Multithreading Optimization Techniques for Sensor Network Operating Systems

Hyoseung Kim **Hojung Cha**

Yonsei University, Korea

2007. 1. 31.

Hyoseung Kim
hskim@cs.yonsei.ac.kr

Motivation (1)

- **Sensor network OS requirements**
  - Support high concurrency
  - Minimal memory usage and low energy consumption

- **TinyOS and SOS**

Event-driven Sensor Operating Systems

Cooperatively-operated tasks | No preemption | Single stack

  - "TinyOS achieved 30x improvement in data memory and 12x reduction in power consumption over GPOS"

S.-F. Li et. al, "Low Power Operating System for Heterogeneous Wireless Communication Systems," **Proc. of the Workshop on Compilers and Operating Systems for Low Power,** 2001.

Motivation (2)

- **Disadvantage of event-driven model**

Split a long-running task into several phases

Manage event handlers' states

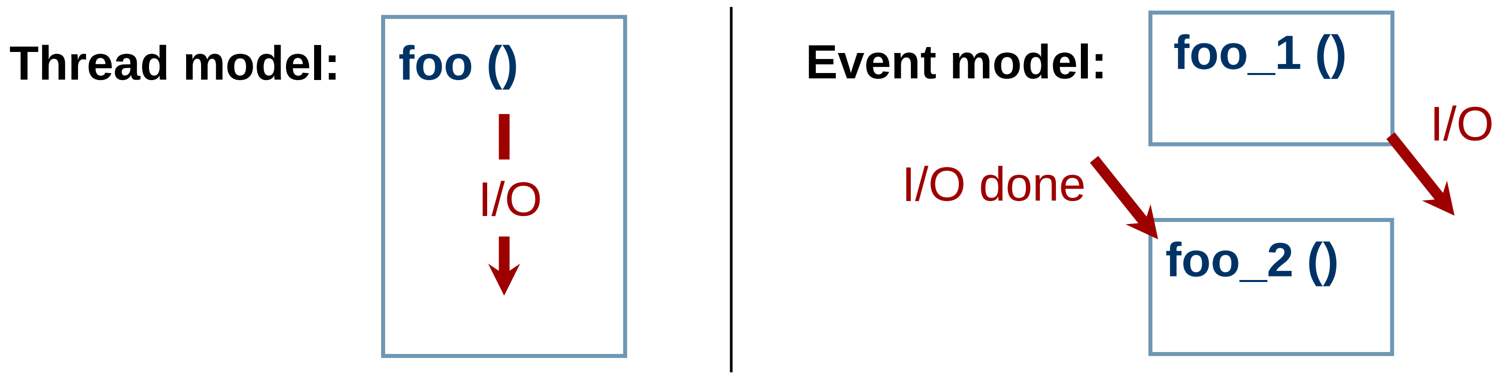
Tasks are inter-dependent in execution context

Changes in application requirements

Repeated analysis of
- **System concurrency**
- **System restructuring**

Motivation (3)

- **Multithreading model**
  - High concurrency with preemption
  - Automatic state management
  - Blocking I/O interface



- **Problem: overhead of multithreading**
  - **Space overhead:** stack reservation for each thread
  - **Time overhead:** scheduler, context switch, time management
  - **Scheduling policy** optimized for sensor applications

Our Goals

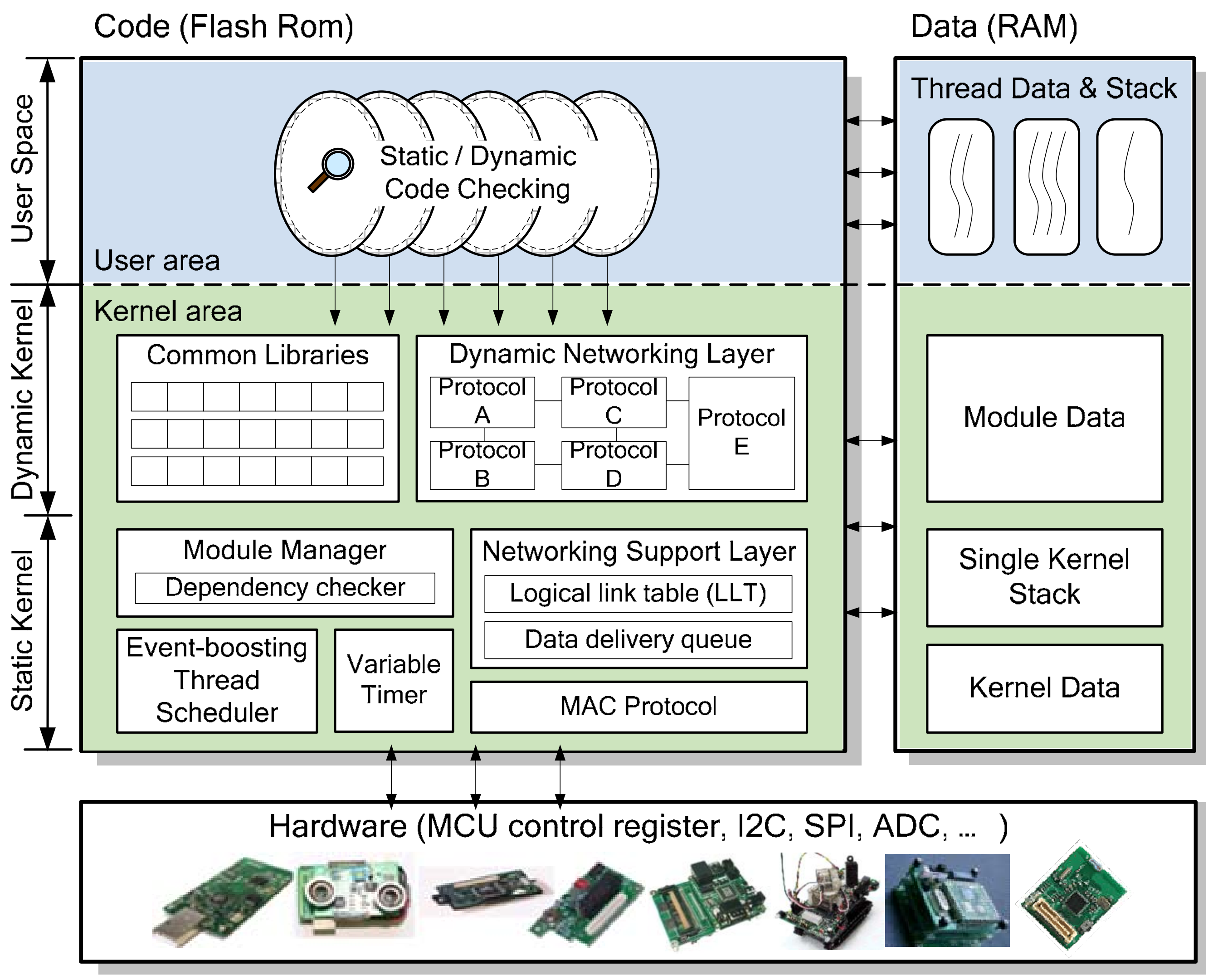
- **Optimized techniques for the implementation of multithreaded sensor network operating systems!**

