Computer Networks II

Ch.5

Wireless Sensor Networks

and the

Internet of Things

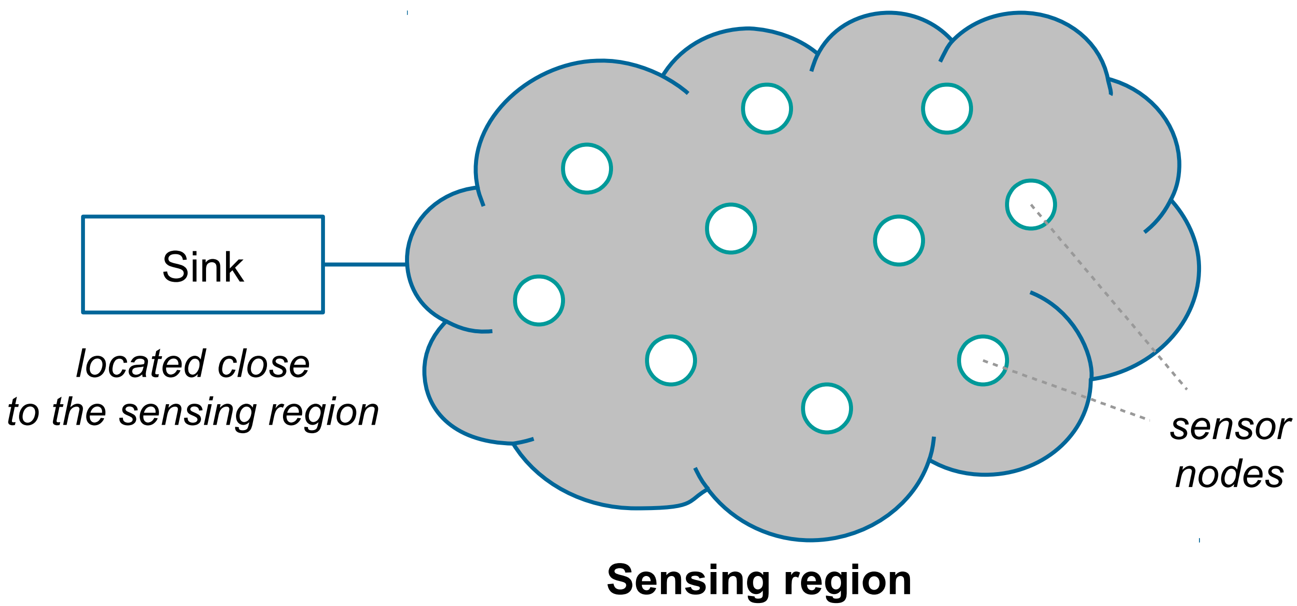
(bruno.quoitin@umons.ac.be)

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

Wireless Sensor Networks

- **Tentative Definition**
  - Large number of low-cost, low-power, multifunctional sensor nodes deployed in a region of interest *[Zheng & Jamalipour, 2009]*
  - Nodes have processing and communication capabilities
  - Nodes collaborate to accomplish a common task



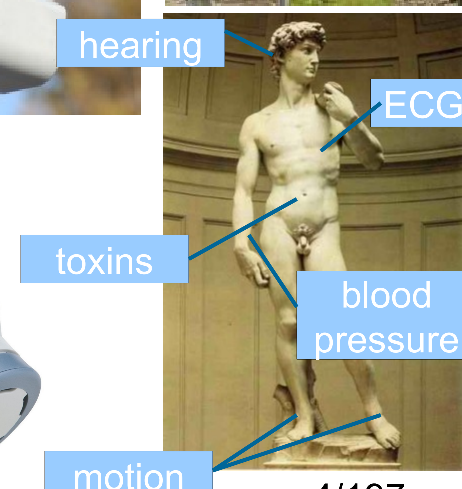
Typical Applications

- **Monitoring, Tracking**
  - Civil engineering infrastructures
  - Agriculture, forests, rivers, oceans
  - Logistics: real-time tracking and status
  - Power grid ("smart grid")
- **Home & building automation**
  - Lighting, HVAC, security, ...
- **Body area networks**
  - e-Health
  - Wearable computing
    (watches, glasses, ...)
- ...
- **No single WSN device !**

LV-45 Linear sensor
railroad bridge
Alliance Sensors (2015)

Station SimTéo
potato/tomato disease
DFI-Elec (30/11/10)

Power line sensor
Sentient (30/07/12)



HR20 Rondostat
Electronic radiator control
Honeywell (30/11/10)

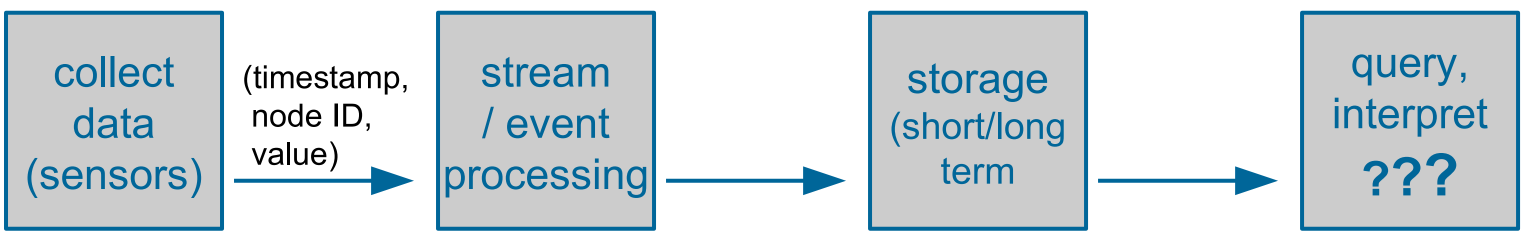
Internet of Things (IoT)

- ***Federate as a single network...***
  - *… physical objects embedding computing and communication resources (traditional computers, smartphones, WSN, …)*
  - diverse communication speeds, access to energy, processing resources
  - avoid protocol specific gateways
- **Predictions**
  - **5.10^{10} IoT nodes in 2020** *[Ericsson, 2009]*
  - IoT 1000 times the size of the Internet *[Cisco, 2009]*
- ***One protocol to rule them all*** **: IPv6**
  - open standard, interoperable
  - IP well tested, well known, widely adopted
  - 2^{128} addresses available: each node globally addressable !

Internet of Things (IoT)

- **Why should computer scientists care ?**
  - 50 billions of devices in 2020... → what's the probability you will be developing an application for one of them ?
  - or take their constraints into account in your research ?
  - still many technical challenges to overcome

- **Not only computer networking**
  - process / mine / interpret large amounts of data (“big data”)



  - algorithmic challenges (path computation, scheduling, …)
  - security !!!

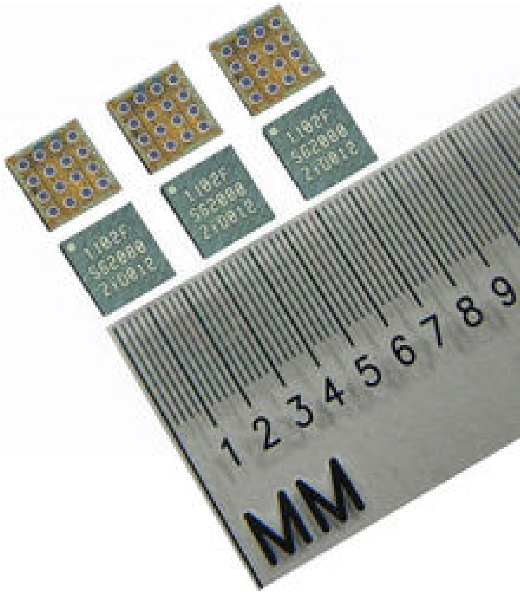
Wireless Sensor Networks

- **How do WSNs differ from Wireless LANs ?**
  - Higher node density (more nodes /m^2 or /m^3)
  - More constrained nodes
    - Limited energy resources → not always ON
    - Limited computation / storage capabilities
  - Low power communication
    - Short range, lossy wireless link
    - Fast changing topology
  - Application specific design
    - not your *general purpose* computer
  - Self-configuration
    - many nodes ; nodes deployed in hard to reach areas
  - Many-to-one traffic pattern (sources to sink)

→ New challenges ! :-)

Wireless Sensor Networks

- **Key enablers**
  - low-power micro-controllers (MCU)
  - higher scale of integration
  - energy harvesting techniques
  - cheap, low-power radio transceivers



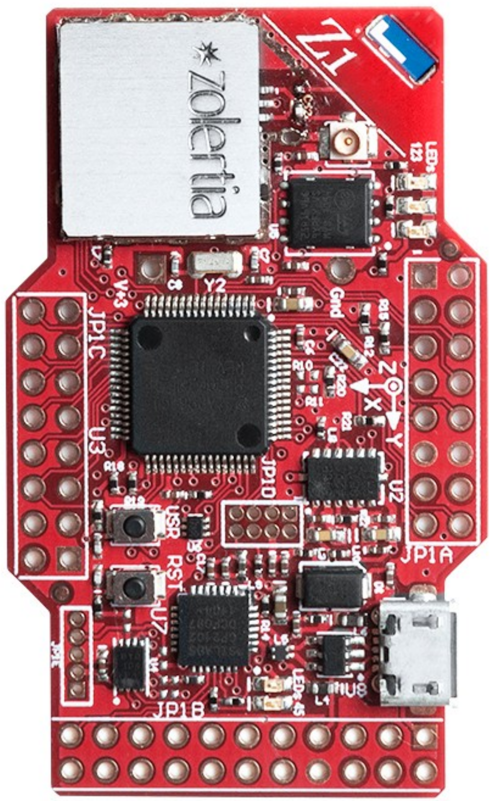
LPC1102 ARM Cortex-M0
Source: NXP (April 20th, 2010)

- **Computer science challenges**
  - energy aware task scheduling
  - energy aware communication protocols
    - asynchronous communications, routing algorithms, in-network processing/data aggregation, ...
  - new programming paradigms
    - real-time, event-driven, limited resources, deeply embedded computing, fault-tolerant, adaptative behavior

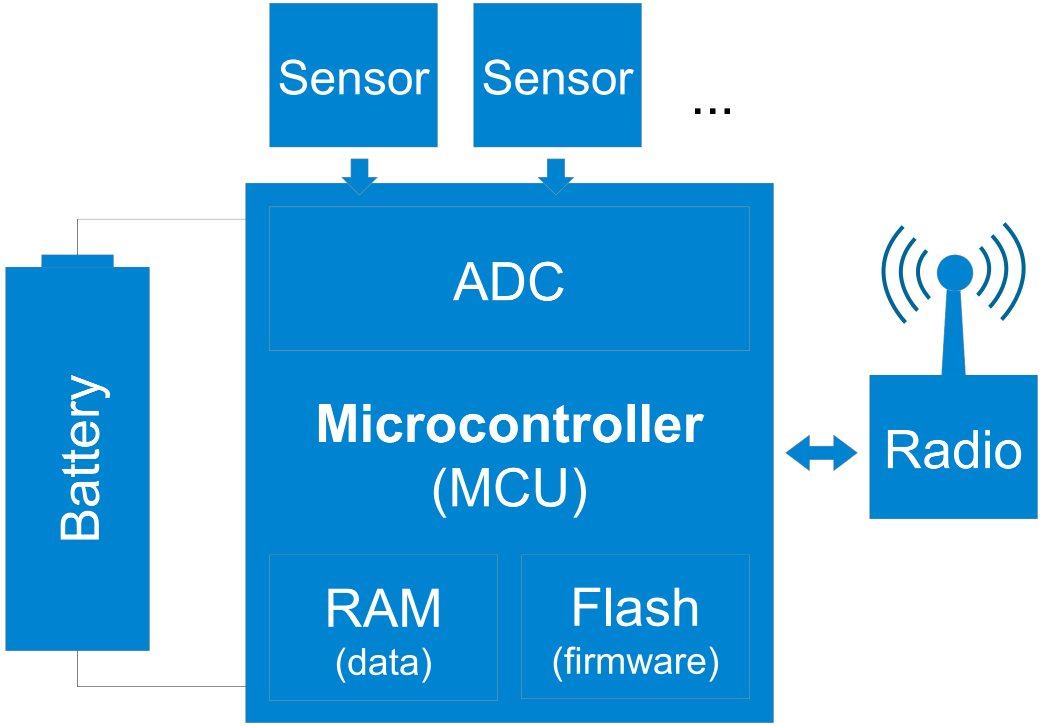
Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

What's in a Sensor Node ?



Zolertia Z1



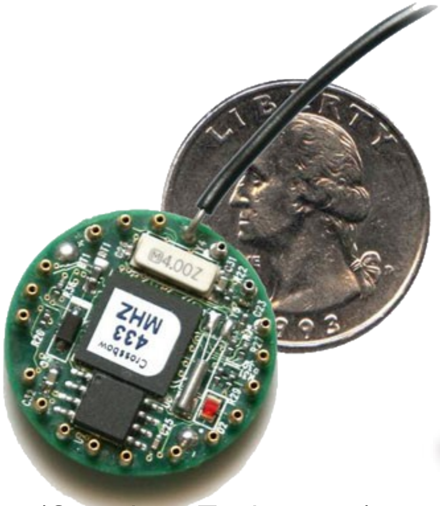
Typical Sensor Node Hardware

- **Limited MCU/CPU**
  - low frequency clock : ~ 10 MHz — *200 times slower*
  - typically 8/16-bits MCU (trend towards 32-bits)
  - no hardware floating point operations (FPU)
  - no cache, no MMU, no MPU, basic instruction pipeline
- **Limited memory**
  - RAM ~ 10 KB — *10^5 times smaller*
  - Flash (program + data) ~ 100 KB — *10^7 times smaller*
- **Low-power radio**
  - Range ~ 10-100 m
  - Data rate ~ 100 Kbps — *10^4 times slower*
- **Power supply constrained**
  - battery, solar, harvesting (vibrations), ...

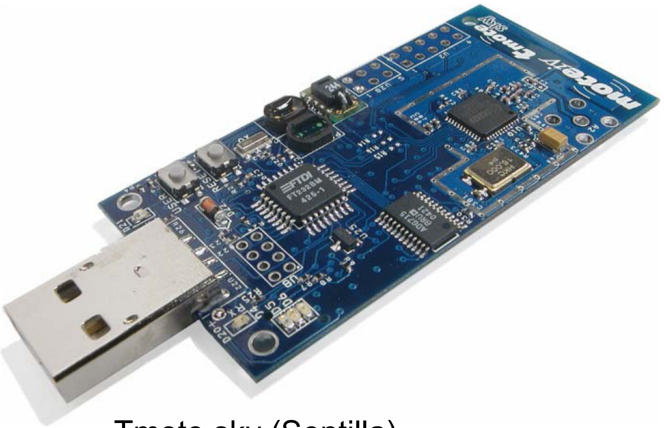
FPU: *floating-point unit*; MMU: *memory management unit*; MPU: *memory protection unit*

Example prototypes

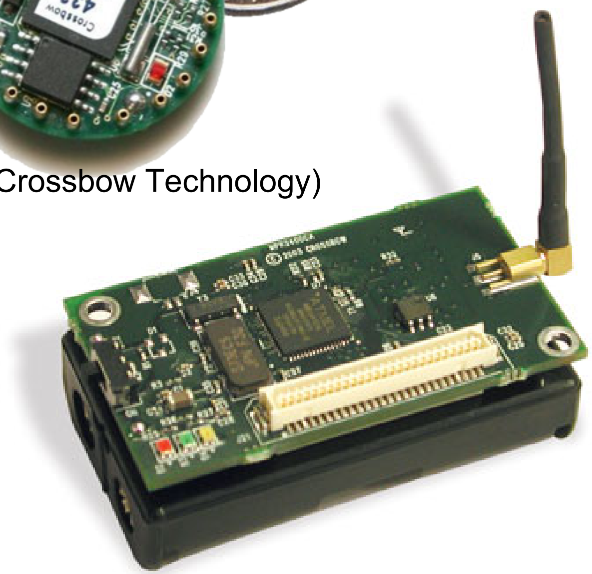
- *Motes*...



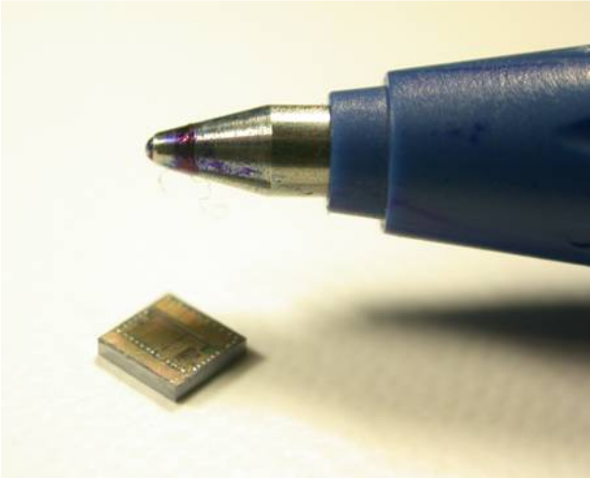
mica2dot (Crossbow Technology)



Tmote sky (Sentilla)



micaZ (Crossbow Technology)



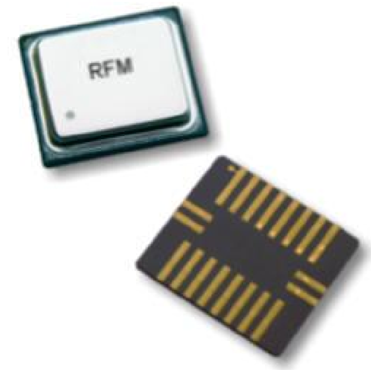
Spec (Berkeley)

Example microcontrollers (MCUs)

Name	Manufacturer	Datapath width (bits)	RAM (kB)	ROM (kB)	Current consumption: active/sleep (mA)
MSP430xF168	Texas Instr.	16	10	48	2 / 0.001
AVR ATmega128	Atmel	8	8	128	8 / 0.02
8051	Intel	8	0.5	32	30 / 0.005
PIC18	Microchip	8	4	128	2.2 / 0.001
CC2538	Texas Instr.	32	32	512	N/A / 0.0016

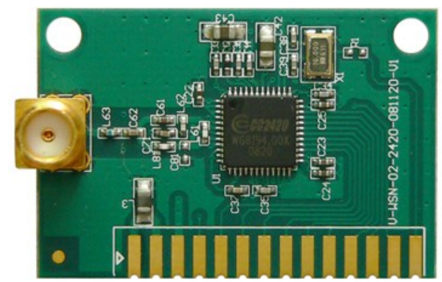
Adapted from “Interconnecting Smart Objects with IP”, J.-P. Vasseur and A. Dunkels, Morgan-Kaufmann, 2010

Example radio transceivers



10.8 X 9.52 mm

- RFM TR1000 family (RF Monolithics)
  - 868MHz and 916 MHz ranges, up to 115.2 kbps, ON-OFF keying or ASK
- CC1000 and CC2420 families (Texas Instruments)
  - CC1000: 300-1000MHz, FSK
  - CC2420: 2.4GHz, IEEE 802.15.4
- MRF24J40 (Microchip)
  - 2.4GHz, IEEE 802.15.4
- AT86RF23x (Atmel)
  - 2.4 GHz, IEEE 802.15.4



where IEEE 802.15.4 in the 2.4 GHz band
uses DSSS and provides a data rate of 250kbps

Sensors

- **Passive**
  - Light, sound
  - Humidity, pressure, temperature
  - Angular/linear velocity
  - Vibration, mechanical stress, tension in material
  - Chemical sensor sensitive to specific substance
  - Smoke detector
  - Camera, ...
- **Active**
  - sonar/radar, seismic, ...

