Painless
Class 1 Devices Programming
Subject of the matter

• Sharing experience implementing:
  – RFC 4944 (6lowpan)
  – RFC 6282 (header compression)
  – TCP
  – UDP

• On constrained devices (class 1)
Programming Constrained Devices

• Class 0 devices
  – RAM << 10k and ROM << 100k
  – Use of specialized OS (Contiki, TinyOS)
    • Event-loop + cooperative multi-threading

• Class 1 devices
  – RAM ≈ 10k and ROM ≈ 100k
  – Alternatives to event-loop paradigm
    • We used RIOT for our implementations
Pains of Coding on Class 0 Devices

• Learning curve
  – Event-loop programming paradigm is different
  – Coders must learn that and cope with lack of RAM/ROM
Easier Coding on Class 1 Devices

• Average programmer background is OK
  – The OS allows full multi-threading as “usual”
  – No need to change programming paradigm
Pains of Coding on Class 0 Devices

• Implementing from scratch
  – Imposes a non-standard programming language or “misuse” of an existing one
  – Cannot one-to-one port existing code: need a new code base
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
    PROCESS_BEGIN();

    printf("Hello, world\n");

    PROCESS_END();
}
#include <stdio.h>

int main(void)
{
    puts("Hello, world!\n");
}

Easier Coding on Class 1 Devices

• Leveraging more well-known tools
  – The OS allows ANSI C
    • Allows for easy porting of existing code for Unix, BSD or Linux (for example BSD socket API)
  – Reuse well-known development and debugging tools
Pains of Coding on Class 0 Devices

• Increased design complexity, e.g.
  – Split phase execution to not block the system
  – Complex state machine to enable multiple connections
    \texttt{(uip\_process()) > 1200 lines)}
Easier Coding on Class 1 Devices

• Safer and quicker coding
  – Reducing the need for new code development and maintenance

• Additional benefits:
  – Microkernel architecture increases robustness
  – IPC API facilitates distributed programming and M2M communication
Efficiency aspect

- event-loop based systems
  - Considered to be more efficient in terms of memory usage and energy efficiency than multi-threading
  - One of the reasons why FreeRTOS etc. not used
Efficiency aspect

- **RIOT application w/ 6lowpan on MSP430:**
  - ~9.7kB RAM usage
  - 43kB ROM usage
- Preliminary results show RIOT on par with Contiki
- **Contiki:**
  - ~35kB RAM usage (w/ uip & RPL on ARM7)
  - Energy consumption: 50mA/40mA/80mA
- **RIOT:**
  - ~40kB RAM usage (w/ 6lowpan & RPL on ARM7)
  - Energy consumption: 58mA/30mA/80mA
Work in progress

• Implementation of COAP on RIOT
  – Compare with aspects reported in draft-kovatsch-lwig-class1-coap-00

• Broaden deployment by support more hardware platforms (16 bit MCUs and 32 bit CPUs)
Thanks

Questions?