  (1) Memory optimization
  - **Single kernel stack** reduces the size of thread stack requirement
  - **Stack-size analysis** assigns an appropriate stack size to each thread automatically

  (2) Energy reduction
  - **Variable timer** reduces energy consumption of time management, such as CPU-time sharing and software timers

  (3) Sensor-aware scheduling policy
  - **Event-boosting thread scheduler** satisfies the response time requirement for sensor applications

RETOS Operating System



Issues

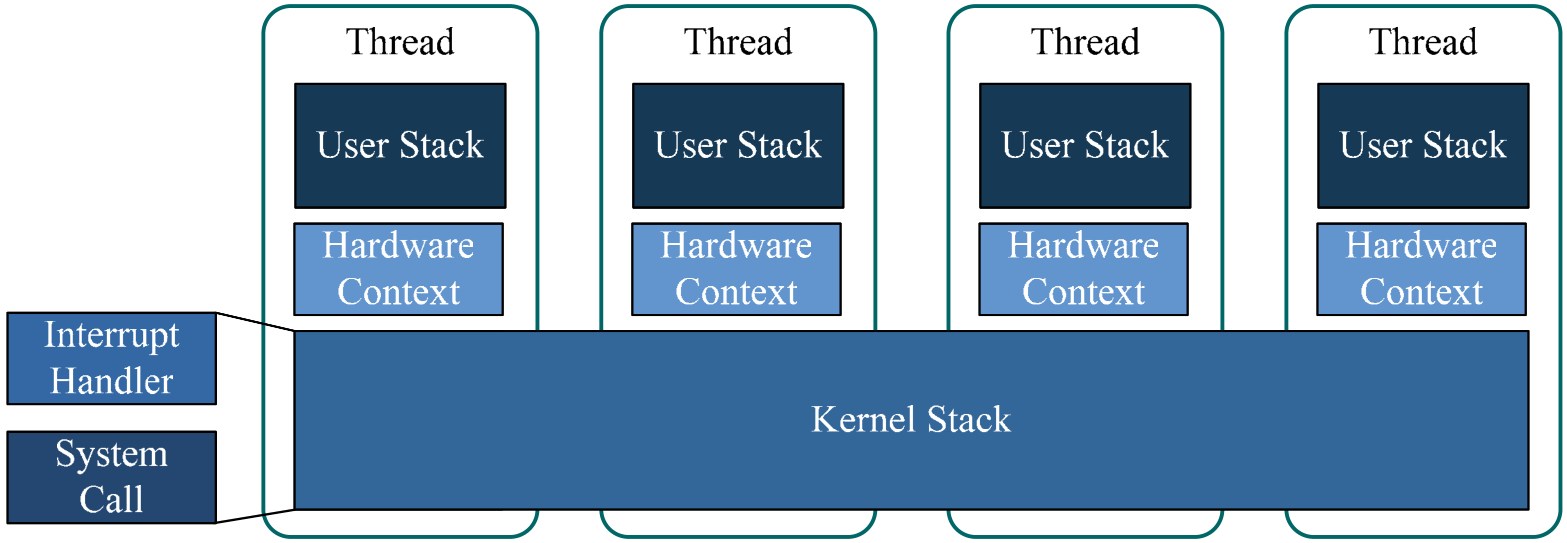
Memory Optimization

Energy Reduction

Scheduling Policy

Single Kernel Stack (1)

