# TinyOS

Arvind Easwaran (arvinde@seas)

# Outline

- TOS Architecture
  - Challenges
  - Features
- Subsystems
  - Scheduler
  - Active Messaging
  - TinyDB
  - Virtual Machine – Mate, Bombilla
  - TinySEC

# Outline

- **TOS Architecture**
  - **Challenges**
  - **Features**
- Subsystems
  - Scheduler
  - Active Messaging
  - TinyDB
  - Virtual Machine – Mate, Bombilla

# TinyOS - Challenges

- Power Efficiency
- Modularity
  - Diversity in Design and Usage
- Security
  - Insecure wireless communication
  - Negligible computation power for cryptography
  - Minimal overhead data
- Process Management
  - Unacceptable context switch overhead
    - Time, Storage
  - Thread driven approach
    - Power intensive

# Challenges Contd.

- Limited physical parallelism

- Data Management
  - Files
    - Conventional file systems – Unix etc
      - Resource consuming
  - Databases
    - RDBMS ?

- Network Management
  - Sensor nodes - communication oriented
  - Concurrency intensive
  - No point to point routing → Multi hop networks
  - Minimal packet overhead
  - TCP/IP will not suffice

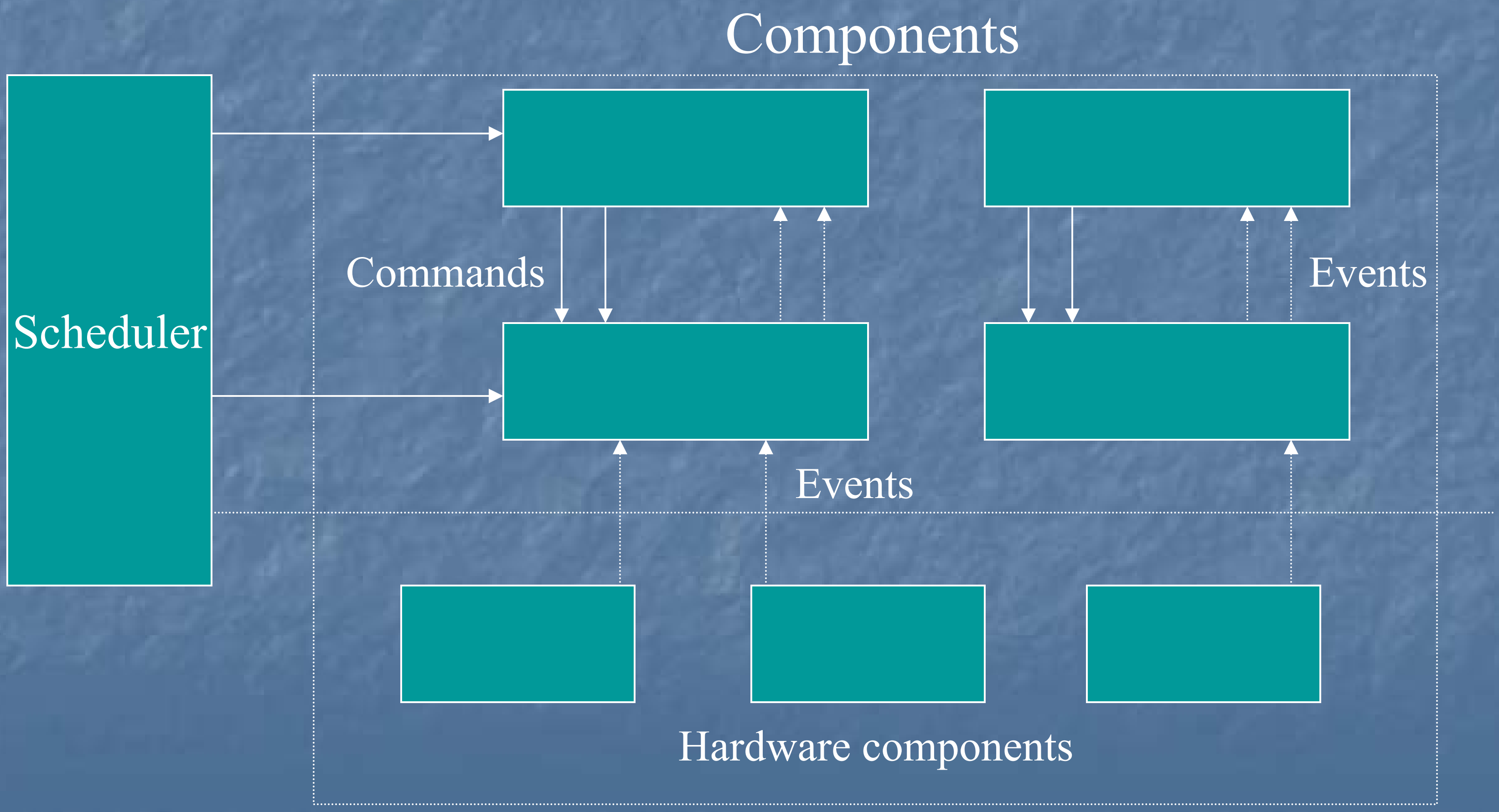
# TinyOS - Features

- Event-driven architecture
  - Lower layer sends events to higher layer
  - Low overhead – No busy-wait cycles
- Interrupt driven → Two kinds of interrupt
  - Clock
  - Radio
- Component driven programming model
  - Size - 400 bytes
  - Extremely flexible component graph
- Single-shared stack

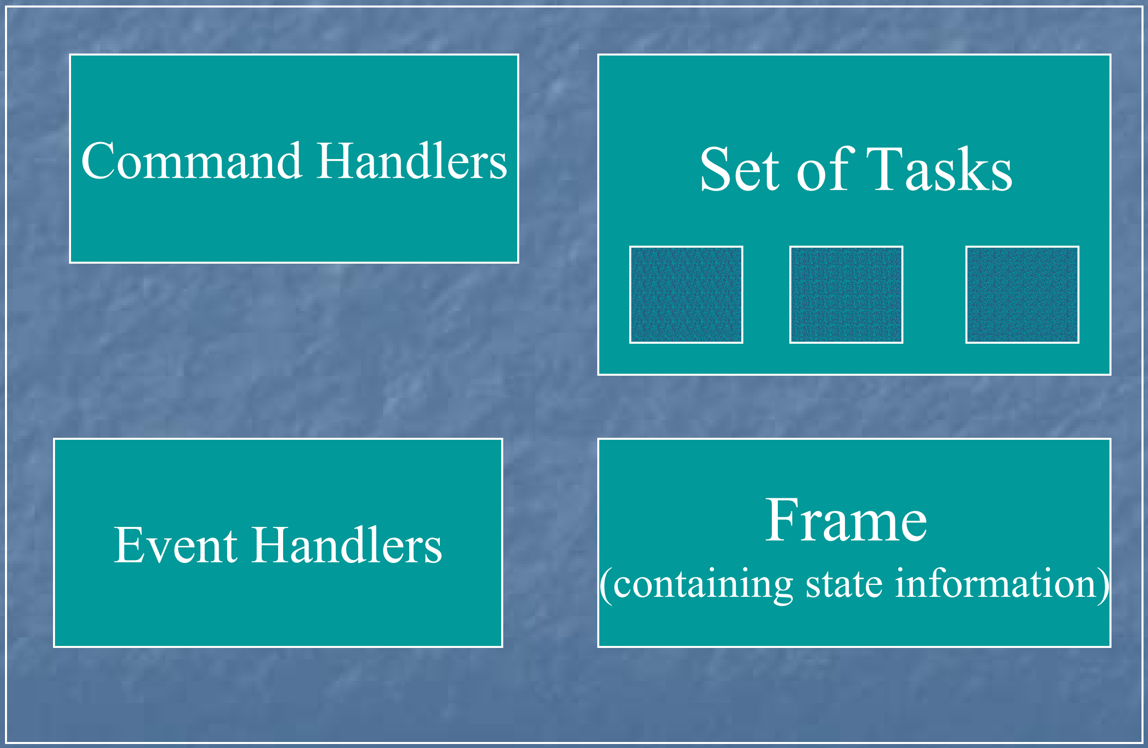
# Features Contd.

- Network management - Active Messaging
- No kernel, process management, virtual memory
- File management - Matchbox
- 2-level FIFO scheduler – events and tasks
- Complete integration with hardware

