Why TinyOS 2.x?[1]

- TinyOS 1.x widely used for programming sensor network platforms
- Fairly easy to pick up: based on nes-C
- Satisfactory for resource constrained platforms
- Reliable in most cases

- Has grown to a huge codebase: changing key aspects of TinyOS a big undertaking
  ➔ So why do it?

Problems in TinyOS 1.x[2]

- Multiplying platforms: CPU + Radio combinations
- Evolving platforms: avrmote → mica → mica2 → micaZ
- Limitations on tasks: low reliability, can post only 8 tasks, no priority, no parameter
- Experience has shown current interfaces are insufficient
- Need for higher level of abstraction for shared hardware resources

Presentation Outlook

- TinyOS 2.x Overview
  - What’s new?
  - How does it compare to TinyOS 1.x?
- Migrating from TinyOS 1.x to TinyOS 2.x
  - TinyOS 2.x Installation Guidelines
  - Porting TinyOS 1.x Code to TinyOS 2.x
  - How to code new features?
TinyOS 2.x Overview[1]

- Still component-based OS in nesC
- Clean-slate rewrite of the OS
- Work done in 3 areas:
  - Greater platform flexibility (interoperability)
  - Improved robustness and reliability
  - Service distributions (application development)
Platform / Hardware Abstraction[^3]

- As for Boomerang, platforms located in `tos/platforms`
  - `platform = chips + glue`
  - `includes .platform` file that contains info to pass to nesC compiler: tells what chips to look for
- **Chip code in** `tos/chips`
- **Hardware abstraction**: 3 level hierarchy called the HAA (Hardware Abstraction Architecture)

... Platform / Hardware Abstraction[3]‡

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIL</strong>: Hardware Independent Layer</td>
<td>Provides abstractions that are hardware independent. HIL does not provide all functionality that HAL can</td>
</tr>
<tr>
<td><strong>HAL</strong>: Hardware Abstraction Layer</td>
<td>Higher-level of abstraction easier to use but still provide full functionality of hardware</td>
</tr>
<tr>
<td><strong>HPL</strong>: Hardware Presentation Layer</td>
<td>Presents IO pins, registers etc. interfaces</td>
</tr>
</tbody>
</table>

Example: Send packet

**GenericComm.c:**
- Provides Send and Receive interfaces

**CC2420RadioM.c:**
- Write to TXFIFO
- Flush FIFOs

**HPLCC2420M.c:**
- Provides the cmd function that sets pins and registers to write to TXFIFO

‡ Familiar to Boomerang / moteiv directory users
Scheduler\textsuperscript{[3]}
Booting / Initialization[3]

