Power-Aware "Scheduling"

Azer Bestavros

September 16, 2003

Scribe: Kanishka Gupta

References (and quotations)

- H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez; "Power-Aware Scheduling for Periodic Real-Time Tasks". TOCS'03.
- R. Melhem, D. Mosse and E.(Mootaz) Elnozahy; The Interplay of Power Management and Fault Recovery in Real-Time Systems". IEEE TOCS, 2003.
- W. Ye, J. Heidemann and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, IEEE Infocom 2002.

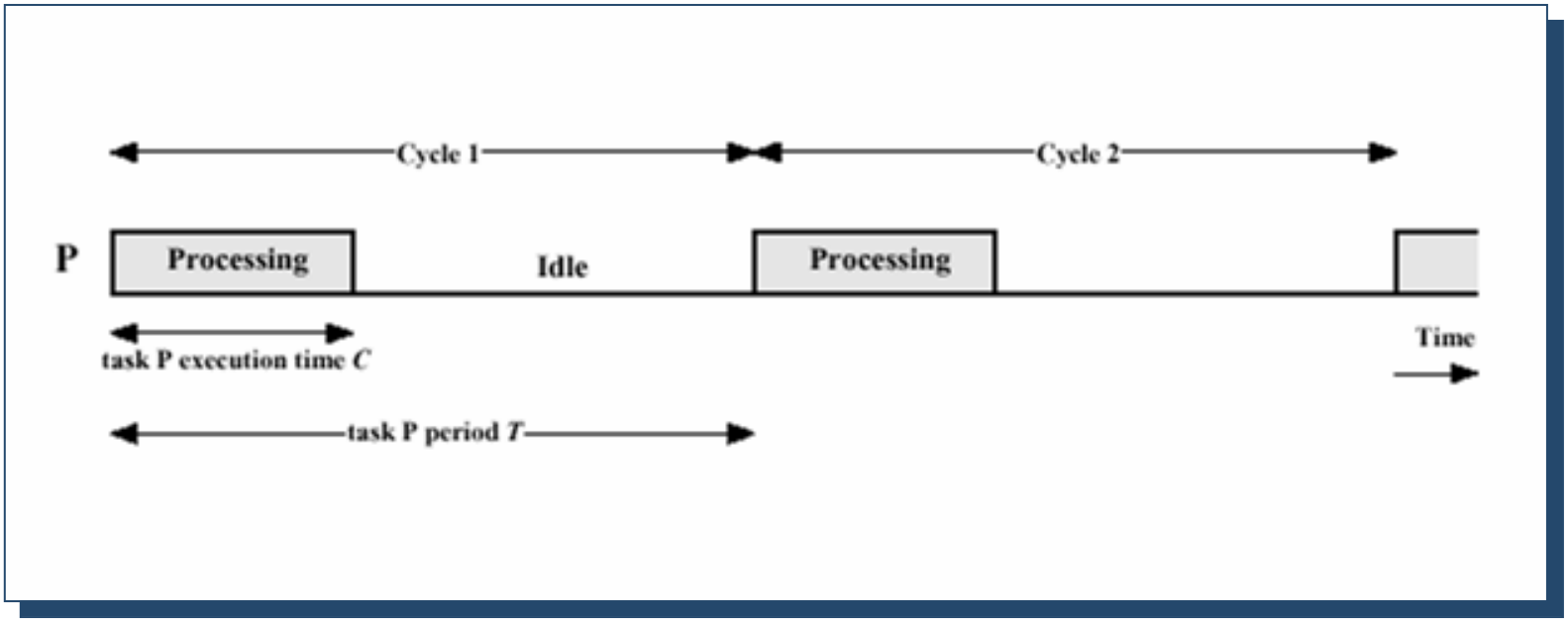
RT Scheduling Basics

Computer Science

- The notion of a task encompasses many instantiations of a specific job. Examples:
  - The task of sensing temperature in a room involves the periodic execution of a job that measures the temperature
  - The task of sending a video stream involves the periodic execution of a job that sends individual frames

- Aspects of task model:
  - Preemptability: Once scheduled could a job be preempted?
  - Periodicity: How are jobs dispatched?
  - Imprecision: Could job execution benefit from more time?

Real-Time Periodic Tasks



- A Periodic Task is specified by
  - **P:** The "Period" (between) job submissions
  - **C:** The "Time of Resource Usage per Period"

Schedulability Analysis

- Check if a given task set can execute without having any job miss its deadline.

- Schedulability analysis is used for admission control of real-time task sets
  - Could a Video Sensing Element (VSE) accept a new streaming task?
  - Can I open a new video-conferencing channel without degrading existing channels?
  - Can a video server accept a new VOD request without degrading service to established VOD requests?

Dynamic Priority Scheduling

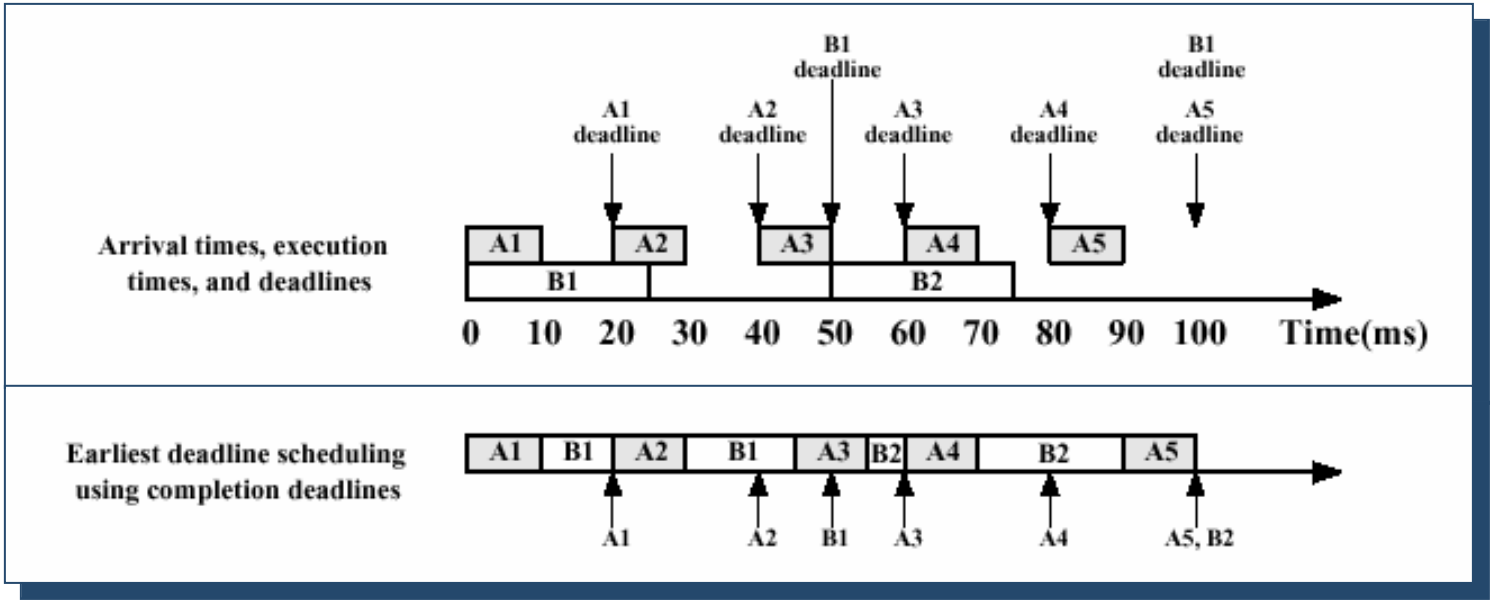
- The Model:
  - Tasks are periodic. Task i has a period Pi.
  - Task i need to compute for up to Ci units of time every period Pi.
  - Jobs of task i are ready at the start of each period Pi.
  - Jobs of task i have deadlines by the end of period Pi.
  - Jobs are preemptable.
  - Tasks are scheduled according to their priorities, which are allowed to change dynamically.

- The Problem
  - Find a priority assignment and a schedulability test such that for any set of tasks that passes the schedulability test, it follows that using the priority assignment technique, each task in the set will meet its deadline.

Earliest Deadline Scheduling

EDF Priority Assignment: Assign to task i a priority inversely proportional to the amount of time remaining until the task's deadline (ties broken arbitrarily).

Earliest Deadline Scheduling



Selection Function: Job with the closest deadline among all jobs ready to execute

Decision Mode: Preemptive (at job arrival)

EDF Schedulability Analysis

Computer Science

- A task set is schedulable if

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq 1$$

+ Obviously optimal (since when the equality holds, we are using 100% of the resource)
- But, uses dynamic priorities and is unpredictable under overload!