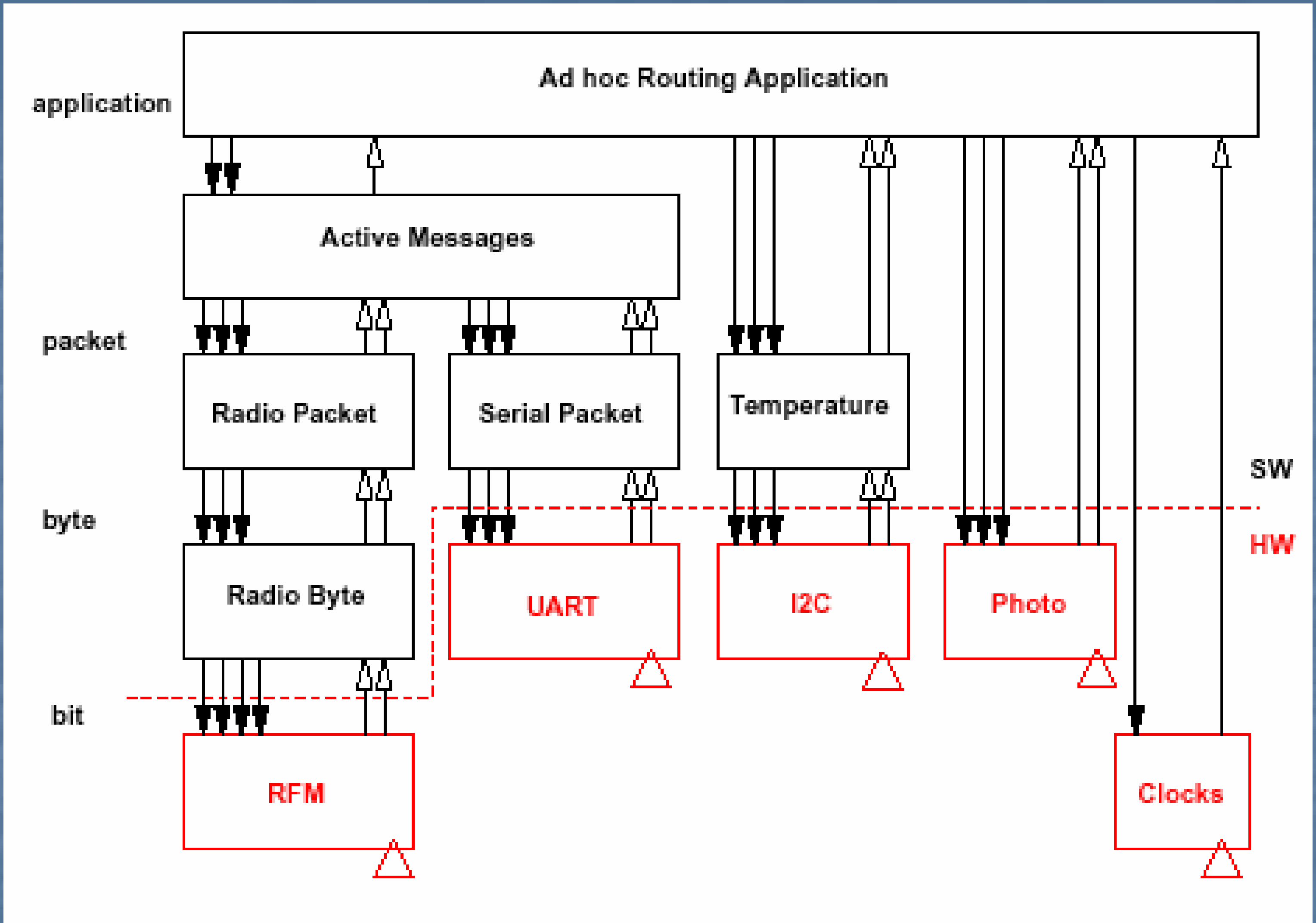
# TinyOS - Design



# Structure of a Component



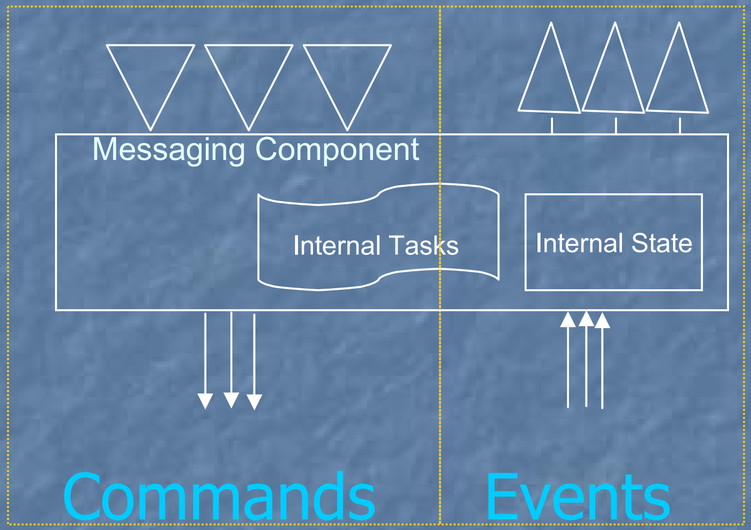

TinyOS Component




Sample Application shown with components

# Programming Model : Review

- Component interface
  - Commands accepted (implemented)
  - Commands used
  - Events accepted (implemented)
  - Events used
- Component implementation
  - Handlers
  - Tasks
- Component description
  - Component graph



# Component Interface

<CompName>**.comp**

```
TOS_MODULE <CompName>;
ACCEPTS {
	// command_signatures
};
HANDLES {
	// event_signatures
};
USES {
	// command_signatures
};
SIGNALS {
	// event_signatures
};
```

# Component Implementation

<CompName>**.c**

```
#define TOS_FRAME_TYPE
TOS_FRAME_BEGIN(< CompName >_frame) {
    // state declaration
}
TOS_FRAME_END(< CompName >_frame);

char TOS_COMMAND(<command_name)(){
    // command implementation
}

char TOS_EVENT(<event_name>)(){
    // event implementation
}
```

# Component Description

<CompName>**.desc**

```
INCLUDE {
        MAIN;
        <CompName>;
        <Comp_I>;
        <Comp_J>;
        …
};

// Wiring
<CompName>.<command>     <Comp_I>.<command>
…
<CompName>.<event>       <Comp_J>.<event>

…
```

# TOS - Issues

- Programming perspective
  - No memory protection -> easy to corrupt/crash the system
  - Heavy use of macros
- System perspective
  - Simplistic FIFO scheduling -> no real-time guarantees
  - Bounded number of pending tasks
  - No "process" management -> resource allocation
  - Software level "bit manipulation"

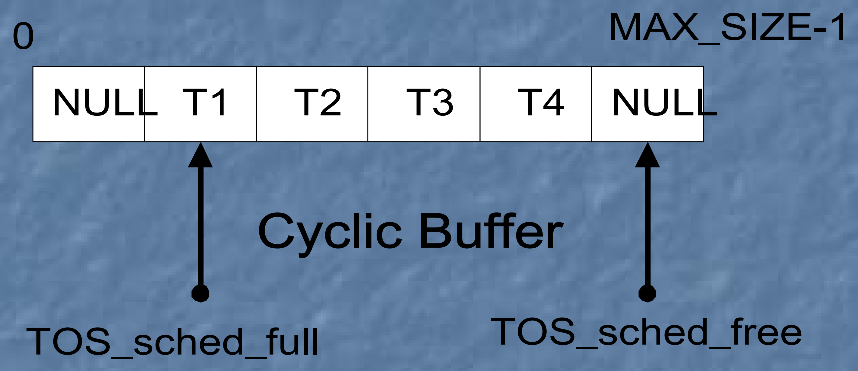
# Outline

- TOS Architecture
  - Challenges
  - Features
- Subsystems
  - **Scheduler**
  - Active Messaging
  - TinyDB
  - Virtual Machine – Mate, Bombilla

# TOS – Scheduling

- Scheduling
  - 2-level scheduling (events and tasks)
  - FIFO scheduler – Queue of size 7
- Tasks
  - Preempt able by events
  - May call commands, events
  - Not preempted by other tasks
- Events
  - Hardware interrupt supported lowest level events

# The FIFO Scheduler




```
int Exec_Next_Task(){

        if (TOS_sched_full == TOS_sched_free) return -1;

        TOS_queue[TOS_sched_full].tp(); // execute the task

        TOS_queue[TOS_sched_full].tp = 0; //remove the task

        TOS_sched_full = (TOS_sched_full +1 == MAX_TASKS  ) ? 0 :
        TOS_sched_full +1;   //increment TOS_sched_full

        return 0;  }
```

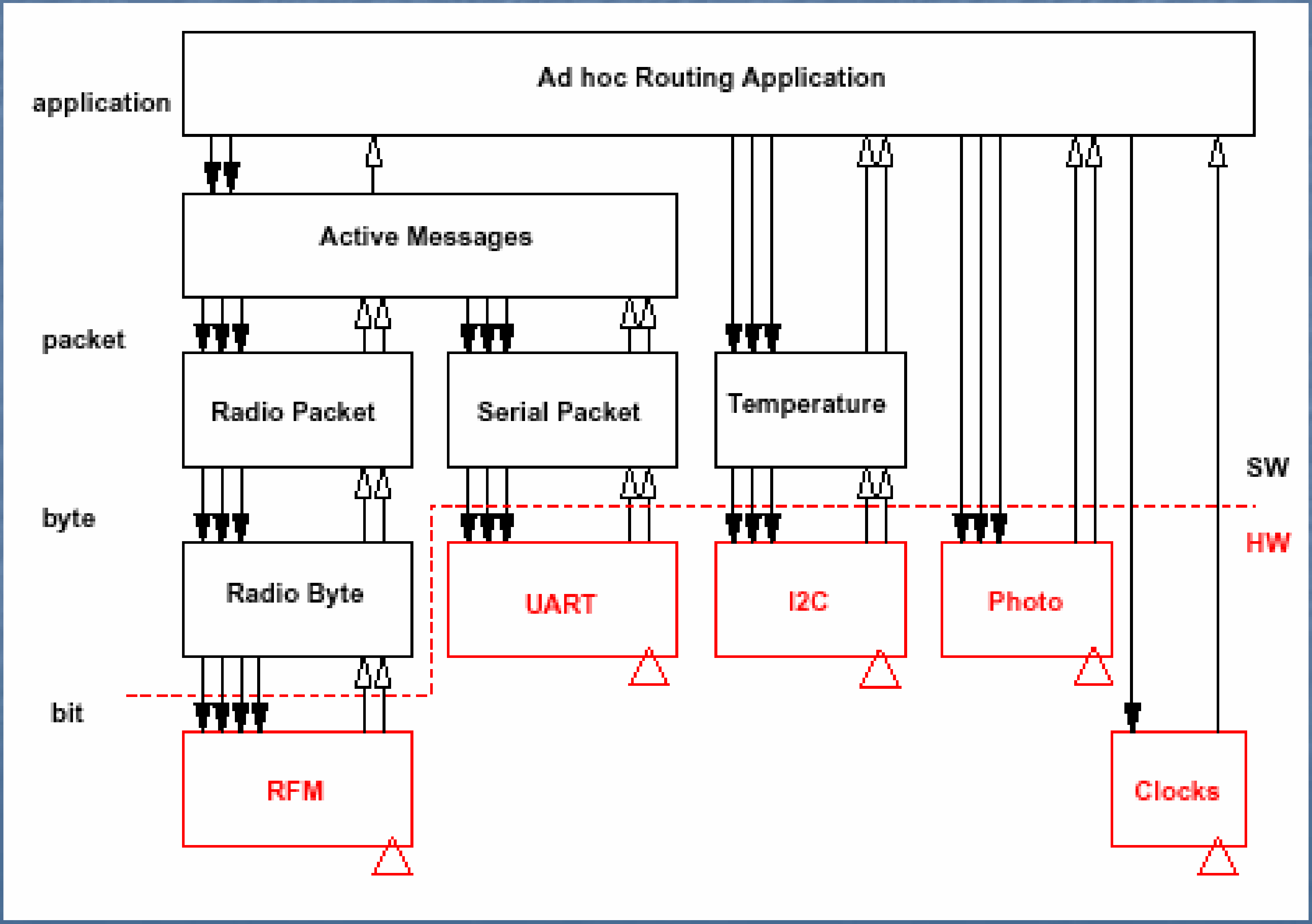
# Prioritized scheduling - Motivation

- Sensor Node tasks
  - Receive packets for forwarding
  - Send packets received for forwarding
  - Process locally sensed data and send it
- Local Processing
  - Raw Data sent to Base Station
  - Aggregation of data done

# Motivation contd.

- Raw Data sent to Base Station
  - Increased Network traffic
  - Rate of transmission >> network capacity
- Aggregation of data done
  - Volume of data high
  - Not enough computational capability
- Handling overload conditions
  - Determine criticality of tasks
  - Prioritize on criticality

