



System Architecture Directions for Networked Sensors

Jason Hill, Robert Szewczyk, Alec Woo,
Seth Hollar, David Culler, Kristofer Pister

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Presented By Camilo Rostoker
CPSC 538a – Topics in Computer Systems
February 28, 2005



Outline

- Introduction
- Network Sensor Characteristics
- Example Design Point
- Tiny Microthreading OS (TinyOS)
- Evaluation
- Related Work
- Architectural Implications

Introduction

- Moore's Law: Push functionality into smaller, cheaper, lower-power units
- Cheap and reliable communications: short-range RF, infrared, optical, etc
- New interesting sensors: light, heat, position, movement, chemical presence, etc
- The Grand View: completely integrated, inch-sized chips containing communications, multiple sensors, CMOS logic...at a low cost
- Result: new design regimes must be established with two main issues in mind – concurrency intensive and efficient modularity

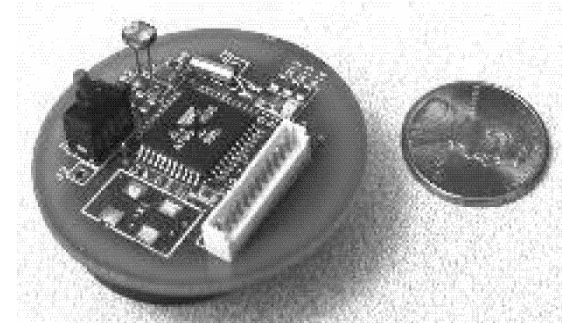


Network Sensor Characteristics

- ❑ Small physical size and low power consumption
- ❑ Concurrency-intensive operation
- ❑ Limited Physical Parallelism and Controller Hierarchy
- ❑ Diversity in Design and Usage
- ❑ Robust Operation

Prototype Design

- Basic processor
 - 8-bit Harvard architecture, 16-bit addresses
 - 32 x 8-bit registers
 - Runs at 4MHz on 3v
 - Can't write to memory
- 8KB flash memory
- 512 bytes of SRAM
- 3 sleep modes: idle, power-down, power-save
- 3 LED's
- Photo-sensor
- Asynchronous radio device, up to 19.2 Kbps (contains no buffering)
- Temperature Sensor
- Serial Port
- Co-processor



Tiny Operating System (TinyOS)

- Similar design problem to efficient network interfaces → requires many concurrent flows and simultaneous events
- Power is the most precious resource
- Event-based approach is a standard for high-performance computing
- Solution: use a multithreading engine with two-level scheduling with event-based interrupts
 - hardware events can be processed by interrupting long-running tasks



TinyOS Design

- Tiny scheduler
- Components
- Each component has four parts:
 - Command handlers
 - Event handlers
 - Frames
 - Tasks

Frames

- ❑ Contains all permanent state for component (lives across events, commands, and threads)
- ❑ Threads, events, and commands execute inside the component's frame
- ❑ Only one per component
- ❑ Like static class variables, not internal class variables.
- ❑ Fixed size
- ❑ Statically allocated at compile time

Commands

- ❑ Commands deposit request parameters into its frame
- ❑ Commands can call lower level commands
- ❑ Commands can post tasks
- ❑ Commands no-blocking
- ❑ Commands needed to return status
- ❑ Declaration:
`TOS_COMMAND (cmd_name) (cmd_args_list)`
- ❑ Usage:
`TOS_CALL_COMMAND (cmd_name) (cmd_args_list)`

Events

- ❑ Deposit information into frame
- ❑ Events can call lower level commands
- ❑ Post tasks and fire higher level events
- ❑ Events can not be signaled by commands

- ❑ Declaration:

```
char TOS_EVENT(evnt_name)(evnt_arg_list)
```

- ❑ Usage:

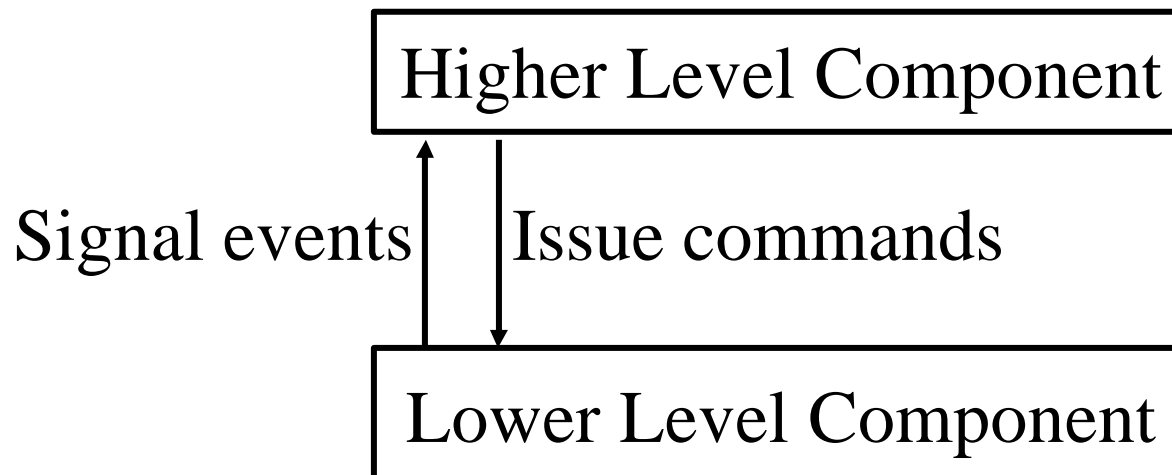
```
TOS_SIGNAL_EVENT(evnt_name)(evnt_arg_list)
```

Tasks

- ❑ Tasks perform work that is computationally intensive.
- ❑ FIFO scheduler
- ❑ Run-to-completion
- ❑ Non-preemptable among tasks (concurrent)
- ❑ Preemptable by events
- ❑ Task declaration:
`TOS_TASK (task_name) { ... }`
- ❑ Posting a task (schedule the task for later execution)
`TOS_POST_TASK (avg_task) ;`

Commands, Events & Tasks

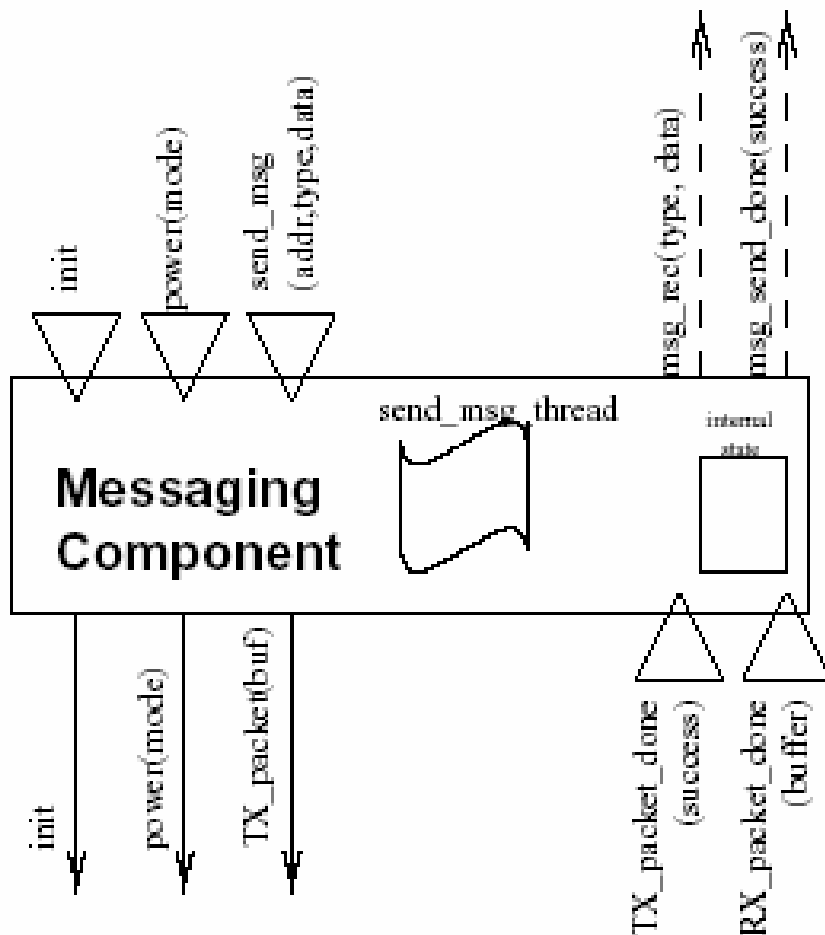
- Both event and command can schedule tasks
- Commands can not signal Events
- Tasks preemptive by Events
- Tasks non preemptive by Tasks