- **Also actuators...**
  - LED, relay, motor, ...

Power sources

- **Storing energy: batteries**
  - Traditional batteries: rechargeable/non-rechargeable
  - ex.: 2100 mAh NiMh AAA battery (9072 J)
- **Energy harvesting**
  - Photovoltaics (solar cells)
  - Temperature gradients
  - Vibrations (e.g. piezoelectric principle)
  - Pressure variations
  - Flow of air/liquid

- **Metrics: energy/power per volume/area**
  - units : Joules or Watts per m^3 or m^2
  - above NiMh battery ~1000 J/cm^3
  - Solar (outdoor) : up to 15 mW/cm^2
  - Vibrations: 0.01-0.1 mW/cm^3

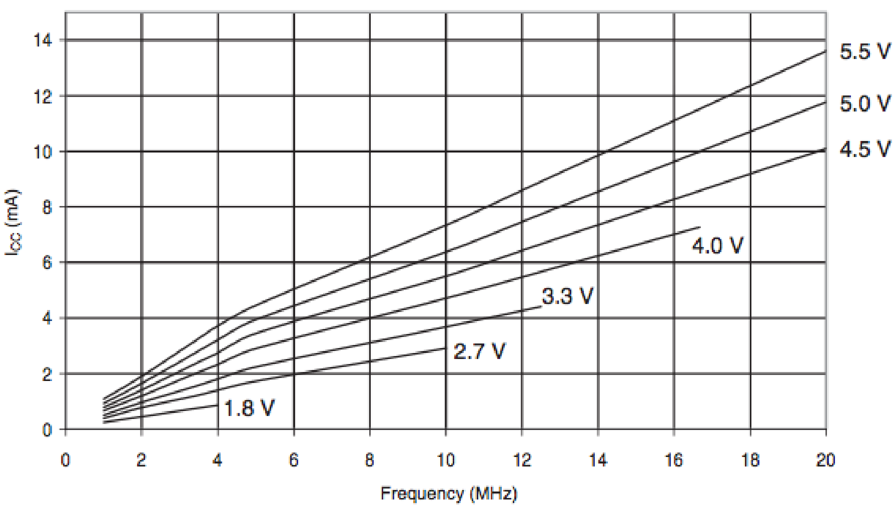
Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
  - **MCU energy consumption**
  - Radio energy consumption
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

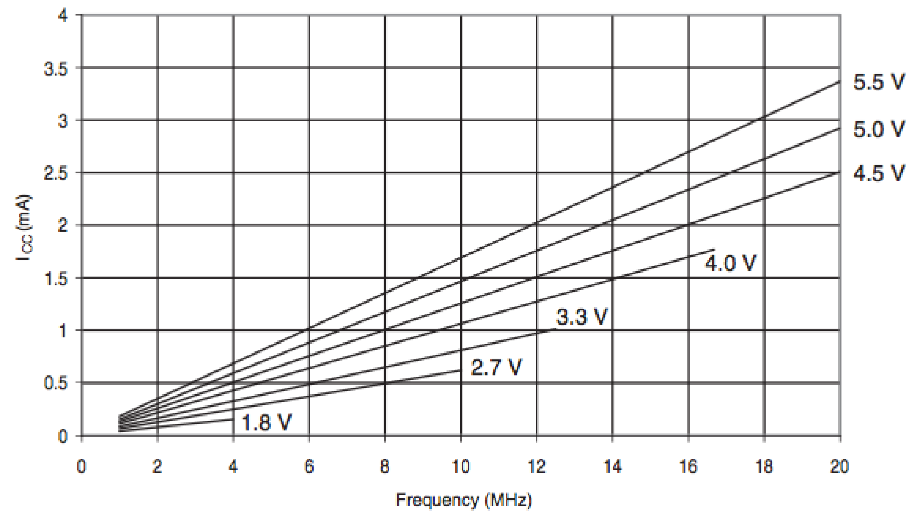
MCU Energy Consumption

- **Operation states with different power consumption**
  - Core technique for energy-efficient wireless sensor node
  - Example: 3 states
    - "active" = normal operation mode, maximum power cons.
    - "idle" = lower frequency, some peripherals power off
    - "sleep" = no operation but low-freq. timer to wakeup node

"deeper" sleep



active mode



idle mode

Source: ATmega328P user manual, Atmel Corporation, Rev. 8271C–AVR–08/10

MCU Energy Consumption

- **Operation states with different power consumption**
  - Core technique for energy-efficient wireless sensor node
  - Example: 3 states

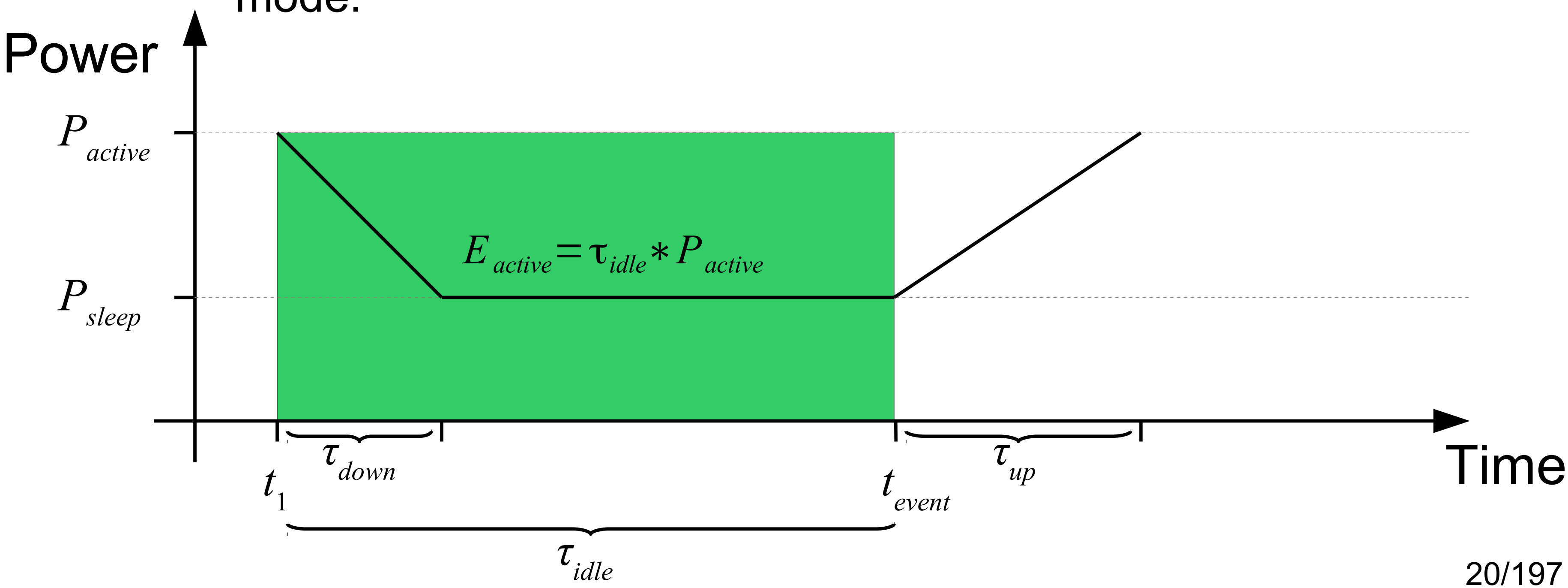
“deeper” sleep

- “active” = normal operation mode, maximum power cons.
- “idle” = lower frequency, some peripherals power off
- “sleep” = no operation but low-freq. timer to wakeup node

  - BUT transitions between states are not free: require time, hence energy.
    - Causes: wait for clock to stabilize, wait for PLL to settle, ...
    - Usually, the deeper the sleep state, the more time and energy it takes to recover (can be in the order of several thousand clock cycles !)
  - Question: when is it worth switching state ?

MCU Energy Consumption

- **Example model**
  - Suppose at t_1, nothing to do until t_{event}.
  - Need to take decision to go to sleep or not.
  - Going to sleep takes time τ_{down} while leaving sleep takes τ_{up}. Power consumption is P_{active} in active state and P_{sleep} in sleep mode.



MCU Energy Consumption

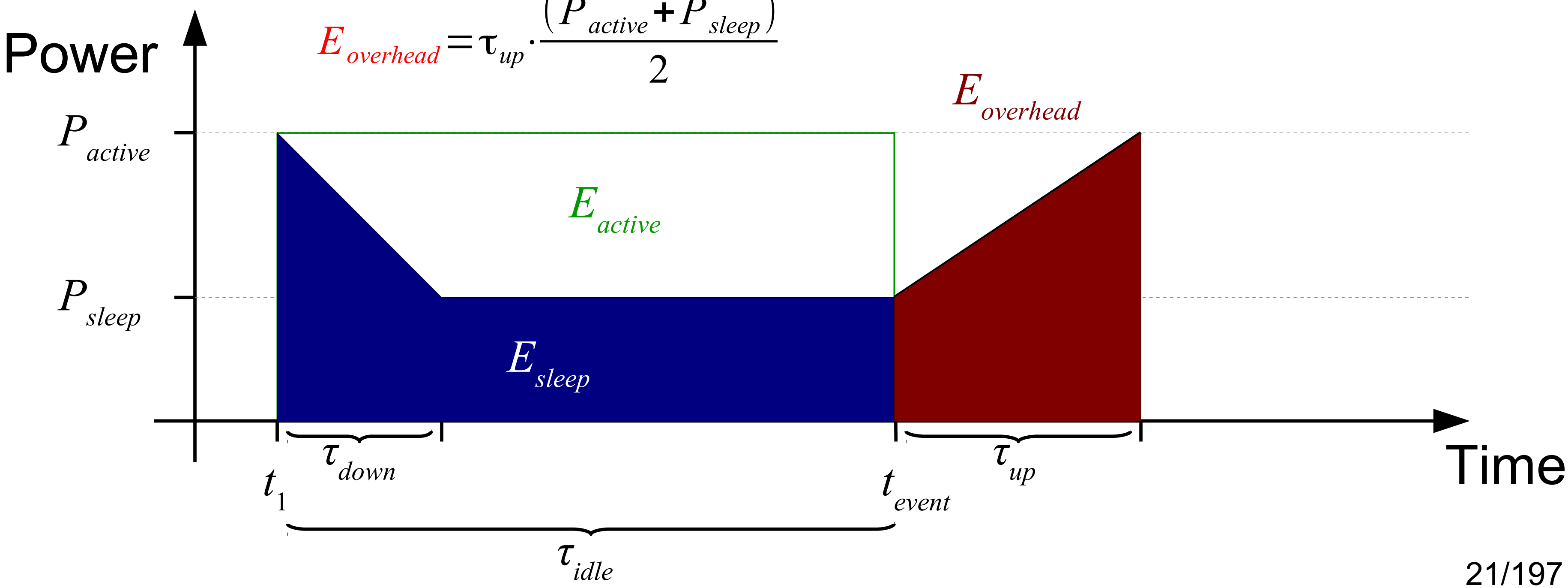
- **Example model**
  - If node does not go to sleep

$$E_{active} = \tau_{idle} * P_{active}$$

  - If node goes to sleep

$$E_{sleep} = \tau_{down} \cdot \frac{(P_{active} + P_{sleep})}{2} + (\tau_{idle} - \tau_{down}) \cdot P_{sleep}$$

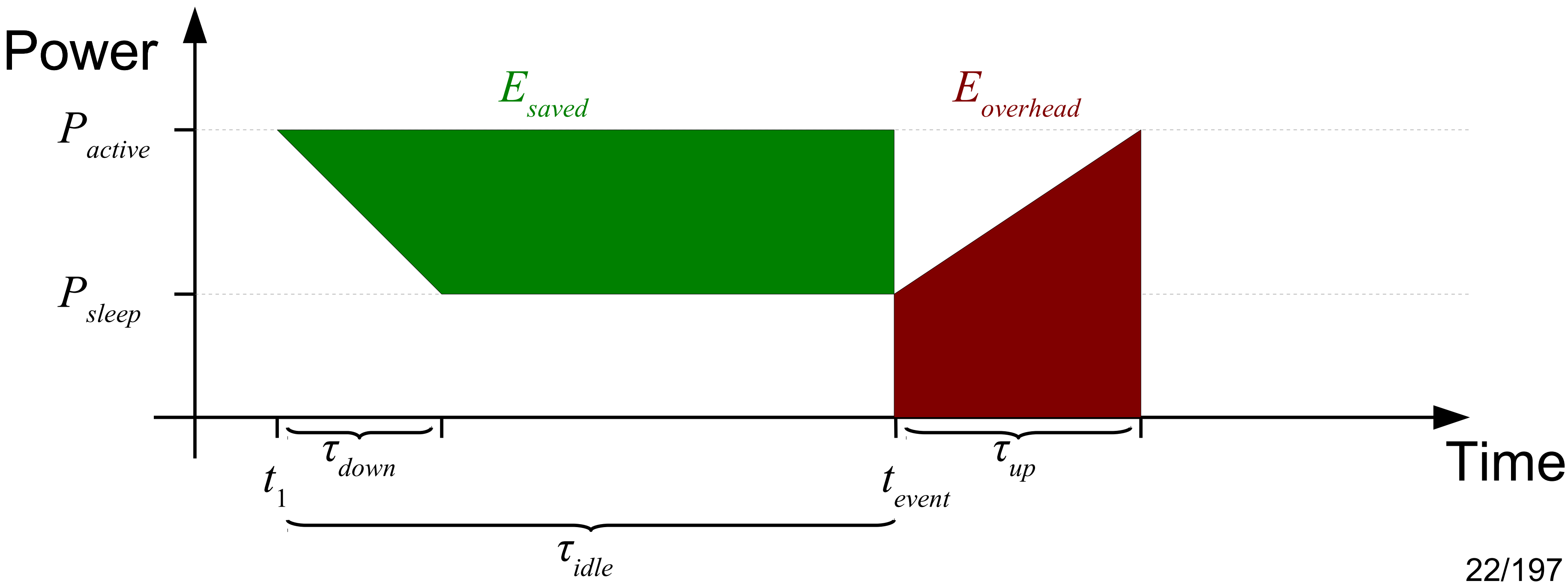
$$E_{overhead} = \tau_{up} \cdot \frac{(P_{active} + P_{sleep})}{2}$$



MCU Energy Consumption

- **Example model**

$$E_{saved} = E_{active} - E_{sleep}$$
$$= \tau_{idle} \cdot P_{active} - \left(\tau_{down} \cdot \frac{\left(P_{active} + P_{sleep} \right)}{2} + \left(\tau_{idle} - \tau_{down} \right) \cdot P_{sleep} \right)$$
$$= \tau_{idle} \cdot \left(P_{active} - P_{sleep} \right) - \tau_{down} \cdot \frac{\left(P_{ative} - P_{sleep} \right)}{2}$$

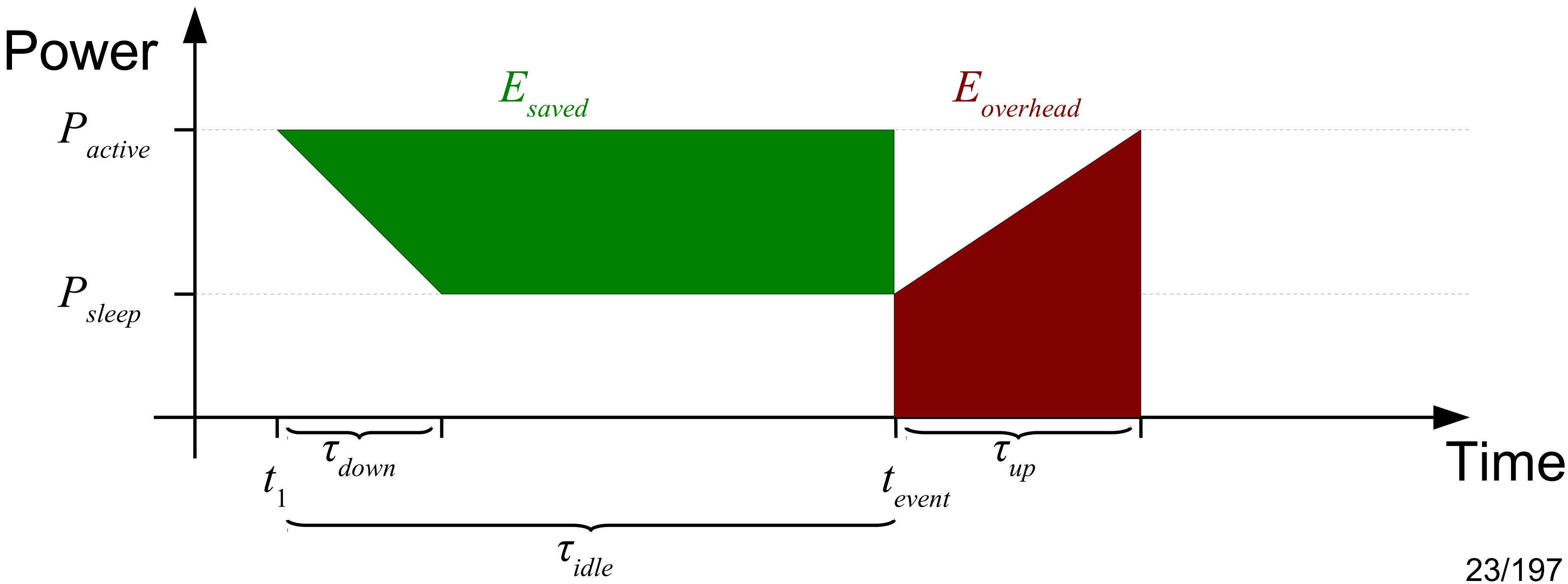


MCU Energy Consumption

- **Example model**
  - Going to sleep is interesting iff $E_{overhead} < E_{saved}$

$$\tau_{up} \cdot \frac{(P_{active} + P_{sleep})}{2} < \tau_{idle} \cdot (P_{active} - P_{sleep}) - \tau_{down} \cdot \frac{(P_{ative} - P_{sleep})}{2}$$