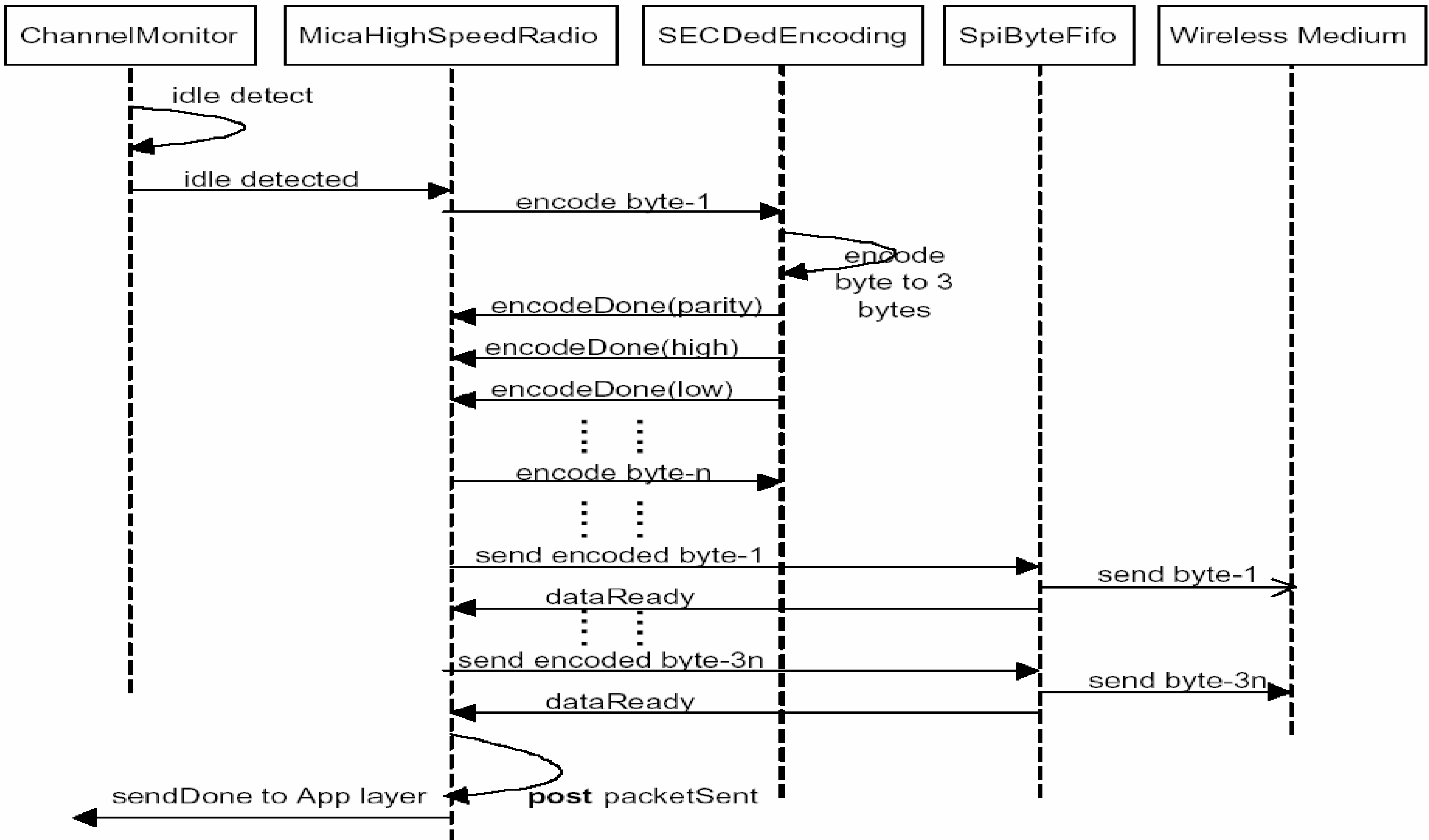
# Example – Radio Stack



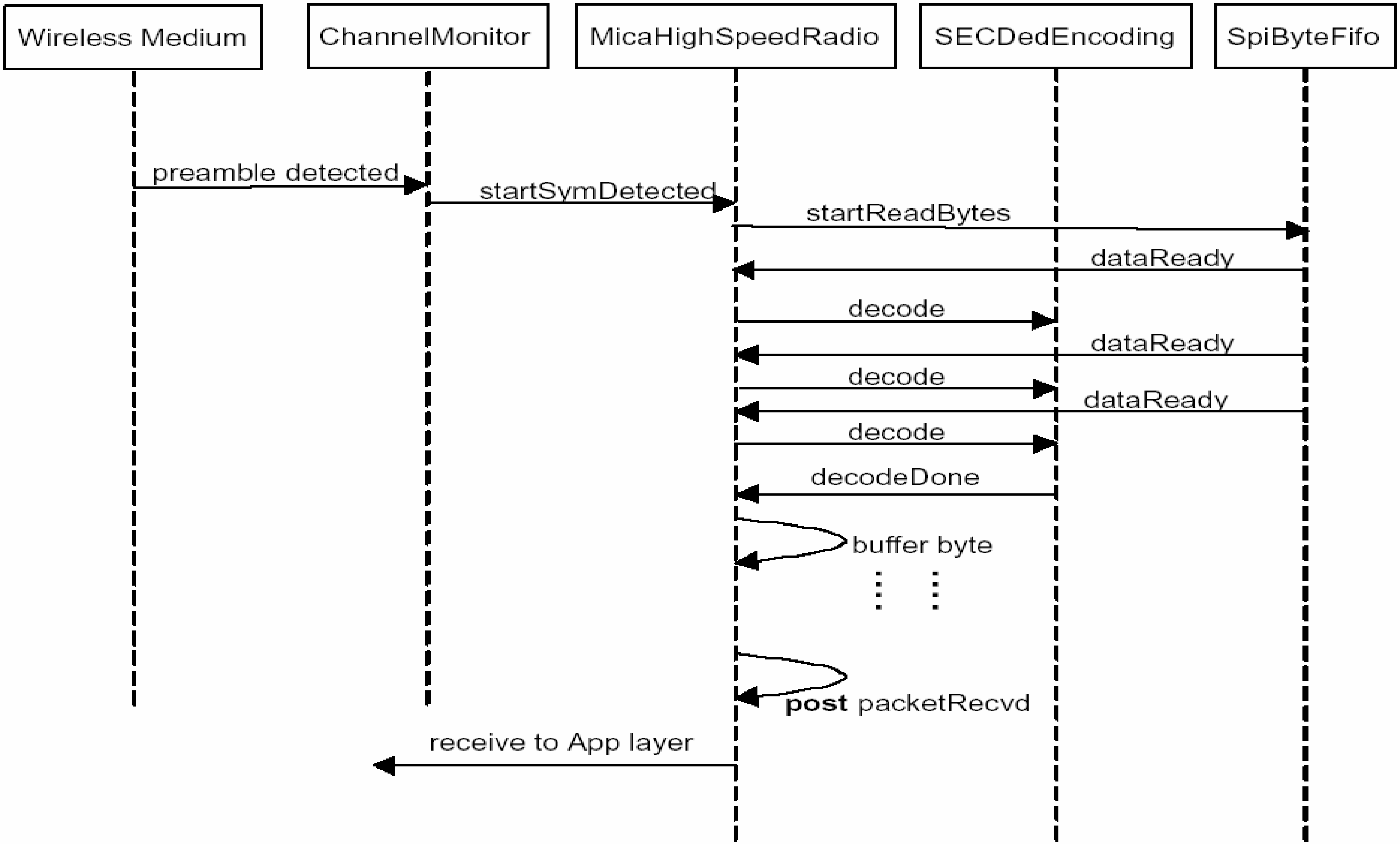
- Sequence of events
  - Radio bits received by node - RFM
  - Radio bits converted to bytes – RadioByte
  - Bytes to packets – RadioPacket
  - Packets to Messages – Active Messages
- CPU involved in processing every interrupt
  - Every radio bit processed by CPU
  - Lack of network interface processor
  - 2 MIPS – CPU
- Interrupts have higher priority
- **Tasks preempted for every bit received**

- High rate of radio bit interrupts
  - No tasks get executed
  - Receiver overload – live lock
- Bit interrupts can post tasks → Forwarding
  - Interrupts prevent tasks from executing
  - They also add tasks to the queue
  - None get executed → No forwarding !!
- Task queue is full (limited size)
- New tasks (critical ??) ignored

