Multithreading optimization techniques for sensor network operating systems

Final Presentation

By Aldarwich Yaser

Overview

- Programming model
- Advantages & disadvantages of models
- Thread optimization for sensor network
  - Memory optimization
    - Singel kernel stack
    - Stack size analysis
  - Energy consumption
    - Variable timer
  - Scheduling policy
    - Retos scheduling
    - Event-boosting thread scheduling
- Tinyos vs. Retos
- Conclusion

Programming models

- Multithread model
  - Retos
  - Mantis
- Event-driven model
  - TinyOS
  - SOS

Model advantages

- Multithread Model
  - Supports High concurrency with preemption
  - Automatic state management
  - Blocking I/O interface
- Event-driven Model
  - Lower memory requirements
  - Less energy consumption

Model disadvantages

- Multithread model
  - Large data memory
  - Large energy consumption
- Event-driven model
  - Low simulation performance
  - Manual configuration
  - Splits a long-running task into several phases

Optimization techniques for implementing thread on sensor network

(1) Memory optimization
- Single kernel stack
- Stack size analysis

(2) Energy reduction
- Timer variable

(3) Scheduling policy
- Event-boosting thread scheduler

Memory optimization (1)

(1) Single kernel stack

- Reduces the size of thread stack requirement :
  - Separates the thread stack into kernel and user stacks
  - Maintain a unitary kernel stack for system calls & interrupt handlers

- Size of kernel stack :
  - SUM{MAX(system call)+MAX(ISR)+h/w context}
  - MAX(system call)+MAX(ISR)+ SUM(h/w context).

Effect of Stack Optimization

- Test requirements
  - Seven sensor applications: MPT_mobile,MPT_backbone,R_send,R_recv,Sensing,Pingpong, Surge)
  - Two versions of RETOS

	Kernel stack size(byte)	Increase of TCP+H/W context (byte)
Singel Kernel stack System	76	18
Multiple Kernel stack System	76	16

Effect of Stack Optimization

Applications	N of threads	User Stack (byte)	Data section (byte)	Kernel stack (byte)	
				Single	Multiple
MPT_backbone	1	68	131	76	152
MPT_mobile	2	78	416	76	228
R_send	3	78	217	76	304
R_recv	3	50	214	76	304
Sensing	2	18	157	76	328
Pingpong	1	8	106	76	152
Surge	4	98	336	76	380

Memory optimization (2)

(2) Stack-size analysis :

Kernel assigns exact stack size automatically to every thread

- Determine optimal stack requirement
- Generate a control flow graph of an application
- Calculates the maximum thread stack size using DFS

Instruction	Stack usages	Description
push var	+ 2	Push a value
pop var	- 2	Pop a value
call #label	+ 2	Push return address
add/sub SP, N	+-N	Directly adjust stack pointer