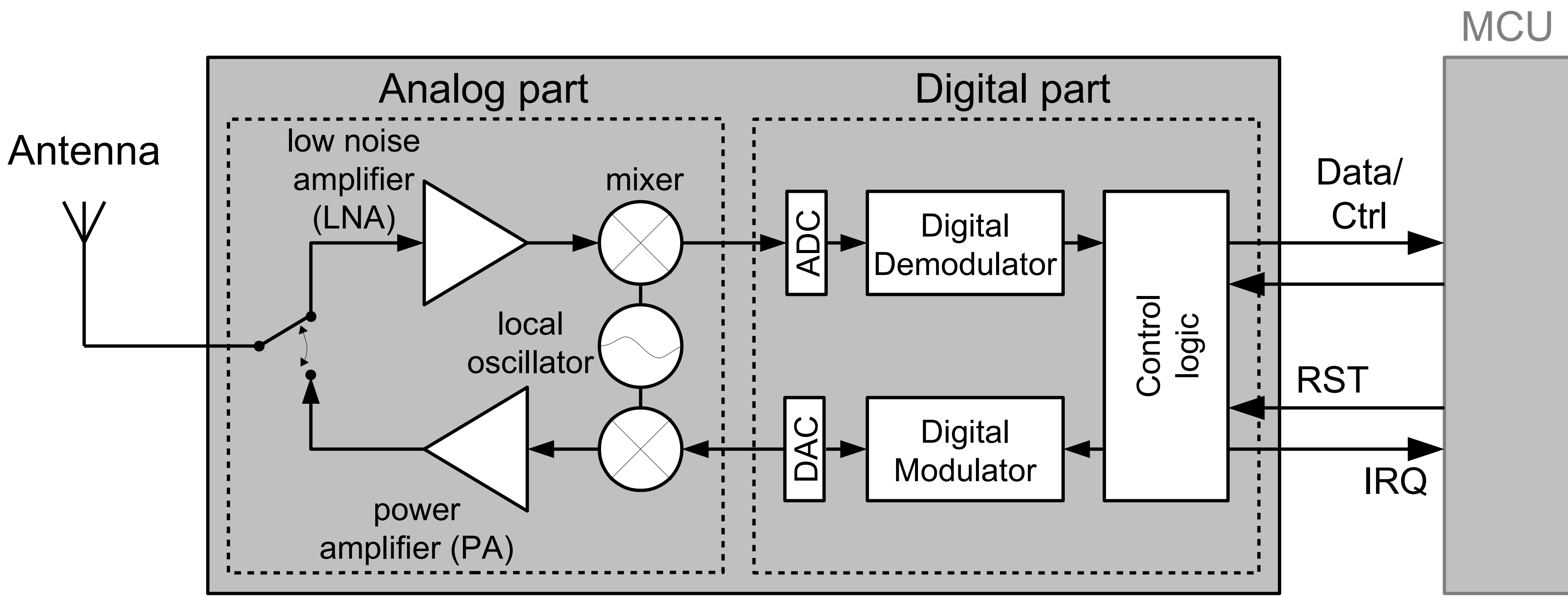
$$\boxed{t_{event} - t_1 > \frac{1}{2}\left(\tau_{down} + \frac{P_{active} + P_{sleep}}{P_{active} - P_{sleep}} \cdot \tau_{up}\right)}$$



Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
  - MCU energy consumption
  - **Radio energy consumption**
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

What's inside a Radio Transceiver ?



Radio Transceiver States

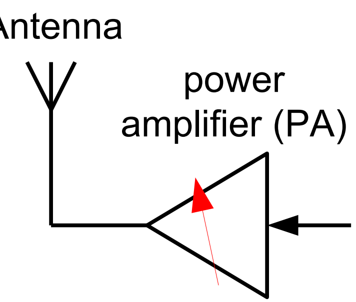
- **Transmit**
  - transceiver active (analog/digital Tx), antenna radiates energy
- **Receive**
  - transceiver active (analog/digital Rx), receive part
  - power consumption mainly due to LNA
- **Idle**
  - ready to receive, but not currently receiving
  - analog Rx is active (LNA as well)
  - power consumption similar to *Receive* state
- **Sleep**
  - most parts of the transceiver are switched OFF
  - usually, it takes some time to recover
  - hard to only wakeup for frames addressed to the local node (needs complex filtering circuitry, hence power)

Power consumption example

- **CC2420 transceiver**
  - Power supply: 3.3 V
  - Transmit current: 22.7 mA (~74.91 mW)
    - for a radiated power of ~0.9 mW !
    - TX efficiency : 0.9 mW / 74.91 mW = 0.01 %
      (welcome to the wireless world !)
  - Receive current: 25.2 mA (~ 83.2 mW)
  - Sleep current: 12 μA

- **Achieved autonomy with 2500 mAh battery**
  - sinking 25 mA current, device would get power during approximately 100 hours (~ 4 days)
  - when sleeping most of the time, almost 23 years
    (ignoring battery self-discharge + MCU)

Dynamic Power Management ?



- **Transmit side**
  - Adapt Tx power (PA[1] control knob)
  - BUT Tx power consumption not directly proportional to energy radiated by antenna → reducing the Tx power might not reduce power consumption a lot !
- **Receive side**
  - can't change
- **Radio duty cycle**
  - only solution is to **go to sleep as much as possible**
  - "active" period: node can listen to others
  - "sleep" period: node does nothing.
  - Fundamental to WSNs...

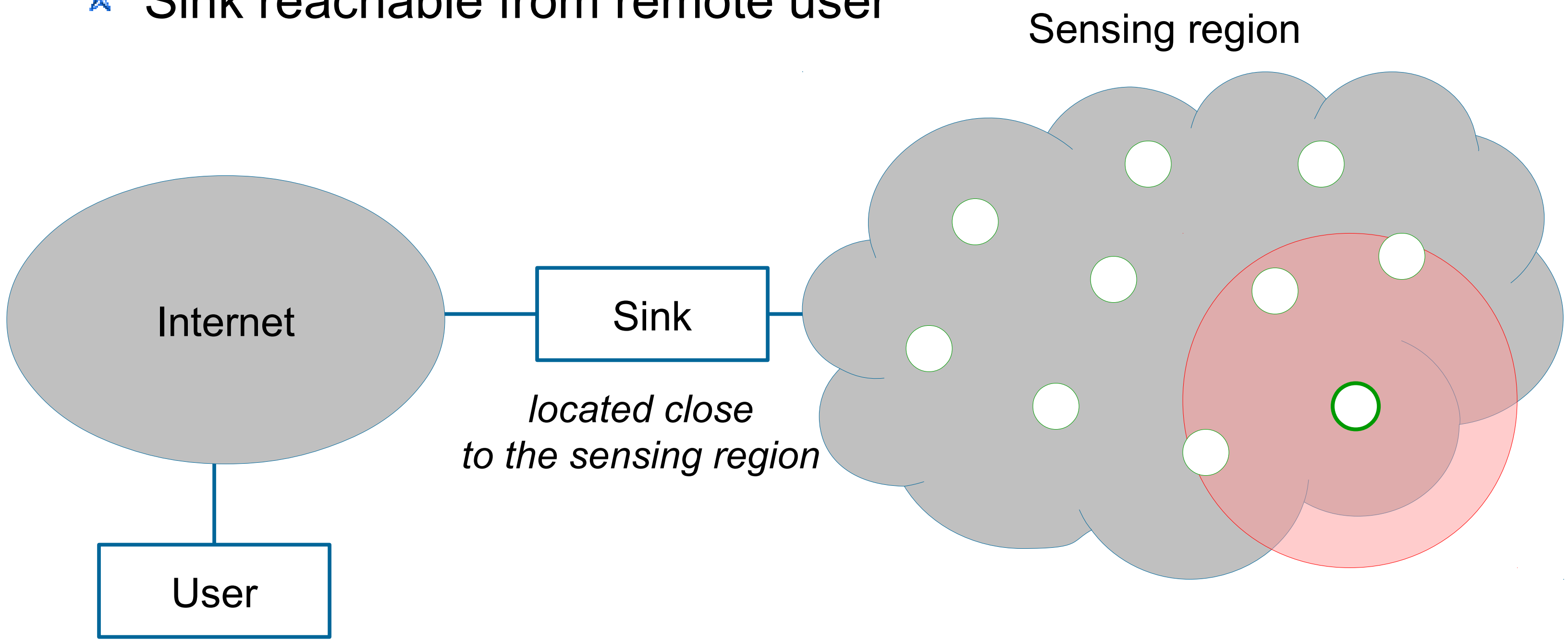
(1) Power Amplifier

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

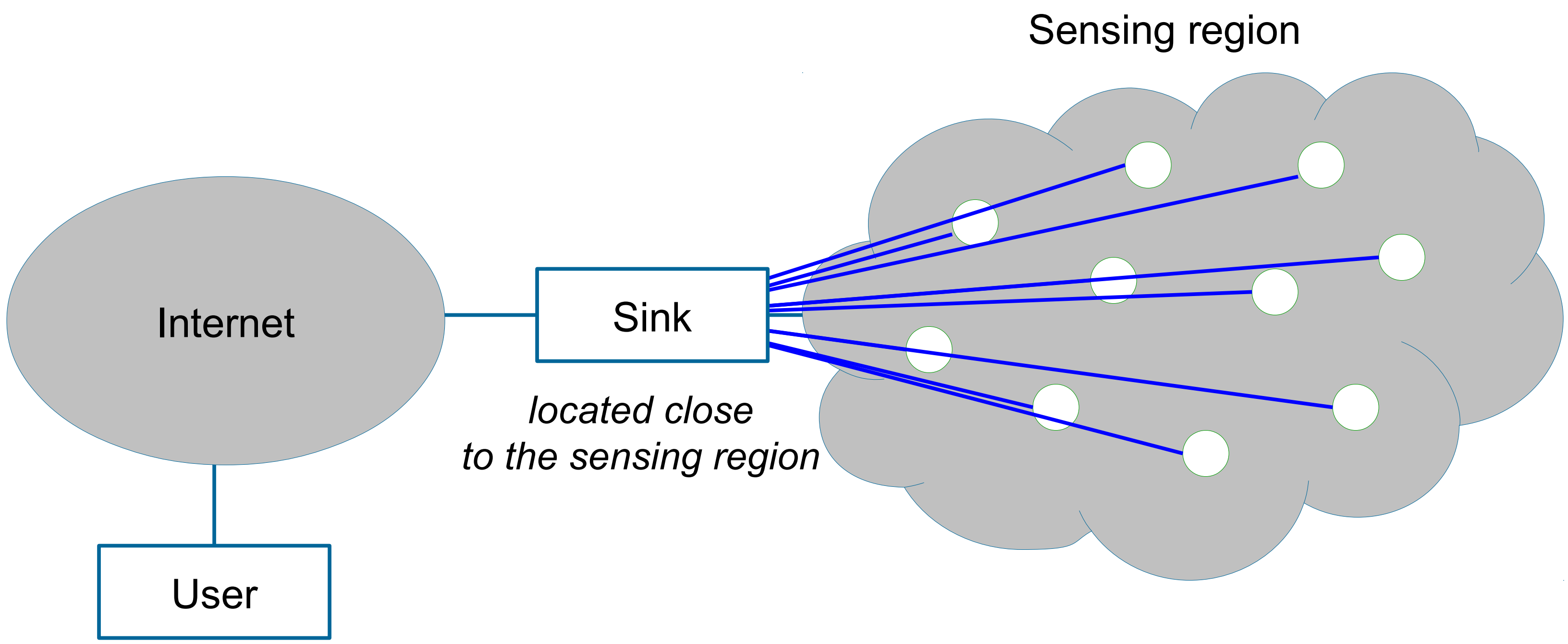
Sensor Network Architecture

- **Sensor network architecture**
  - Sensors perform measurements
  - Measures collected by special "*sink*" node
  - Sink reachable from remote user



Sensor Network Architecture

- **Single-hop**
  - long distance → high transmission cost
    (recall that power increases exponentially with distance)



Sensor Network Architecture

- **Recall path-loss equation**
  - Friis free-space equation

$$P_{rx}(d) = P_{tx} G_t G_r \frac{\lambda^2}{(4\pi)^2 d^2 L}$$

$$= P_{tx} G_t G_r \frac{\lambda^2}{(4\pi)^2 d_0^2 L} \cdot \left(\frac{d_0}{d}\right)^2 = P_{rcvd}(d_0)\left(\frac{d_0}{d}\right)^2$$

  - where
    - P_{tx} is the transmission power
    - G_t and G_r are antenna gains at transmitter and receiver
    - d_0 is the reference, far-field, distance
    - d is the distance between receiver and transmitter
    - λ is the wavelength
    - L summarizes tx/rx circuit losses.

Sensor Network Architecture

- **Recall path-loss equation**
  - Generalization to non-free-space environments

$$P_{rcvd}(d) = P_{rcvd}(d_0) \cdot \left(\frac{d_0}{d}\right)^{\gamma}$$

    - where γ is the path-loss exponent and depends on the environment

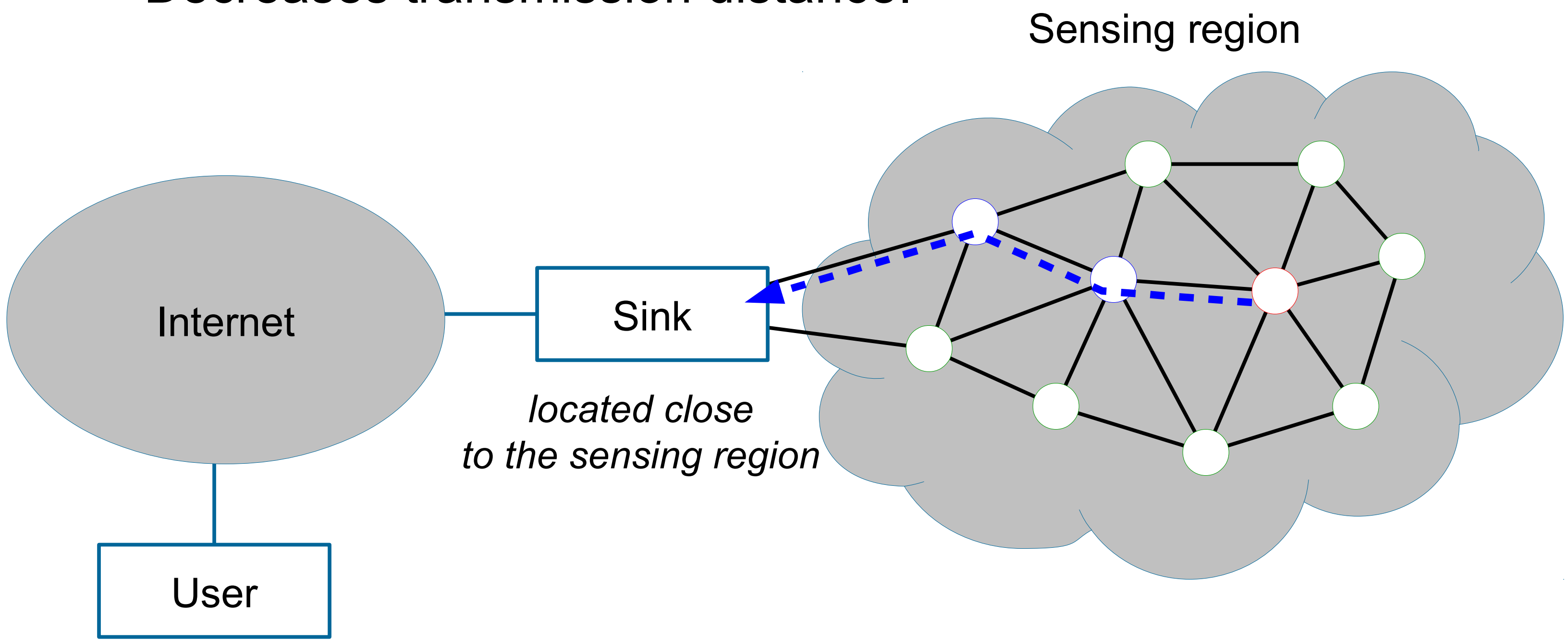
  - Consequence: if distance doubles, P_{tx} must be multiplied by 2^{γ} to keep the same received power !