# Packet Send Protocol



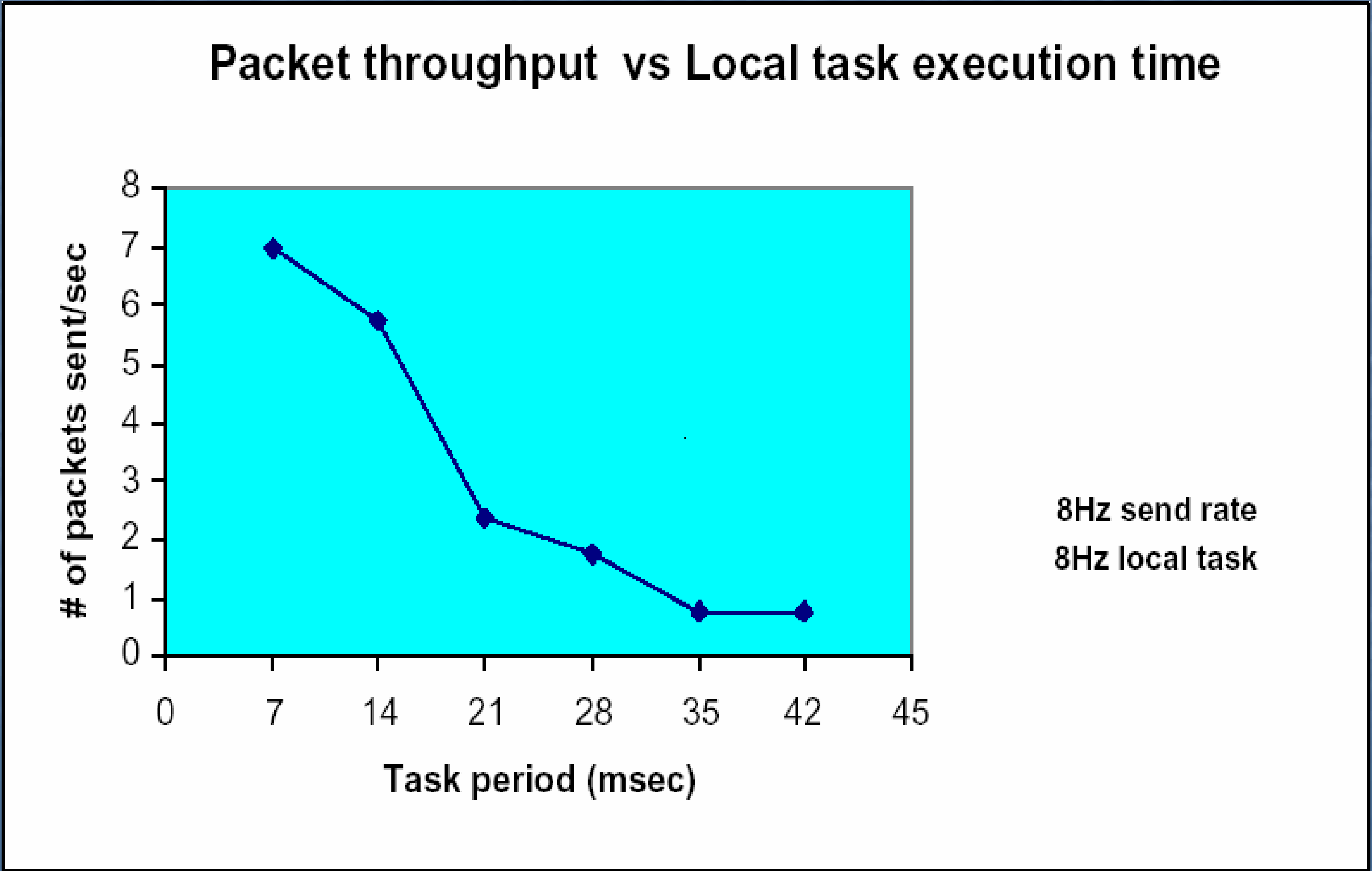
# Packet Receive Protocol



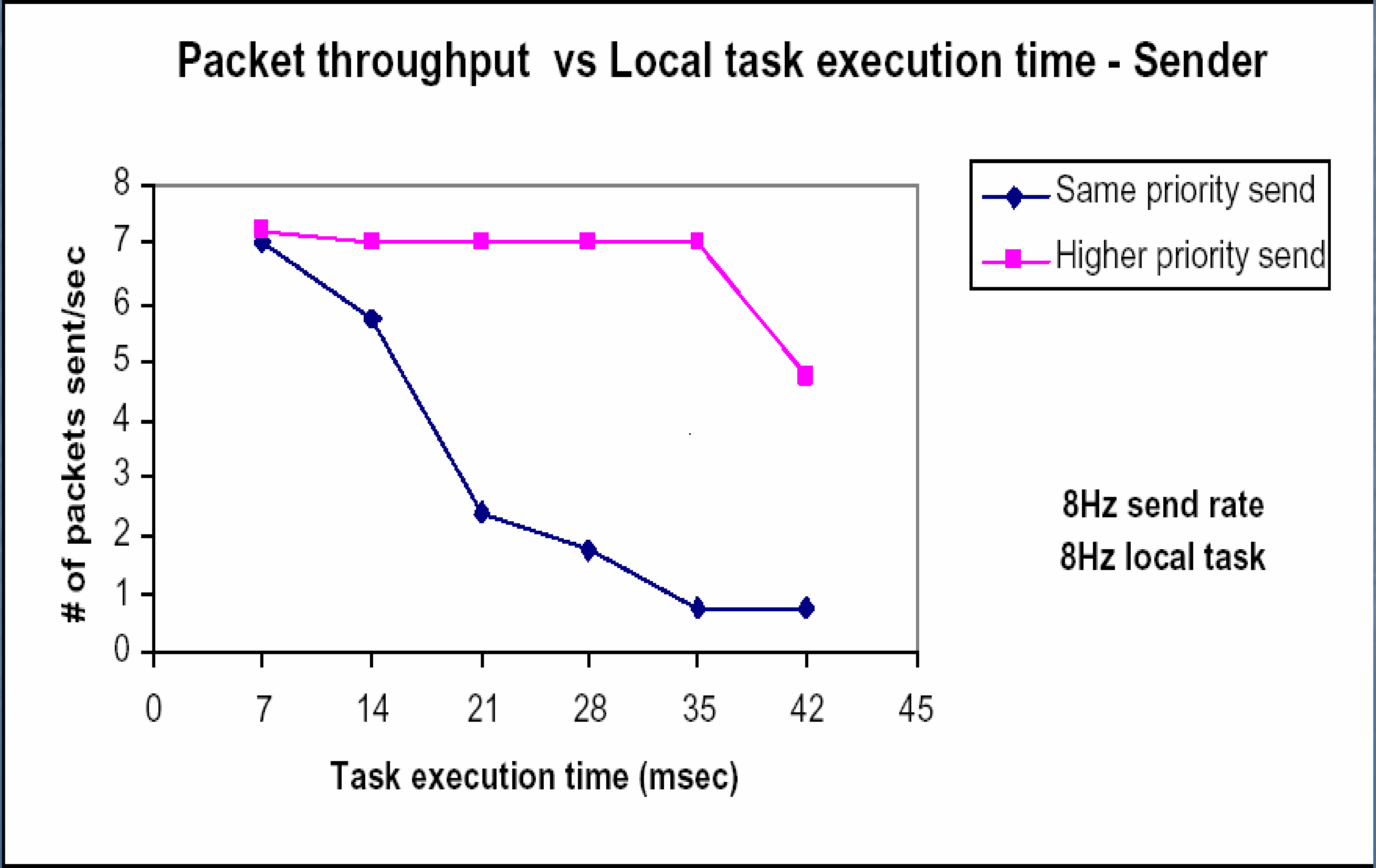
# Prioritized scheduling

- Each task given priority
  - Incorporated in the programming model
- Send/Receive, Encryption tasks given higher priority
- Higher priority task inserted ahead in the FIFO queue
- Queue full → Lower priority posted task dropped
- Semantics of task post modified

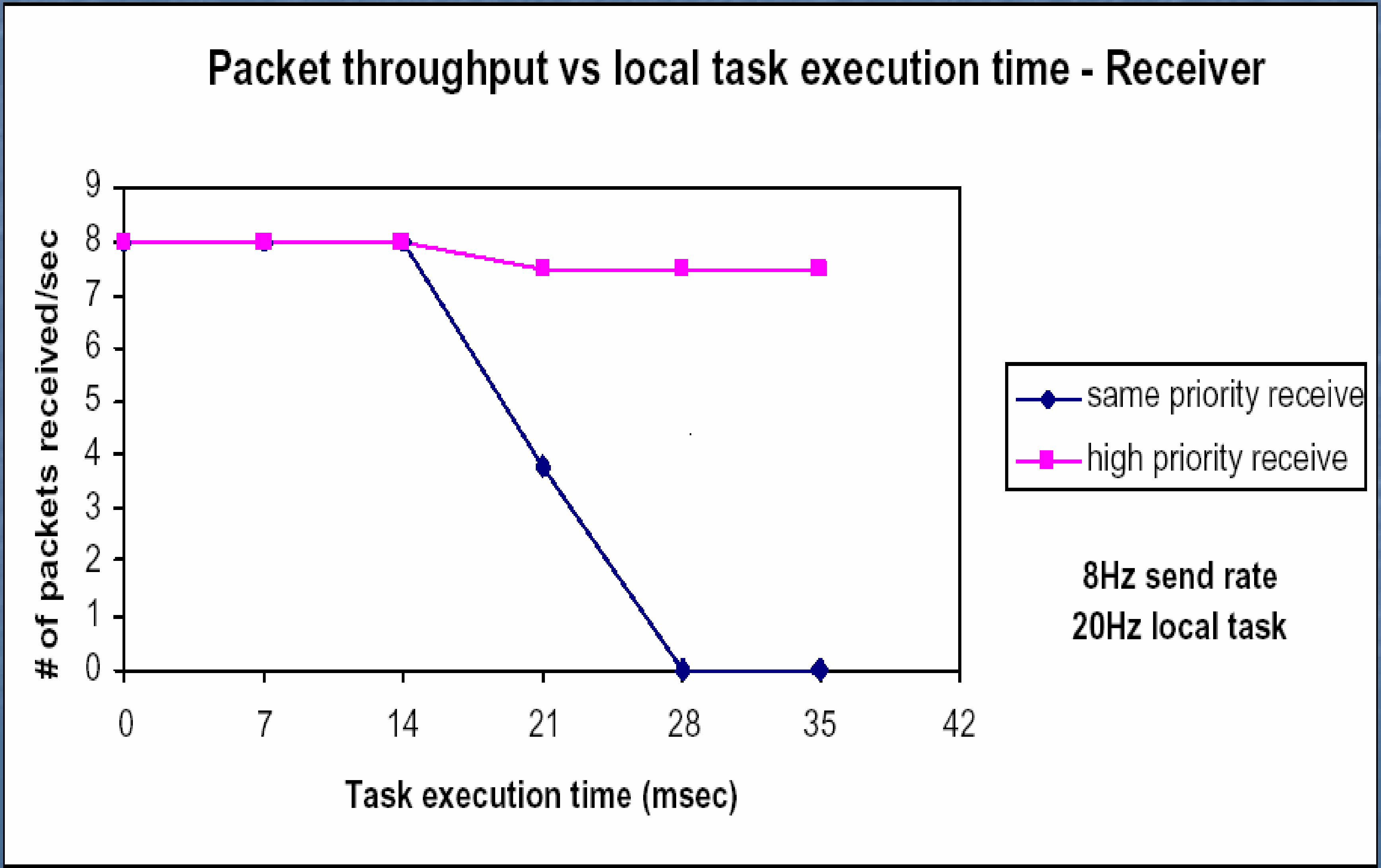
# Throughput – FIFO scheduler



# Throughput – Prioritized Send



# Throughput – Prioritized Receive



# Outline

- TOS Architecture
  - Challenges
  - Features
- Subsystems
  - Scheduler
  - **Active Messaging**
  - TinyDB
  - Virtual Machine – Mate, Bombilla

# Active Messages : Motivation

- Sensor nodes – Communication requirements
  - Communication intensive
  - No point to point routing → Multi hop networks
    - Power conservation
  - Minimal packet overhead
  - Efficient in memory, processor, power
  - Tolerant to high level of concurrency

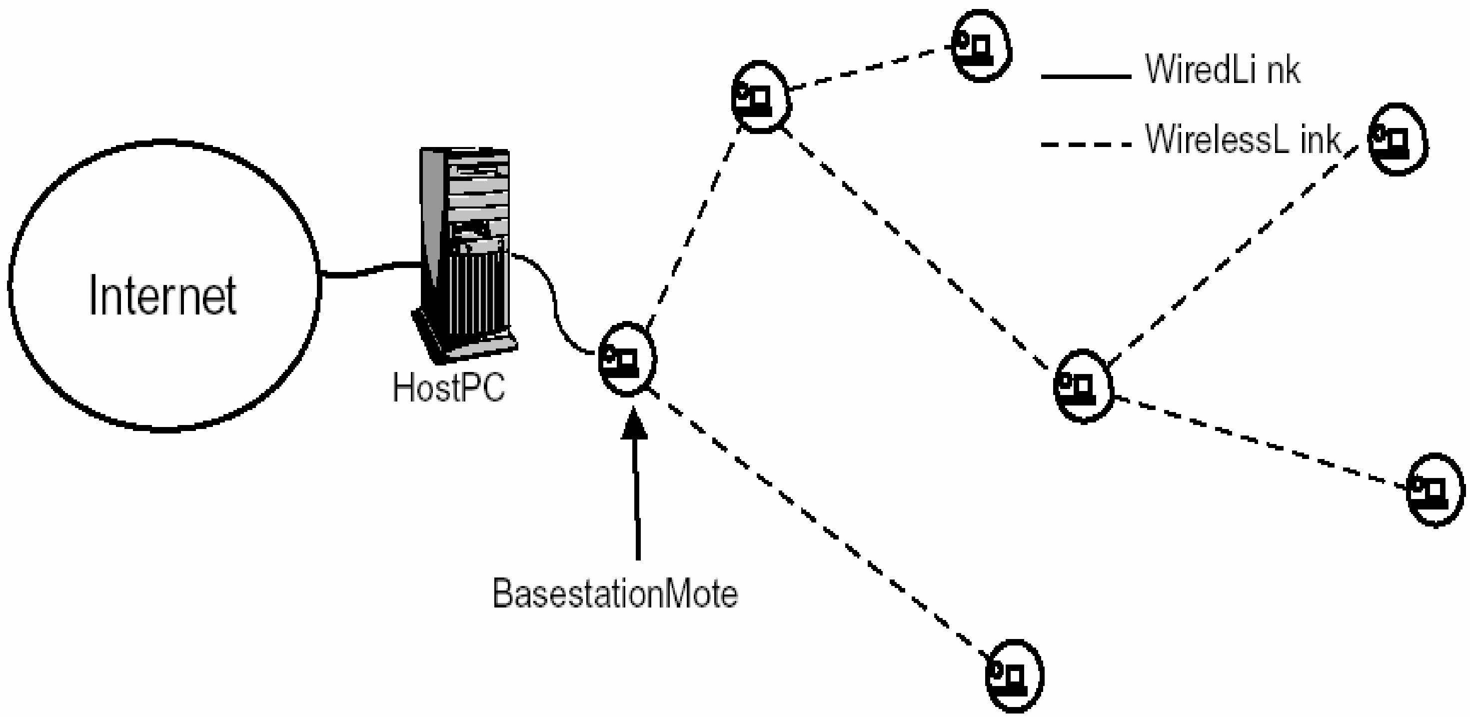
# Motivation Contd.