Tiny Scheduler

- Simple FIFO scheduler (dependant on application)
- Scheduler is power-aware: puts processor to sleep when queue is empty, leaves peripherals on so they can wake system
- Once queue is empty, only an event can schedule another task
 - Scheduler can put itself to sleep and wait for a hardware event

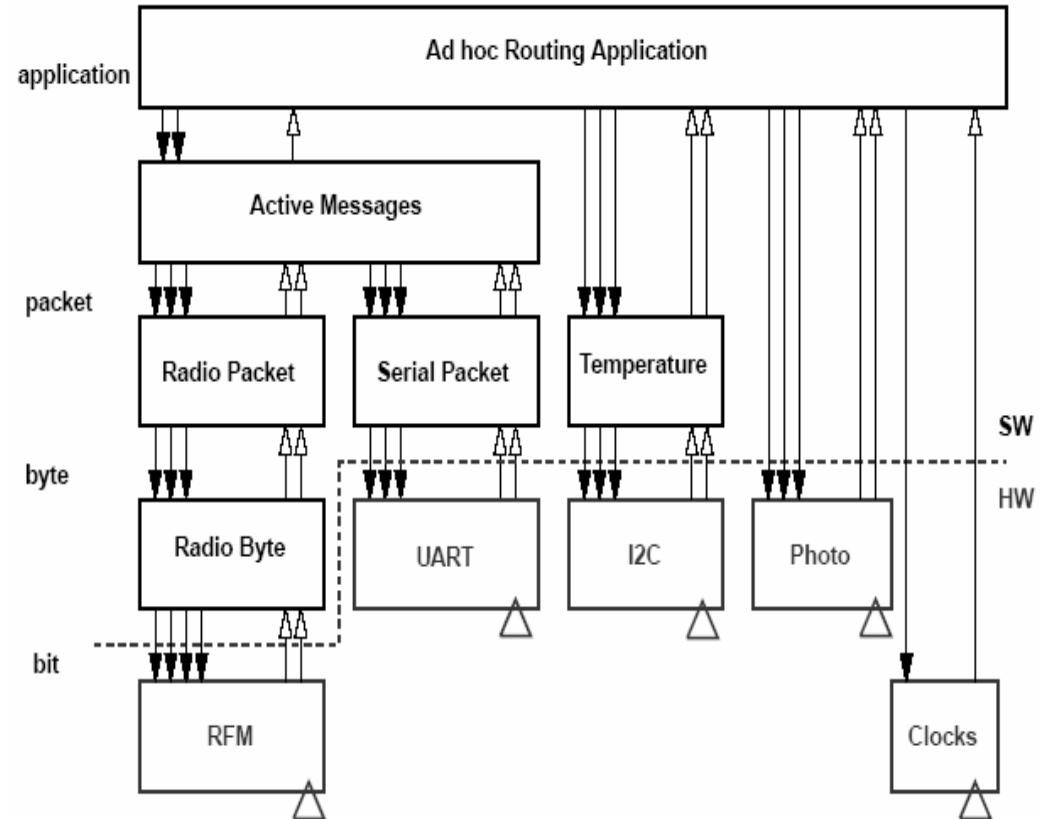
Example Component



```
/* Messaging Component Declaration */  
  
//ACCEPTS:  
char TOS_COMMAND(AM_send_msg)(int addr,int type,  
                               char* data);  
  
void TOS_COMMAND(AM_power)(char mode);  
char TOS_COMMAND(AM_init)();  
  
//SIGNALS:  
char AM_msg_rec(int type, char* data);  
char AM_msg_send_done(char success);  
  
//HANDLES:  
char AM_TX_packet_done(char success);  
char AM_RX_packet_done(char* packet);  
  
//USES:  
char TOS_COMMAND(AM_SUB_TX_packet)(char* data);  
void TOS_COMMAND(AM_SUB_power)(char mode);  
char TOS_COMMAND(AM_SUB_init)();
```