    → Objective: minimize Tx distance

  - Two approaches
    - multi-hop (mesh) networks
    - clustering

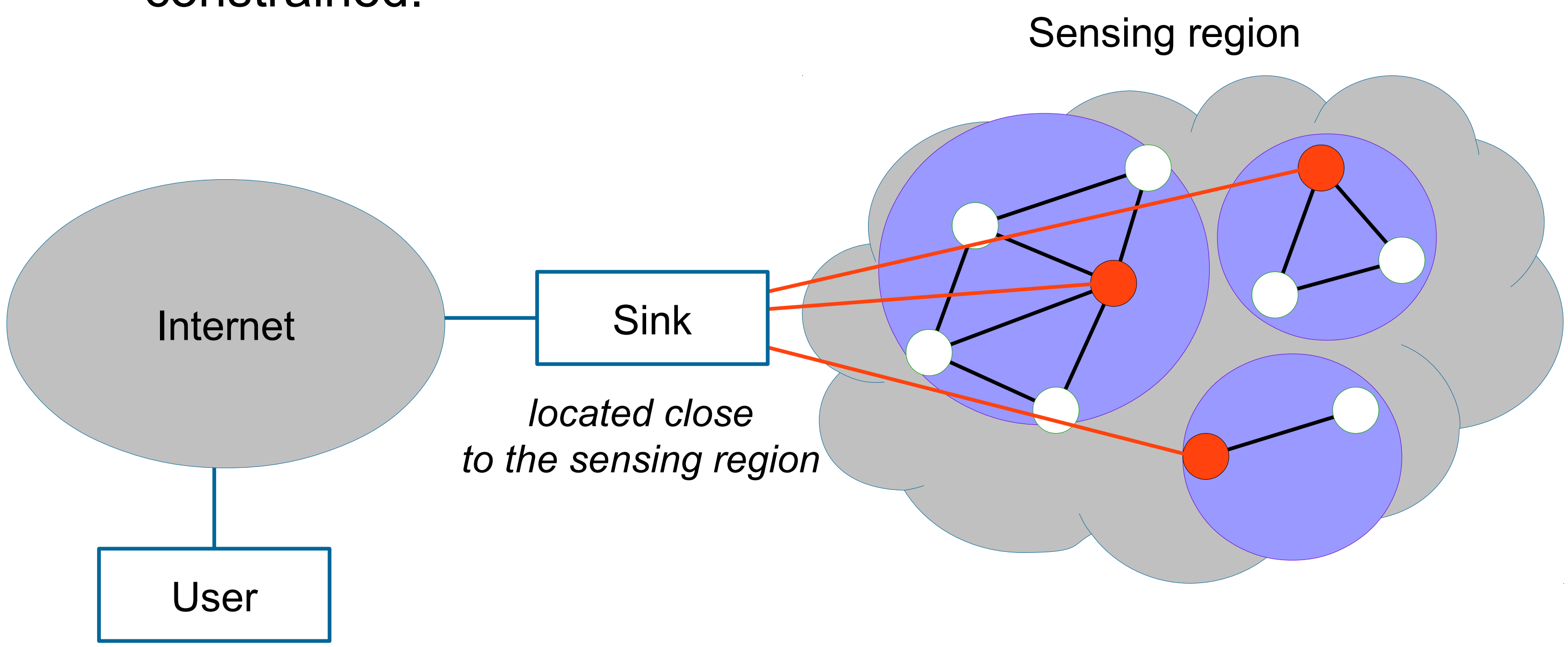
Sensor Network Architecture

- **Multi-hop**
  - In addition to sending own measurements, each node can act as a **relay** for other nodes' messages.
  - Decreases transmission distance.



Sensor Network Architecture

- **Multi-hop clustering**
  - Multiple sinks aggregate traffic from a cluster of nodes.
  - Cluster head nodes expected to be more powerful, less energy constrained.



Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
  - centralized
  - synchronous
  - asynchronous
  - IEEE 802.15.4
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

MAC layer

- **Lessons learned from traditional Wireless MAC protocols**
  - CSMA/CD cannot be used: emitter not able to sense collision at the receiver

    → **CSMA/CA**
  - Hidden-terminal, exposed-terminal issues
    → **RTS/CTS**
    - but RTS/CTS introduce significant overhead (latency before transmission + bandwidth for control messages)

- **Why not use existing wireless protocols ?**
  - Bluetooth: requires a permanent master to do polling, limited number of active slaves in a “piconet”
  - Bluetooth Low Energy (BLE): only single-hop
  - Wi-fi (802.11): requires all nodes to be constantly listening

MAC layer for WSNs

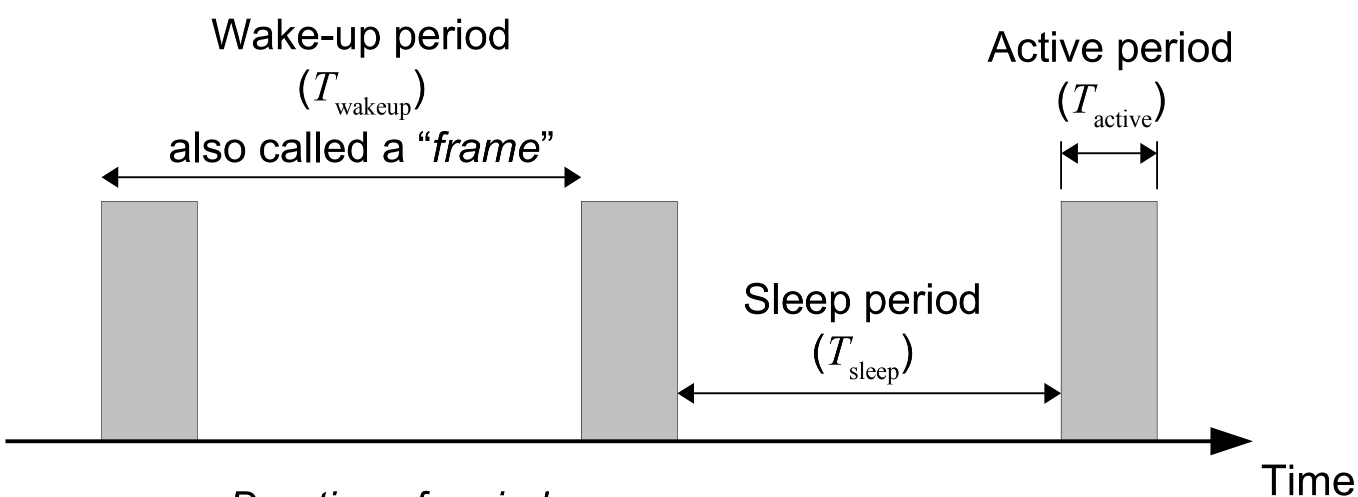
- **Requirements**
  - Need to conserve energy
    - very different from traditional WLANs
  - Scalability and robustness against frequent topology changes
    - nodes powering down temporarily (save energy)
    - mobility
    - deployment of new nodes
    - death of existing nodes (failure, battery power exhausted)

MAC layer for WSNs

- **Energy problems**
  - **Collisions**
    - Energy wasted at transmitter and receiver
      → avoid collisions as much as possible !
  - **Overhearing**
    - Wireless = broadcasting → messages received by several nodes not interested → listen, then drop
      → avoid listening for useless messages !
  - **Protocol overhead**
    - MAC-related control frames (e.g. RTS, CTS), packet headers
      → keep protocol simple, avoir unnecessary messages !
  - **Idle listening**
    - Energy wasted when nothing to send/receive
      → go to sleep as frequently as possible !

MAC layer for WSNs

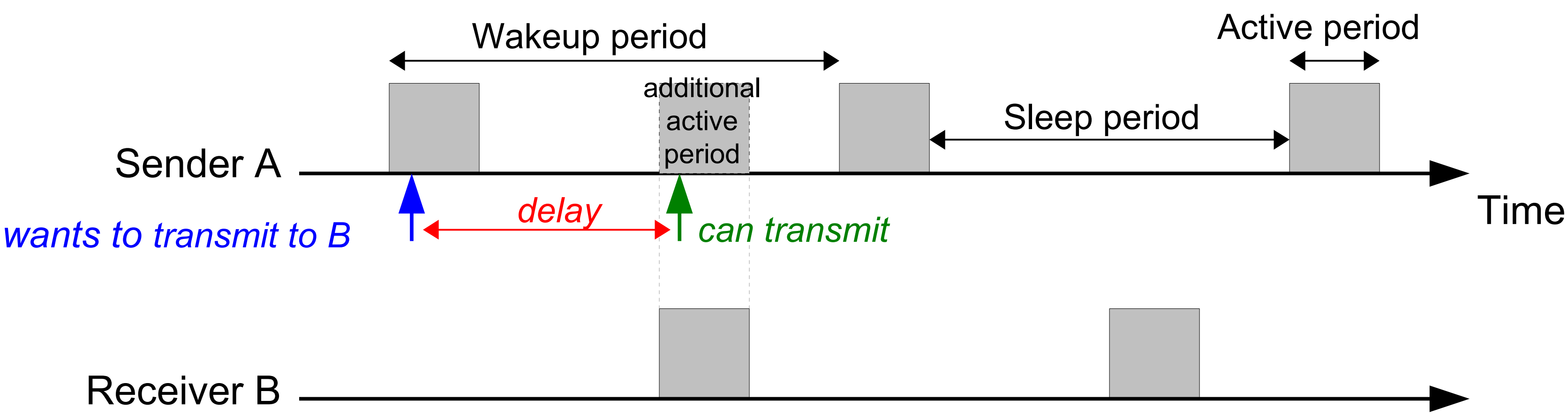
- **Periodic wakeup scheme** - *radio duty-cycling* (RDC)
  - Principles
    - Nodes alternate between *active* and *sleep* periods according to their own schedule
    - Active period used to receive and transmit
    - Duty-cycle ratio $\dfrac{T_{active}}{T_{wakeup}} = \dfrac{T_{active}}{T_{active} + T_{sleep}}$



Duration of periods
- active (fixed) : depends on PHY- and MAC-layers.
- sleep : depends on APP-layer requirements.

MAC layer for WSNs

- **Periodic wakeup scheme**
  - Requirement
    - need to coordinate the schedules of neighboring nodes such that their active periods start at the same time.



  - Consequence
    - The sleep period introduces additional latency.
    - For multi-hop communications (between non-adjacent nodes), this latency is introduced at each hop !

Low Duty-Cycling Protocols

- **Centralized solution**
  - e.g. Mediation device
- **Distributed protocols.**
  - **Synchronous**
    - predetermined periodic wake-up schedule ($T_{\text{sleep}} + T_{\text{active}}$)
    - explicit sharing of schedule with neighboring nodes
    - synchronization maintained at local scale
    - e.g. PACT, LEACH, **S-MAC**, **T-MAC**
  - **Asynchronous**
    - no a priori wake-up schedule shared
    - frequent channel sampling (*low-power sampling* – LPL)
    - optimization: automatically learn neighbor schedule
    - e.g. WiseMAC, **B-MAC**, **X-MAC**, **ContikiMAC**

see also *The MAC Alphabet Soup*
http://www.st.ewi.tudelft.nl/~koen/MACsoup/index.php

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
    - **centralized**
    - synchronous
    - asynchronous
    - IEEE 802.15.4
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

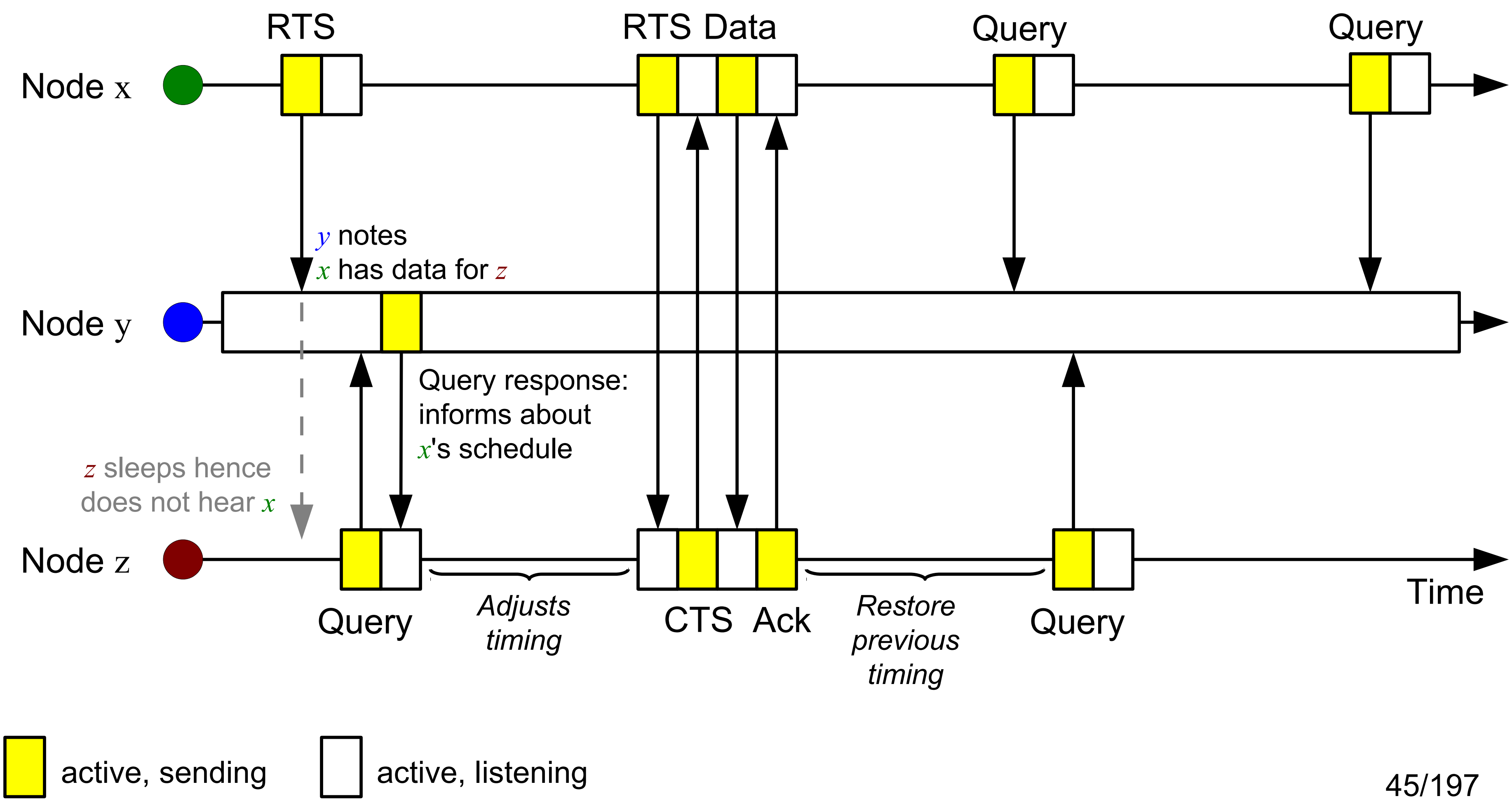
MAC layer - Mediation Device Protocol

- **Principles**
  - No global time reference
  - Each node has its own sleep schedule
  - Assumption : mediation device has no energy constraint

  - Operations
    - When a node wakes up it transmits a short query beacon
      → indicates it is willing to receive others' packets
    - Then, it stays awake for a short time. If no packet is received it goes back to sleep.
    - If a node wants to transmit a packet to a neighbor, it must synchronize with it.
    - The mediation device allows nodes to synchronize without having to stay awake for a long time !

MAC layer - Mediation Device Protocol

- **Example**



MAC layer - Mediation Device Protocol

- **Summary**
  - Advantage
    - No need for global synchronization
    - Most of the energy burden is shifted to the mediation device
  - Drawbacks
    - Requires an energy unconstrained mediation device
    - If multiple nodes pick the same schedule, they might send their query beacon at the same time → collisions[1].
  - Further work
    - reschedule message in case of repeated collisions
    - increased coverage : distributed mediation device protocol

(1) can be solved by e.g. jitter or CSMA/CA.