- **Size of thread stack**
  - Thread function + System calls + Interrupt handlers
- **Single kernel stack**
  - Separate the thread stack into kernel and user stacks
  - Maintain a **unitary kernel stack** for system calls and interrupt handlers to reduce the thread stack bound



Single Kernel Stack (2)

- **Results of running multithread sensor applications**

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

- **If each system executes all applications in this table,**

Multiple kernel stack system

1748 bytes for kernel stack

⟷

Single kernel stack system

76 bytes for kernel stack

Stack-size Analysis (1)

- **With MMU-less hardware, application developers must estimate accurate thread stack size**
  - Stack overflow ⟷ RAM overhead
- **Stack-size analysis**
  - Estimate optimal stack requirement and pass the size to the kernel
  - Produce a control flow graph of an application
  - Calculate the maximum possible thread stack size with DFS

Ex) TI MSP430 stack instructions

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

Stack-size Analysis (2)

- **Exception 1: Recursive call**
  - Create cycles in the flow graph
- **Exception 2: Indirect call**
  - Cause a disconnect in the flow graph
- **Results**
  - The results of the technique were equal to 1 or 2 words more than the results of simulation
  - The same call graph with the program's control flow

Issues

Memory Optimization

Energy Reduction

Scheduling Policy

Variable Timer (1)

- **Major energy overhead of the multithreaded system**
  - Scheduling

Frequency of Thread scheduling	<<	Frequency of Message passing
Context saving/restoring (TI MSP430)	<	90 clock cycles

  - Time management

Timer interrupt handler & timer bottom half handler	>	320 clock cycle

100HZ of periodic timer interrupt : 32,000 clock cycles/sec

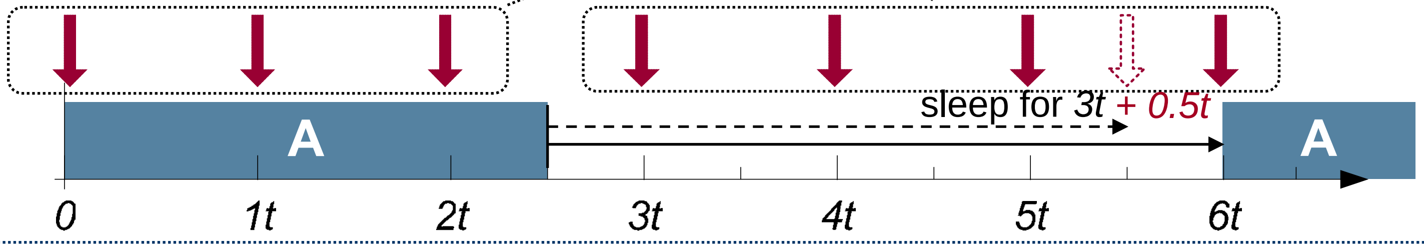
We have focused on reducing the overhead of time management!!

Variable Timer (2)

- **Problem: Time management of Multi-threaded GPOS**
  - Based on the periodic timer interrupt
  - Continuously trigger the interrupt handler even if it is not required

► Let the timer interrupt interval be *t* and A have *3t* of time quantum. Let A be executed until *2.5t.*

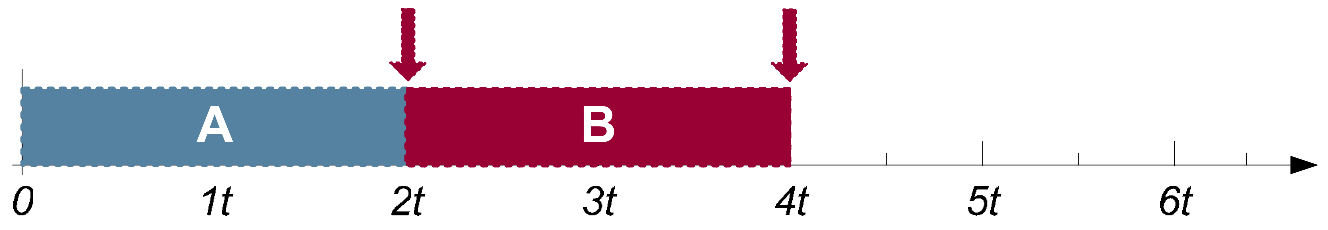
− If there is no timer request to be handled, then these are useless.
− If there is no thread to be scheduled, then these are useless.
− If A wanted to be woken up after *3t* of sleep, then the time drift is inevitable.



Variable Timer (3)

- **Principle: Adjust the next timer interrupt to the earliest upcoming timeout**

► Let A and B be threads with *2t* of time quantum.
Let there be no external interrupt, except the timer interrupt.

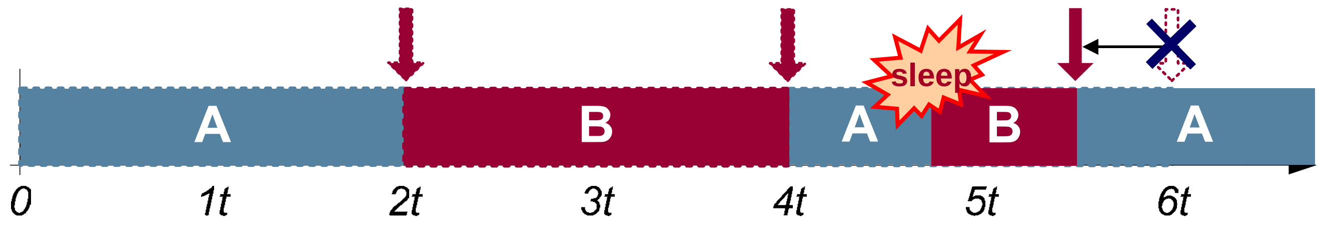


1. Next thread: A, Earliest upcoming timeout: A's quantum expiration
2. Timer interrupts → A has exhausted its time quantum
3. Next thread: B, Earliest upcoming timeout: B's quantum expiration
4. Timer interrupts → B has exhausted its time quantum
5. All running threads (A, B) have exhausted their quanta, Scheduler re-allocates time quantum durations

