against context-aware safety specifications written as SATABS assertions. We report benchmarks on running our tool on standard applications distributed with TinyOS’s sources, and on a more complex healthcare application; we find routine violations of safety requirements in staple TinyOS code.

2 The Automatic Verification of TinyOS Applications

This section presents our verification method. We first overview TinyOS and the structure of a TinyOS application, which then allows us to underline possible sources of TinyOS software bugs. Finally, we assess performance with a set of benchmarks and point to the cause and nature of the bugs found.

Modelling the TinyOS Kernel

Fig. 1. A TinyOS application programmed in the TosThreads C API, calling TinyOS kernel services, as available on a Tmote Sky sensor node with integrated sensors for temperature, humidity and light intensity.

TinyOS 2.x applications written in TinyOS’s C TosThreads API.
Our tool builds on SATABS [3], a generic software verification tool for ANSI C; SATABS takes specifications written as user-specified assertions of boolean conditions inserted in the code. The verification is sound (and complete for finite-state applications): The program's state space is exhaustively explored for violations of the specification, including e.g. behaviours triggered by unexpected, but possible, events such as scrambled incoming network packets. An execution trace is returned as a bug witness, allowing the programmer to correct the fault before deploying the application.

We (i) add native support for the C TOSThreads API to SATABS, (ii) implement a SATABS-readable C model of the TinyOS system calls to stand in for the OS kernel, and finally (iii) verify application and kernel model against context-aware safety specifications written as SATABS assertions. We report benchmarks on running our tool on standard applications distributed with TinyOS’s sources, and on a more complex healthcare application; we find routine violations of safety requirements in staple TinyOS code.
Which programming errors?

Table 2. Categories of bugs in generic concurrent software

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink 4/64</td>
<td>yes</td>
<td>2.9s</td>
<td>interface use</td>
<td>-</td>
</tr>
<tr>
<td>SenseAndSend 6/347</td>
<td>no</td>
<td>32.2s</td>
<td>interface use</td>
<td>order violation</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>1m08s</td>
<td>sensing exception</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>4m25s</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>PatientNode 6/439</td>
<td>yes</td>
<td>29.9s</td>
<td>(interface use)</td>
<td>(order violation)</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>3m55s</td>
<td>(sensing exception)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>35m07s</td>
<td>network exception</td>
<td>deadlock</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>2m38s</td>
<td>(false reasoning)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>61m12s</td>
<td>(false reasoning)</td>
<td>-</td>
</tr>
</tbody>
</table>
Software Correctness for TinyOS

with Marta Kwiatkowska

In preparation.
Which software?

TinyOS x.x applications written in any API.

Application (C, nesC)

Kernel (nesC)

Drivers

Platform-specific drivers
What method?

Model the microcontroller’s working:
memory map, interrupt system.

```c
#define "tinyos-1.x/tos/platform/telos/hardware.h"
static inline
void TOSH_MAKE_GREEN_LED_OUTPUT(void)
{
    static volatile uint8_t r __asm ("0x0032");
    r |= 1 << 5;
}
```

```c
#define "tinyos-1.x/tos/platform/msp430/MSP430TimerM.nc"
void __attribute__((interrupt(12))) __attribute__((wakeup)) sig_TIMERx0_VECTOR(void)
{
    MSP430TimerM$CompareA0$fired();
}
```
Thank you!

Software Verification for TinyOS Applications