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
  - centralized
  - **synchronous**
  - asynchronous
  - IEEE 802.15.4
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

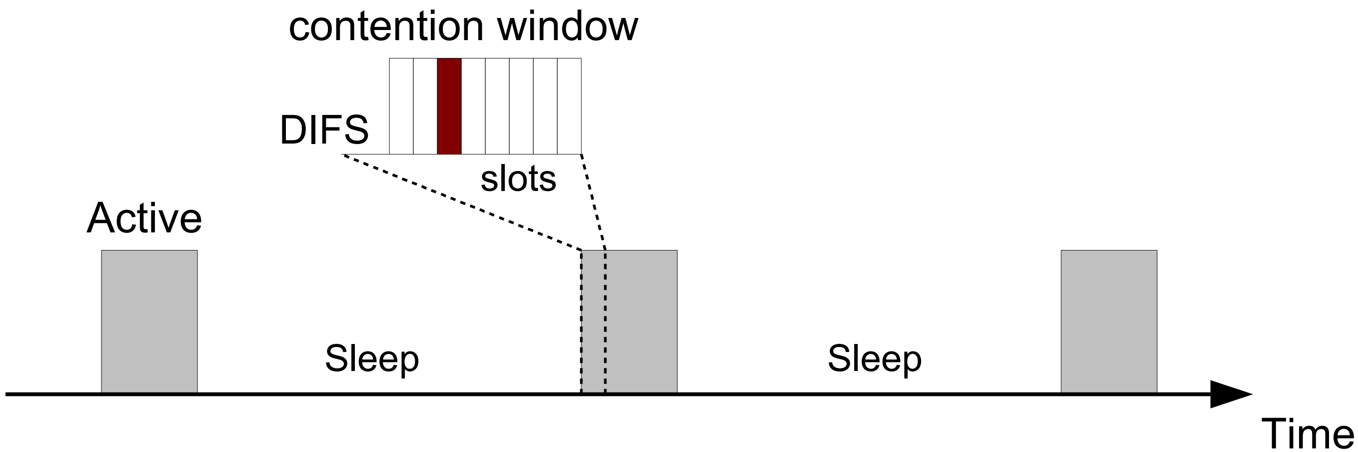
MAC layer – Sensor MAC (S-MAC)

- **Principles**
  - Minimize idle-listening: low duty cycle
  - Minimize contention (and collision)
    - Broadcast → CSMA/CA
    - Unicast → CSMA/CA with RTS/CTS
  - Overhearing
    - RTS frames contain destination field + duration (NAV)
    - Other nodes can go to sleep and know for how long
  - Synchronous
    - Explicit sharing of schedules (SYNC frames)
    - Local synchronization of schedules (virtual clusters)

Reference: ***Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks***", W. Ye, J. Heidemann, and D. Estrin, IEEE/ACM Transactions on Networking, Volume 12, issue 3, 2004

MAC layer - S-MAC

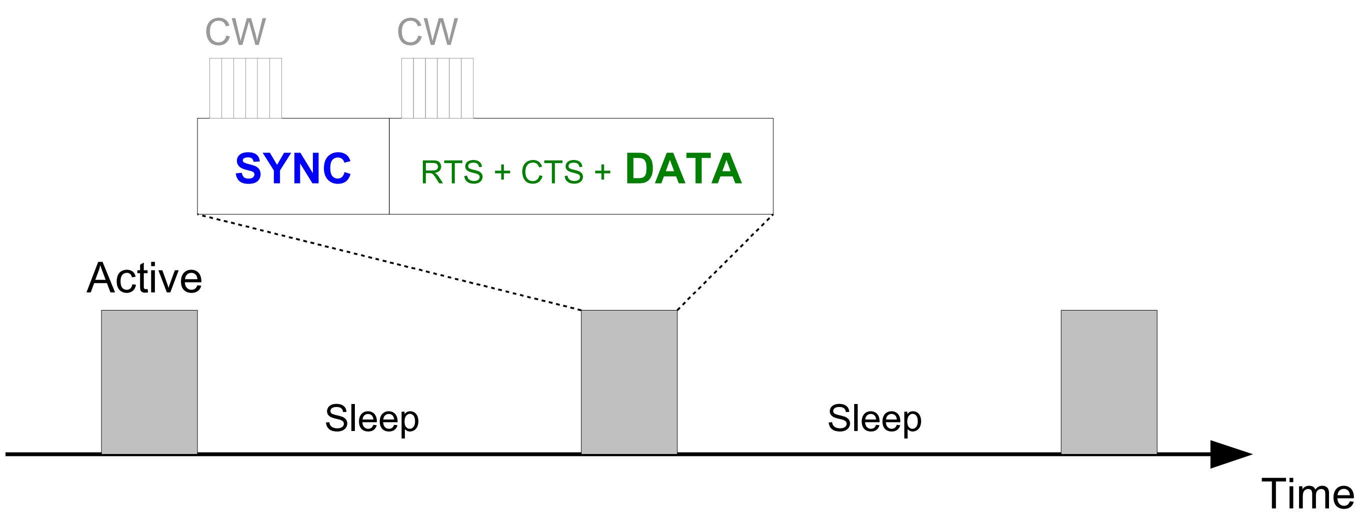
- **Collision avoidance**
  - CSMA/CA with random back-off[1]
    - Limits the likelihood of collision
    - A node that wants to transmit picks a slot randomly in contention window, checks if the channel remains free until its slot. If the channel is free, transmission starts. Otherwise the node backs off until the next wake-up period.



(1) In the original TinyOS implementation, the contention window had a fixed duration.
This differs from IEEE802.11 where the contention window's size doubles with each re-transmission.

MAC layer - S-MAC

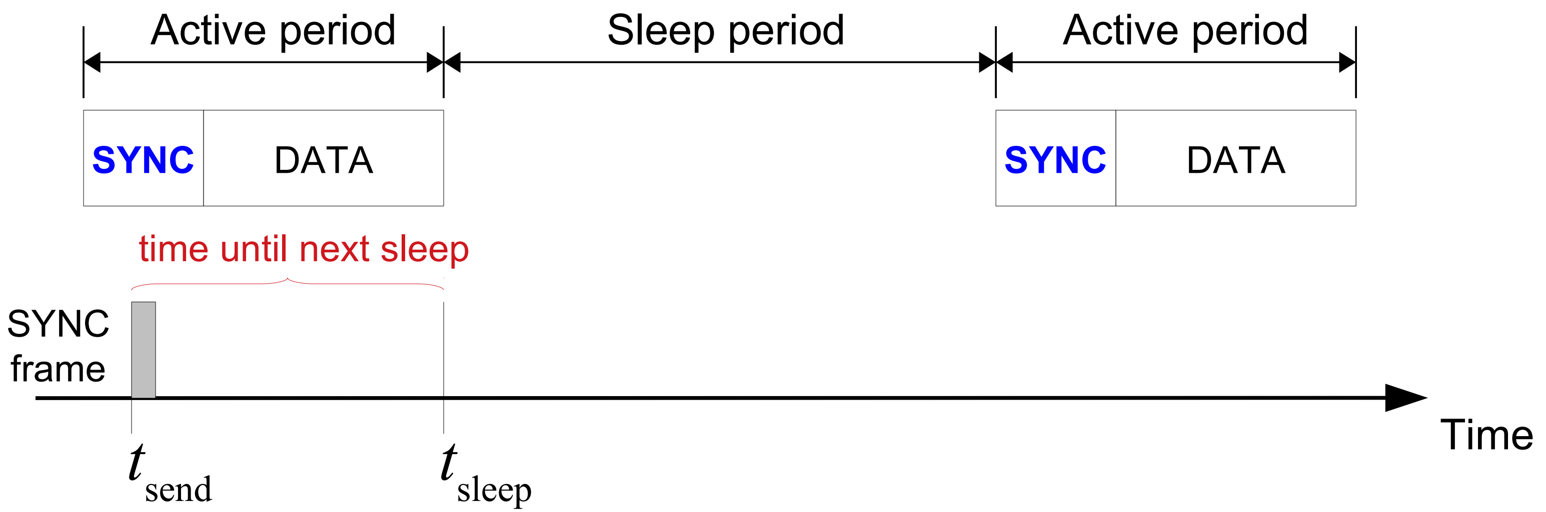
- **Active period**
  - Divided in two phases
    - **SYNC** : used for nodes to transmit their own wake-up schedule.
    - **DATA** : used to send frames. Unicast frames come after an RTS/CTS exchange (similar to IEEE 802.11). Broadcast frames are sent without RTS/CTS.



Note 1 : there is an SFIS delay between CTS / RTS / DATA frames and ACK frames.

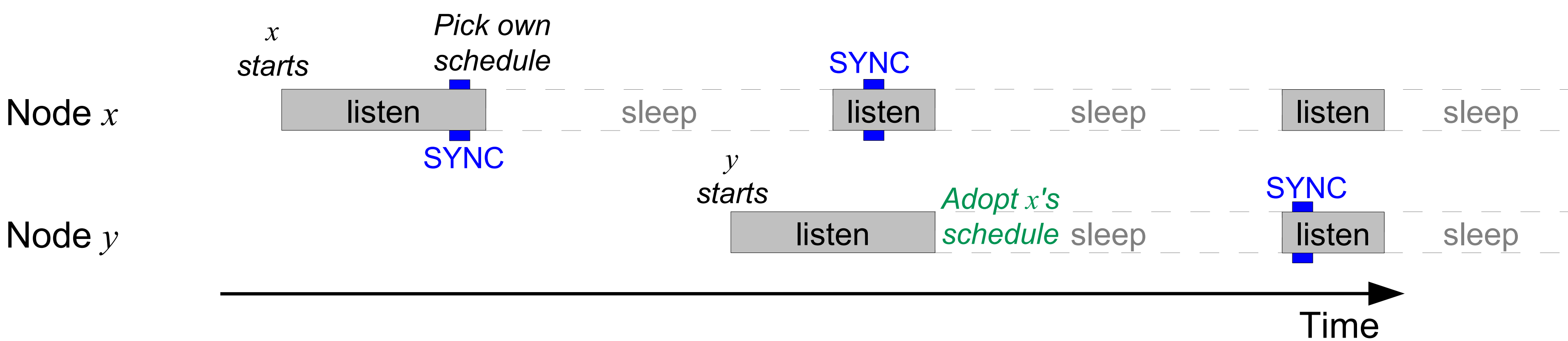
MAC layer - S-MAC

- **Sharing schedules - SYNC period**
  - Nodes accept or broadcast SYNC frames from/to neighbors.
  - SYNC frame includes the *sender ID* and the amount of time until next sleep. This time is relative to when the SYNC frame is sent (t_{send}).



MAC layer - S-MAC

- **Picking a wake-up schedule**
  - When a node boots up : it must determine its wake-up schedule. It first listens for SYNC frames during a fixed amount of time(1)
    1. no SYNC frame heard → pick its own schedule and advertise it with a SYNC frame.
    2. SYNC frame heard → follow the received schedule

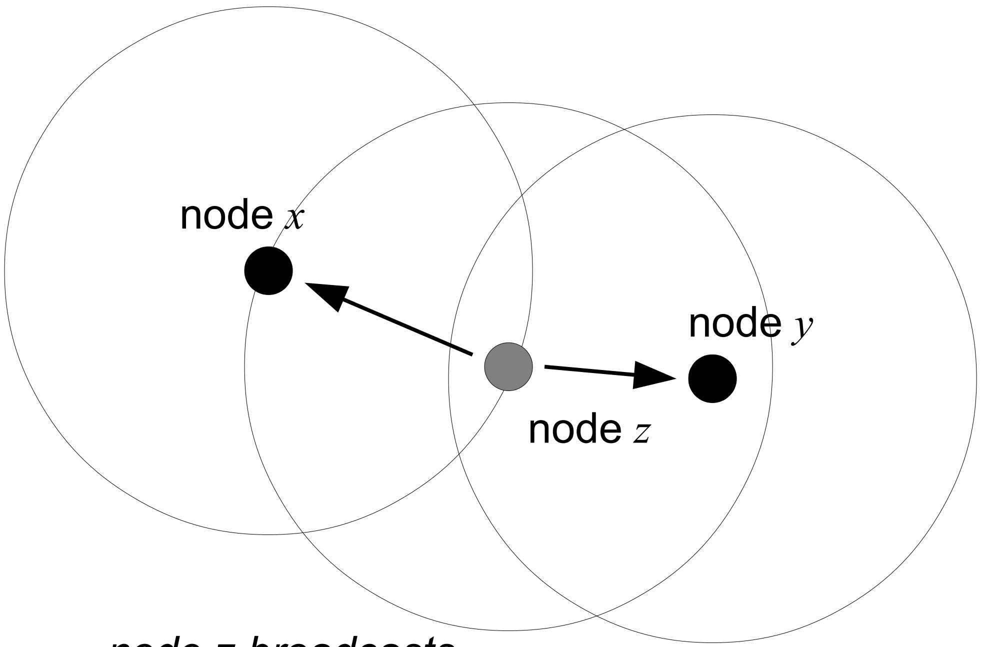


- when a node first listens for neighbors it should listen for at least a full wakeup period.

(1) The initial listen period should obviously be at least as long as a full cycle. Since SYNC frames are not sent at every cycle, the initial listen period is usually longer than multiple cycles.

MAC layer - S-MAC

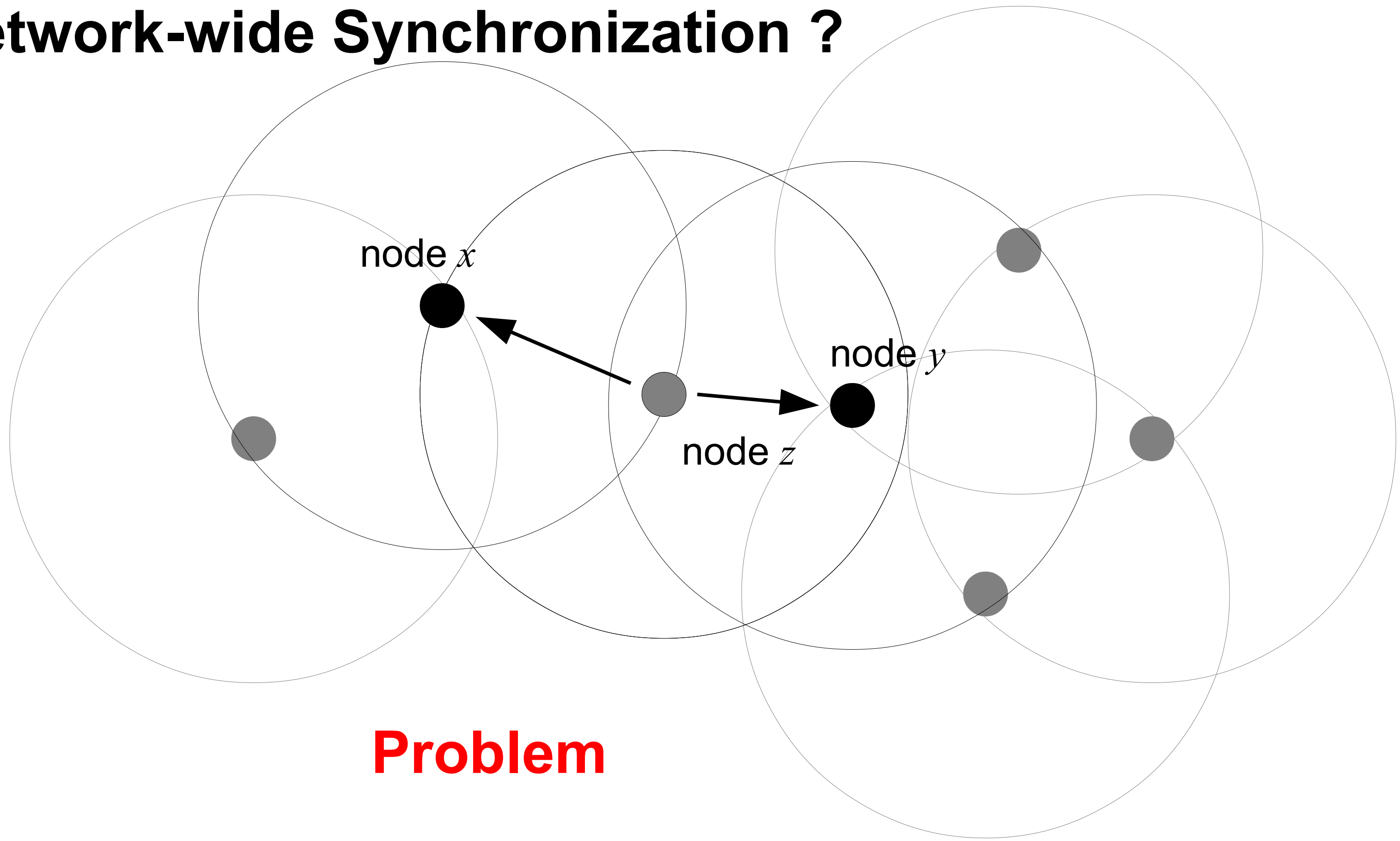
- **Example**



node z broadcasts
its schedule table
to its neighbors

MAC layer - S-MAC

- **Network-wide Synchronization ?**



Problem

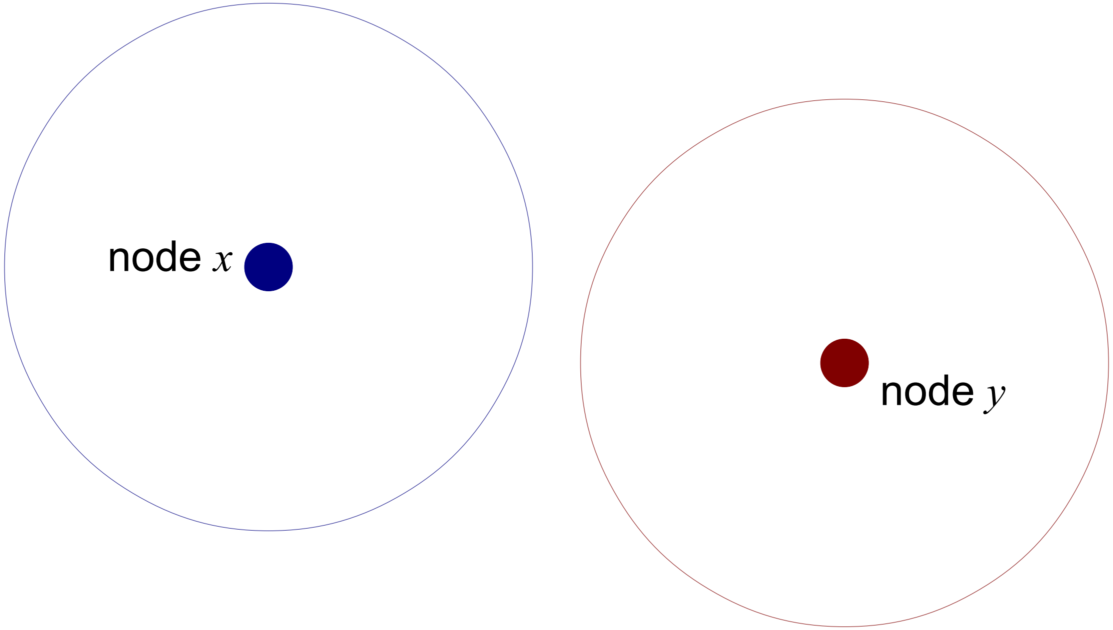
- Network-wide synchronization of schedules is not desirable: would increase contention, as all nodes wake up together and contend for the limited time slots

MAC layer - S-MAC

- **Virtual clusters - Local synchronization**
  - If a node receives a schedule different from its own schedule, 2 cases to consider
    - **Node currently has no neighbor** → discard its own schedule and adopt received one. Nodes that have the same schedule belong to the same cluster.
    - **Node already has neighbors** → keep its own schedule and adopt additional received schedule. This will occur for nodes at the border of two clusters.
  - Corner case: failure to discover schedule of neighbor
    - Occurs if node has adopted a schedule that does not overlap with that of neighbors.
    - Solution: S-MAC uses a **periodic neighbor discovery** where nodes listen for a full wakeup period. This must not be done too frequently → consumes energy.

