Tickless Contiki
Efficient timekeeping for low-power sensor nodes
Franck Rousseau
Université Grenoble Alpes – LIG
The problem

• Low-power, scarce energy
  • Must sleep most of the time

• Distributed system
  • Time is not a local internal reference
  • Need for precise synchronization with neighbors
    • DSME beacons, GTS/ TSCH slots

• High resolution timers
  • Software MAC implementation
Traditional approach

• Periodic interrupt, software clock counting ticks : Hz (polling !)
  • Every 1/Hz seconds, increment absolute clock value ⇒ waste of energy

• Local clock in every node
  • Neighbors drift apart

• Resolution for timers
  • Hz = 100 means a resolution of 0.01 seconds (Linux jiffies)
    • Cannot sleep less and be more precise than 10 ms
  • Waking up more often ⇒ waste even more energy
Rounding issues

• Problem with timer resolution

• Periods that are not integer numbers of clock tick
  • 32 kHz, 31.25 µs resolution clock
  • How to wait for 40 symbols (2.4 GHz radio) ?
    • 640 µs = 20.48 ticks !

• Need for high resolution timers
Timers in Contiki

- **clock**: system time
- **timer, stimer**: needs polling
- **ctimer, etimer**: callbacks and events
  - for protocols and applications
- **rtimer**: real-time, architecture specific timers
  - preempt any running process
- Naive and inefficient implementation
  - Polling, O(n) list search, …
Precision

- **TMoteSky**
  - 16 bit counter with Hz = 128
  - $65536 / 128 = 512$ wrap around every $\sim 8'30"$

- **Longest Beacon Interval**
  - $960 \times 2^{14} \times 16 \times 10^{-6} \sim 251 \text{ s} \sim 4 \text{ min } 11 \text{ s}$
  - $251 \times 128 - 1 = 32127$ useless interrupts between two beacons!
What do we need?

• Efficient implementation, less running code ⇒ energy savings

• Long sleep periods, tickless timekeeping ⇒ energy savings

• High resolution timers
  • Software MAC implementation: order of symbol ~ 16 µs

• Precise synchronization with neighbors
  • Clock calibration / drift compensation
  • Smaller wake-up margins ⇒ energy savings
Modern timekeeping

- Tickless or dynamic ticks for a long time in GPOSes (Linux, BSD, …)
  - Prevent waking up idle CPUs and/or cores
  - Reduce load in virtualized environments
- HiRes timers: timeouts vs. timers
  - High performance NICs, multimedia
- Deferrable timers / Timer coalescing
  - Group non critical work in batches
Tickless for sensor nodes

• Some RTOSes for embedded systems support tickless
  • FreeRTOS, RIOT OS, FireKernel, ...

• In Contiki
  • Implemented for one target (not found yet)
  • Close enough in the current ST GreenNet implementation
  • Should be architecture dependent code
    • Factor out this code in the core Contiki
Requirements (i)

• Support for multiple clock sources
  • Several HW sources
    • HiRes and LowRes
  • Handle wrap around transparently
    • 16 bit architectures
Requirements (ii)

- Multiple virtual clocks
- Clock calibration
- Tracking several neighbors
  - DSME
  - TSCH
- Wake on Idle
Clock drift

eZ430, 12 kHz VLO
Efficiency of drift compensation

![Graph showing error distribution and cumulative frequency against frequency and error (clock ticks).]
Requirements (iii)

• Timer correction
  • Rounding issues

• Keep compatibility with existing API
  • Wake up tasks just before polling
Pending issues

• Efficient data structures and algorithms
• Generic transparent calibration possible ?
  • No need to worry in user code
Tentative architecture

Applications
  Protocols
    Timers
    Virtual clocks
  Calibration process
  Clock/timer sources

Sources:
  Src1
  Src2

Virtual clocks:
  VClock1
  VClock2
  VClock3

Timer:
  BeaconTimer

Calibration

External source:
  ExtSrc: Parent Beacon

802.15.4 beacon-enabled