Variable Timer (3)

- **Principle: Adjust the next timer interrupt to the earliest upcoming timeout**

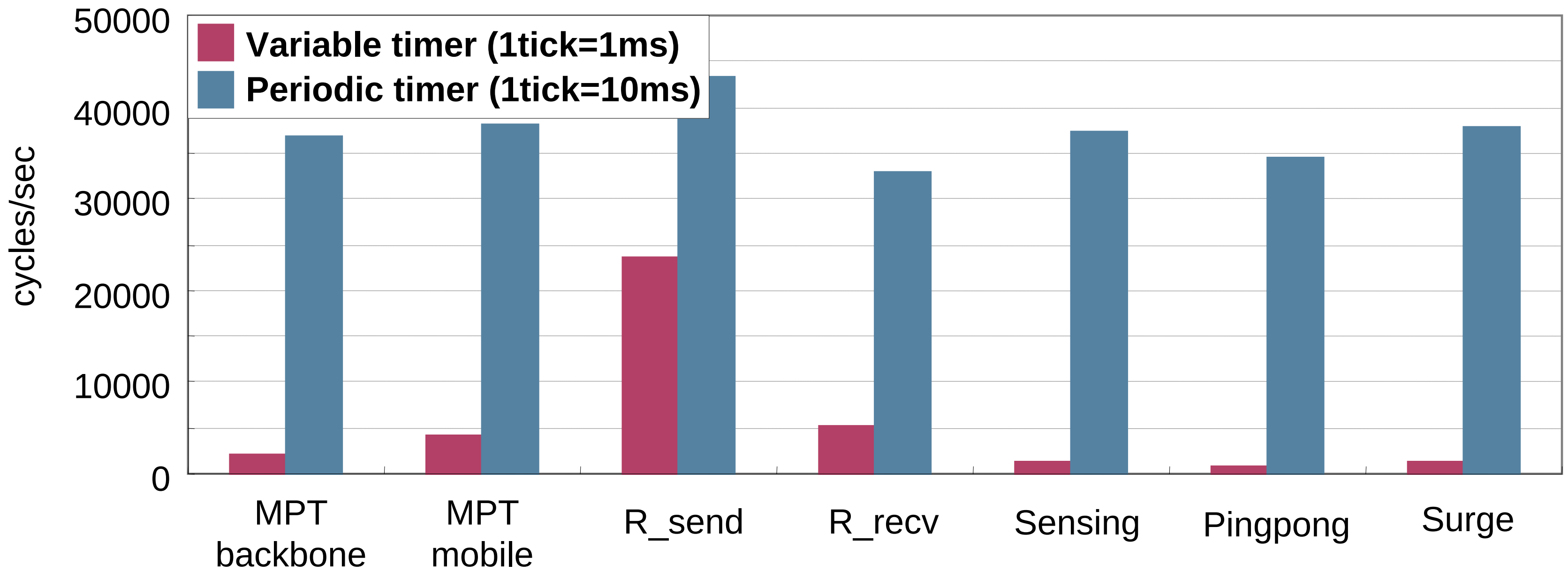
► Let A and B be threads with *2t* of time quantum.
Let there be no external interrupt, except the timer interrupt.



6. Next thread: A, Earliest upcoming timeout: A's quantum expiration
7. A issued ***sleep(0.75t)*** and blocked
   Adjust next timer interrupt: *current time + 0.75t = 5.5t*
8. Next thread: B, Timer is not changed
   (B's time quantum expiration is later than A's wakeup time)
9. Timer interrupts: *sleep(0.75t)* expired
10. Next thread: A.

Variable Timer (4)

- **Energy efficiency of variable timer**
  - Measured the active CPU time to estimate the energy consumption
  - The effectiveness differs from the execution interval of applications