Fixed Priority Scheduling

- The Model:
  - Tasks are periodic. Task i has a period Pi.
  - Task i need to compute for up to Ci units of time every period Pi.
  - Jobs of task i are ready at the start of each period Pi.
  - Jobs of task i have deadlines by the end of period Pi.
  - Jobs are preemptable.
  - Tasks are scheduled according to their priorities, which must be fixed (i.e. they cannot change dynamically).

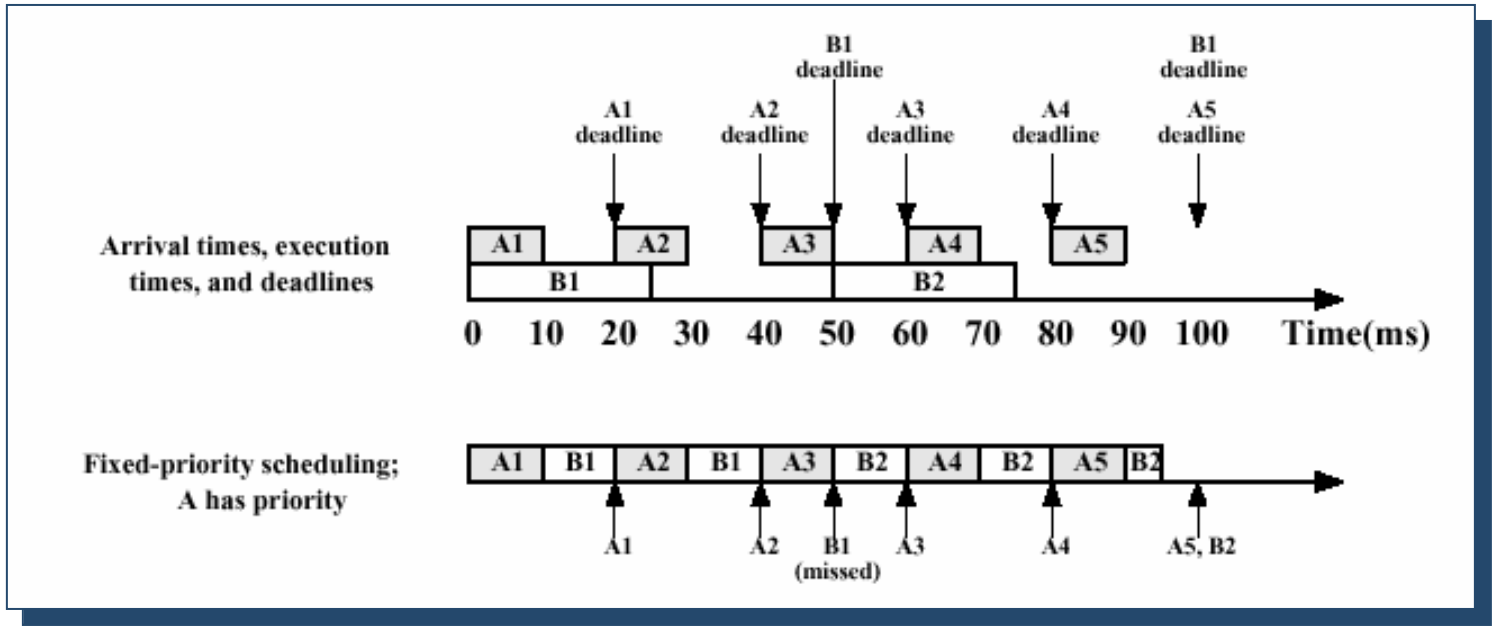
- The Problem
  - Find a priority assignment and a schedulability test such that for any set of tasks that passes the schedulability test, it follows that using the priority assignment technique, each task in the set will meet its deadline.

Rate Monotonic Scheduling

RMS Priority Assignment: Assign to task i a priority inversely proportional to its period Pi (i.e. priority is proportional to rate). Ties are broken arbitrarily.

Rate Monotonic Scheduling

Computer Science



Selection: Job with highest rate that is ready to execute

Decision: Preemptive (at job arrival times)

RMS Schedulability Analysis

Computer Science

- A task set is schedulable if

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq \mathrm{N}\left(2^{\frac{1}{\mathrm{N}}} - 1\right) \approx \ln(2)$$

+ Optimal among class of fixed-priority schedulers
+ Predictable under overload
- But, wastes upwards of 30% of the scheduled resource!

Execution Time Variability

- Execution time per period can vary by as much as 1 or 2 orders of magnitude

- Should one do schedulability analysis based on worst-case or based on some other combinatorial or statistical assumptions?

Statistical RMS

Computer Science

- The Model:
  - Tasks are periodic. Task i has a period Pi.
  - Task i need to compute Ci units of time every period Pi and Ci follows a known distribution.
  - Jobs of task i are ready at the start of each period Pi.
  - Jobs of task i have deadlines by the end of period Pi.
  - Jobs are preemptable.
  - Tasks are scheduled according to their priorities, which must be fixed (i.e. they cannot change dynamically).
  - Tasks need to satisfy some QoS (expressed as % deadlines met)

- The Problem
  - Find a priority assignment and a schedulability test such that for any set of tasks that passes the schedulability test, it follows that using the priority assignment technique, each task in the set will meet its QoS constraints.

RT Scheduling & Power

Computer Science

- CPU consumes 30%-50% of notebook power
- To conserve power, system could operate at lowest voltage/frequency possible, but:
  - Embedded devices typically involve periodic, real-time tasks (e.g., measure *x* every *t*).
  - DVS is constrained by the need to meet the deadlines of all periodic tasks in the system.

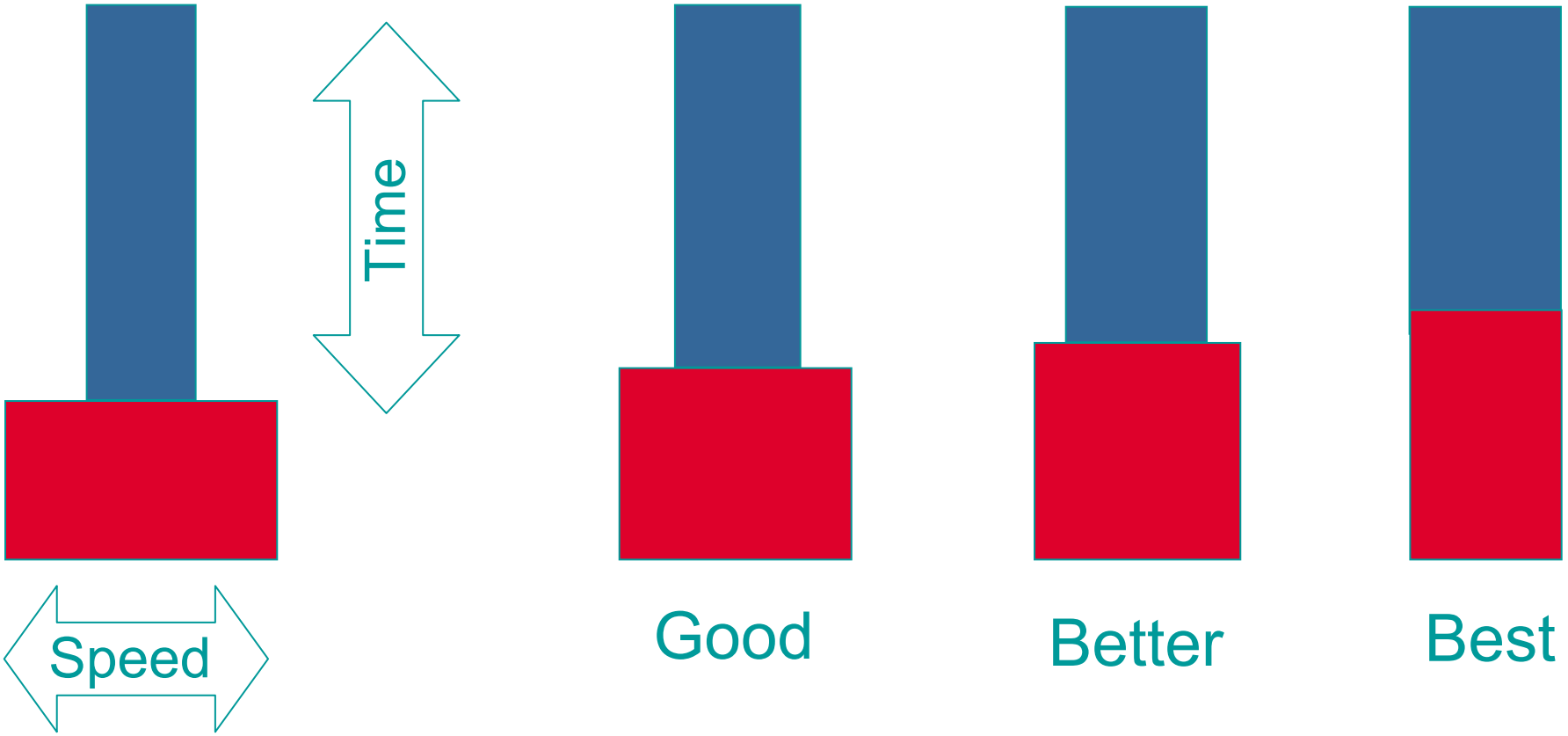
- **Problem:** Devise a scheduling algorithm *and* a DVS scheme that minimizes power consumption while satisfying temporal constraints on RT tasks.