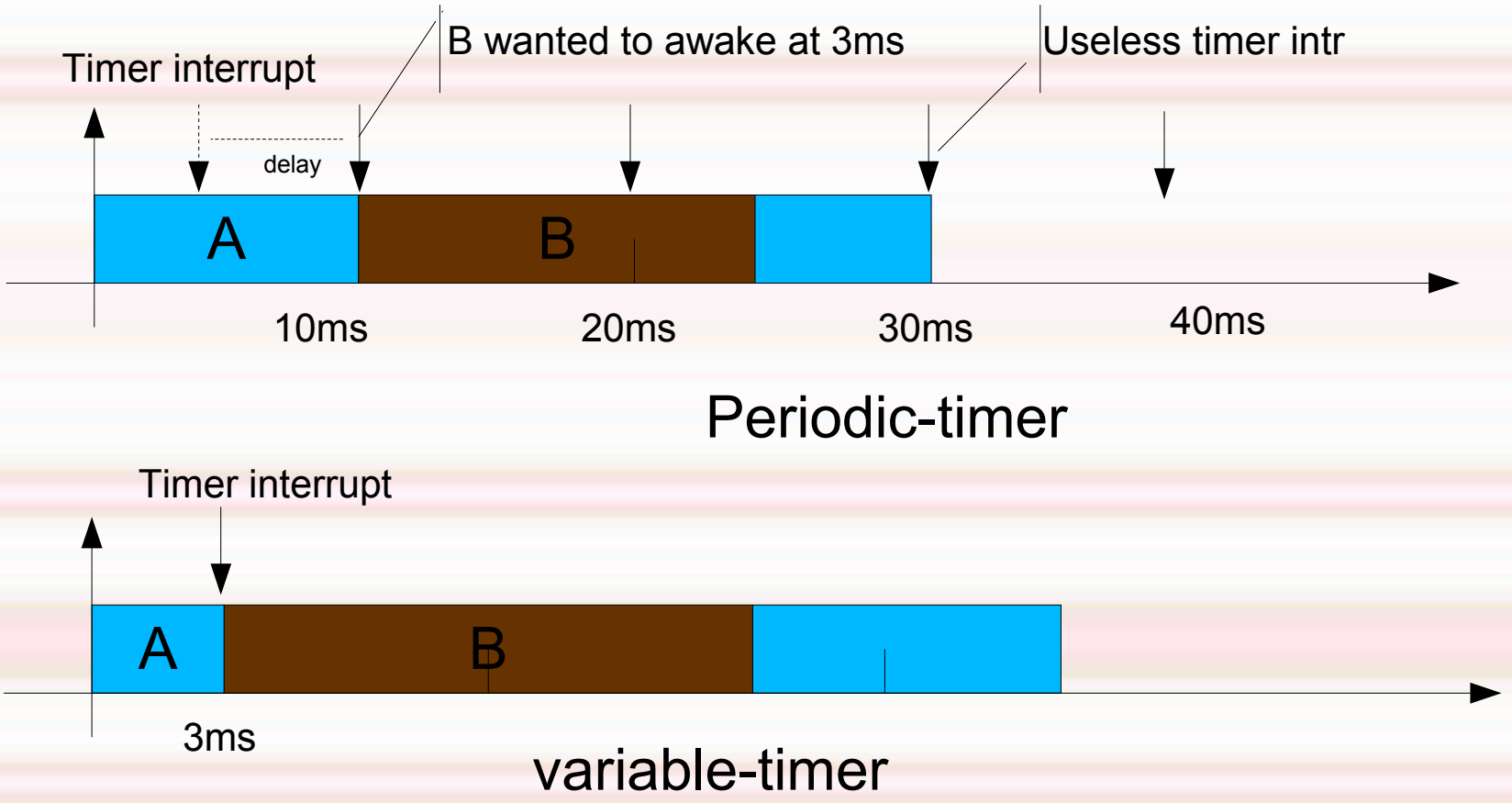
TI MSP430 stack instructions

Energy reduction (1)

- Variable timer (1)
  - Technique to reduce energy consumption
  - Reasons of overhead multithreaded system:
    - Context switching
    - Scheduling
    - Time management