Issues

Memory Optimization

Energy Reduction

Scheduling Policy

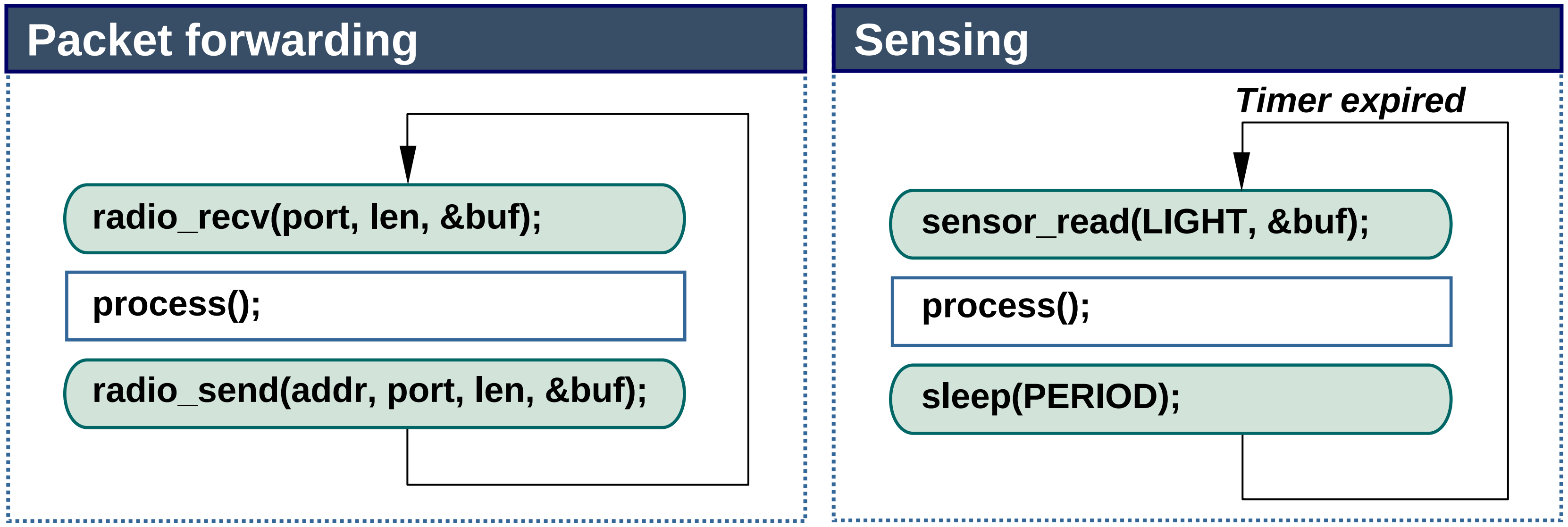
RETOS Scheduling

- **Priority-based, time-sharing, preemptive scheduling**
  – When time quantum expires, timer interrupts
  – Support both *static* and *dynamic* priority
  – Threads are preemptive (*but the kernel is non-preemtive*).
  – The scheduler is invoked in a deferred way
  : before returning to the user mode

- **POSIX.4 Compatibility**
  – SCHED_FIFO: static priority "real-time" class with no time limit
  – SCHED_RR: static priority "real-time" class with time quantum
    - Cannot be blocked by best-effort threads → not hard real-time
  – **SCHED_OTHER:** best effort class (conventional threads)

Event-boosting Thread Scheduling (1)

- **Typical sensor applications on multithreaded systems**


Packet forwarding
radio_recv(port, len, &buf);
process();
radio_send(addr, port, len, &buf);
Sensing
Timer expired
sensor_read(LIGHT, &buf);
process();
sleep(PERIOD);

  – Mostly I/O bound job
  – Switch between I/O bound and CPU bound

Event-boosting Thread Scheduling (2)

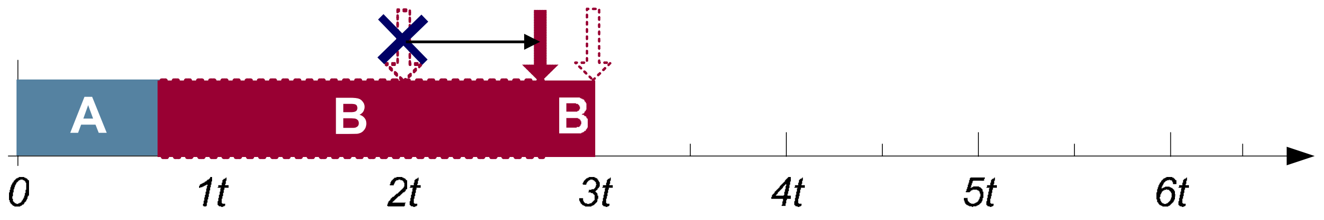
- **Principle: Priority adjustment for event-boosting**
  - Boosts the priority of the thread requesting to handle a sensor application specific event
  - Expiration of the timer request
  - Receiving a packet
  - Sensing

	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	Decrease dynamic priority

Event-boosting Thread Scheduling (3)

- **Principle: Priority adjustment for event-boosting**

  ► Let A and B have the same initial priority and 2t of time quantum
  Let A be periodic sensing thread (interval: 3t)
  Let B be CPU-bound thread

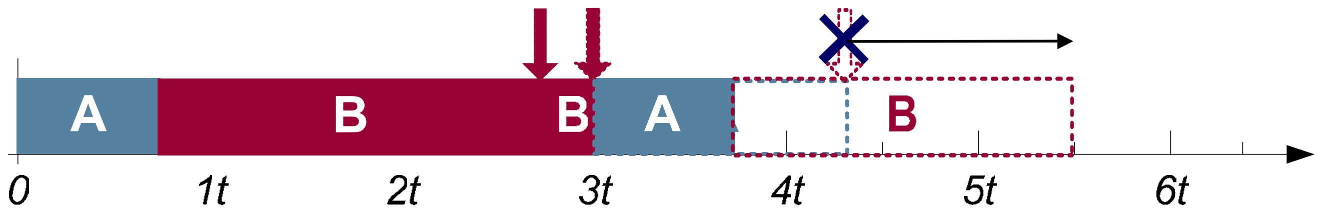


1. A : wakeup, variable timer sets the next timer interrupt at 2t
   A : → process → sleep, priority ↑
2. Next thread: B, Variable timer adjusts the next timer interrupt
3. Timer interrupts, B : CPU time consumed, priority ↓
4. B exhausted its time quantum → Re-allocate time quantum.
   Next thread: B.
   Earliest upcoming timeout: 3t (A's sleep expiration)