- In TOS 1.x: StdControl interface powers up and starts at boot
- In TOS 2.x:
  - StdControl broken into Init and StdControl
  - boot sequence now only initializes component
  - Boot.booted event signals end of scheduler, HW, and SW initialization
  - Components can be started as needed (saves energy)

```plaintext
configuration AppC{
}

implementation{
    components MainC, AppM;
    AppM.Boot -> MainC;
    Initializes the radio and other components
}

module AppM{
    uses interface Boot;
}

implementation{
    event void Boot.booted(){
        call ...
    }
}
Virtualization\textsuperscript{[3]}\textsuperscript{†}

- Many TinyOS 2.x services are now *virtualized*: no more parameterized interface
- Instead: program instantiates service component
- See new Timers
Timers[^3]‡

- TOS 2.x introduces timers with 32kHz and 1kHz granularity
- A couple of high-precision async timers may be used
- New functions provided by the timer interface:
  - how much time before it fires
  - start timer in the future
  - no more “unique” for timer instantiation
... Timers[3]

configuration AppC{
}

implementation{
    components AppM, new TimerMillisC();
    AppM.Timer -> TimerMillisC;
...

module AppM{
    uses interface Timer<TMillis>;
}

implementation{
...

Communication[^3]

- `message_t now replaces TOS_Msg`:
  - it is now completely opaque
  - `AMPacket.destination(msg);` will return the destination of `msg`

- Component --- provides --- interface:
  - `AMReceiverC Receive, Packet`
  - `AMSenderC AMSend, Packet`

- `Packet provides a function that returns a pointer to the payload`
Error Codes[3]

- TinyOS 1.x had simple SUCCESS / FAIL error codes
- But doesn’t distinguish between errors!!
- TinyOS 2.x replaces this by
  - SUCCESS (packet was transmitted successfully)
  - FAIL (packet failed transmission)
  - EBUSY (the instantiation of AMSenderC.AMSend has a packet to send)
  - ECANCEL (packet has been canceled)
Arbitration

Many hardware systems may be used by various protocols (running the radio, sensors, etc.)

Need resource arbitration: Resource interface

Components request and release shared resources

RXFIFO
Comparison with TinyOS 1.x[4]

- A Sense & Send Application:
  - Sense and Send requires up to 32% more ROM on mica platforms, but 13% less on Telos
  - RAM usage reduced by up to 42%
  - Reduced CPU utilization (5-10%)

Presentation Outlook

- TinyOS 2.x Overview
  - What’s new?
  - How does it compare to TinyOS 1.x?
- Migrating from TinyOS 1.x to TinyOS 2.x
  - TinyOS 2.x Installation Guidelines
  - Porting TinyOS 1.x Code to TinyOS 2.x
  - How to code new features?
Installing TinyOS 2.x[5]

- From a clean drive: use CD or install manually
- Usual steps in 2nd case: install…
  - Java
  - Cygwin (Windows only)
  - Native compilers (msp430tools*.rpm for telos motes)
  - nesC compiler
  - TinyOS source tree
  - Graphviz visualization tool

Upgrading from TinyOS 1.x[6]

- Upgrade external tools (compilers):
  msp430tools*.rpm

- Upgrade TinyOS specific tools
  - Use `rpm -ivh` to keep TinyOS 1.x tree
  - Install nesC compiler BEFORE the TinyOS tools

- Install TinyOS 2.x tree
  - Use `–i` rpm argument to install, not `–U` (removes TinyOS 1.x tree)
  - Configure the environment

- Switching back to TinyOS 1.x requires only to change 4 env. var.

Porting TinyOS 1.x Code[7]

- **SUCCESS** no longer has a non-zero value

```c
if (call Packet.function())
{
    // SUCCESS!: do...
}
```

```c
if (call Packet.function() == SUCCESS)
{
    // SUCCESS!: do this...
}
```

- Every module implementing `StdControl` and calling `init()` must now implement the interface `Init`.

```c
Main.StdControl -> AppM;
MainC.SoftwareInit-> AppM;
```

```c
command result_t StdControl.init()
{
    // initialize component
}
command error_t StdControl.start()
{
    // start component
}
command error_t SoftwareInit.init()
{
    // initialize component
}
```

...(2) Porting TinyOS 1.x Code\textsuperscript{[7]}

- Radio needs to be started manually using:

\begin{verbatim}
AppM.AMControl -> ActiveMessageC;
\end{verbatim}

\begin{verbatim}
module AppM{
    uses interface SplitControl as AMControl;
}
implementation{
    (after boot sequence completed)
    call AMControl.start();
}\end{verbatim}

- Use of the \texttt{nx\_} prefix: means that an element is external type: its representation is chip-independent (storing LSB or MSB first)
...(3) Porting TinyOS 1.x Code[7]

- Replace bool data types by nx_uint8_t
- Packets are now abstract

```c
msg->addr = nodeId;
AMPacket.setDestination(msg, nodeId);
```

- Replace SendMsg and ReceiveMsg interfaces by AMSend and Receive, which at the lower layer are wired to AMSenderC and AMReceiverC

- TOS_LOCAL_ADDRESS no longer exists: distinction between:
  - Node ID: TOS_NODE_ID, set at compile-time
  - Active message address: AMPacket.localAddress() and can be changed during runtime
 ...(4) Porting TinyOS 1.x Code[^7]

- There are no red, green, yellow leds any longer. Instead call `Leds.led1Toggle(); Leds.led2On();`
Conclusion

- More involved than I thought it would be
- Some aspects of TinyOS 2.x already familiar to Boomerang users

Questions?