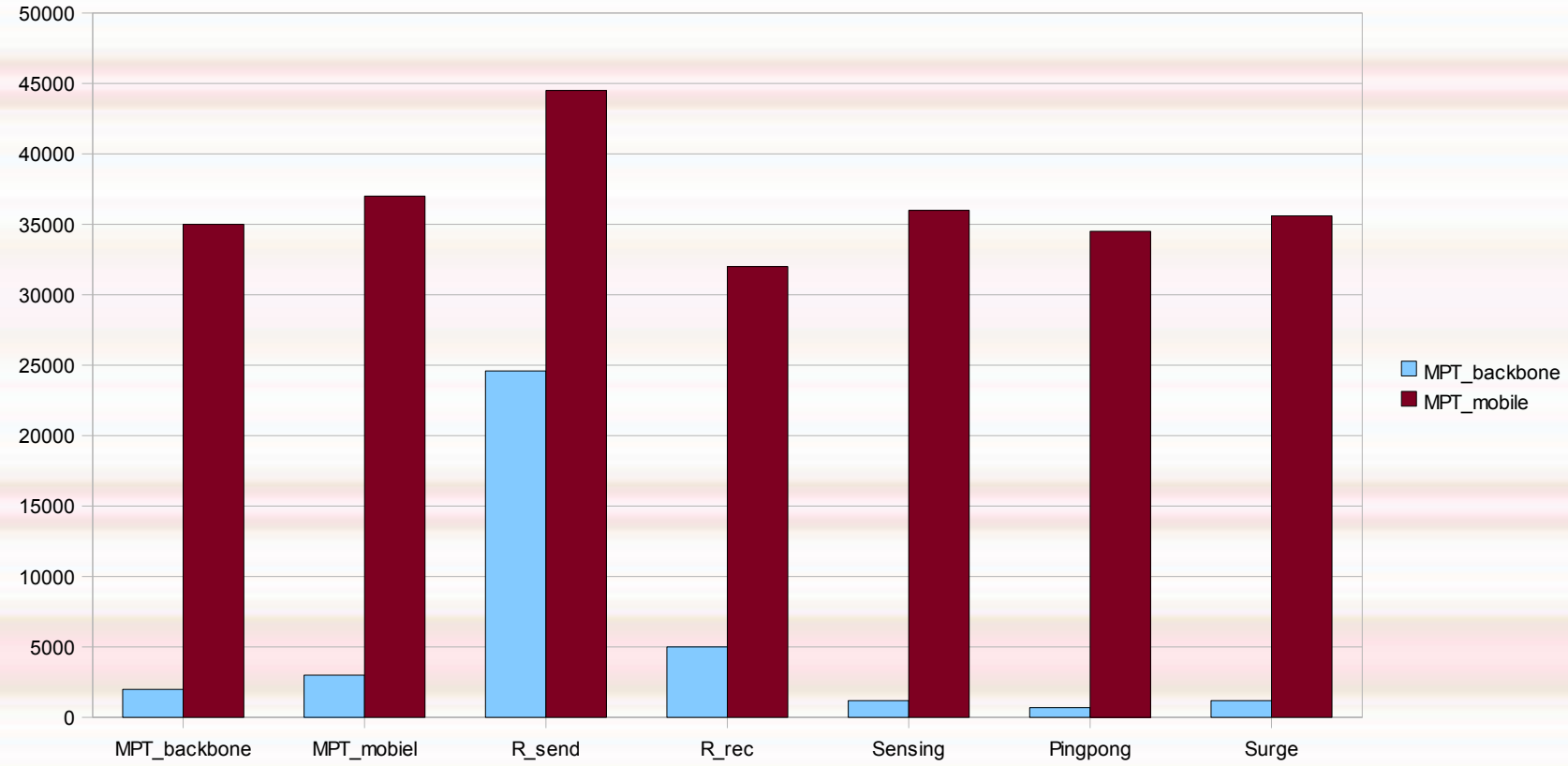
Energy reduction (2)

- Variable timer (2)



Energy reduction (3)

- Effect of variable timer



Scheduling policy

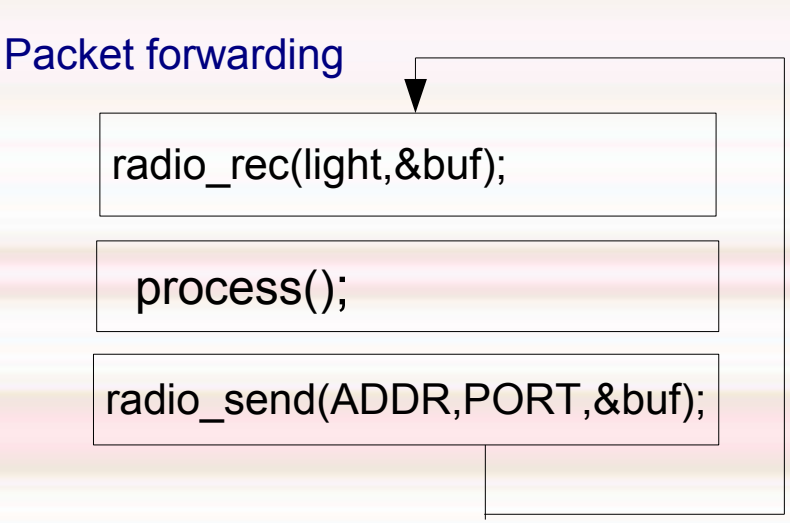
- RETOS scheduling
  - Priority-based and preemptive scheduling
  - POSIX.4 Compatibility
- Event-boosting thread scheduling
  - Typical sensor applications on multithreaded systems
  - Priority adjustment for event-boosting