Event-boosting Thread Scheduling (4)

- **Principle: Priority adjustment for event-boosting**

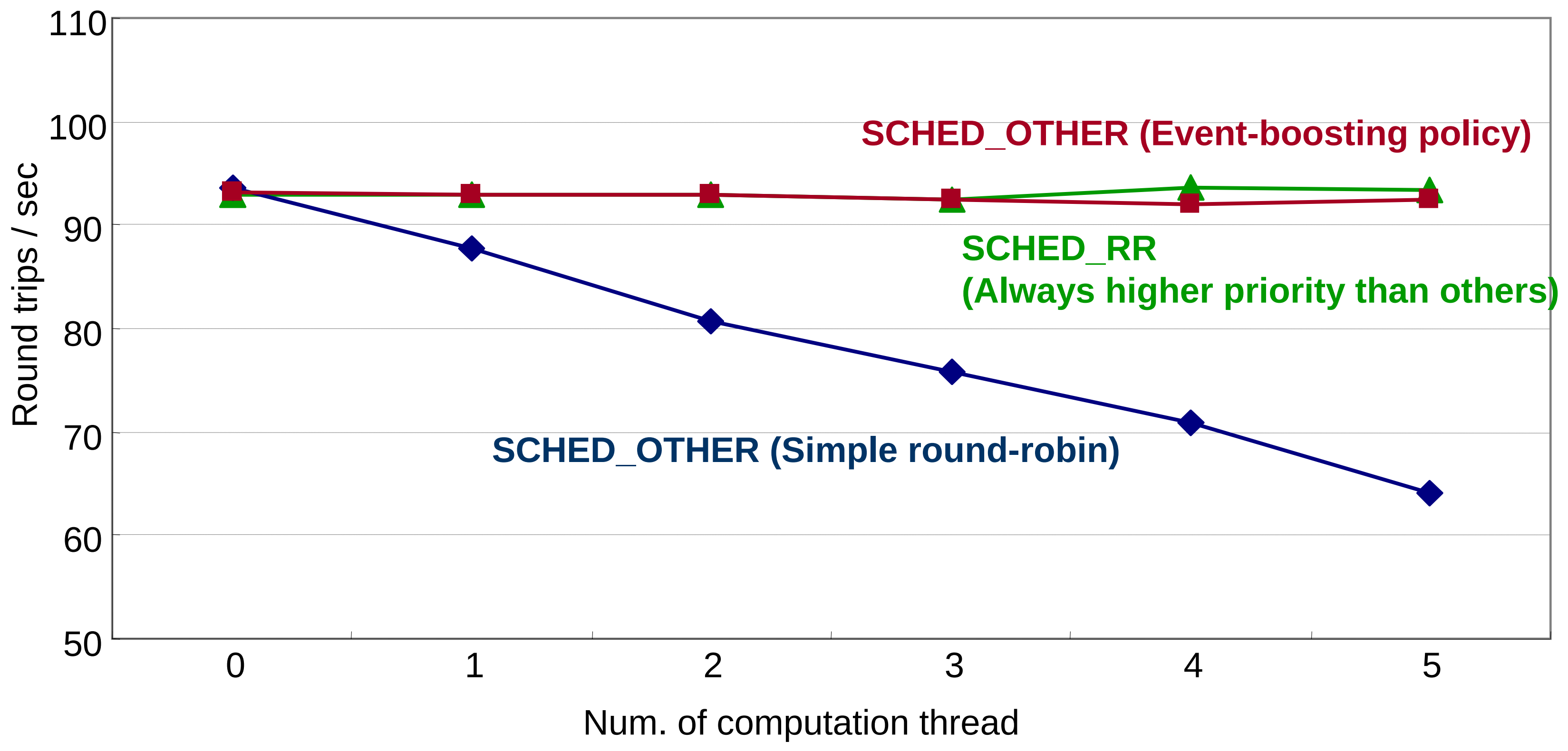
► Let A and B have the same initial priority and 2t of time quantum
Let A be periodic sensing thread (interval: 3t)
Let B be CPU-bound thread



5. Timer interrupts → Thread A wakes up
6. A : priority boosted by sleep() system call
   B : CPU time consumed, priority ↓ → preemption (next thread: A)
7. A issued sleep() and blocked
8. Next thread : B,
   B will execute until the time quantum expiration

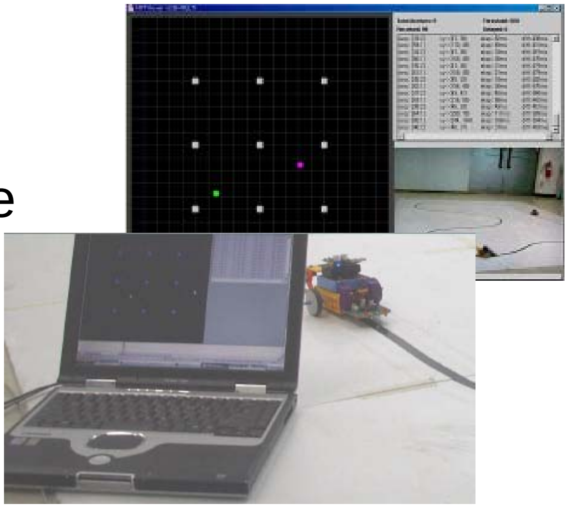
Event-boosting Thread Scheduling (5)

