Lessons from TinyOS

Vlado Handziski (TU Berlin)
@vlahan
TinyOS?

One of the first customized operating systems for resource-constrained (wireless) networked embedded systems
TinyOS Goals and Features

● High concurrency under limited resources
  ○ Event-driven
  ○ Single-stack execution

● Flexibility and adaptivity
  ○ Component-based, expressed using nesC
  ○ Both portable and platform-specific abstractions

● High robustness
  ○ Static allocation
  ○ Static binding
A Bit of History...

● 1999  UC Berkeley project part of DARPA NEST program
● 2000  *Initial public release, mix of C macros and Perl*
● 2002  First version using the nesC language
● 2004  UC Berkeley and TU Berlin release MSP430 port
● 2006  *TinyOS Alliance formed, Complete rewrite, TinyOS 2.0 released*
● 2008  TinyOS 2.1.0 released
● 2010  TinyOS 2.1.1 released
● 2012  TinyOS 2.1.2 (BLIP:6lowPAN, TinyCoAP), GitHub
● 2014  *TinyOS 2.2, new build system*
TinyOS Alliance

- Representatives from both academia and industry, organized in a number of topical “Working Groups”
  - Guided by a charter and having “ownership” of a subset of the code tree
  - Individual membership and decision policies

- Core [2005-2014]: TinyOS core, hardware abstractions
- Net2 [2005-2014]: multihop protocols, CTP, BLIP
- 802.15.4, Zigbee, Doc, Sim, Tools, Testbeds
TinyOS Development Process

- Centered around proposing and finalizing “TinyOS Enhancement Proposals (TEPs)”
- Document core design principles, major abstractions and APIs
  - Best Current Practice TEPs
  - Documentary TEPs
  - Experimental and Informational TEPs
- Very valuable especially for new developers
  - Faster on-boarding
  - Higher-quality code aligned with core principles
TEP Examples

● BCP
  ○ TEP2: Hardware Abstraction Architecture
  ○ TEP3: Coding Standards
  ○ ...

● Documentation
  ○ TEP101: Analog-to-Digital Converters (ADCs)
  ○ TEP102: Timers
  ○ TEP103: Permanent Data Storage (Flash)
  ○ TEP106: Schedulers and Tasks
  ○ TEP107: TinyOS 2.x Boot Sequence
  ○ TEP108: Resource Arbitration
  ○ TEP109: Sensors and Sensor Boards
  ○ ...

Impact of Process

“good-enough is sometimes better than perfect”

- TEP-centered development process focused on completeness, finding the “optimal” solution
  - Required a lot of iterations, long discussions
  - Many WGs (including core) pushing for consensus among all members before finalization
  - Results were “immutable” after finalization
- Slow speed and lack of agility were the price
  - Some TEPs needed years to be finalized
- Need for sponsoring WG deterred external contributions
- Incompatible with modern development processes
  - 2013 attempt was made to merge GitHub feature request issues management with the TEP process, without big success
Impact of Process
“faster is sometimes better than complete”

● Very conservative and comprehensive release testing process
  ○ All testing apps were (re)tested on all platforms after each change post code freeze
  ○ Many test applications include multi-node protocol tests, not easily evaluated by classical continuous integration tools
  ○ Initial efforts by ETH and TUB to more closely integrate release process with testbeds did not get traction

● As a result, major releases roughly each two years
  ○ But HEAD on the development branch was kept almost as stable as a release
Impact of Academia
“too much innovation can kill you”

- Constant tension between innovating at levels relevant for research vs. the needs of the “average” user
- Constant tension between designing the OS towards flexibility vs. streamlined implementation of a single “standard” solution
  - E.g. protocol stack architecture (heaps vs. layered)
- Underestimating compounding effects on complexity when layering new concepts
  - Innovating concurrently on language and code organization, and OS architecture and abstractions
Impact of Academia
“surviving the PhD student half-life”

● Lack of continuity in the developer community
  ○ A lot of code in some ways related to research work
  ○ Significant churn in developers as a result of students graduating, projects ending
  ○ “Generation” changes are synchronized, minimizing “overlaps”

● Lack of real incentive for producing highest quality code
  ○ Code often missing the last 10-20% polishing that is needed to minimize bugs and facilitate long term maintenance
  ○ E.g. BLIP leading to non-aligned memory access errors on specific 32-bit CPUs (NXP JN 516x) due to implicit assumptions of working on a 16-bit platform, etc.
Impact of Industry
“beware of companies bringing gifts”

- TinyOS has benefited from close cooperation with many companies: Intel, Crossbow/Memsic, Moteiv/Sentila, ArchRock/Cisco, PeoplePower, Zolertia, etc.
  - Full-time, well-trained software engineers, resulting in high quality code contributions
- But interaction with industry can also be a liability and source of tensions in the community
  - Exposing the project to the business interests
  - Differing goals, stability vs. continuous innovation, leading to dissonance in design decisions
  - Can change focus, leave code unsupported, poach core developers
Impact of Licensing
“liberal licenses are a double-edged sword”

● All TinyOS core code is open sourced under a very liberal “modified BSD” license
  ○ It reduces the barriers in the development process among many diverse entities (including academia and industry) and is especially suited to the nature of the project (OS component library)

● However, it also has important shortcomings and has caused major tensions in the TinyOS community
  ○ Not being able to fully track “dark” usage of TinyOS code in industrial products
  ○ Companies forking TinyOS and innovating into external projects
    ■ Motiv: Boomerang [Apache License]
    ■ PeoplePower: OSHAN
Thanks!

vlahan@vlahan.net