MAC layer - S-MAC

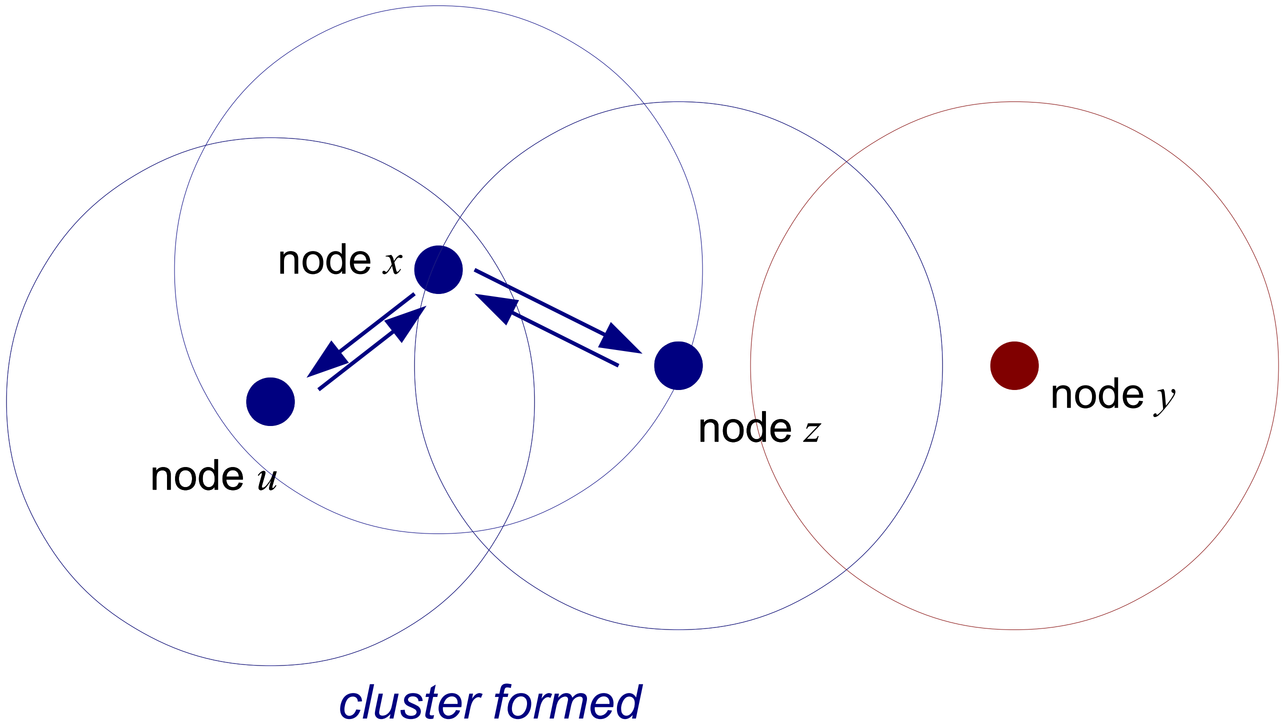
- **Virtual Clusters**



  - Nodes x and y can't hear each other and pick their own schedule.

MAC layer - S-MAC

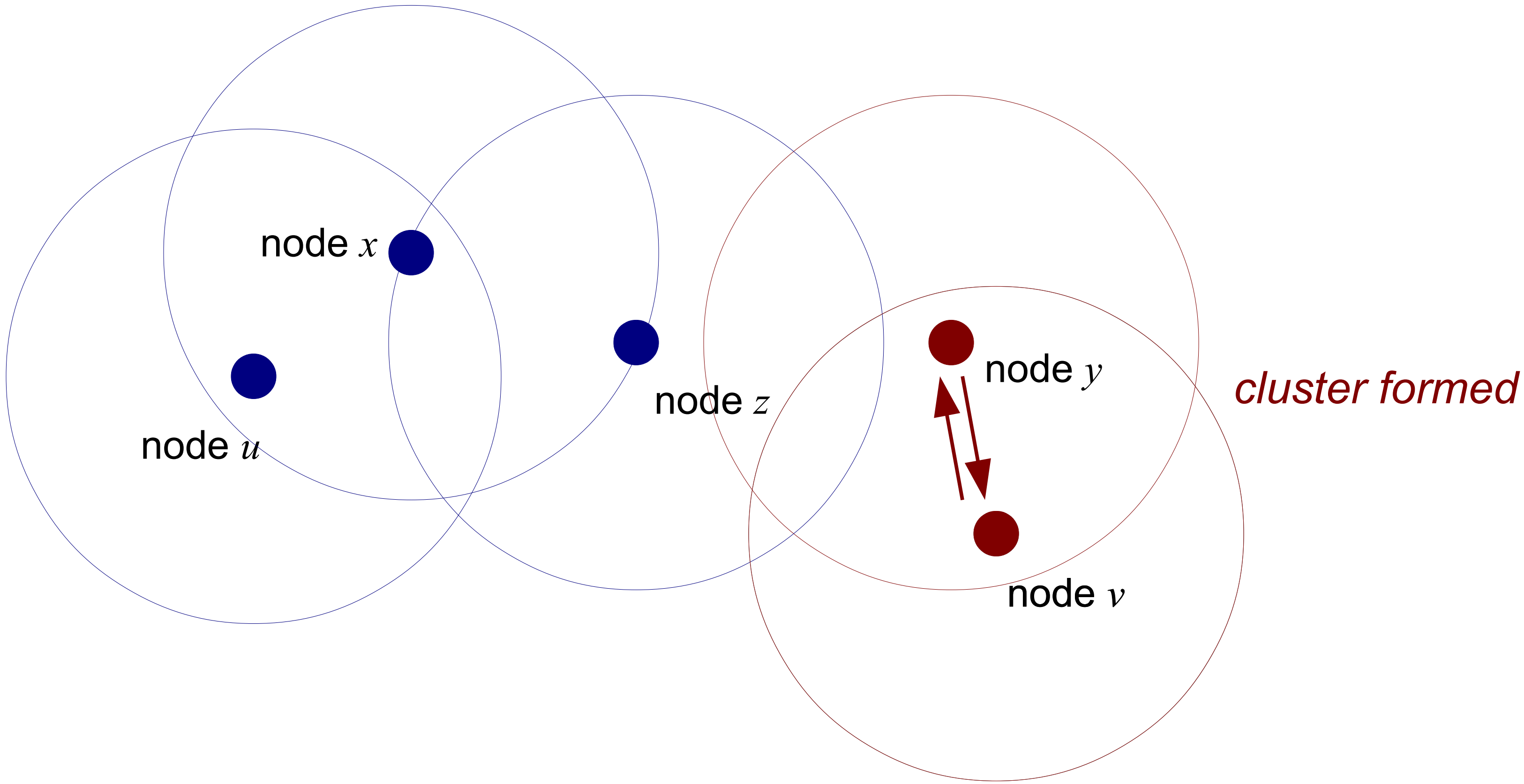
- **Virtual Clusters**



- Node u and z first hear and adopt x's schedule after they are switched ON. Later, they will advertise x's schedule. Node x learns that someone else uses its schedule.

MAC layer - S-MAC

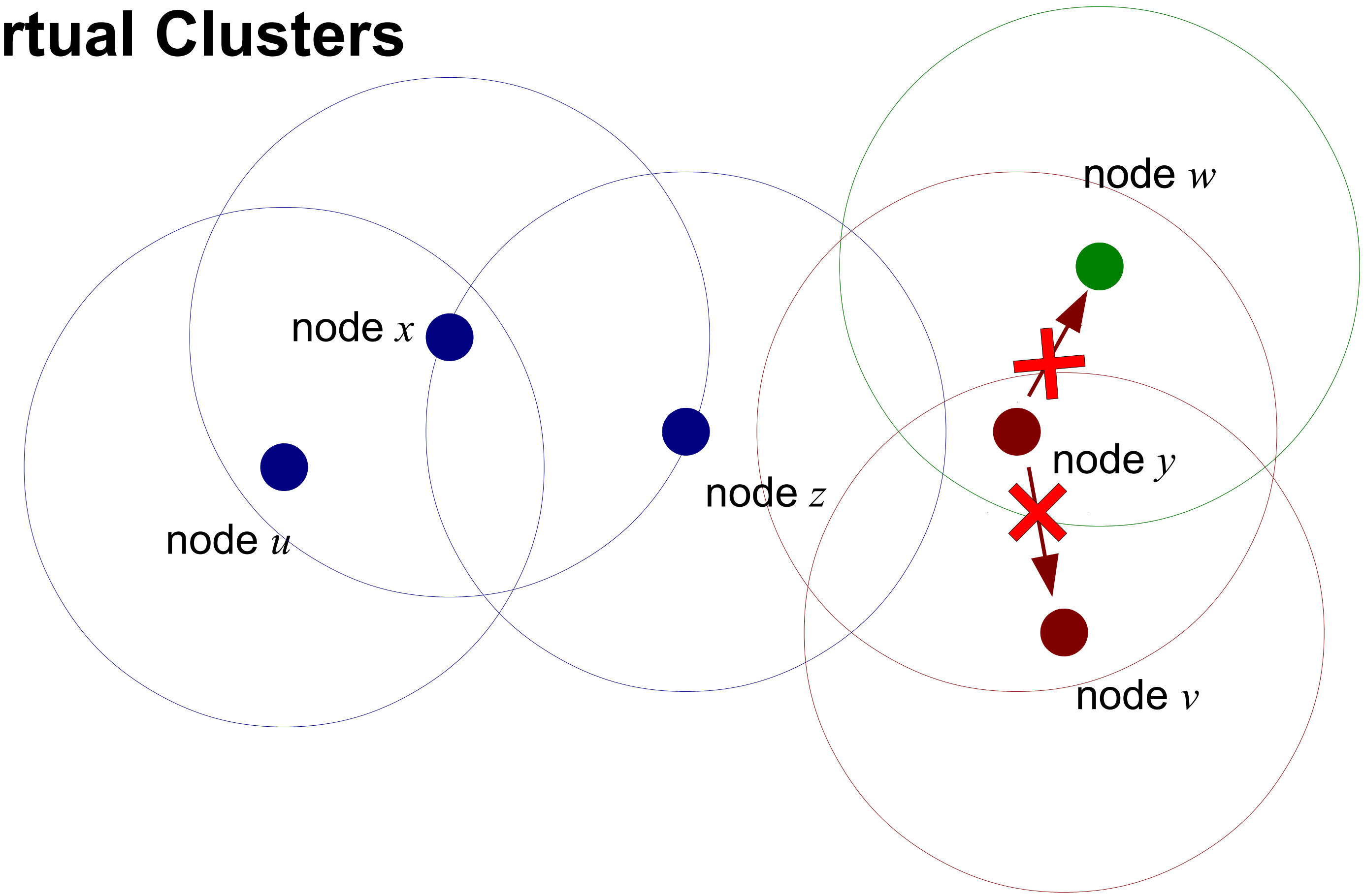
- **Virtual Clusters**



- Node v first hears and adopts y's schedule after it is switched ON. Later, it will advertise y's schedule. Node y learns that someone else uses its schedule.

MAC layer - S-MAC

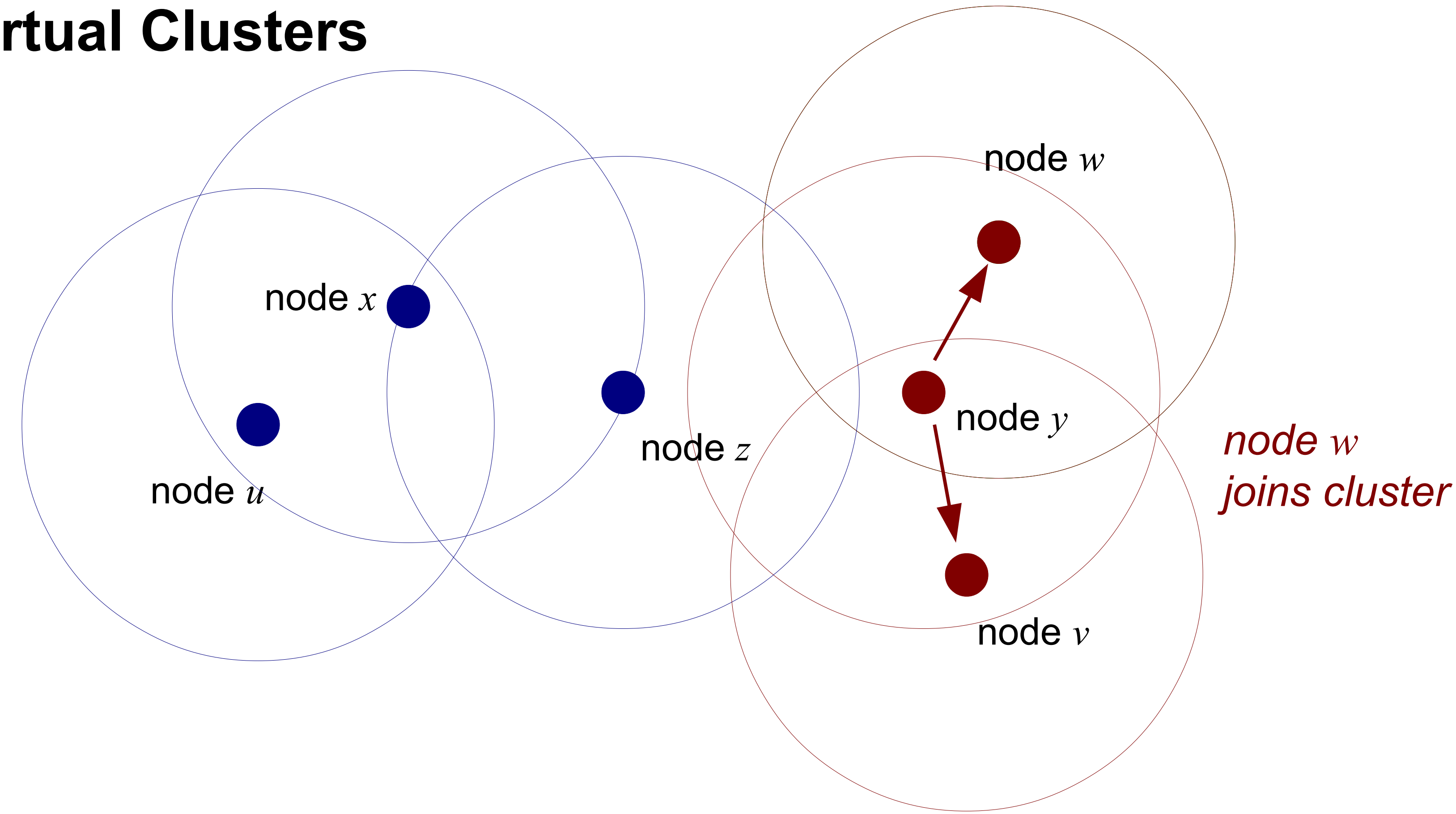
- **Virtual Clusters**



☆ Node w is switched ON. The schedule from y arrives with incorrect checksum and is discarded. Node w picks its own schedule.

MAC layer - S-MAC

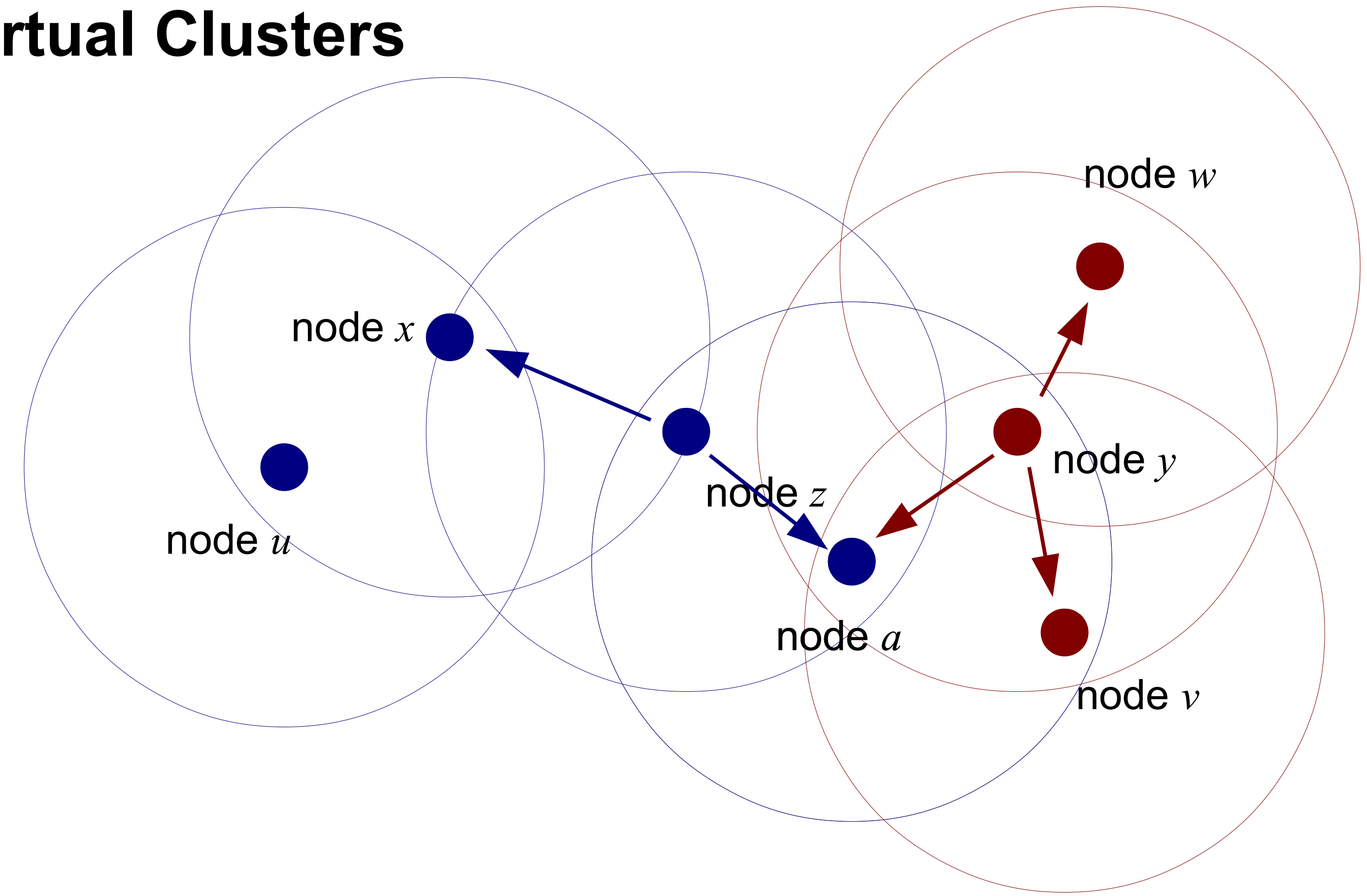
- **Virtual Clusters**



- Later, node w receives a different schedule from node y. As node w has not heard that another node shares its schedule, it switches to y's schedule.