- Sensor nodes – Communication requirements
  - Real time constraints
  - Almost no physical parallelism available
  - Dynamic deployment → ad hoc network formation
  - RF interference
  - Mobile → Node failures
  - Highly modular communication subsystem

# Motivation Contd.

- Conventional network protocols
  - TCP/IP, sockets, routing protocols
    - Bandwidth intensive → Acknowledgements
    - High overhead per packet → Headers
    - Centered on "stop and wait" semantics
    - High memory requirements/ Computational power demands
    - Sockets not suited to constrained TOS environment

# Example Sensor Network



# Active Messages

- Simple, extensible paradigm
- Widely used in parallel and distributed systems
- Integrating communication and computation
- Distributed event model where networked nodes send events

# Active Messages : Basic structure

- Light weight architecture
- Each Active Message contains
  - User-level handler to be invoked on arrival
  - Data payload passed as argument
- Event-centric nature
  - Enables network communication to overlap with sensor-interaction

# Active Messages : Basic structure

- Handler functions
  - Extract message quickly from network
  - Provide data for computation/forward data
  - Prevent network congestion
- Minimal buffering → Pipeline analogy
  - Quick execution of handlers prevents use of send/receive buffers

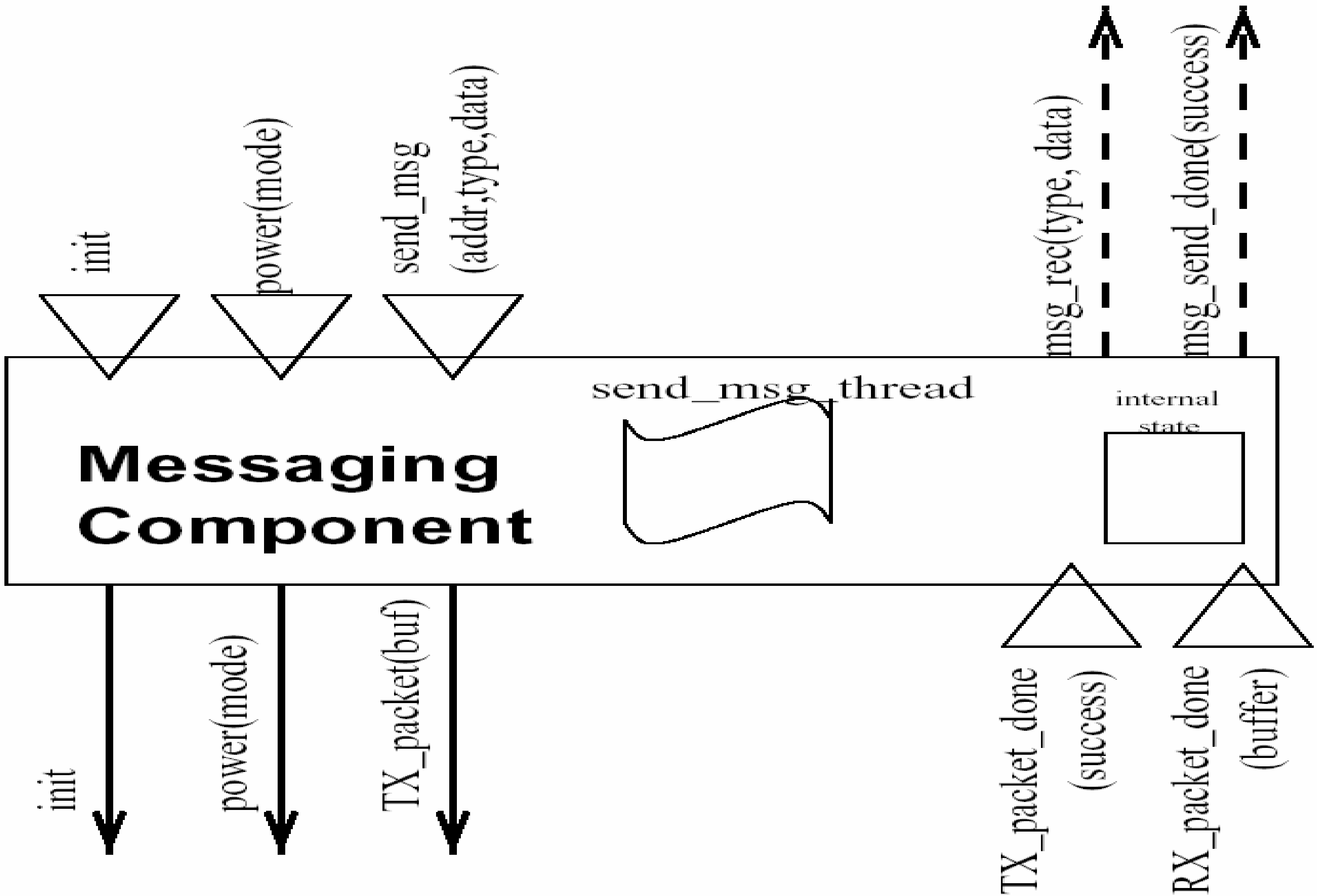
# Tiny Active Messages

- Three basic sufficient primitives
  - Best effort message transmission
  - Addressing → Address checking
  - Dispatch → Handler invocation
- Components provide modularity
  - Applications choose between types/levels of error correction/detection
- Consistent interface to communication primitives
  - Portability to hardware platforms

# Tiny Active Messages

- Applications can have additional components
  - Flow control
  - Encryption
  - Packet fragmentation
- Event based → Threaded
  - Simple → FIFO queue
- Use of buffers possible but not mandatory
  - Applications can create their own

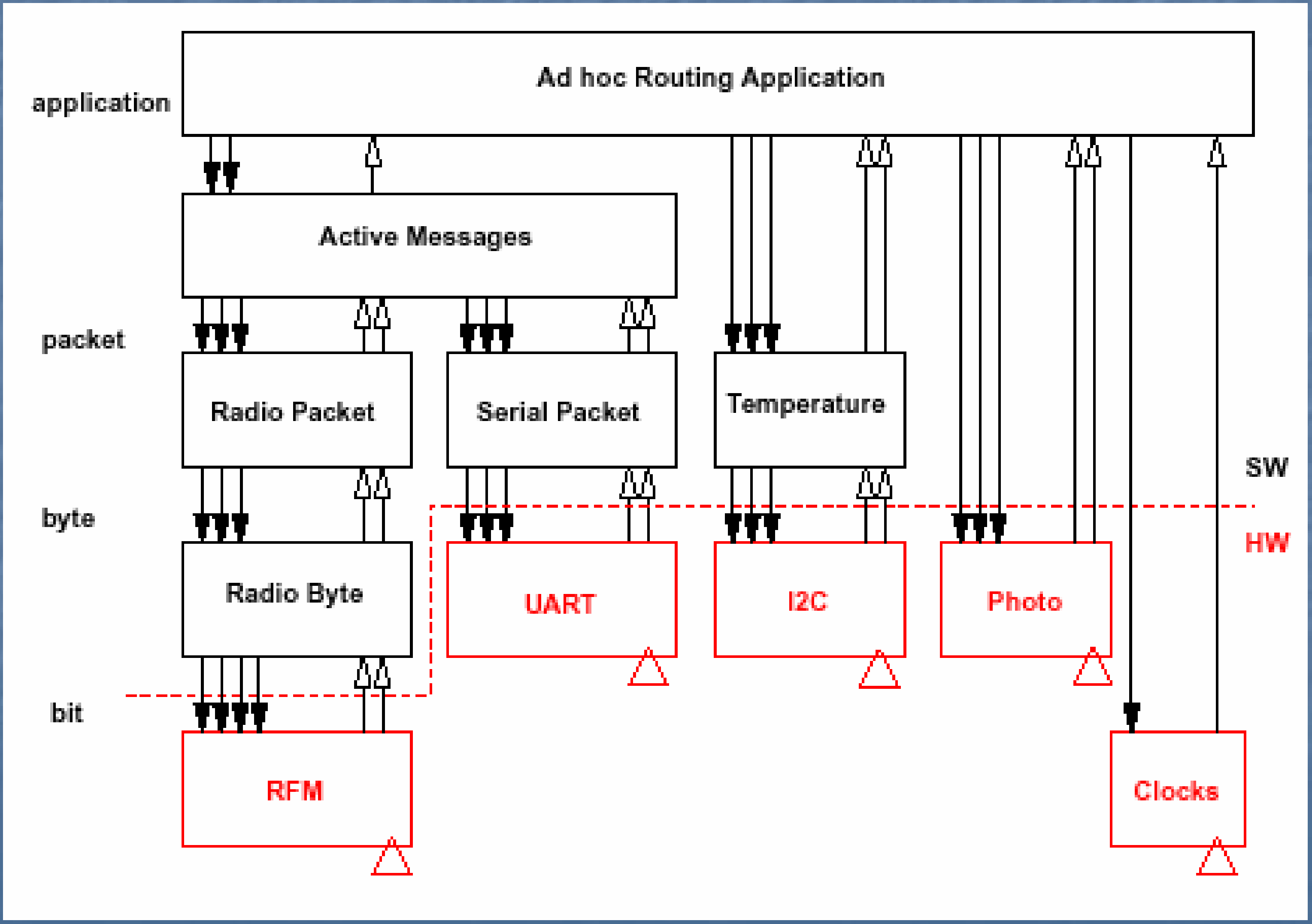
# Tiny Active Messages



# Tiny Active Messages - Component

- Accepts TOS commands from application
- Fires events to message handlers
- Event to signal completion of transmission
- Send command includes
  - Dest. Address, Handler ID, Message body
- Address checking and dispatching
- Relies on components for packet transmission

# Example – Radio Stack



# Component functions

- RFM, RadioByte, RadioPacket
  - Best effort transmission mechanism
- Active Messages → Error correction component
  - Basic → None
  - CRC → Error detection
  - Error corrected packets → Correction and detection
- Host PC package
  - Communicates to base station through serial port
  - Simple bridge to get data to the internet

# Packet Format

- Byte 1 → Destination address (R_0)
- Byte 2 → Message handler (H_0)
- AM component
  - Address match
  - Handler invocation
  - Remaining 28 bytes → Message body passed as argument to handler
- Dispatch routine for handlers statically linked

# Active Messaging - Example

- Ad hoc networking application
- Collects information from nodes randomly distributed
- Routing topology explored using Active Message primitives
- Automatic re-configuration with new routing topology
- Application closely mirrors real world sensor applications
- DSDV algorithm used

# Multi-Hop Packet Format

| $R_0$ | $H_0$ | N | $H_f$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | S | $D_0$ | $D_1$ | . . . |
|---|---|---|---|---|---|---|---|---|---|---|---|

$R_0$ - Next Hop
$H_0$ - Next Handler
N - Number of Hops
$H_0$ - Destination Handler
$R_1$, $R_2$, $R_3$, $R_4$ - Route Hops
S - Sending Node
$D_0$, $D_1$ ... - Payload

- 4-hop communication → 7 extra bytes
- H_0 set to 0
- At each hop routing handler
  - Decrements hop count
  - Rotates next hop, pushes current address to end
- If next hop is final destination (N = 1)
  - H_F moved to H_0

# Route Discovery

- Broadcast address
  - Useful for route discovery
  - Application sends a 2-hop packet to broadcast address followed by self-address
  - Returned packet contains address of neighbors
  - Efficient communication with neighbors

# Routing Topology Discovery

- Base station periodically broadcasts its identity
- Shortest path discovery done from every node to BS
- Cycles prevented using epochs
- Identity of route stored in packet
  - To generate statistics
- Message types
  - Routing message → Update message handler
  - Forwarding message → Data message handler
  - Clock event → Sensing and sending data

# Evaluation

- Round Trip Time

# Evaluation

- Power consumption

| | |
|---|---|
| Idle State | 5 $\mu$Amps |
| Peak | 5 mAmps |
| Energy per bit | 1 $\mu$Joule |

TABLE II

POWER AND ENERGY CONSUMPTION MEASUREMENTS.

