Software Verification for
TinyOS Sensors

Doina Bucur and Marta Kwiatkowska
Sensor platforms

TinyOS and NesC:
OS and language for embedded systems
Sensor networks

- Pervasive Healthcare.
- Body Sensor Networks
Overview: Node-based verification

- Language:
  - NesC, C
- “Bounded” verification:
  - few IRQ calls
  - little recursion unwinding (CBMC)
- Specifications:
  - assertions upon program states
A TinyOS application

Drivers

Platform-specific drivers

Kernel (nesC)

Application (C, nesC)
...and TinyOS’s compile stages

Components
(threaded, C, NesC)

NesC compiler

Platform-specific
inlined program
(sequential, C + asm)

Compiler

Platform compiler

Machine code

Deployment on sensors

inline static void
RealMainP$Scheduler$init(void ){
  SchedulerBasicP$Scheduler$init();
}

# 45  'tos/chips/msp430/
pins/
HplMsp430GeneralIOP.nc''
static inline void
HplMsp430GeneralIOP
$38$I$set(void )
{
  * (volatile uint8_t
  * )49U |= 0x01 << 6;
}

000001c0 l  *ABS*
00000000 DAC12_0CTL
000001c2 l  *ABS*
00000000 DAC12_1CTL
000001c8 l  *ABS*
00000000 DAC12_0DAT
000001ca l  *ABS*
00000000 DAC12_1DAT
00000122 l  *ABS*
00000000 DMACTL0
00000124 l  *ABS*
00000000 DMACTL1
000001e0 l  *ABS*
00000000 DMA0CTL

model extraction?
Solution 1, high-level

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.
Solution 2, close to hardware

Preserve system-wide code  
but
Model the microcontroller’s working:  
memory map, interrupt system.

```c
#include "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
#include "tinyos-1.x/tos/platform/msp430/MSP430TimerM.nc"

void __attribute__((interrupt(12))) __attribute__((wakeup)) sig_TIMERA0_VECTOR(void)
{
    MSP430TimerM$CompareA0$fired();
}
```
Our toolchain

nesC/C modules, configurations, wiring

MCU-specific C/asm (mspgcc)

CProver-readable standard C

assertion instrumentation (application-based)

tos2cprover (source transformation, analysis, and IRQ instrumentation)

cprover assertion instrumentation (memory violations, exceptions)

inline static void McuSleep (uint16_t temp;  
if(McuSleepC_dirty) {     
  McuSleepC$computePowerState();     
  temp = McuSleepC_msp430PowerBits[]  
__asm volatile ("#_R2 | temp;     
sig_ADC_VECTOR();     
__nesc_disable_interrupt; }

assertion violations with program trace
## Problem size

<table>
<thead>
<tr>
<th></th>
<th>Blink</th>
<th>Sense</th>
<th>TestDissemination</th>
</tr>
</thead>
<tbody>
<tr>
<td>functionality</td>
<td>timer</td>
<td>sensor, timer</td>
<td>CC2420 radio, timer</td>
</tr>
<tr>
<td>lines of code, number of loops</td>
<td>3340, 8</td>
<td>7181, 16</td>
<td>13388, 31</td>
</tr>
<tr>
<td>memory-violation assertions</td>
<td>35</td>
<td>132</td>
<td>747</td>
</tr>
<tr>
<td>expected interrupts</td>
<td>TIMERBO</td>
<td>TIMERBO, ADC</td>
<td>TIMERBO, PORT1, PORT2, UARTORX, UARTOTX</td>
</tr>
<tr>
<td>potentially raced global variables</td>
<td>TIMERBO: 6</td>
<td>TIMERBO: 7, ADC: 11</td>
<td>TIMERBO: 15, PORT1: 13, PORT2: 0, UARTORX: 19, UARTOTX: 0</td>
</tr>
<tr>
<td>IRQ instrumentations</td>
<td>initial 21, minimized to 4</td>
<td>initial 92, minimized to 8</td>
<td>initial 422, minimized to 30</td>
</tr>
</tbody>
</table>
Verification times

Figure 3: Verification times for selected memory-violation assertions in Sense 0.