Speed vs Power

Computer Science

- CMOS Chips
  - Power consumed is a function of voltage ~ $C_{eff}.V^2.f$
  - Frequency (1/delay) is proportional to voltage ~ k. $V^2/(V-V_t)$
  - Reducing voltage necessitates reducing frequency (MHz)
- Power consumption ~ $O(V^3)$
  - A strictly increasing convex function
  - Dynamic Voltage Scaling (DVS) allows voltage to be set to a continuous spectrum (e.g. in increments of 33MHz)

Speed vs Power



When to Adjust Power

Computer Science

- Inter-Task Power Adjustments:
  - Only at job dispatch time
  - Done by the “dispatcher” and not by the “program”
  - Takes 10s of microseconds on today’s processors
- Intra-Task Power Adjustments:
  - Job may adjust its power consumption
  - Requires OS/compiler support to allow/use API power management API

Problem Statement

Computer Science

- Model
  - Tasks are periodic. Task i has a period Pi.
  - Task i need to compute for up to Ci cycles every period Pi
  - Jobs of task i are ready at the start of each period Pi
  - Jobs of task i have deadlines by the end of period Pi
  - Jobs are preemptable
  - Tasks are scheduled according to dynamic priorities (a.k.a. EDF)
  - ACi < Ci is actual # of cycles needed by task Ti
  - Speed S can be set to continuous values between 0 and 1
  - Speed can be changed at context switching times
  - Power is given by a function g(S)
- Problem

  Find speed setting that minimizes power consumption while satisfying all deadlines

Overview of Solution(s)

1. **Off-Line:** Determine optimal speed at task level assuming worst-case execution time for each arrival.
2. **On-Line:** Reclaim “unused time” and use that to slow-down task speeds in a manner that guarantees meeting all deadlines.
3. **On-Line:** Predict future underutilization and use that to achieve better power consumption

Off-Line Solution

Computer Science

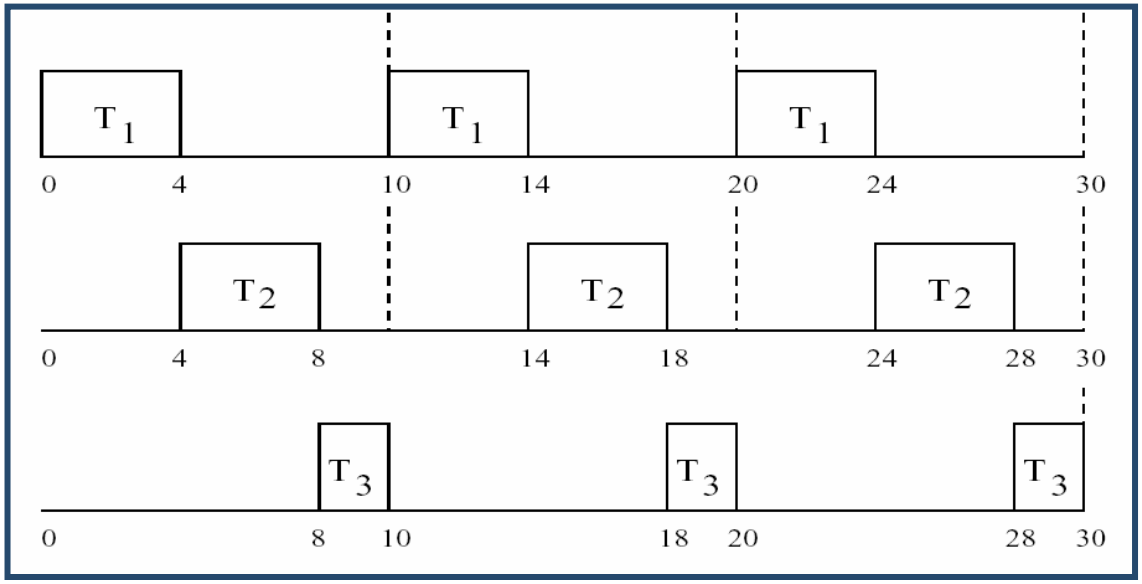
Proposition 1 *The optimal speed to minimize the total energy consumption while meeting all the deadlines is constant and equal to* $\bar{S} = max\{S_{min}, U_{tot}\}$. *Moreover, when used along with this speed* $\bar{S}$, *any periodic hard real-time policy which can fully utilize the processor (e.g., Earliest Deadline First, Least Laxity First) can be used to obtain a feasible schedule.*

- Proof follows directly from convexity argument
- Basically, if we use an optimal RT schedule that delivers a utilization U_{tot} when S=1 then we are guaranteed a feasible schedule when S= U_{tot}
- What about scheduler that do not fully utilize the processor (for good reasons) a la RMS or SRMS?

Worst-Case vs Actual

- Actual execution times could be up to 2 orders of magnitude less than worst cases
- Possible benefit from slowing speed “on the fly”
- Doing the “dumb” thing (e.g., assign lowest speed to idle process) is not good due to convexity
- Greedily using “slack” is not good either

On-Line Solution: Motivation



- Sum(Ci/Pi)=1; no power saving based on WCET
- If T3 is done at t=8, assigning the slack of 4 to T1 (making speed = 0.5 in [10-18]) would make T2 miss its deadline!
- Need a mechanism that recognizes that T1 and T2 must share the slack (i.e., making speed = 0.8 in [10-30])

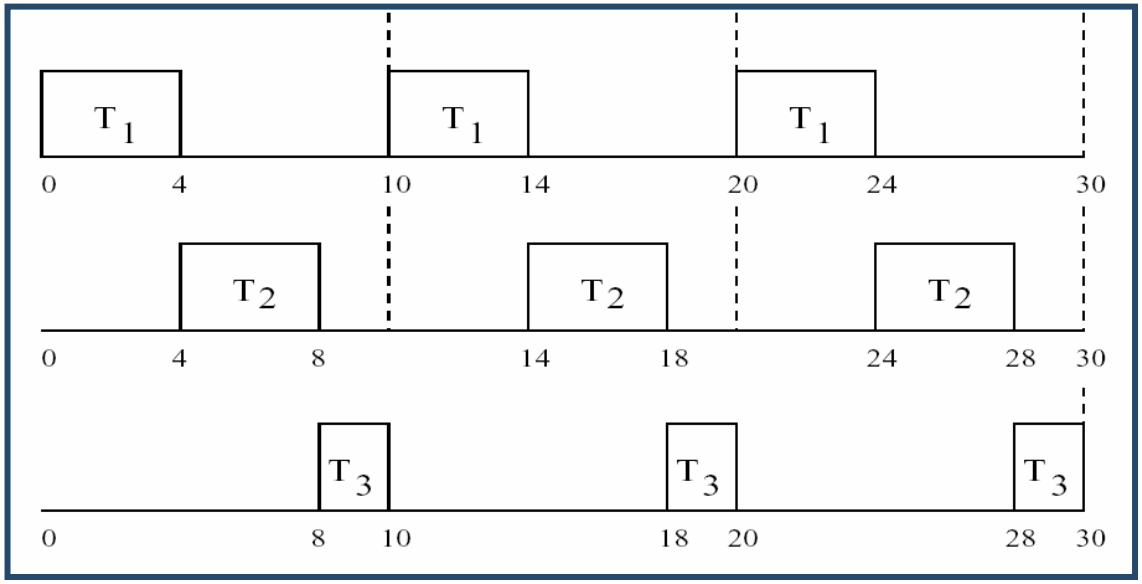
On-Line Solution

Computer Science

- Need to keep track of tasks likely to be affected by "reclamation":
  - Assume EDF* (EDF with non-arbitrary tie-breaking)
  - Keep an efficient data structure (α-queue) that returns the set of tasks whose slack can be used by a task at time t

Proposition 2 *For any task T_x which is about to execute, any unused computation time (slack) of any task in the α-queue having strictly higher priority than T_x will contribute to the earliness of T_x along with already finished work of T_x in the actual schedule. That is,* total earliness *of T_x is no less than* $\epsilon_x(t) = \sum_{i|d_i^* < d_x^*} rem_i(t) + rem_x(t) - w_x^{\widehat{S}_x}(t) = \sum_{i|d_i^* \leq d_x^*} rem_i(t) - w_x^{\widehat{S}_x}(t)$.