MAC layer - S-MAC

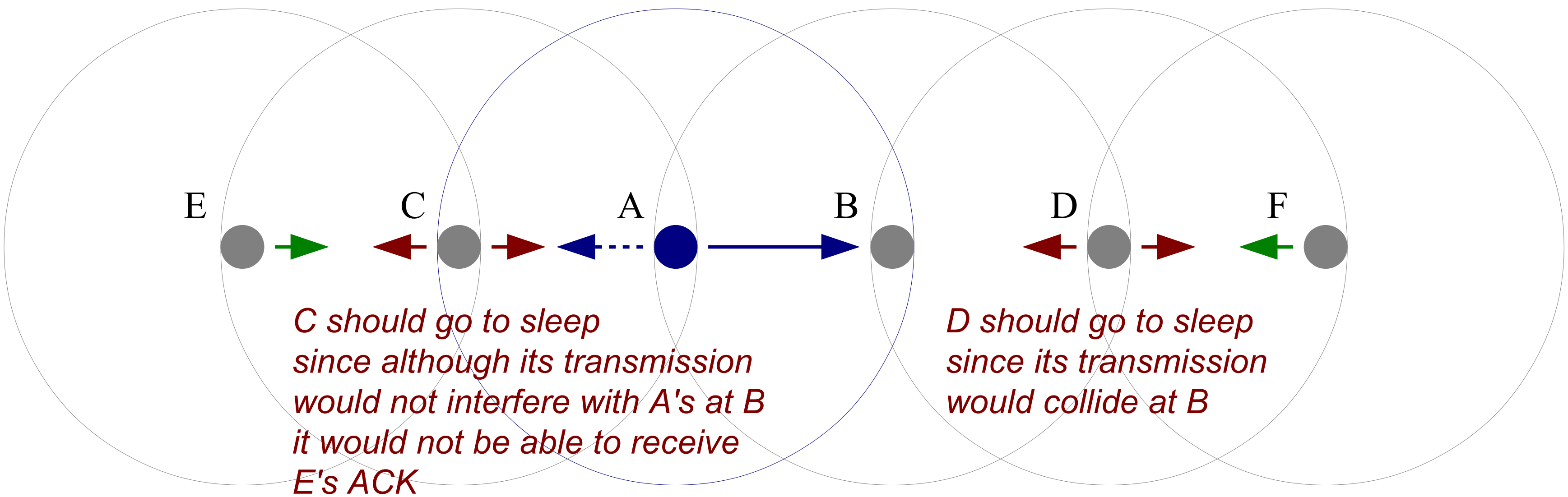
- **Virtual Clusters**



- Node a is switched ON. It first hears node z's schedule. Later it receives a different schedule from node y. It adopts both schedules. Node a is a border node.

MAC layer - S-MAC

- **Limiting Overhearing**
  - Principle: nodes can go to sleep as soon as they hear an RTS for another node or a CTS(1).
  - Question: which nodes need to go to sleep ? Example: A, B, C, D, E and F can only hear their immediate neighbors. A wants to send to B.

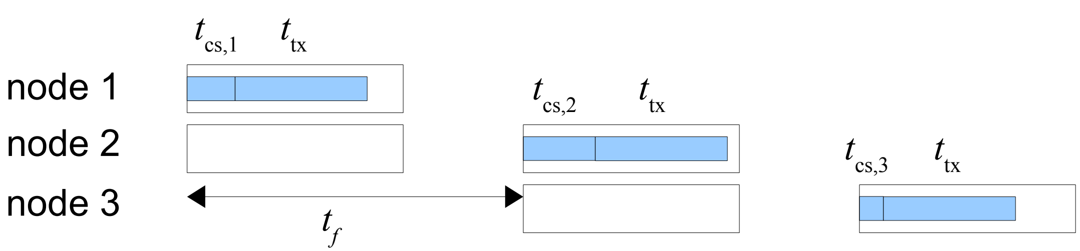


→ Neighbors of sender / receiver should go to sleep

(1) RTS and CTS packets indicate the length of data transmission so that other nodes know when they can wakeup.

MAC layer - S-MAC

- **Summary**
  - Benefits
    - Nodes can spend much time in sleep mode (limits idle listening).
    - Avoids overhearing as much as possible (RTS/CTS).
  - Drawbacks
    - Increased latency. Exacerbated with multi-hop transmissions. Worst incurred latency is on the order of $N * T_{sleep}$ where N is the number of hops and T_{sleep} the length of the sleep period.



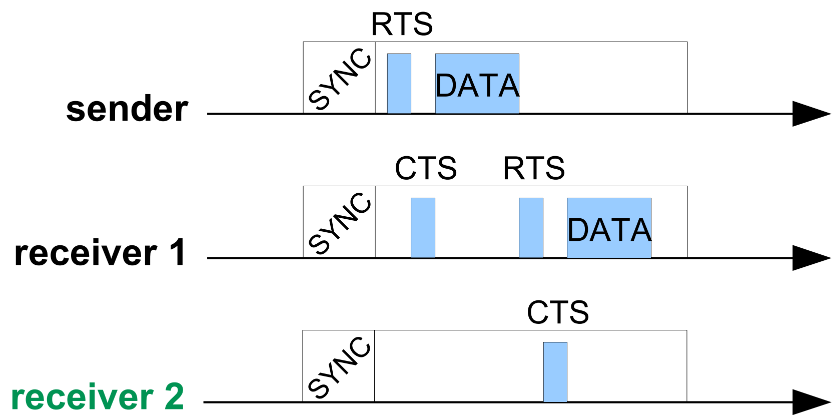
$$t_{total} = (N-1)t_f + t_{cs,N} + t_{tx}$$

    - Listen period is fixed and allows a single transmission (see adaptive listening).

Note : $t_{cs,n}$ time spent in CSMA ; t_{tx} transmission time ; t_f wake-up frame duration ; SYNC period ignored

MAC layer - S-MAC

- **Adaptive listening**
  - Principle
    - A node that overhears an RTS or CTS transmission can schedule an “extra” listen period later in its frame (it goes to sleep meanwhile).
    - This allows e.g. a next-hop node to stay awake and receive the forwarded frame in the same wake-up cycle.



MAC layer - T-MAC

- **Principle**
  - Observation
    - S-MAC uses a fixed listen (active) period. Not practical for networks where the traffic load varies.
    - Example : sensor networks subject to bursts of frames after an event is sensed.
  - Solution
    - Variable active period
    - T-MAC's active period ends when there is nothing to hear during a time TA.

      → minimal duty cycling = TA / frame length[(1)]

(1) Recall that in this context, the frame length denotes the duration of a wakeup cycle.

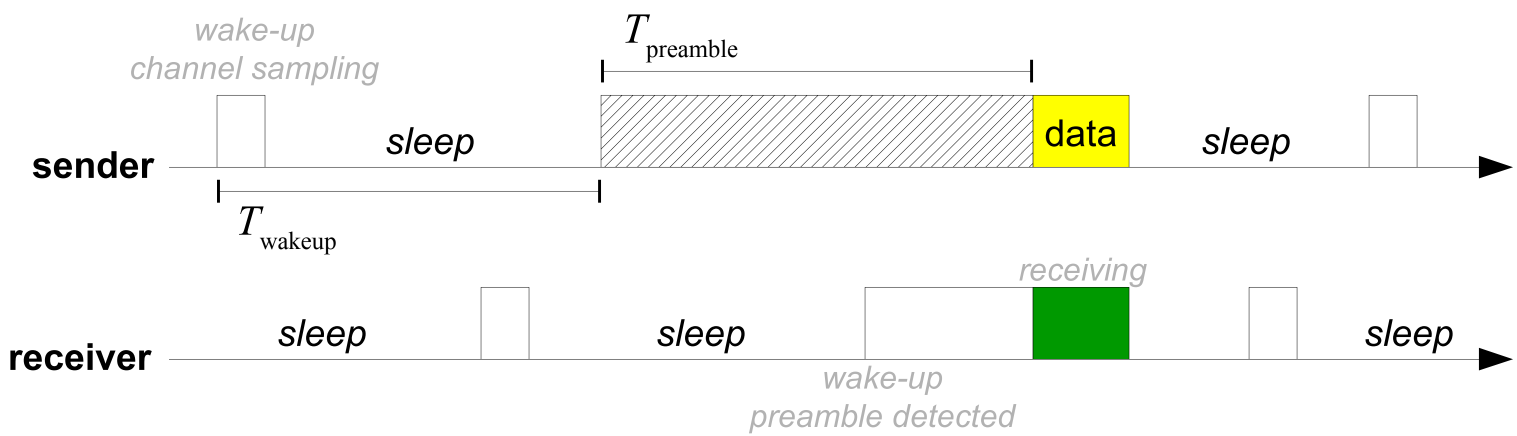
Reference: ***An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks***, T. van Dam and K. Langendoen, ACM Sensys 2003

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
  - centralized
  - synchronous
  - **asynchronous**
  - IEEE 802.15.4
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

MAC layer – Berkeley MAC (B-MAC)

- **Principles**
  - Asynchronous : no wake-up schedule shared
  - Receivers sample the channel at regular interval
  - Long preamble to signal data transmission
  - Preamble length T_{preamble} > wake-up cycle length T_{wakeup}



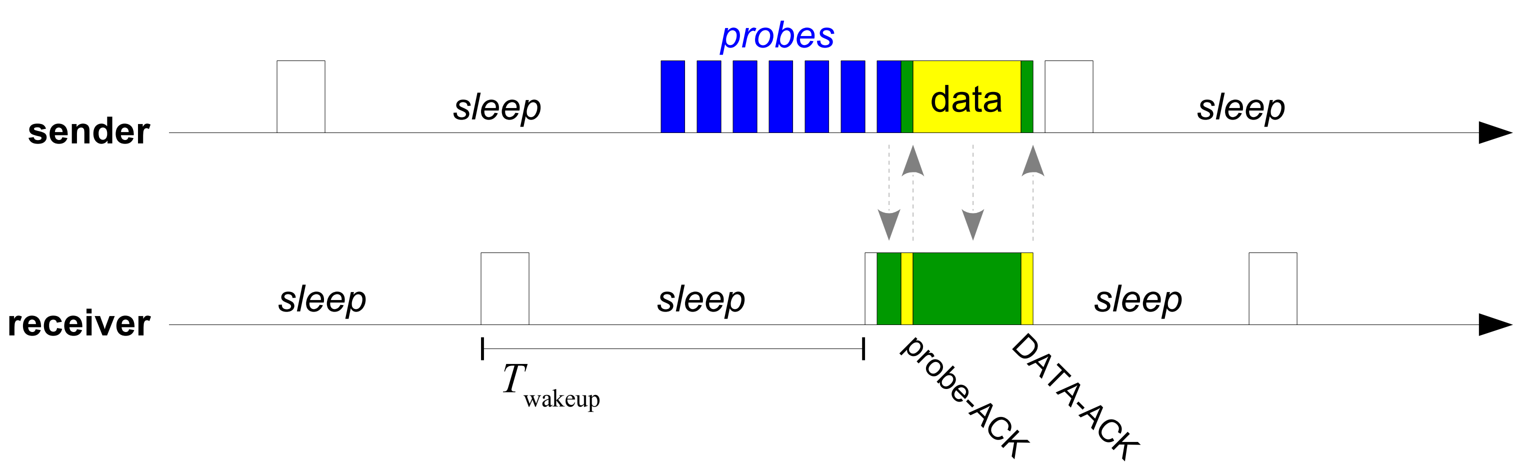
Reference: ***Versatile low power media access for wireless sensor networks***, J. Polastre, J. Hill and D. Culler, ACM SenSys, 2004

MAC layer – Berkeley MAC (B-MAC)

- **Discussion**
  - Benefits
    - Receivers go to sleep immediately when channel sampling detects no preamble → limit idle listening
    - Simpler than synchronous protocols such as S-MAC
  - Drawbacks
    - Decreasing the duty-cycling (increasing T_{wakeup}) implies increasing the preamble time (T_{preamble})
    - Complete preamble transmitted even if receiver already awaken (no way to know)
    - Not possible with every radio transceiver (need control on PHY layer)
    - No limitation to overhearing → impossible to know the frame destination before the frame is completely received.

MAC layer – X-MAC

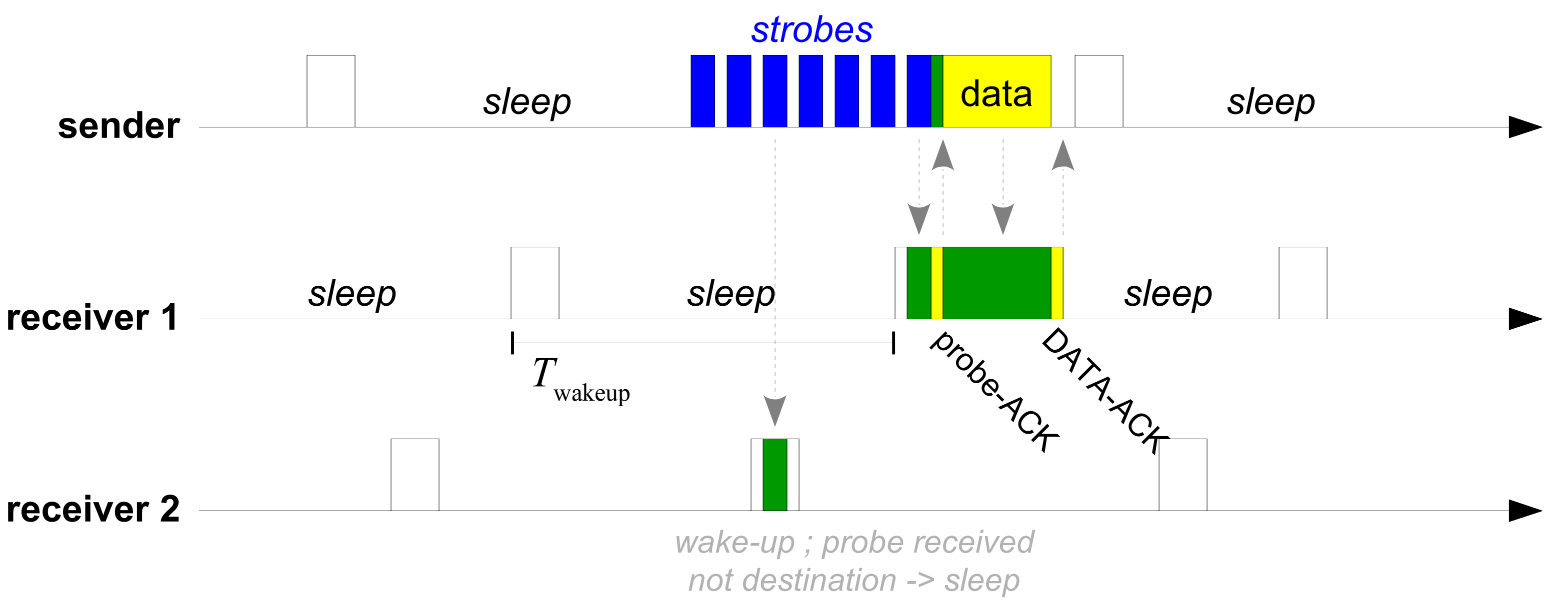
- **Principles**
  - Objective : limit overhearing
  - Preamble replaced with short “strobe” frames that contain the destination address : non-interested receivers can go to sleep earlier
  - Preamble stopped when receiver awakened (ACK frame)



Reference: ***X-MAC: A Short Preamble MAC Protocol For Duty-Cycled Wireless Networks***, M. Buettner, G. Yee, E. Anderson and R. Han, ACM SenSys, 2006

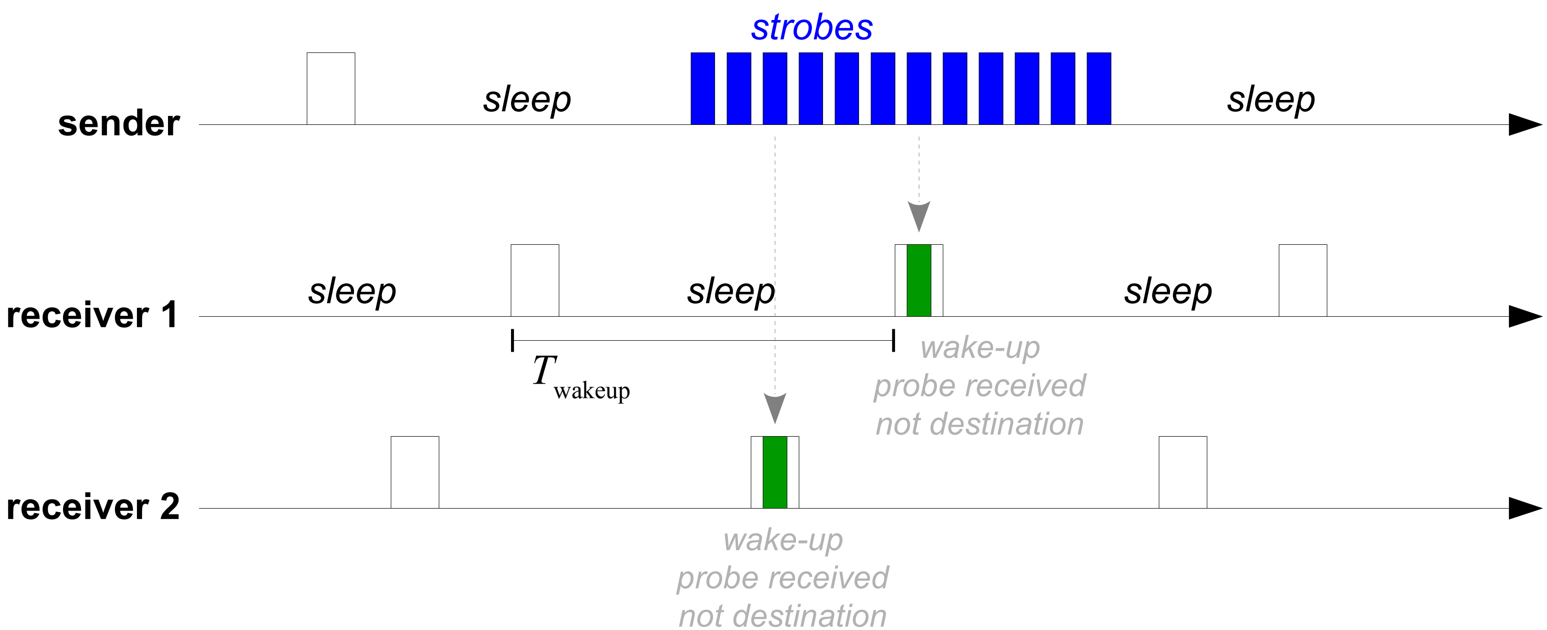
MAC layer – X-MAC

- **Principles**
  - Limited overhearing : a non interested receiver can go to sleep immediately after a probe has been received.
  - no need to listen to whole DATA frame.