# Outline

- TOS Architecture
  - Challenges
  - Features
- Subsystems
  - Scheduler
  - Active Messaging
  - **TinyDB**
  - Virtual Machine – Mate, Bombilla

# TinyDB - Motivation

- Traditional query processing systems
  - RDBMS
  - Passive systems
    - Assume a priori existence of data
- Two solutions
  - Power constrained version of RDBMS
    - Data aggregation, filtering techniques
  - Acquisitional Query Processor - AQP

# AQP

- Sampling – Where, When and How often
- Focus on location and cost of acquiring data
- Reductions in power consumptions
- Simple extensions to SQL
  - Controls data acquisition
  - Achieves query optimization, dissemination and execution

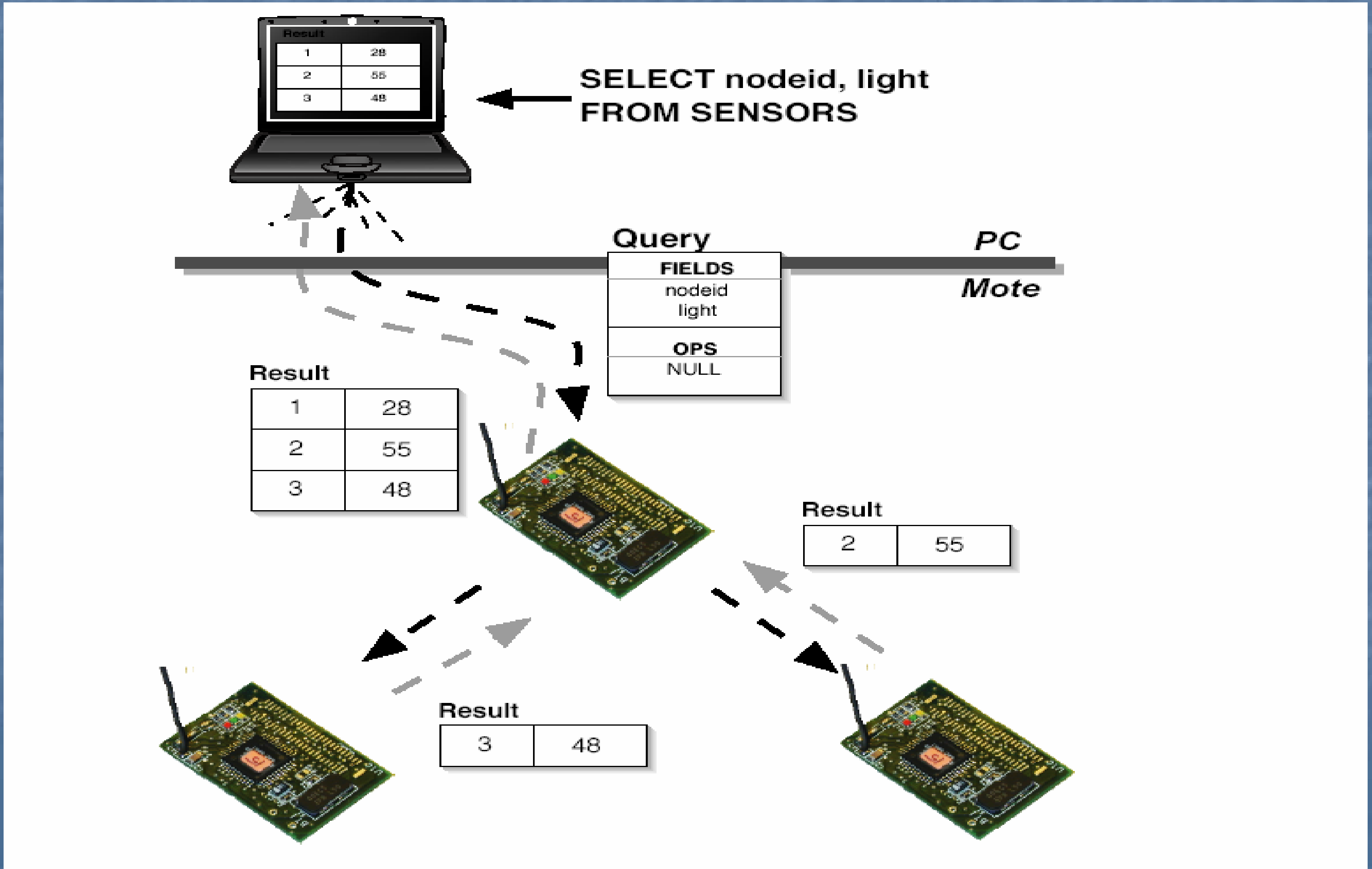
# AQP - Characteristics

- Query Optimization
  - Significant cost of sampling
  - Prioritizing sampling attributes important
  - Done at base station
- Query Dissemination
  - Query nodes which have data
  - Done at each node
- Query Execution
  - When to sample
  - Which samples to process
  - Done at nodes where query disseminates

# TinyDB Features

- Distributed AQP
  - Runs on each sensor node
- Ability to select, join, project and aggregate
- Acquisitional techniques to reduce power consumption
- Interleaving query processing with local computations
- Quantitative analysis for data aggregation

# Basic Architecture



# Acquisitional query language

- SELECT-FROM-WHERE Clause
- Supports join, projection and aggregation
- Explicit support for sampling intervals, windowing
- Sensor data
  - Single table with one column per sensor type

# Specifying sampling interval

- SELECT nodeid, light, temp
  FROM sensors
  SAMPLE INTERVAL 1s FOR 10s

- " sensors " → Virtual table

- Results stream to base station using multi-hop topology

- Output consists of ever growing sequence of tuples
  - Streaming data
  - Timestamp with each tuple

# “ sensors ” table

- Virtual unbounded table
- Continuous data stream of values
- Blocking operations not allowed
  - Sorting, Symmetric Join etc
  - Unless window is specified
- Query ID associated with every query
  - Used to stop running queries

# Window creation

- Window
  - Fixed size materialization points over stream
- CREATE
  STORAGE POINT recentlight SIZE 8
  AS (SELECT nodeid, light FROM sensors
          SAMPLE INTERVAL 10s )
- recentlight → Shared local location
  - Local to node
- Joins allowed between
  - Two storage points on same node
  - Storage point and "sensors"

# Aggregate functions

- SELECT COUNT(*)
  FROM sensors AS s, recentlight as r1
  WHERE r1.nodeid = s.nodeid and
  s.light < r1.light
  SAMPLE INTERVAL 10s

- SELECT WINAVG (volume, 30s, 5s)
  FROM sensors
  SAMPLE INTERVAL 1s

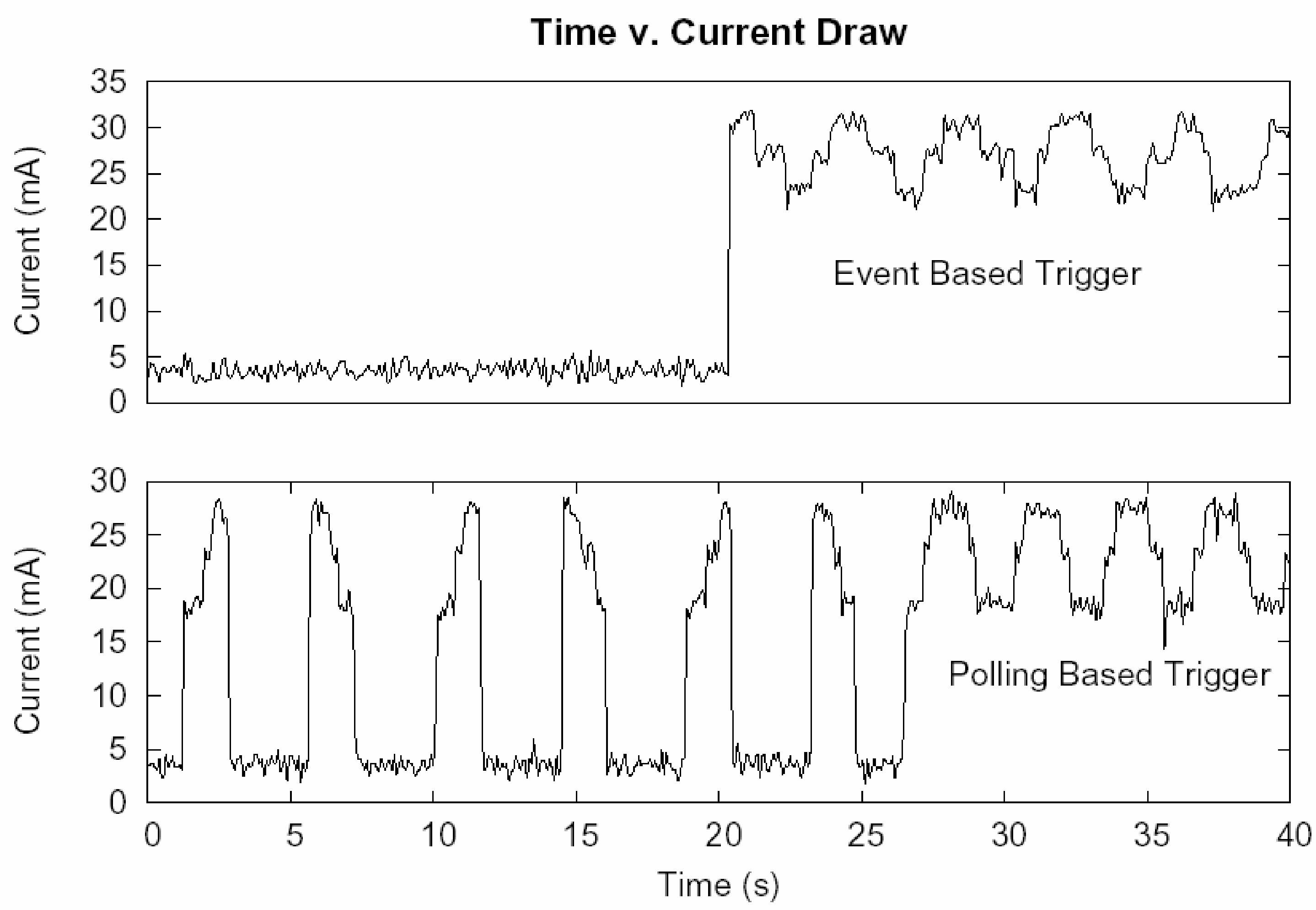
- Query reports average volume
  - Over last 30 seconds
  - Once every 5 seconds

- Sliding window query

# Event based queries - Triggers

- Events initiate data acquisition
  - Event generated either by TOS or another query
- ON EVENT bird-detect (loc)
  SELECT AVG(light), AVG(temp), event.loc
  FROM sensors AS s
  WHERE dist(s.loc, event.loc) < 10m
  SAMPLE INTERVAL 2s FOR 30s
- Events triggered only on local node
  - Queries can be distributed for execution
- Avoids polling or blocking

# Triggers – Power Savings



# Lifetime Based Queries

- SELECT nodeid, accel
  FROM sensors
  LIFETIME 30 days

- Much more intuitive to reason about power consumption

- Lifetime estimation performed by TinyDB
  - Compute a sampling and transmission rate
  - Given energy remaining

# Estimation : Individual Node

```
SELECT a_1, ... , a_numSensors
  FROM sensors
  WHERE p
  LIFETIME l hours
```

| Parameter | Description | Units |
|---|---|---|
| $l$ | Query lifetime goal | hours |
| $c_{rem}$ | Remaining Battery Capacity | Joules |
| $E_n$ | Energy to sample sensor $n$ | Joules |
| $E_{trans}$ | Energy to transmit a single sample | Joules |
| $E_{rcv}$ | Energy to receive a message | Joules |
| $\sigma$ | Selectivity of selection predicate | |
| $C$ | Number of children nodes routing through this node | |

$$p_h = c_{rem} / l$$

$$e_s = \left(\sum_{s=0}^{numSensors} E_s\right) + (E_{rcv} + E_{trans}) \times C + E_{trans} \times \sigma$$

$$T = p_h / e_s$$

# Estimation - Network

- Deciding network transmission rate
  - Sleep-Wakeup cycles are co-coordinated
  - Maximum rate of network
    - Transmission rate of root
  - Slower transmission
    - Transmit at integral multiples of root rate
  - Parent includes rate in queries forwarded to children