On-Line Solution



- At t=10 neither T1 nor T2 can “inherit” the slack of T3
- At t=3, T2 can inherit T1’s slack
- How about at t=23? Can T2 inherit T1’s slack?

Betting on the "future"

- We are still too pessimistic—assuming that all future jobs will request the worst-case utilization
- What if we "steal" time from the future?
  - Clearly, we may be able to further reduce power by having more uniform (lower) speeds
- Can we steal time from the future "safely" (i.e., without the risk of missing any deadlines)?
  - Yes, as long as we can make up for it (by possibly pushing down on the accelerator beyond the statically-computed WCET speed)—i.e., as long as $U_{tot} < 1$

Amdahl Law

- A powerful system design principle is to make the common case efficient…
  - This translates (in settings where the worst-case workload occurs only rarely) into having a power-efficient schedule for average or close to average cases, which can be achieved by reducing further the CPU speed, but without sacrificing correctness (i.e., QoS).
- … and the rare case correct
  - This means that we should reduce speeds only to the extent that we can “recover” by increasing speeds later (if necessary).

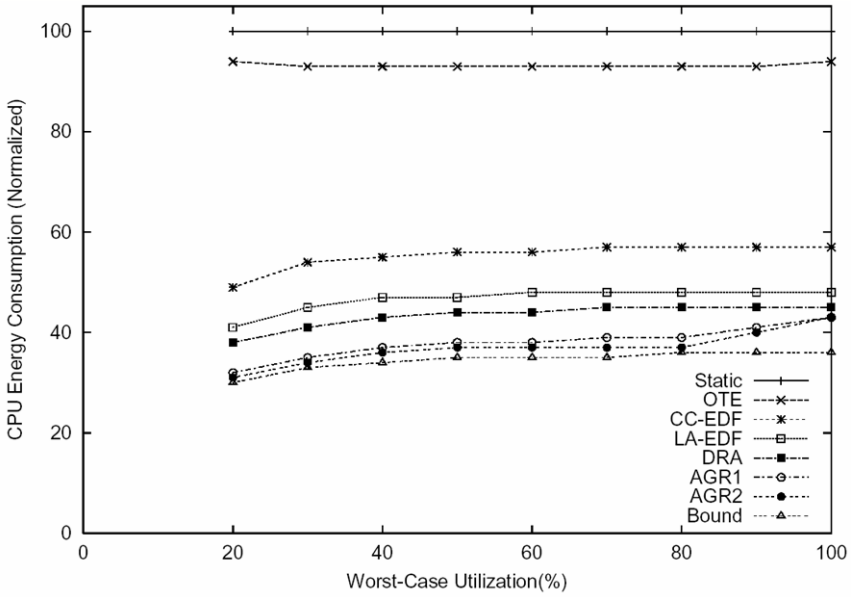
Performance Evaluation

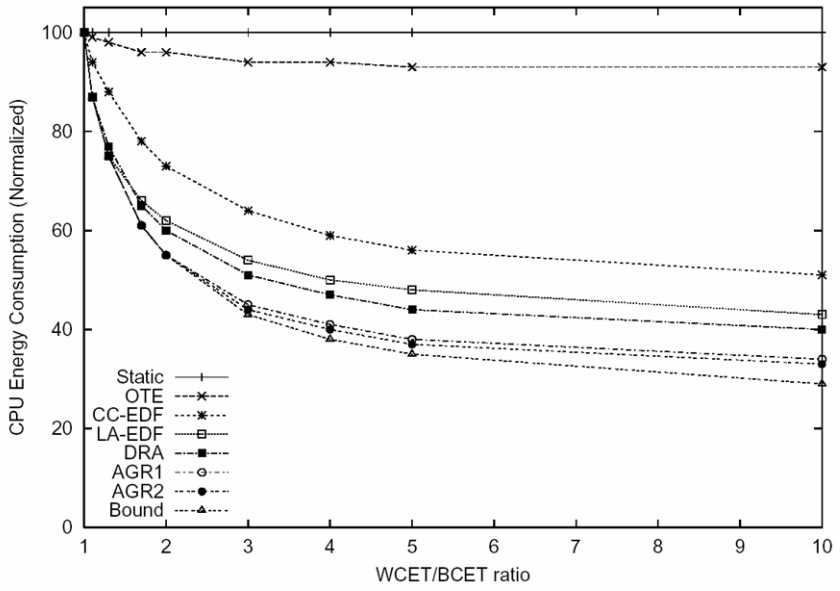
- Simulations of proposed techniques and comparison with trivial and clairvoyant approaches

  - **Static:** Uses constant speed, switching to minimum speed when idle
  - **OTE:** Static speed scheme (with One Task Extension optimization)
  - **CC-EDF:** Prior art (Cycle-conserving EDF)
  - **LA-EDF:** Prior art (Look-ahead EDF)
  - **DRA:** Dynamic reclamation Algorithm
  - **AGR1:** Speculative DRA with aggressiveness factor *k* is set to 1
  - **AGR2:** Speculative DRA with aggressiveness factor *k* is set to 0.9
  - **Bound:** *Clairvoyant* oracle

Experimental Results

Computer Science





- Energy consumption independent of utilization
- Variability in execution time a good differentiator

Thoughts

Computer Science

- CPU is an example resource—results could be equally applied to scheduling communication channels where cost/energy ~ convex function of link speed (e.g., flow = task, CPU cycle = packet)
- Paper does not make use of distributional characteristics of execution times and/or correlation in periodic resource utilization from one period to the next
- What about non-EDF schedulers? What about non-deadline-based RT task models (e.g. pinwheel model)?

Thoughts

- Why insist on 100% QoS? What if the QoS per task is lower-bounded (*a la* SRMS)?
- Are all CPU cycles created equal (when it comes to power consumption)?
- Could prolonging a task execution be harmful (e.g., resulting in stale data)? Could it increase power consumption (e.g., increase probability of collisions)?
- What other uses do we have for “elasticity” in scheduling?

Fault Tolerance/Recovery

- Overview of *"The Interplay of Power Management and Fault Recovery in Real-Time Systems"* by Kanishka Gupta

Medium Access Control

- Wireless channel is a shared medium; need access control mechanism to avoid interference
- MAC protocols used to avoid collisions so that two interfering nodes do not transmit at the same time.
- Two main approaches
  - Contention-based: e.g., 802.11, MACAW, PAMAS
  - Reservation and scheduling: e.g., TDMA, CDMA, Bluetooth
- Not as easy as it sounds!

MAC in Sensor Nets

Computer Science

Important attributes of MAC protocols

1. Collision avoidance
2. Energy efficiency
3. Scalability in node density
4. Latency
5. Fairness
6. Throughput
7. Bandwidth utilization

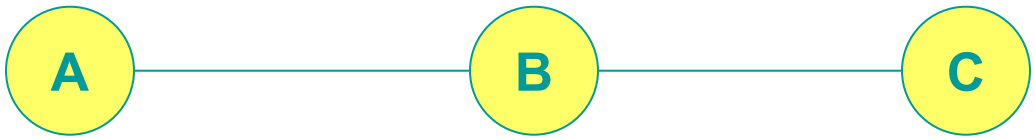
Primary

Secondary

Hidden Terminal Problem

Computer Science

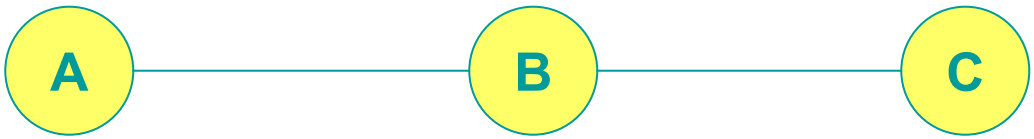
- Node B can communicate with A and C both
- A and C cannot hear each other
- When A transmits to B, C cannot detect the transmission using the *carrier sense* mechanism
- If C transmits, collision will occur at node B