- memset.1
- AdcStreamP_Init_Init.1
- SchedulerBasicP_popTask.1
- SchedulerBasicP_popTask.2
- McuSleepC_McuSleep_sleep.1
- HplAdc12P_HplAdc12_getCtl0.1
- HplAdc12P_HplAdc12_getCtl0.2
- HplAdc12P_HplAdc12_getMem.1
- Msp430Adc12ImplP_HplAdc12_conversionDone.1
- Msp430Adc12ImplP_HplAdc12_conversionDone.2
- HplAdc12P_HplAdc12_setMCtl.1
- HplAdc12P_HplAdc12_setMCtl.2
- HplAdc12P_HplAdc12_setMCtl.3
- HplAdc12P_HplAdc12_setMCtl.4
- HplAdc12P_HplAdc12_getMCtl.1
- RoundRobinResourceQueueC_0_RoundRobinQueue_isEnqueued.1
- RoundRobinResourceQueueC_0_RoundRobinQueue_isEnqueued.2
- RoundRobinResourceQueueC_0_RoundRobinQueue_isEmpty.1
- RoundRobinResourceQueueC_0_RoundRobinQueue_isEmpty.2
- RoundRobinResourceQueueC_0_clearEntry.1
- RoundRobinResourceQueueC_0_clearEntry.2
- ArbitratedReadStreamC_0_Resource_granted.1
- VirtualizeTimerC_0_Timer_stop.1
- VirtualizeTimerC_0_Timer_stop.5
- ArbitratedReadStreamC_0_Resource_granted.3
- ArbitratedReadStreamC_0_Resource_granted.4
- AdcStreamP_SingleChannel_multipleDataReady.1
- AdcStreamP_SingleChannel_singleDataReady.1
- AdcStreamP_SingleChannel_singleDataReady.2
- VirtualizeTimerC_0_updateFromTimer_runTask.1
- VirtualizeTimerC_0_updateFromTimer_runTask.6
- AdcStreamP_readStreamDone_runTask.1
- AdcStreamP_readStreamFail_runTask.2

Verification time (minutes) per assertion

Decision procedure runtime

Program unwinding time

Verification times

5. RELATED WORK

5.1 Runtime error protection

Most existing solutions against software errors in sensor operating systems act at runtime: the code is instrumented such that a statement which is semantically unsafe under its current execution context is detected before it is executed, and one or another diagnosis or recovery measure is taken (which usually consists in reporting the error and rebooting, as summarized in Table 3). While a necessary solution to ensure safe execution in all execution contexts, runtime error detection is necessarily followed by the expensive redeployment of already-deployed sensor software, and could instead be preceded by static error detection to save on redeployment efforts.

Safe TinyOS [9] detects memory and type violations in deployed, running code, and transfers control to a fault handler, which either reboots or powers down after sending a concise failure report to its base station. The failure report identifies the error type and its source code location (but doesn't give an execution trace clarifying the context which caused the error); the failure report is used to debug the code post-deployment. While the method allows the safe execution of existing TinyOS code, with little programmer effort and runtime overhead, debugging every error encountered envolves the software's redeployment.

Adraw baco f SafeT i ny O S ' s Dep ut y co d e i n s tr u -
Thank You!

Software Verification for TinyOS Sensors

Doina Bucur and Marta Kwiatkowska
Doina Bucur


Post-doctoral researcher, Oxford University Computing Laboratory, with Marta Kwiatkowska

Bio

Background in networking, operating systems (teaching for Cisco Systems); ubiquitous computing, especially sensor networks.

Some formal and software verification. Currently working at the intersection of systems and verification.

Mission statement

Model extraction

Real sensor deployments (MSP430 TinyOS nodes)

Suitable specification

Program-state assertions and

Bounded model checking, CBMC (qualit.)