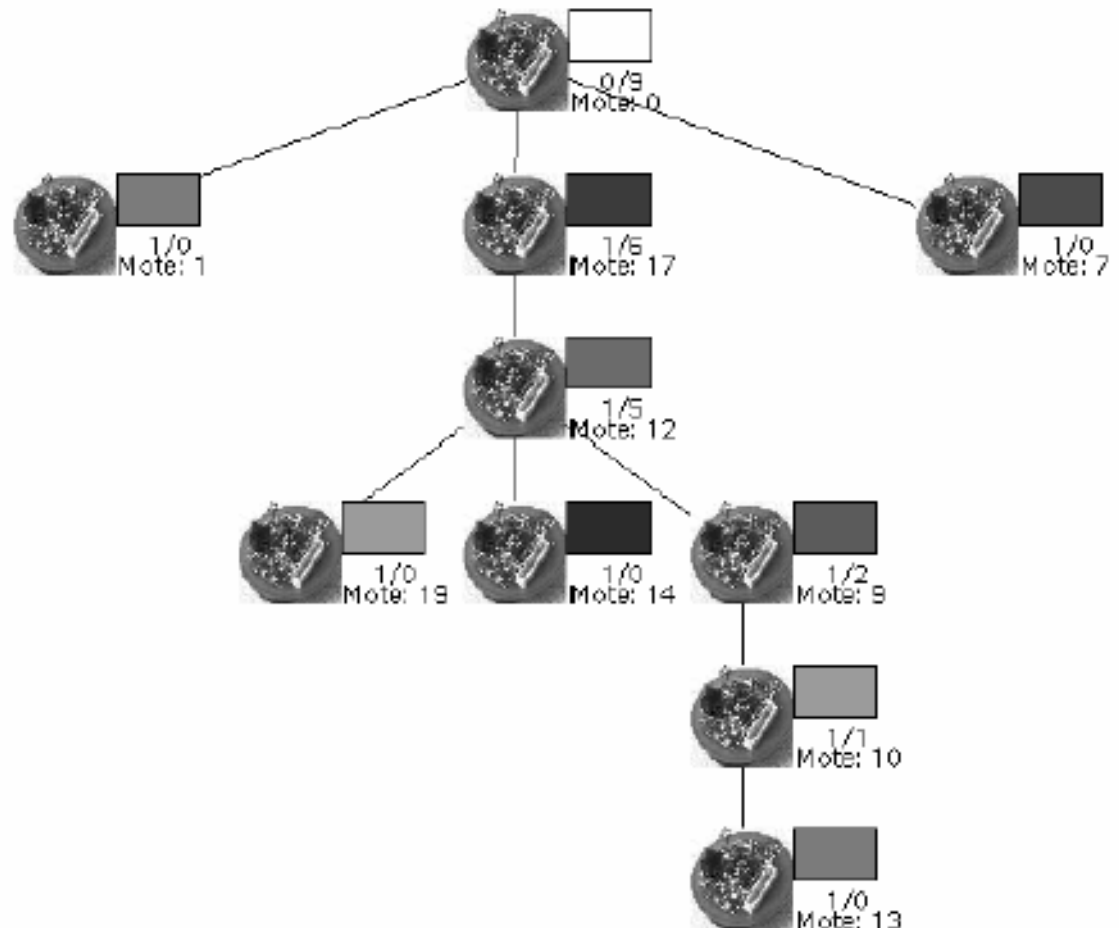
Component Types

- Hardware Abstractions
 - Maps physical hardware into component model
- Synthetic Hardware
 - Simulates behaviour of advanced hardware
- High Level Software Components
 - Perform control, routing and data transformations



Putting it all together

- prototype application:
multiple sensors
distributed in localized
area
- 3 I/O devices:
temperature & light
sensors and network
- periodically sending data
to a base station
- Base station periodically
broadcasts route updates



Evaluation

- Small physical size
 - Table shows code/data size of components
 - Uses only 226 bytes of data → under 50% of 512 bytes available
 - requires only 3KB of instruction memory!

Component Name	Code Size (bytes)	Data Size (bytes)
Multihop router	88	0
AM_dispatch	40	0
AM_temperature	78	32
AM_light	146	8
AM	356	40
Packet	334	40
RADIO_byte	810	8
RFM	310	1
Photo	84	1
Temperature	64	1
UART	196	1
UART_packet	314	40
I2C_bus	198	8
Processor_init	172	30
TinyOS scheduler	178	16
C runtime	82	0
Total	3450	226