# Query Optimization

- Done by Base Station
- Purpose → To choose correct ordering for sampling, selection and joins
- Simple cost based optimizer
  - Reduces power consumption
  - Processing cost and transmission cost
- Cost dominated by
  - Sampling of physical sensors
  - Transmission costs

# Meta Data

- Maintained at each node
- Enlists
  - Local attributes
    - Semantic properties → Used in dissemination
  - Events
  - User defined functions
  - Cost of processing and delivering data
    - Query lifetime estimation
- Periodically copied to root

# Metadata - Types

- Event metadata
  - Name, Signature, frequency estimate
- User defined functions metadata
  - Name, signature and selectivity estimate
- Attribute metadata

| Metadata | Description |
|---|---|
| Power | Cost to sample this attribute (in J) |
| Sample Time | Time to sample this attribute (in s) |
| Constant? | Is this attribute constant-valued (e.g. id)? |
| Rate of Change | How fast the attribute changes (units/s) |
| Range | What range of values can this attribute take on (pair of units) |

# Predicate Ordering

- Sampling → Very expensive in terms of power
  - Selection and Join "FREE" in comparison
- SELECT accel, mag  
  FROM sensors  
  WHERE accel > a  
  AND mag > m  
  SAMPLE INTERVAL 1s
- Order of magnitude cost difference in sampling accel and mag
- Three plans
  - Sample both before either selection
  - Sample mag, apply selection, sample accel, apply selection
  - Sample accel, apply selection, sample mag, apply selection

# Trigger Batching

- ON EVENT e(nodeid)
  SELECT a
  FROM sensors AS s
  WHERE s.nodeid = e.nodeid
  SAMPLE INTERVAL d FOR k

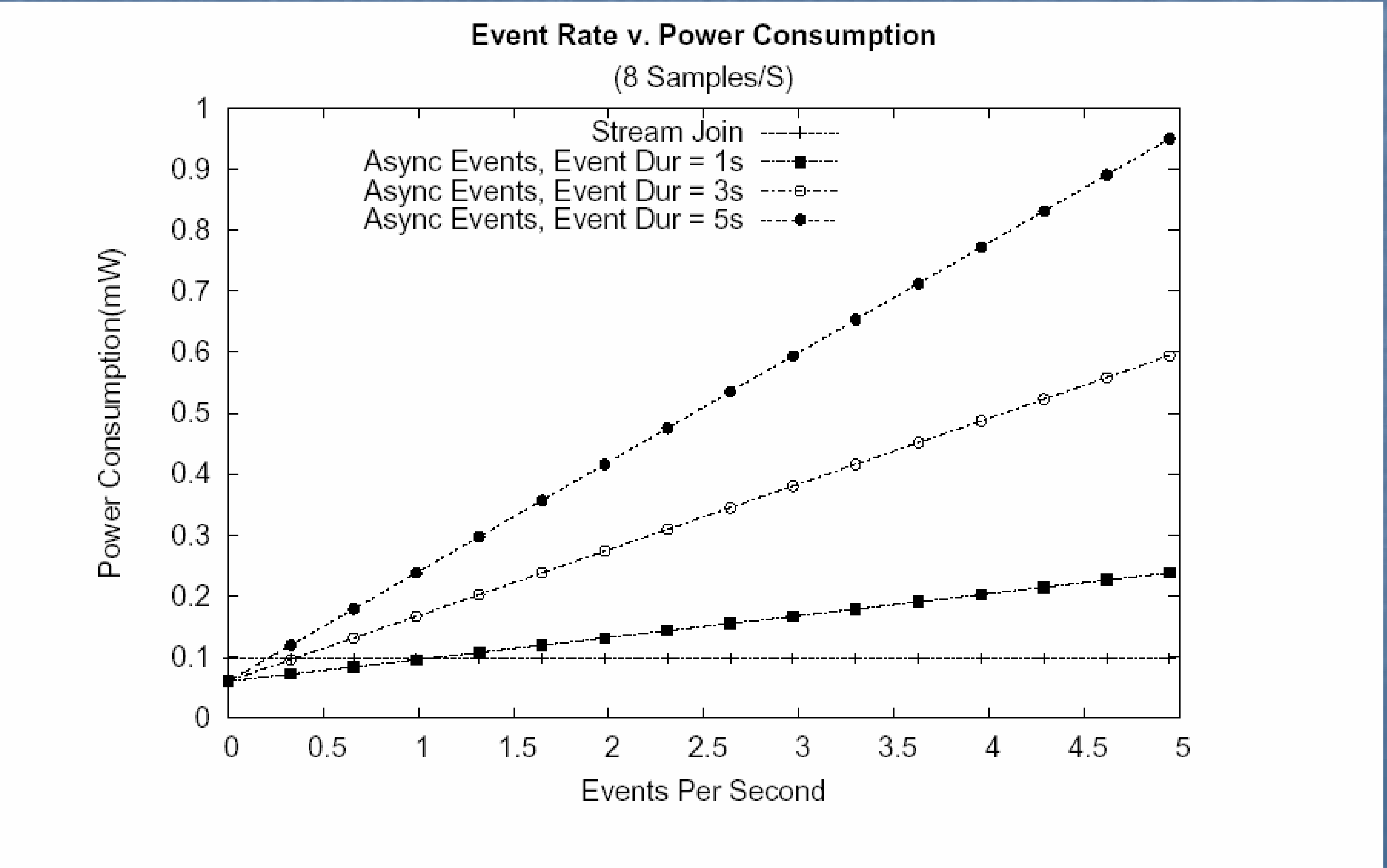
- Query samples every d seconds for k seconds

- Event e → Instance of query executed
  - Multiple instances running simultaneously

# Query rewriting

- External events (e) converted to data stream
- Query rewritten as sliding window join of event and sensors streams
- SELECT s.a
  FROM sensors AS s, events AS e
  WHERE s.nodeid = e.nodeid
  AND e.type = e
  AND s.time - e.time <= k
  AND s.time > e.time
  SAMPLE INTERVAL d
- Only single query executing
- Disadvantage → Reverting back to polling

# Power Consumption - Rewriting



# Query Dissemination

- Deciding where a query should execute
  - Limiting the scope of queries
- Constant valued attributes with selection predicate
  - nodeid
  - Location → Fixed location network
- Solution → Semantic Routing Table (SRT)

# SRT - Features

- Routing tree
  - Efficiently determines children who can execute queries
- Construction
  - Pick parent with
    - Highest link quality
    - Semantic properties
- An index over attribute A
  - Each node stores interval for range of A values
  - Range includes range of children

# SRT - Construction

- Two phase process
  - Phase I
    - SRT build request flooded
    - Request includes name of attribute A
  - Phase II
    - Node has no children
      - Chose a parent p, report range → Parent selection
    - Node has children
      - Propagate build request to children, wait
      - Record ranges with children's id
      - Report to parent with complete range

# SRT – Parent Selection

- Parent Selection Algorithm
  - Random
  - Closest Parent
    - Parent reports its range
  - Clustered approach
    - Snoop sibling's parent selection packet
- Advantages
  - Network topology correlation with geography exploited

# Query Execution

- $T_{awake}$ = Sensor node awake time
- Nodes forced to drop or combine tuples
  - Small $T_{awake}$
  - Very small sample interval
- Solution
  - Data Aggregation
    - Partial state record

# Tuple Aggregation

- Query results en queued onto radio queue
  - Tuples for forwarding
  - Limited queue size → Data aggregation
- Aggregation method
  - Naive
  - WinAVG
  - Delta
    - Involves updating on every transmission

# Outline

- TOS Architecture
  - Challenges
  - Features
- Subsystems
  - Scheduler
  - Active Messaging
  - TinyDB
  - **Virtual Machine – Mate, Bombilla**

# The Origin of MATE

Mate(mah-tay): A tea like beverage consumed mainly in Argentina, Uruguay, Paraguay and southern Brazil…

# Why Do We Need VM ?

- Some nodes will fail during operation
  - Change in network topology/parameters
- Almost impossible to manually recollect and reprogram
  - Adaptive query processing, data aggregation
- Significant energy cost in reprogramming
- Incremental code generation using XML
  - Memory intensive
- Need for viral programming

# System Requirements

- Small (16KB installation memory, 1KB RAM)
- Expressive → versatile
- Concise → limited memory & bandwidth
- Resilience → robustness
- Efficient → energy consumption / transmission
- Tailor able → specialized operations

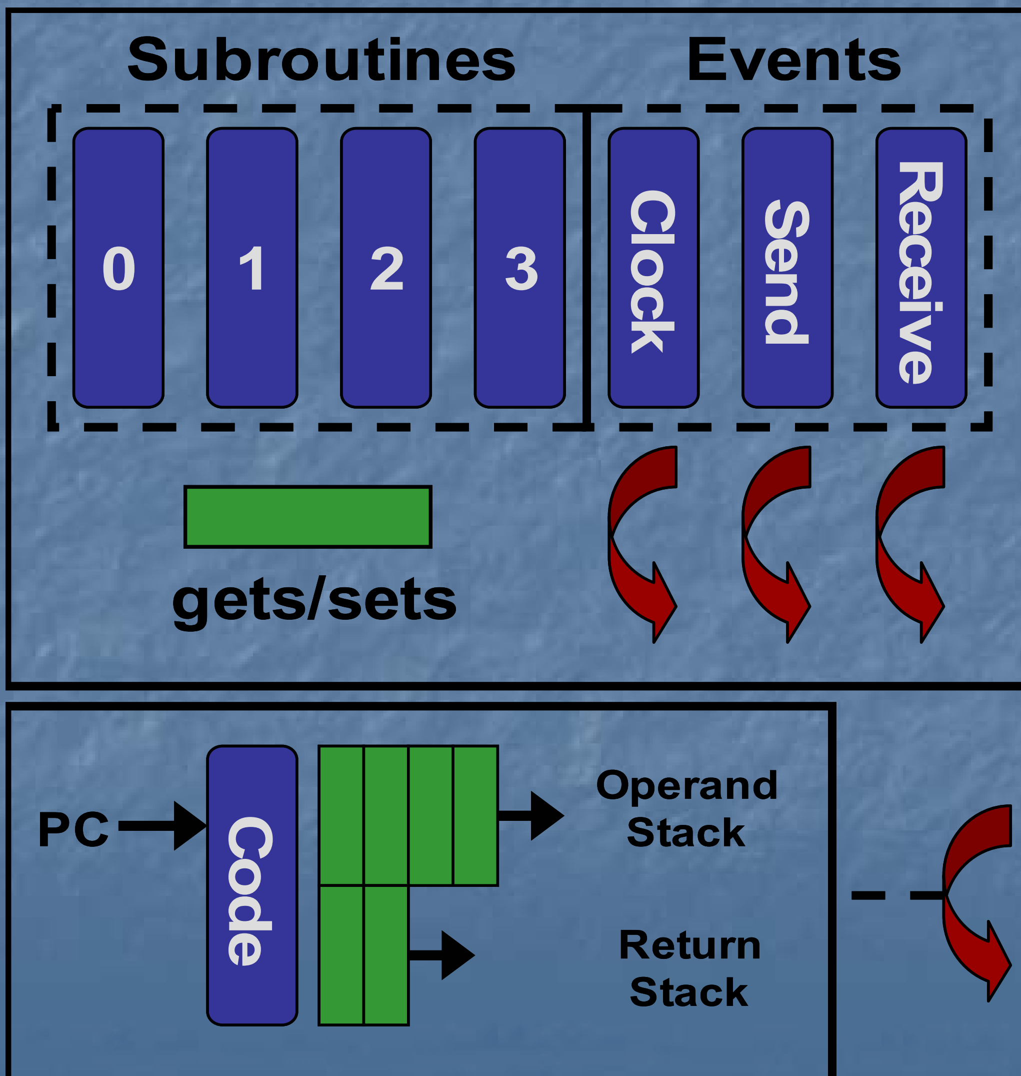
# Mate in a Nutshell

- Tiny communication centric virtual machine
- Byte code interpreter running on motes
- Single TOS component
- Code broken into 24 instruction capsules
- Concise, high level programming
- Safe execution environment
  - Implied user/kernel boundary