- **Results of the event-boosting policy**
  - A packet round-trip program
  - # of round-trips per second according to the scheduling policy



RETOS vs. TinyOS (1)

- **MPT(Mobile object tracking) app.**
  - Based on ultrasound localization technique
  - Consist of Mobile node and Backbone node
  - Mobile node: computes its location using Trilateration

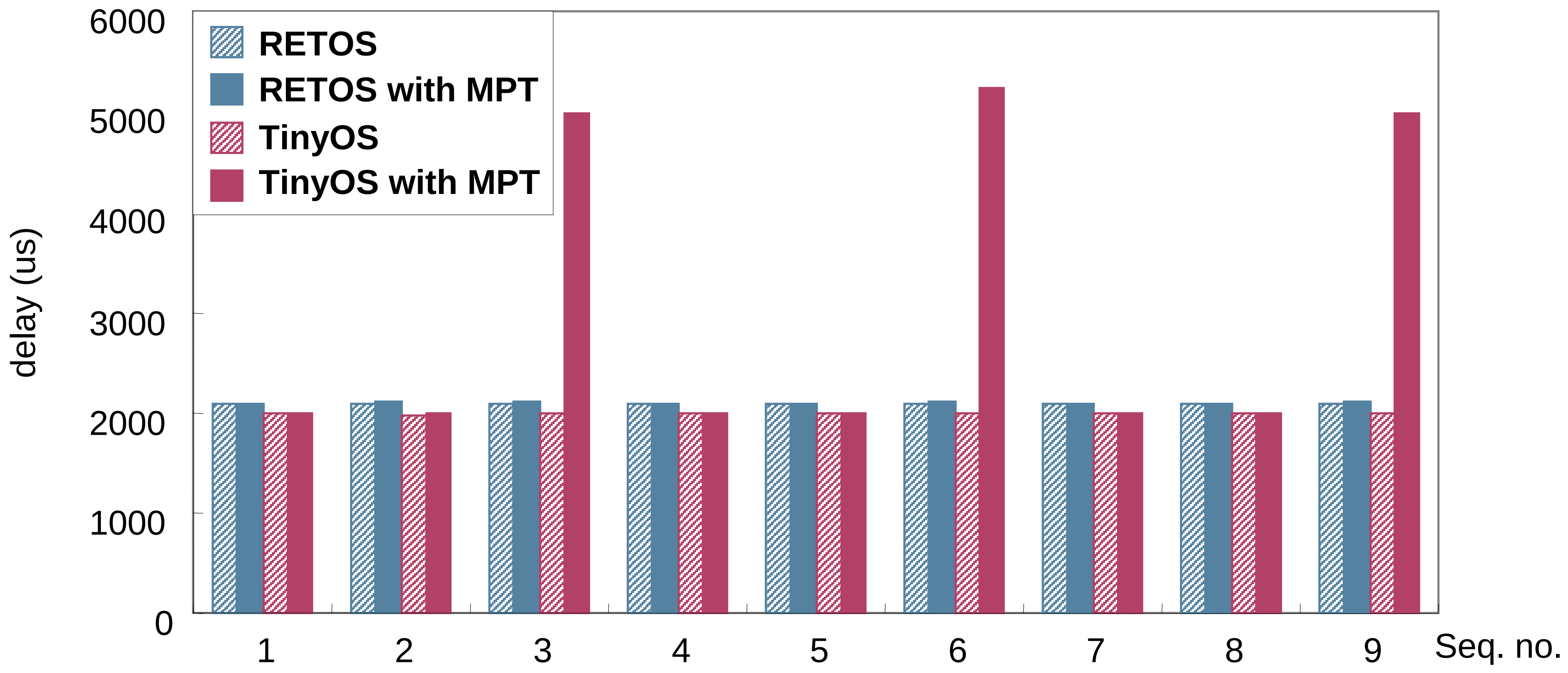


- **Code size for MPT**

	TinyOS (bytes)		RETOS Kernel (bytes)		RETOS Lib. + App. (bytes)		RETOS Total (bytes)	
	ROM	RAM	ROM	RAM	ROM	RAM	ROM	RAM
MPT Backbone	**12,614**	**467**	18,314	748	492	143	**18,806**	**891**
MPT Mobile	**17,222**	**701**	18,314	748	6,848	434	**25,162**	**1,182**