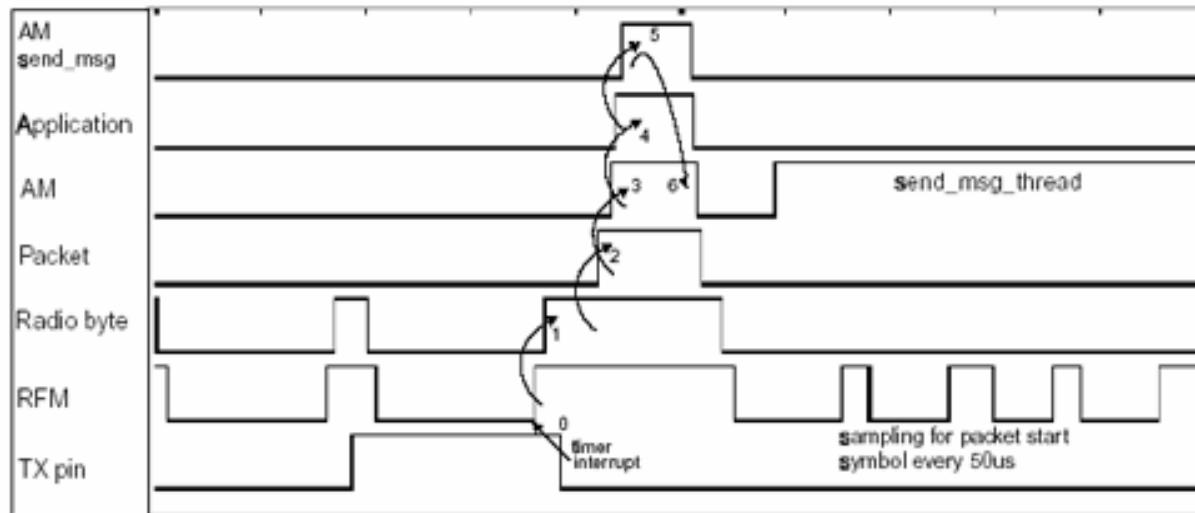
Evaluation (2)

- Concurrency-intensive operations
 - Multiple flows → context switch speed is important!
 - Most expensive operations are interrupts → Partition the register set or user register windows
- Efficient Modularity
 - Total propagation time up five-layer radio communication stack is 40 μ s or 80 instructions

Operations	Cost (cycles)	Time (μ s)	Normalized to byte copy
Byte copy	8	2	1
Post an Event	10	2.5	1.25
Call a Command	10	2.5	1.25
Post a task to scheduler	46	11.5	6
Context switch overhead	51	12.75	6
Interrupt (hardware cost)	9	2.25	1
Interrupt (software cost)	71	17.75	9

Evaluation (3)

- Limited physical parallelism and controller hierarchy



- Diversity in usage and robust operation
 - Tested in multi-hop routing, active-badge-like location detection applications, sensor network monitoring applications
 - Developed in C → target multiple CPU architectures

Related Work

- Traditional RT embedded OS's include VxWorks, WinCE, PalmOS, QNX
 - Good for PDA's, cell phones, set-top-boxes, but do not meet new design requirements

Name	Preemption	Protection	ROM Size	Configurable	Targets
pOSEK	Tasks	No	2K	Static	Microcontrollers
pSOSystem	POSIX	Optional		Dynamic	PII → ARM Thumb
VxWorks	POSIX	Yes	≈ 286K	Dynamic	Pentium → Strong ARM
QNX Neutrino	POSIX	Yes	> 100K	Dynamic	Pentium II → NEC chips
QNX Realtime	POSIX	Yes	100K	Dynamic	Pentium II → 386's
OS-9	Process	Yes		Dynamic	Pentium → SH4
Chorus OS	POSIX	Optional	10K	Dynamic	Pentium → Strong ARM
Ariel	Tasks	No	19K	Static	SH2, ARM Thumb
CREEM	data-flow	No	560 bytes	Static	ATMEL 8051

Architectural Implications

- Individual microcontroller for each I/O device?
 - Tradeoff's between power consumption, chip speeds, flexibility, etc
- Analysis showed hardware support for event would be beneficial
- Reconfigurable (dynamic) systems for versatility
 - Again, introduces many tradeoffs between efficiency and power consumption, etc
- Strong tie between software execution model and hardware execution that supports it

References

- System Architecture Directions for Networked Sensors. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)
- Presentation slides on “System Architecture Directions for Networked Sensors”. Qihua Cao.
http://www.cs.virginia.edu/~qc9b/cs851/SADofNS_2.ppt