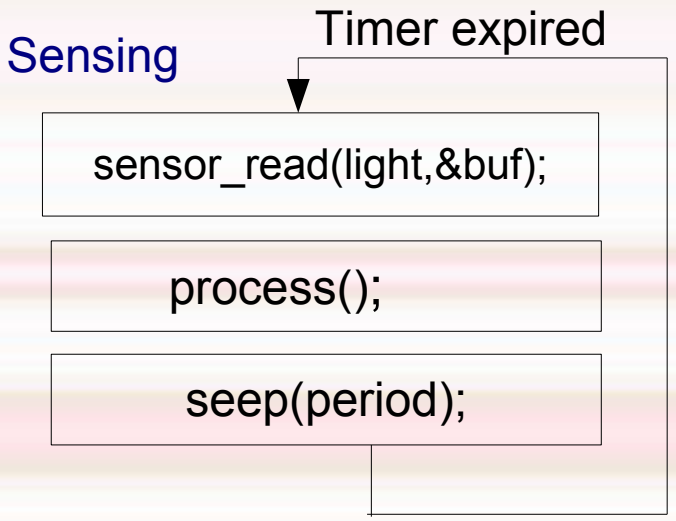
Retos scheduling

- Priority-based and preemptive scheduling
  - When time quantum expires, timer interrupts
  - Support both *static* and *dynamic* priority
  - Threads are preemptive
- Posix.4 compatibility
  - SCHED_FIFO
  - SCHED_RR
  - SCHED_OTHER

Event-boosting thread scheduling

- Typical sensor application on multithreading system
  - Thread classified into I/O bound & CPU bound
  - I/O bound is preferable





Event-boosting Thread Scheduling

- Priority adjustment for event-boosting
  - Boosts the priority of the thread requesting to handle a sensor application specific event
  - Event in Sensor network application defined as:
    - Expiration of the timer request
    - Receiving a packet
    - Sensing

	Dynamic priority	Description
Init	4	Thread created
sleep()	+3	Timer request
radio_recv()	+2	Radio event request
sensor_read()	+1	Sensor event request
Consuming CPU time	-1 per 8ms	Decreasing

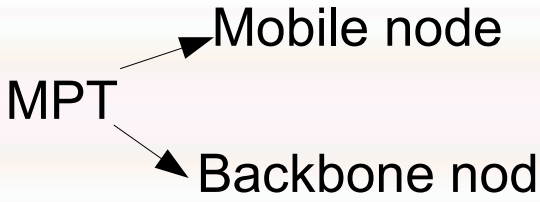
TinyOS and RETOS

- TinyOS :
  - Provides an event driven operating environment
  - Componet-based operating system
  - Dosn't have difference between the kernel and the Application
  - Support multiple streams of data.
- RETOS:
  - Provides a multithreaded programming interface
  -Support functionality

TinyOS vs. RETOS

- Experament‘s requiremnts:

- RetOS v0.96,TinyOS v1.1.13
- Applications : MPT and packet transmission



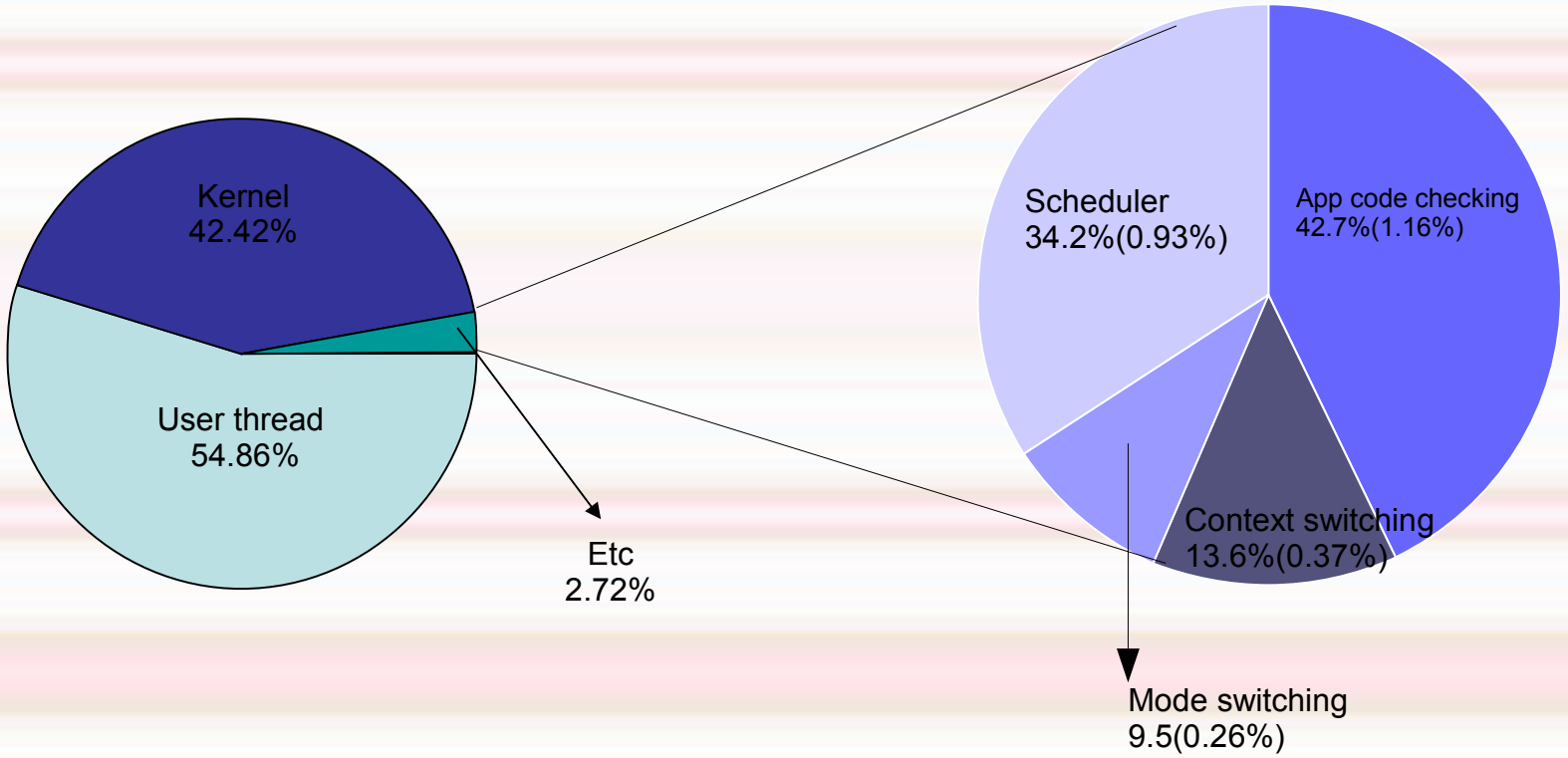
RETOS vs. TinyOS

- MPT(Mobile object tracking)
  - Based on ultrasound localization technique
  - Consist of mobile node and backbone node
  - Mobile nodes: computes their location by Trilateration every 300ms
  - Trilateration takes around 16ms to define location

	TinyOS(byte)		RETOS Kernel (byte)		RETOS LIb. +App(byte)		RETOS Total (byte)	
	ROM	RAM	ROM	RAM	ROM	RAM	ROM	RAM
MPT BB	12614	467	18314	748	492	143	18806	891
MPT Mobile	17222	701	18314	748	3	25162	25162	1182

RETOS vs. TinyOS

- Retos overhead analysis



Conclusion

Optimized multithreading techniques for sensor network operating systems

- **Memory optimization** : Single kernel stack, Stack-size analysis
- **Energy reduction** : Variable timer
- **Scheduling policy** : Event-boosting thread scheduler

Tests

- Overhead of multithreading system can be reduced to minimal
- Response delay to sensor application can be reduced as well
- System provide quick response time of threads without manual setup.

The End

Thank you for your attention

Questions