# Mate in a Nutshell

- Stack Architecture
  - Operand stack
  - Return address stack
- Three concurrent execution contexts
  - Timer → Persistent operand stack
  - Send
  - Receive
- Execution triggered by predefined events
- Tiny code capsules → self-propagate into network
- Built in ad-hoc routing / Customized routing
  - send / sendr

# Mate Architecture



# Send

- Mate calls command in routing component
- Suspends context until send complete event
- No need to manage message buffers
  - Capsule suspended till network component sends packet
- Synchronous model of communication
  - Application programming made simple

# Instruction Set

| | | |
|---|---|---|
| basic | 00iiiiii | i = instruction |
| s-type | 01iiixxx | x = argument |
| x-type | 1ixxxxxx | |

- Three instruction classes
  - basic: arithmetic, LED operation
  - s-type: messaging system
  - x-type: pushc, blez

- 8 instructions reserved for users to define
  - Default no-ops
  - Useful for creating domain specific instructions

# Code Example

- Display Counter to LED

```
gets        # Push heap variable on stack
pushc 1     # Push 1 on stack
add         # Pop twice, add, push result
copy        # Copy top of stack
sets        # Pop, set heap
pushc 7     # Push 0x0007 onto stack
and         # Take bottom 3 bits of value
putled      # Pop, set LEDs to bit pattern
halt        #
```

# Code Capsules

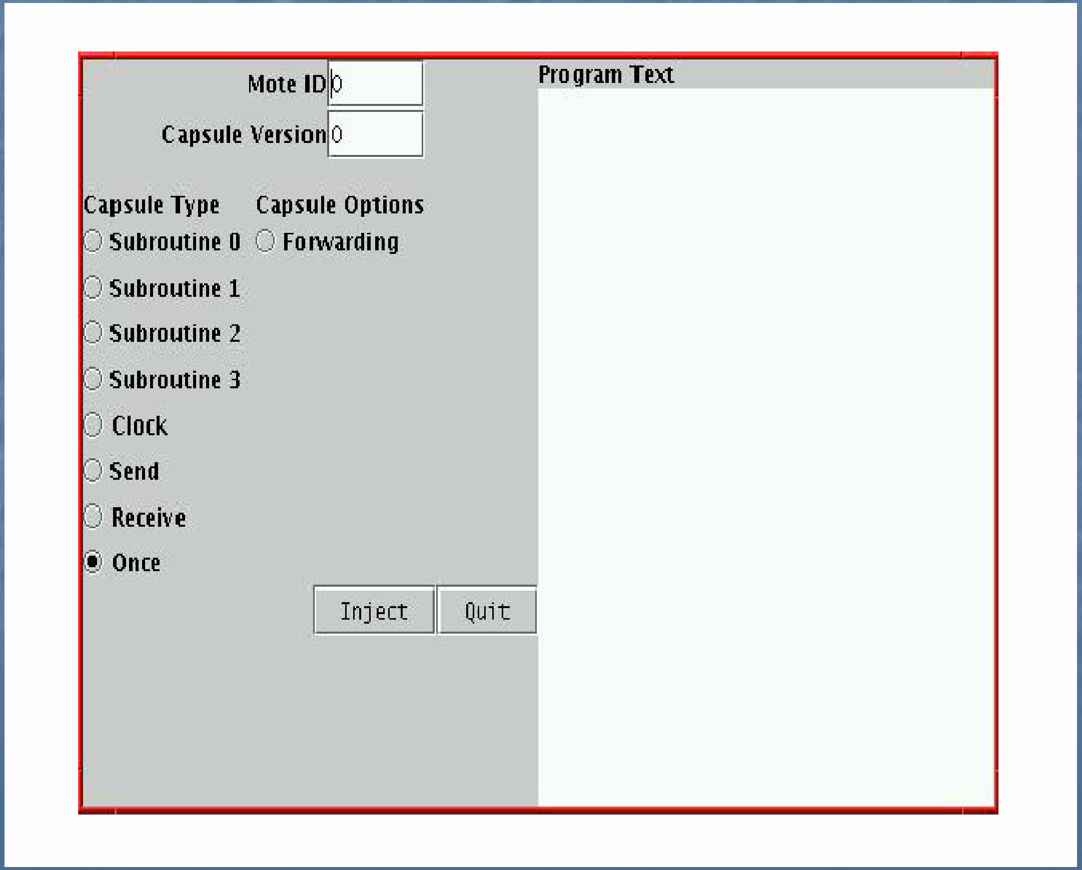
- One capsule = 24 instructions
  - Each instruction is 1 byte long
  - Larger programs → Multiple capsules / subroutines
- Fits into single TOS packet
- Atomic reception
- Code Capsule
  - Type and version information
  - Type: send, receive, timer, subroutine
- Each instruction executed as TOS task

# Capsule forwarding

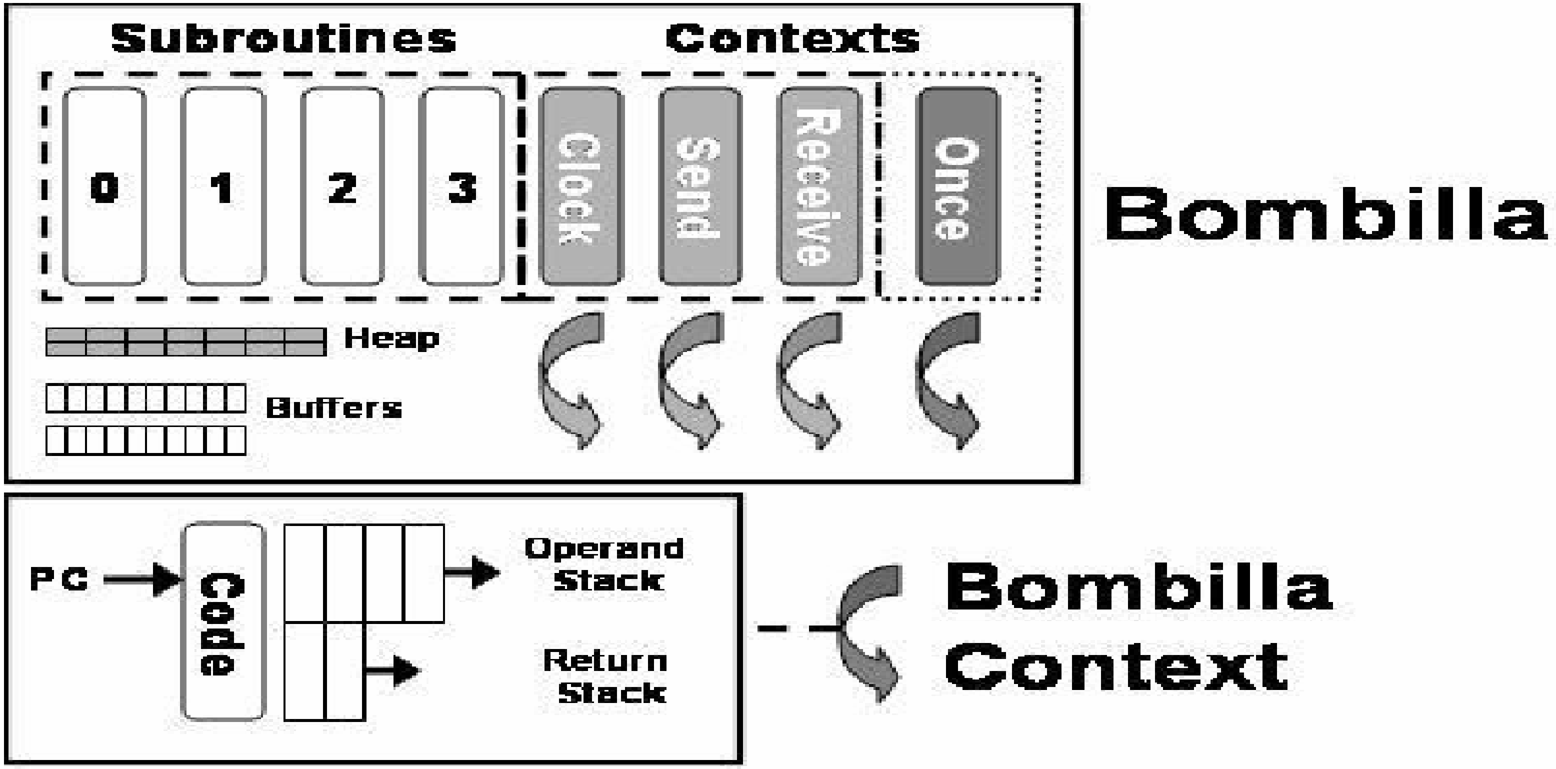
- Capsule transmission → *forw*
- Forwarding other installed capsule → *forwo*
- Mate checks on version number on reception of a capsule
  - If it is newer, install it
- Versioning → 32bit counter
- Easily disseminates new code over the network

# Bombilla

- Next version of Mate?



# Bombilla Architecture




- Once context
- 16 word heap sharing among the context
- Buffer holds up to ten values

# Bombilla Instruction Set

| | | |
|---|---|---|
| basic | 00iiiiii | i = instruction |
| m-class | 010iixxx | i = instruction, x = argument |
| v-class | 011ixxxx | i = instruction, x = argument |
| j-class | 10ixxxxx | i = instruction, x = argument |
| x-class | 11xxxxxx | i = instruction, x = argument |

- m-class: access message header
- v-class: 16 word heap access
- j-class: two jump instructions
- x-class: pushc

# References


- **TinyOS - Architecture**
  - **System Architecture Directions for Networked Sensors**
    Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David C,Kristofer P
    Department of Electrical Engineering and Computer Sciences
    University of California, Berkeley

- **Active Messages**
  - **Active Message Communication for Tiny Networked Sensors**
    Philip Buonadonna, Jason Hill, David Culler
    Department of Electrical Engineering and Computer Sciences
    University of California, Berkeley

  - **Active Messages: a Mechanism for Integrated Communication and Computation**
    Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, Klaus Erik
    Department of Electrical Engineering and Computer Sciences
    University of California, Berkeley

# References


- **Scheduling**
  - **Priority Scheduling in TinyOS - A Case Study**
    Venkita Subramonian, Huang-Ming Huang, Seema Datar, Chenyang Lu
    Department of Computer Science, Washington University
- **Virtual Machine**
  - **Mate: A Tiny Virtual Machine for Sensor Networks**
    Philip Levis and David Culler
    Department of Electrical Engineering and Computer Sciences
    University of California, Berkeley
- **TinyDB**
  - **The Design of an Acquisitional Query Processor For Sensor Networks**
    Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein Wei Hong
    *UC Berkeley Intel Research, Berkeley*