MACAW Solution

Computer Science

- When node A wants to send a packet to node B, node A first sends a ***Request-to-Send (RTS)*** to A
- On receiving **RTS**, node A responds by sending ***Clear-to-Send (CTS)***, provided node A is able to receive the packet
- When a node (such as C) overhears a **CTS**, it keeps quiet for the duration of the transfer
  - Transfer duration is included in both RTS and CTS



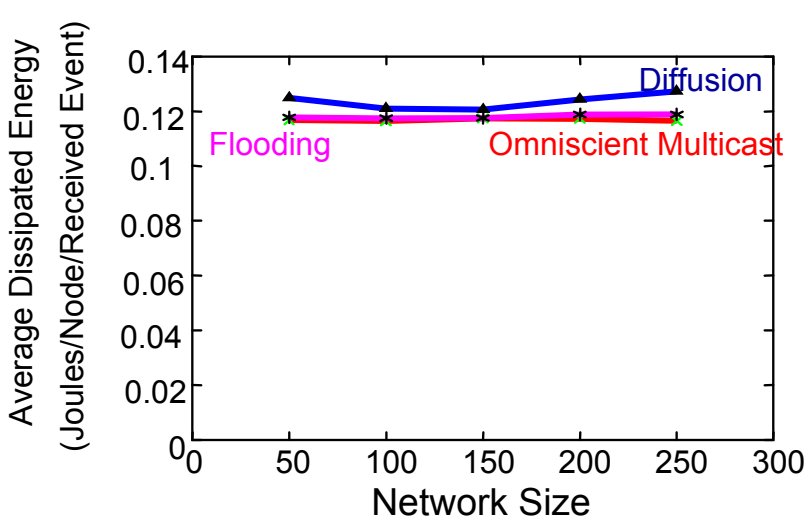
Energy Consumption

- *Packet collisions and retransmissions*. Collisions increases latency as well.
- *Packet overhearing.* A node picks up packets that are destined to other nodes.
- *Control packet overhead*. Sending and receiving control packets consumes energy too.
- *Idle listening.* Listening to receive possible traffic that is never sent. This is the major sink of energy (~ 50-100% of total required for communication).

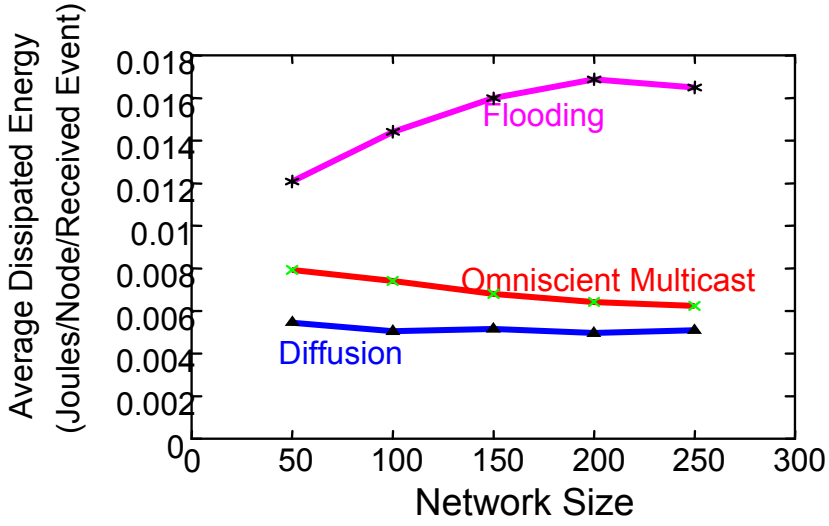
Idle Listening Energy Costs

Computer Science

- Energy consumption of typical 802.11 WLAN cards idle:receive ratio— 1:1.05 to 1:2 [Stemm97]



Always-listening MAC



Energy-aware MAC

What to Trade for Energy?

Computer Science

- Per-Hop Fairness is not important for SN
  - In sensor networks, all nodes cooperate for a single common task; fairness is not important as long as application-level performance is not degraded.
  - Advocates “message passing” (message as opposed to packet store-and-forward)
    - A node with a larger message gets more time to access the medium (LJF scheduling ☺). While this is unfair from a per-hop, MAC level perspective, it results in better energy profile since it reduces control overhead, avoids overhearing.
    - Enables in-network processing (filtering, averaging, etc.) which could potentially improve latency as well!

Sensor-MAC

Computer Science

Latency Fairness    Energy Efficiency

Major components in S-MAC

- Periodic listen and sleep
- Collision avoidance
- Overhearing avoidance
- Massage passing

Periodic Listen and Sleep

Computer Science

Problem: Idle listening consumes significant energy

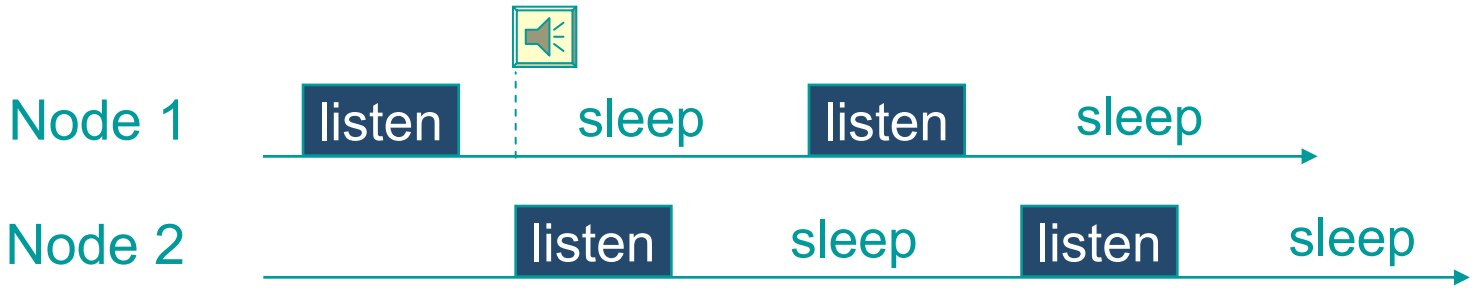
Solution: Periodic listen and sleep

- Turn off radio when sleeping
- Reduce duty cycle to ~ 10% (200ms on/2s off)
- Clearly hurts latency

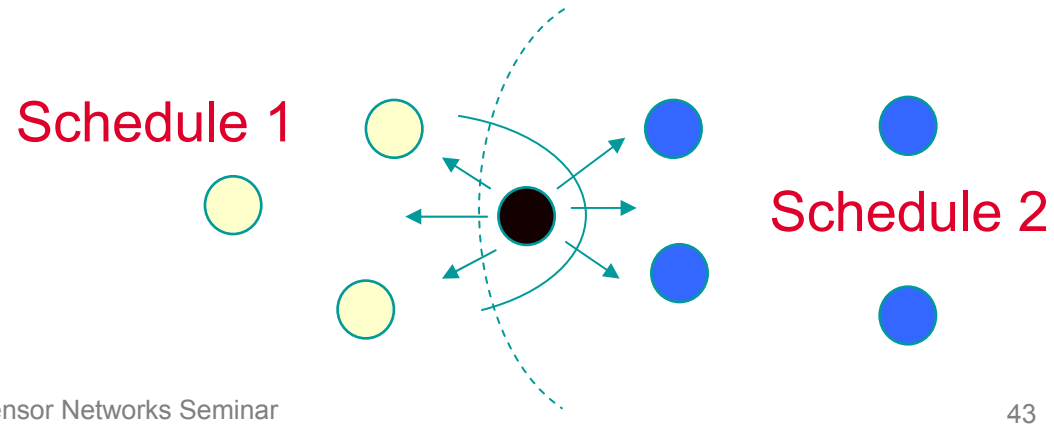
Periodic Listen and Sleep

Computer Science

- Need to coordinate schedules of close-by nodes to avoid the "can you hear me now" syndrome ☺
  - Potential for improved latency
  - Lower control overhead



- Border nodes ?