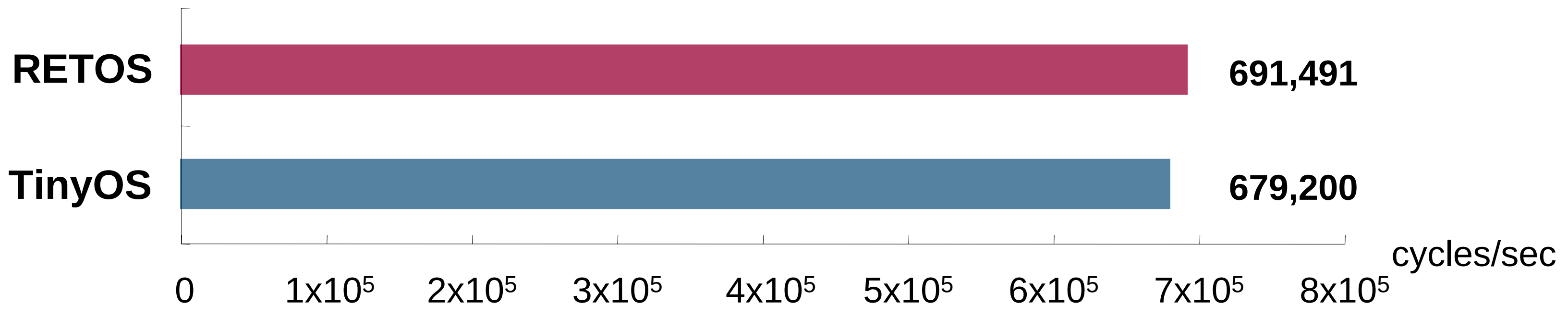
RETOS vs. TinyOS (2)

- **Packet handling delay (MPT + Periodic RF Send/Recv)**
  – RETOS: delay was almost the same whether or not MPT was run
  – TinyOS without MPT: delay was slightly shorter than RETOS
  – TinyOS with MPT: the considerably longer delay was detected periodically

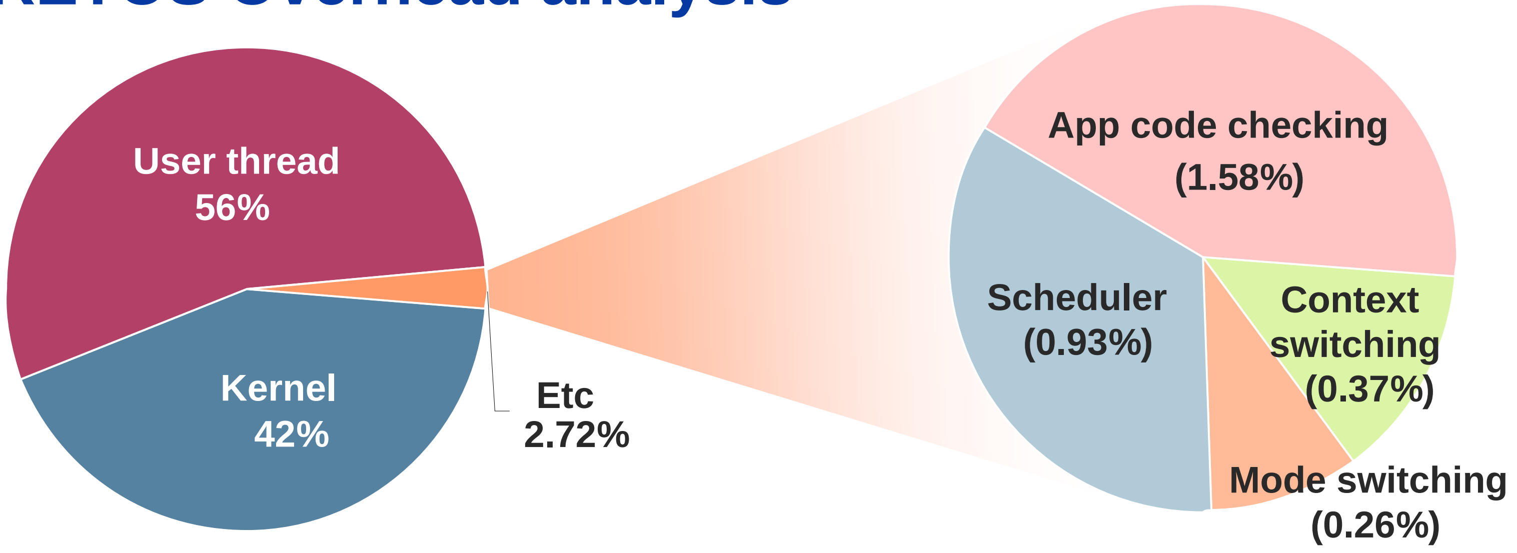


RETOS vs. TinyOS (3)

- Computational Overhead



- RETOS Overhead analysis



Conclusions

- **Optimized multithreading techniques for sensor network operating systems**

  (1) Memory Optimization : **Single kernel stack, Stack-size analysis**

  (2) Energy Reduction : **Variable timer**

  (3) Scheduling Policy : **Event-boosting thread scheduler**

- **Our experiences**
  - The overhead of multithreading is reasonable.
  - The system guarantees minimal response delay to sensor applications without manual configurations.

RETOS Operating System

- **Current status**
  - Ported to TI MSP430, AVR ATmega128 and CC2430
  - System-level power management
  - Monitoring tools
  - Network stack optimization

Safety mechanism

H. Kim, H. Cha, **"Towards a Resilient Operating System for Wireless Sensor Networks",** The 2006 USENIX Annual Technical Conference (USENIX'06), Boston, Massachusetts, June 2006.

Network stack

S. Choi, H. Cha, **"Application-Centric Networking Framework for Wireless Sensor Nodes,"** The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS) 2006, San Jose, California, July 2006.

Loadable modules

H. Shin, H. Cha, **"Supporting Application-Oriented Kernel Functionality for Resource Constrained Wireless Sensor Nodes,"** The 2nd International Conference on Mobile Ad-hoc and Sensor Networks (MSN 2006), Hong Kong, China, December 2006.

Questions