Periodic Listen and Sleep

Schedule Synchronization

- Synchronizer node chooses and broadcasts a schedule, which is followed by all recipients
- Nodes remember neighbors' schedules (to know when to send to them)
- Each node broadcasts its schedule every few periods of sleeping and listening
- Re-sync when receiving a schedule update
- Schedule packets also serve as beacons for new nodes to join a neighborhood

Collision Avoidance

Computer Science

Problem: Multiple senders want to talk

Solution: Similar to IEEE 802.11 ad hoc mode (DCF)

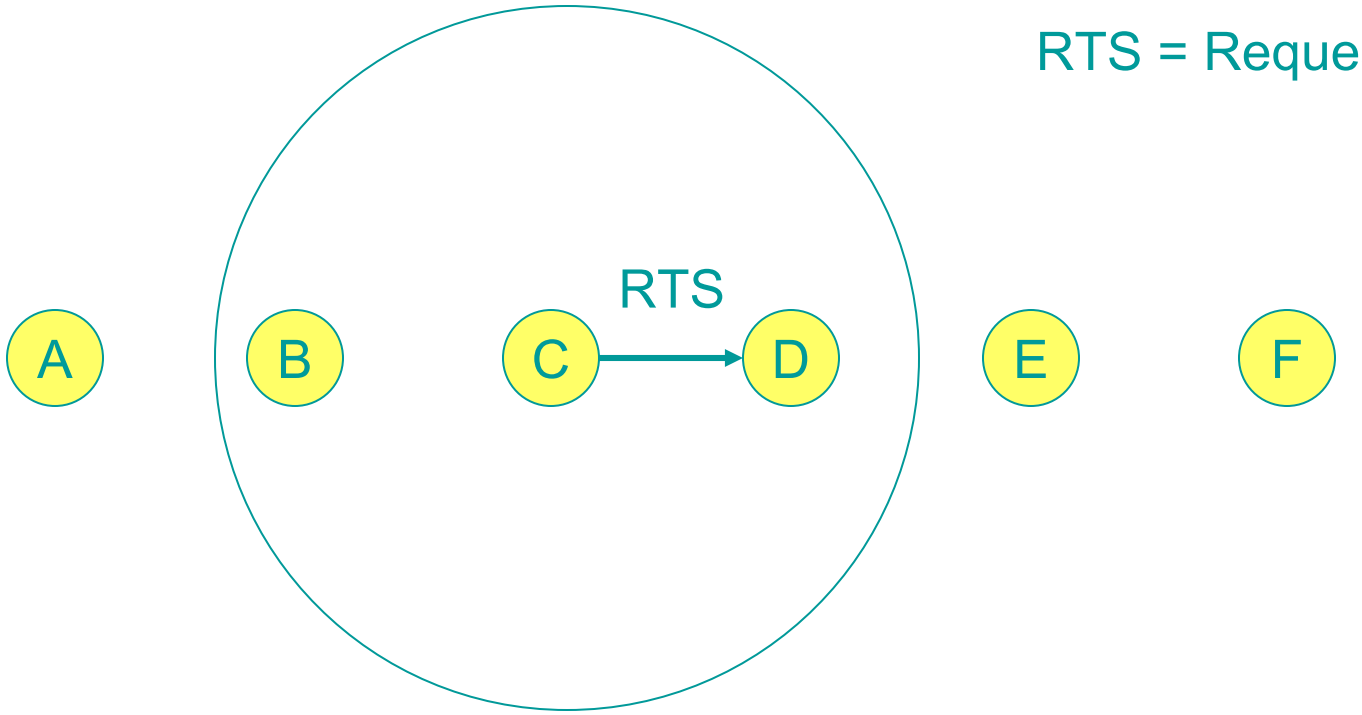
- Physical and virtual carrier sense
- Randomized backoff time
- RTS/CTS for hidden terminal problem
- RTS/CTS/DATA/ACK sequence

IEEE 802.11 DCF

- Uses RTS-CTS to avoid hidden terminal problem
- Uses ACK for reliability
- Any node receiving the RTS cannot transmit for the duration of the transfer
  - To prevent collision with ACK when it arrives at the sender, when B is sending data to C, node A will keep quite

IEEE 802.11

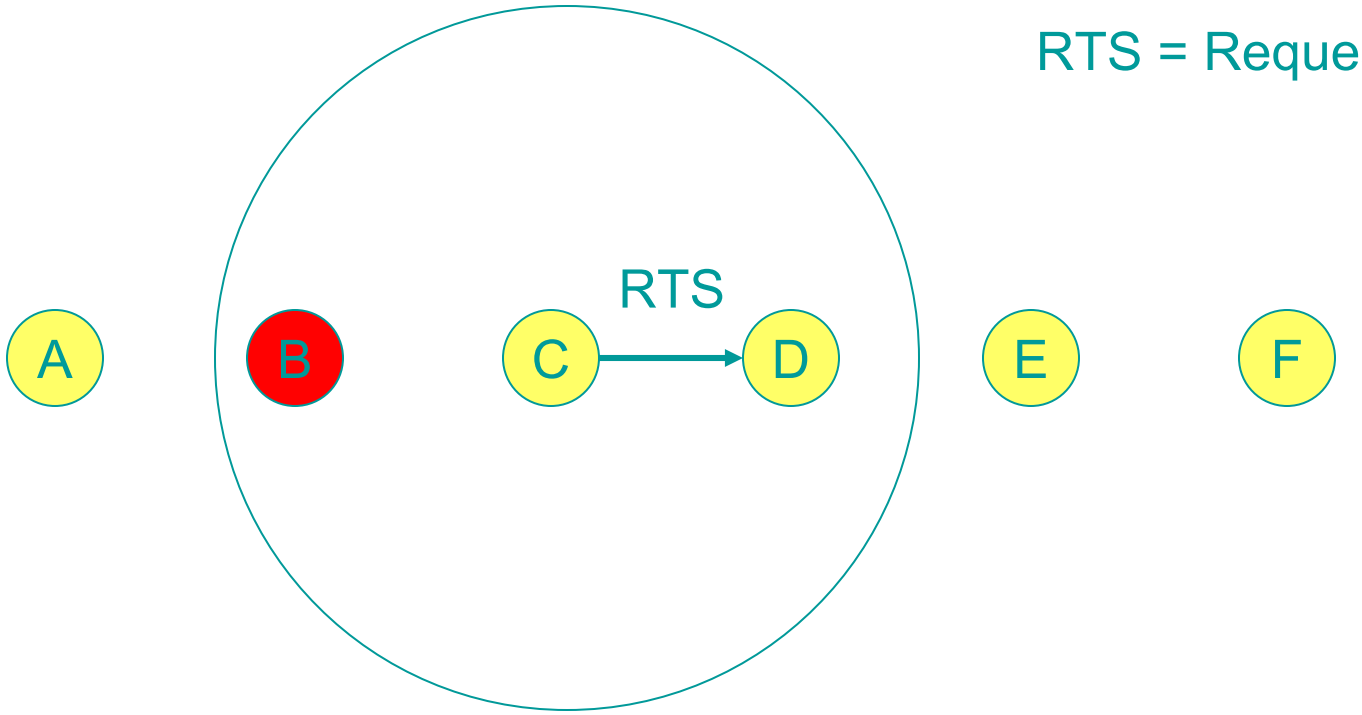
Computer Science



RTS = Request-to-Send

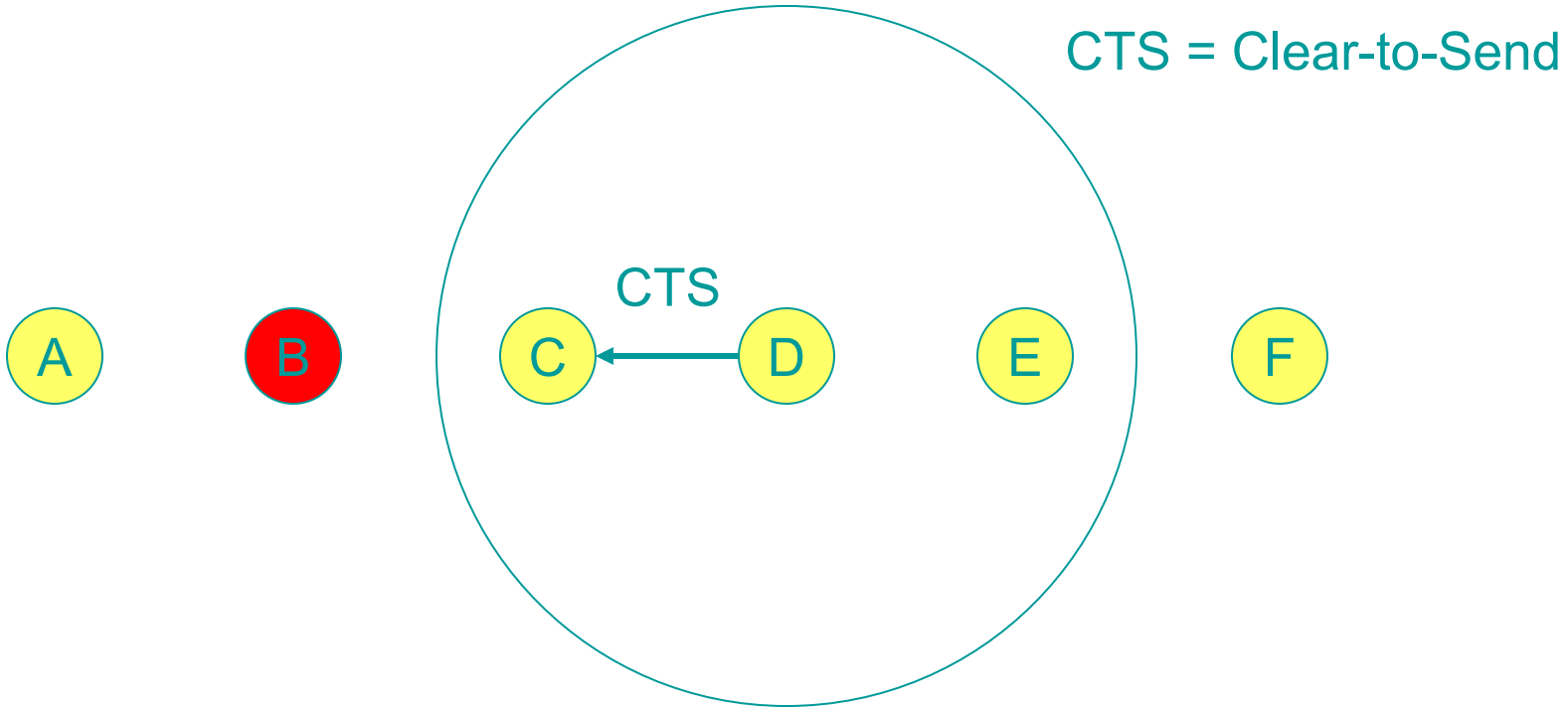
IEEE 802.11

Computer Science

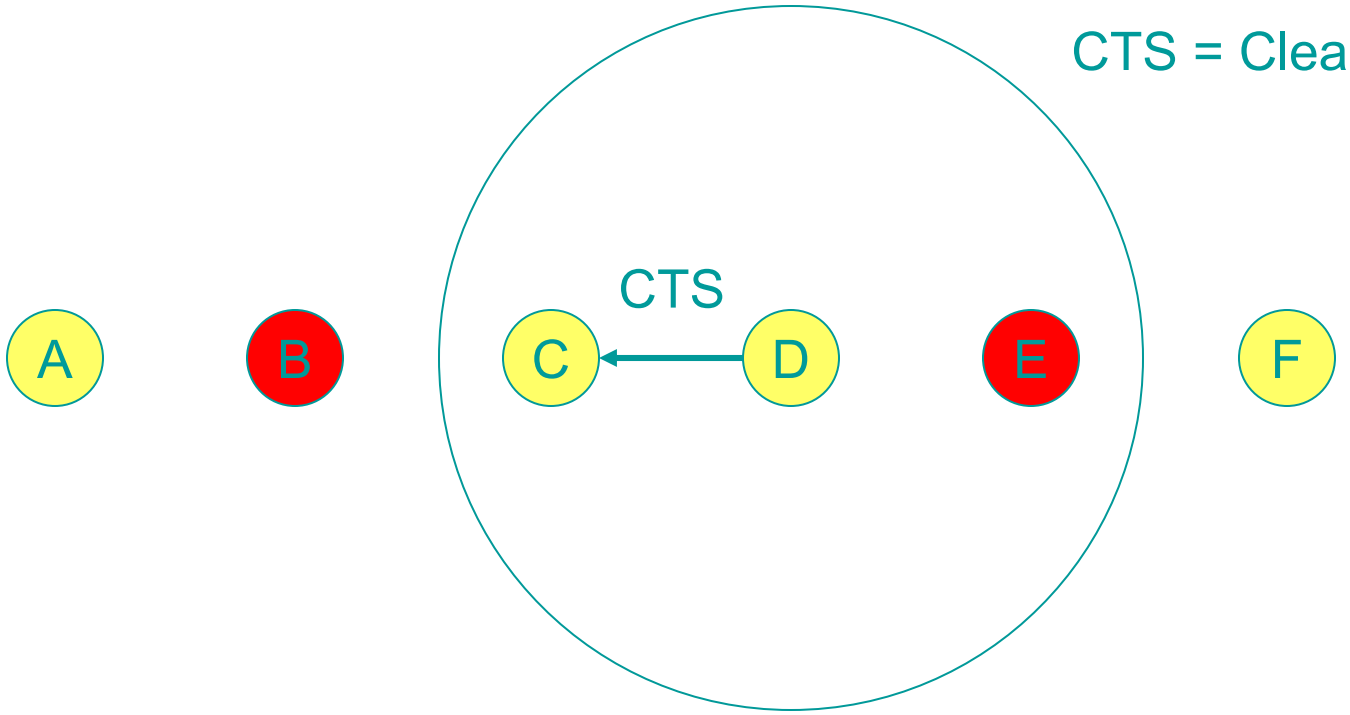


IEEE 802.11

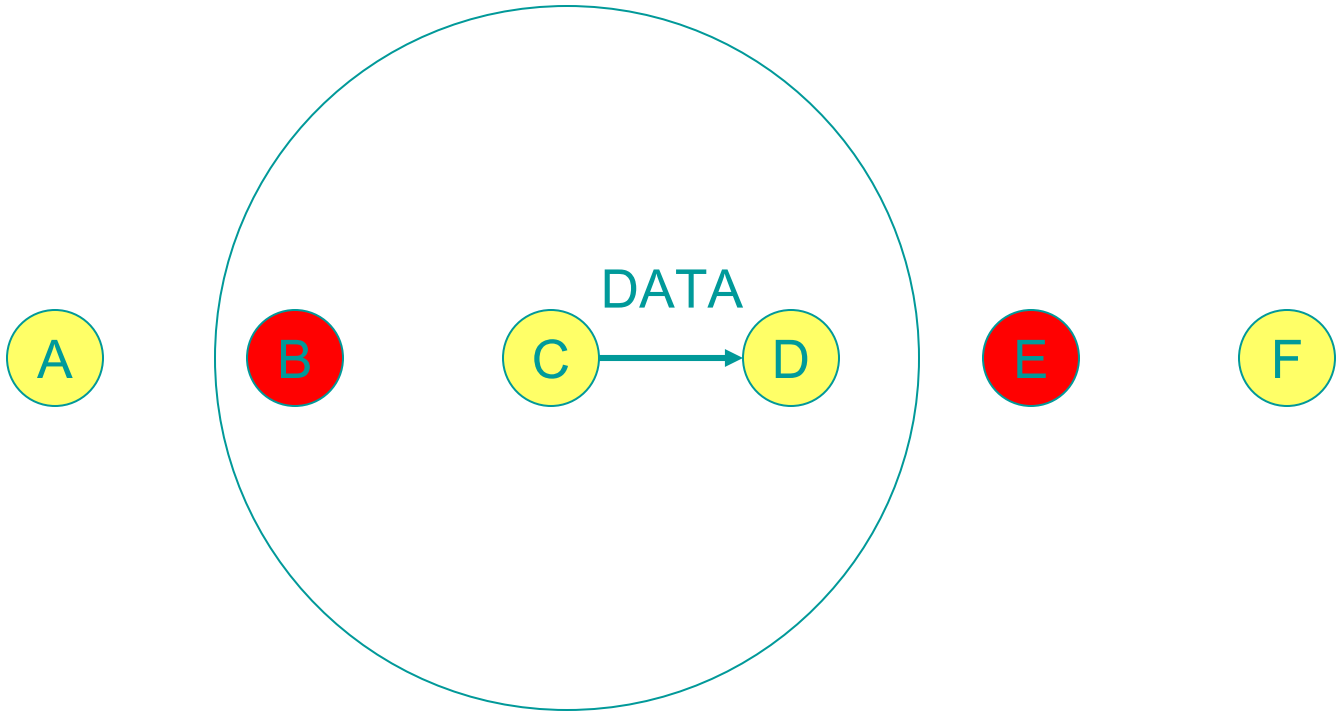
Computer Science



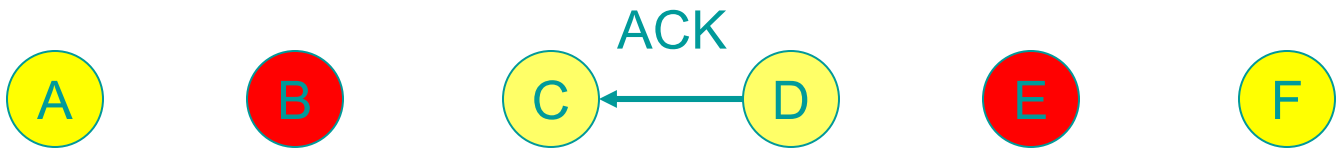
IEEE 802.11



IEEE 802.11



IEEE 802.11



Overhearing Avoidance

Problem: Receive packets destined to others

Solution: Sleep when neighbors talk

- Basic idea from PAMAS [SinghRaghavendra:1998]
- But only use in-channel signaling

- Who should sleep?
  - All immediate neighbors of sender and receiver

- How long to sleep?
  - The duration field in each packet carries the sleep interval

Message Passing

Problem: In-network processing requires *entire msg*

Solution: Don't interleave different messages

- Long message is fragmented & sent in burst
- RTS/CTS reserve medium for entire message
- Fragment-level error recovery
- On retransmission, extend reservation

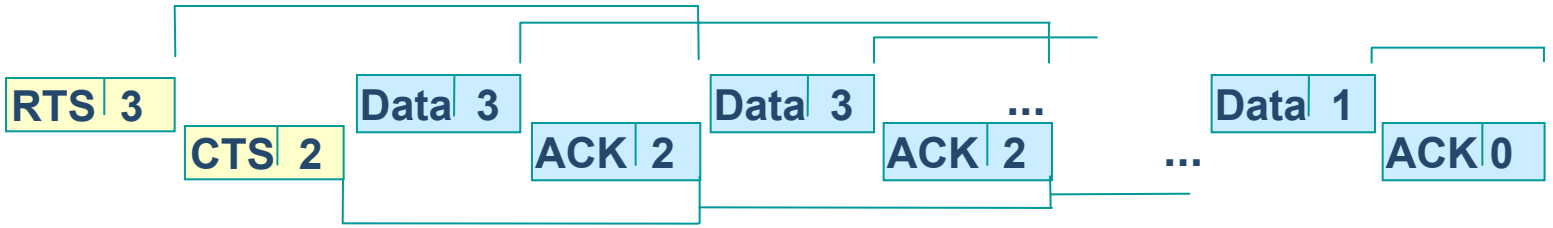
- Other nodes sleep for whole message time

S-MAC vs. 802.11

Computer Science

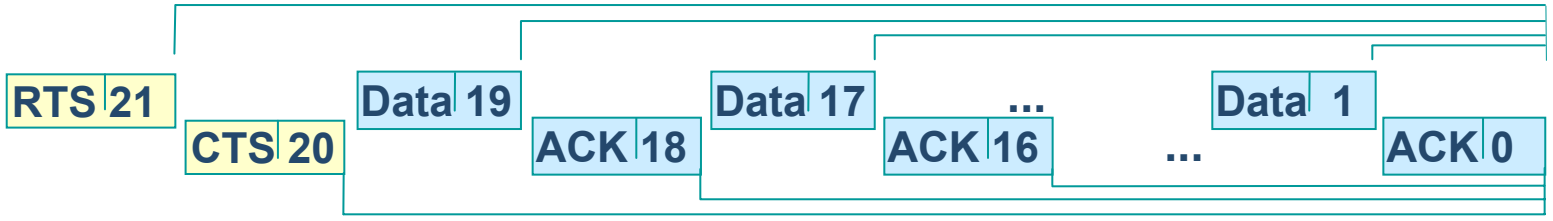
Fragmentation in IEEE 802.11

- No indication of length of msg → idle listening
- If ACK is not received, give up Tx → fairness



Message Passing in S-MAC

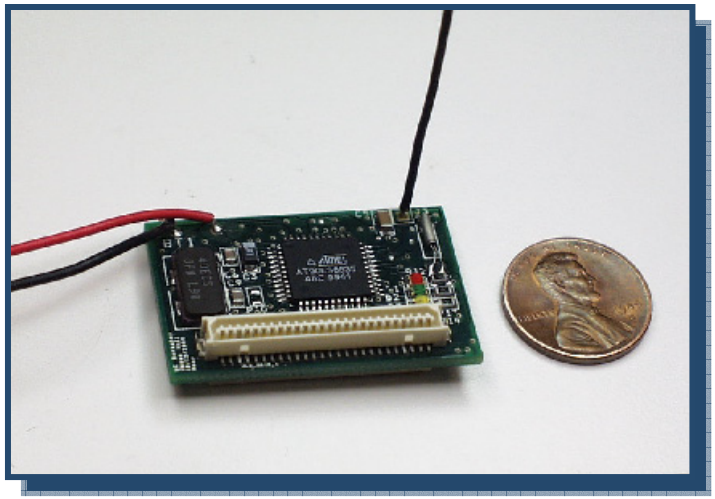
- Each packet carries length of reservation → sleep length
- If ACK is not received, extend reservation → unfair



S-MAC Implementation

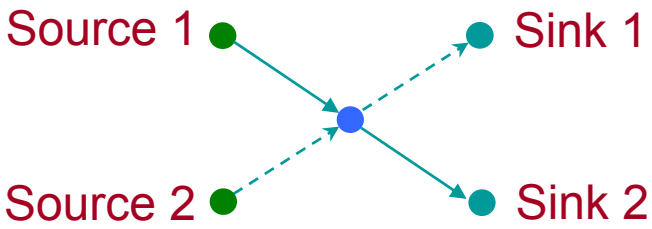
Computer Science

- Platform
  - Motes (UC Berkeley)
    - 8-bit CPU at 4MHz,
    - 8KB flash, 512B RAM
    - 916MHz radio
  - TinyOS: event-driven

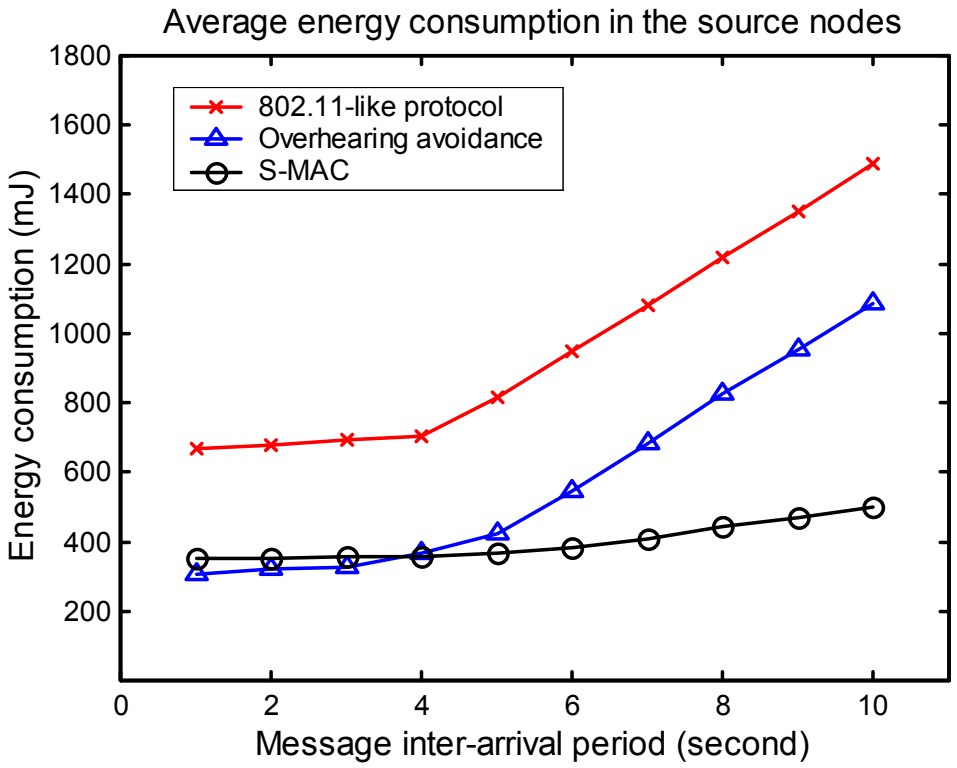


- Compared MAC modules
  - IEEE 802.11-like protocol w/o sleeping
  - Message passing with overhearing avoidance
  - S-MAC (2 + periodic listen/sleep)

Experiments (!)



- Source nodes send 10 msgs, each with 400B in 10 fragments
- Msg interarrivals determine offered load
- Measure total energy over time to send all messages



S-MAC's impact is when load is lowest (of course)

Thoughts

Computer Science

- Effect of node unfairness on “quality” of results
  - Assumption (unsubstantiated) that node unfairness is OK in sensor networks is questionable
- Minimizing aggregate energy isn’t the problem
  - Need to look at issues of balancing energy consumption, or on MAC scheduling subject to node power constraints→ maximize network lifetime
- Is schedule synchronization a good thing?