MAC layer – X-MAC

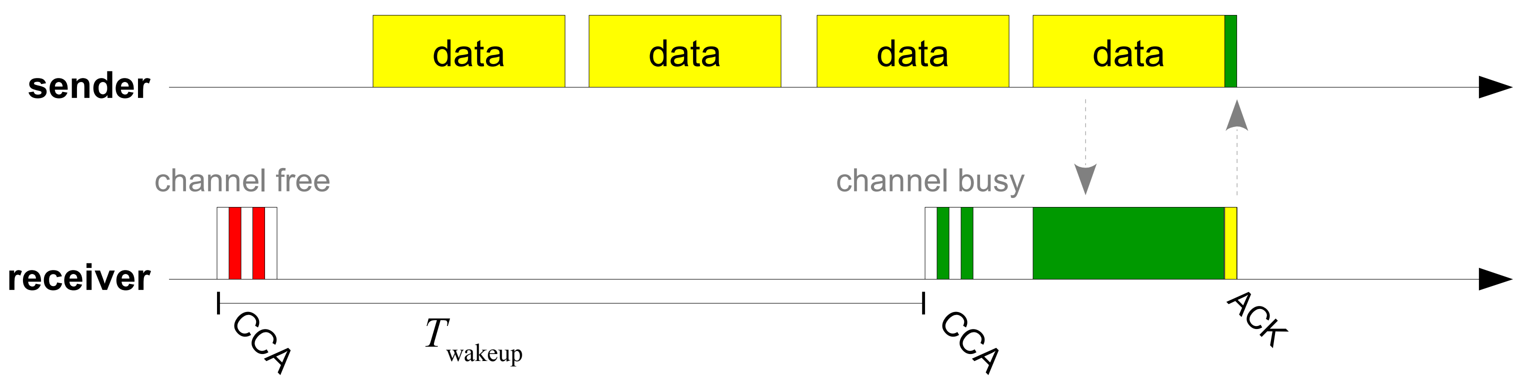
- **Principles**
  - Special case : receiver missing
    - Number of strobes limited to a full wake-up cycle ~ T_{wakeup}



  - Special case : broadcast
    - No probe-ACK is sent ; DATA frame sent after strobes

MAC layer - ContikiMAC

- **Principles**
  - Similar to X-MAC... but full DATA frame sent as strobe.
  - Overhearing can be limited thanks to the DATA frame destination field (whole frame must be received[(1)]).
  - Receiver detects incoming frame by checking channel activity (*Clear Channel Assessment* - CCA)

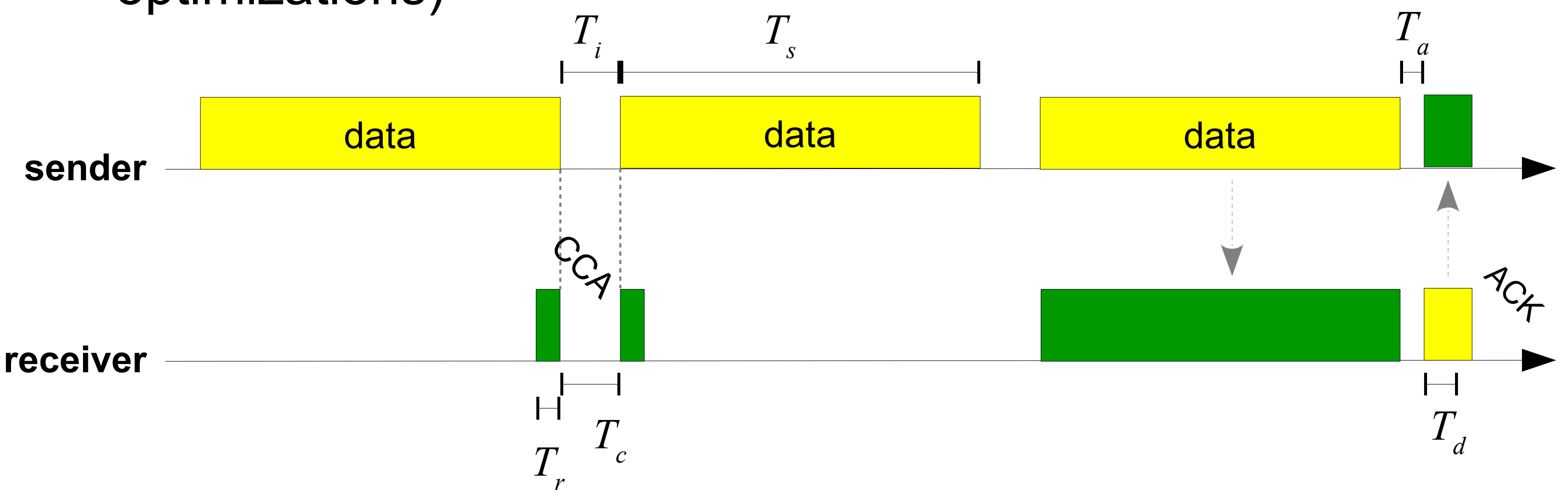


(1) the destination field cannot be used before the CRC (in frame trailer) has been checked.

Reference: ***The ContikiMAC Radio Duty Cycling Protocol***, A. Dunkels, SICS Technical Report T2011:13, Swedish Institute of Computer Science, December 2011

MAC layer - ContikiMAC

- **Timing constraints**
  - Required for the correct operation of the protocol (and optimizations)



$T_s > T_c + 2T_r$ Frame cannot fall between CCAs

$T_i < T_c$ 2 CCAs enough to detect frame

$T_i > T_a + T_d$ Allow space for ACKs

$$T_a + T_d < T_i < T_c < T_c + 2T_r < T_s$$

<u>Case of 802.15.4 PHY</u>

$T_r = 192\ us$

$T_d = 160\ us$

$T_a = 192\ us$

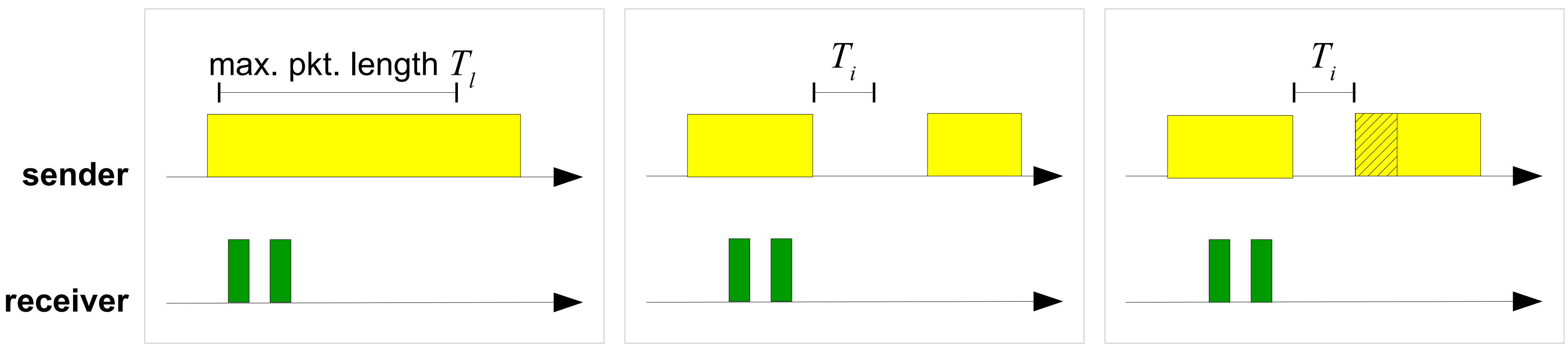
<u>Contiki implementation</u>

$T_i = 0.4\ ms$

$T_c = 0.5\ ms$

MAC layer - ContikiMAC

- **Fast-sleep optimization**
  - Overhearing is limited by looking at the received frame's destination field.
  - The CCA only detects if there is energy transmitted on the channel[1].
  - Detected energy might be noise. This would force a node to remain awaken, but no frame would be received. How to detect this case ?



no interval after packet (signal > T_l)

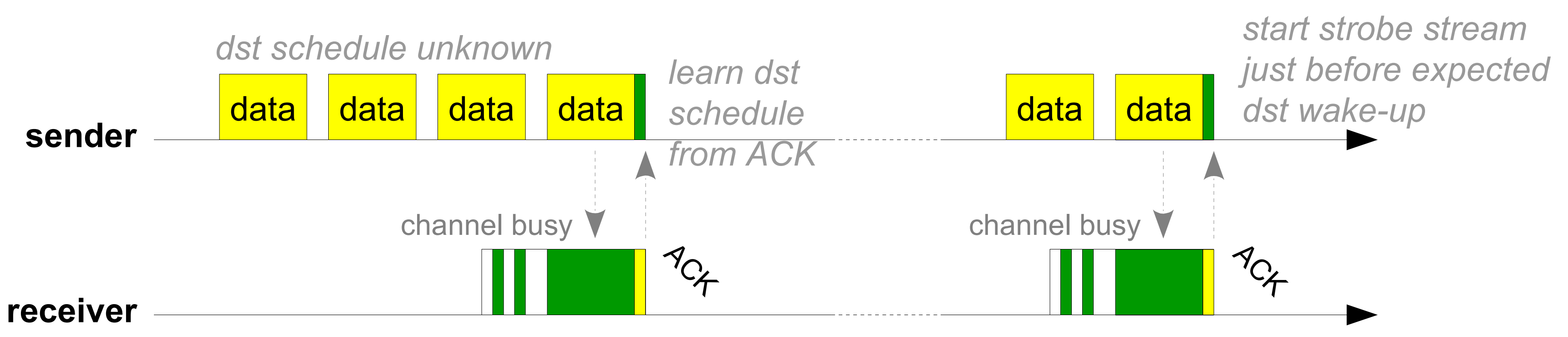
interval too long

no “start” of frame

(1) more complex CCAs can be done where symbols should be correctly demodulated.

MAC layer - ContikiMAC

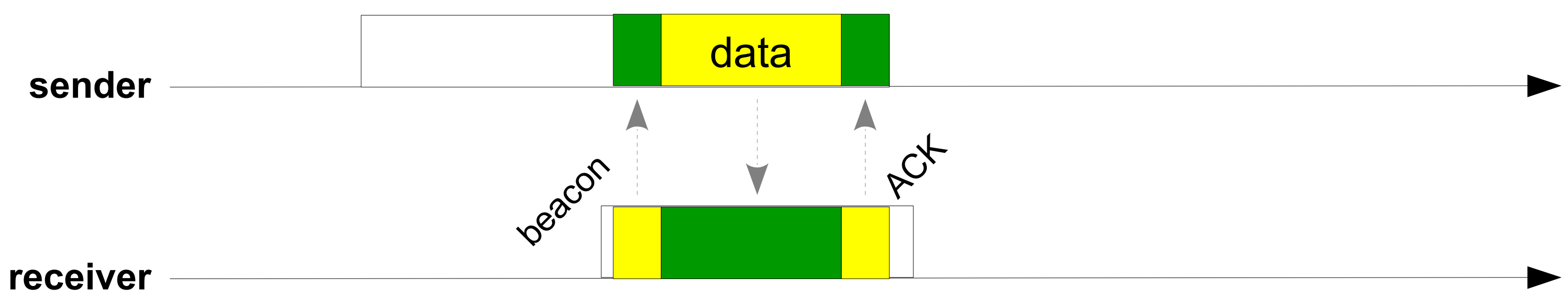
- **Phase lock**
  - Stream of strobe can last up to a full cycle (T_{wakeup}) in the worst case.
  - Phase lock objective: reduce number of strobes sent by learning the destination's wake-up schedule
  - How ? Deduce neighbor's wake-up time from ACK arrival time. Remember neighbor schedule for future frames.



  - Issues: clock drifts (nodes have slightly different clock frequencies) + small error in wake-up time estimation

MAC layer – Receiver-initiated protocols

- **That's not the end of the story...**
  - Protocols studied so far are *sender-initiated* (the sender sends the preamble/strobes)
  - Asynchronous protocols also contain a *receiver-initiated* subfamily. In this family, receivers initiate the transmission by sending a beacon frame when they are ready to receive.



  - Several proposals : LPP, RI-MAC, A-MAC, ...
  - Many nodes = high risk of collisions (beacon frames) even with moderate traffic intensity

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
  - centralized
  - synchronous
  - asynchronous
  - **IEEE 802.15.4**
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

MAC layer - IEEE 802.15.4

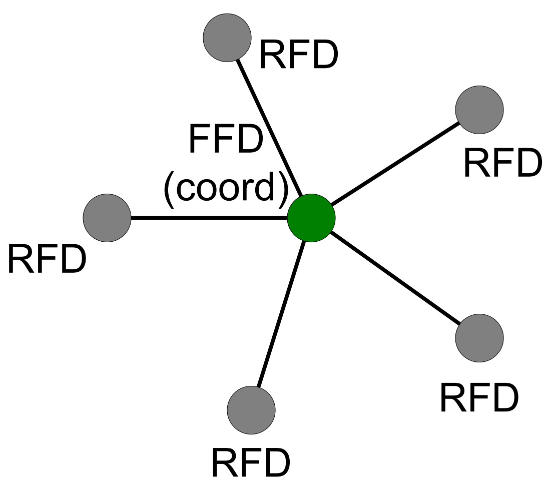
- **Introduction**
  - PHY and MAC layers
  - Low-Rate Wireless Personal Area Network (WPAN)
  - Features
    - low-to-medium bitrates
    - Industrial/Scientific/Medical (ISM) band
    - allows for contention-based and scheduled-based schemes

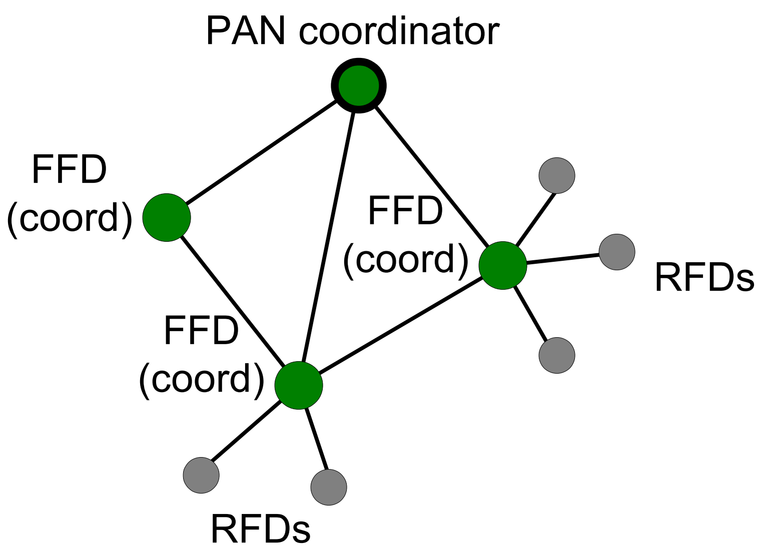
Frequency band (MHz)	Bitrate (kbps)	Number of channels	Modulation
868-868.6	20	1	BPSK
902-928	40	10	BPSK
2400-2483.5	250	16	O-QPSK

  - Often confused with "*ZigBee*" (works on top of 802.15.4)

MAC layer - IEEE 802.15.4

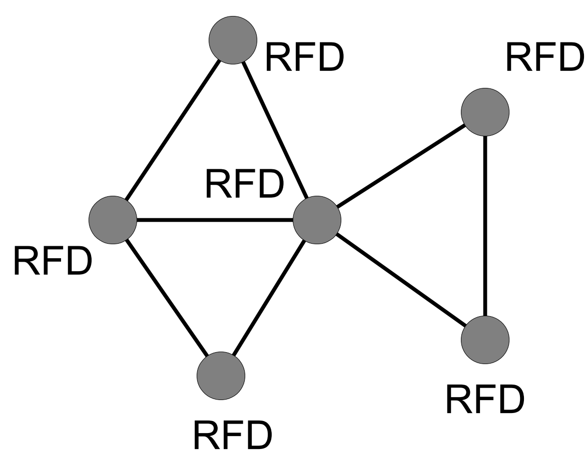
- **Network Architecture**
  - Two different node types
    - Full Function Device (FFD) : can work as PAN coordinator, simple coordinator or device
    - Reduced Function Device (RFD) : only as device
  - Network topology
    - Star network to connect devices to coordinator
    - Peer-to-peer network among coordinators





MAC layer - IEEE 802.15.4

- **Network Architecture**
  - Network topology
    - Star network to connect devices to coordinator
    - Peer-to-peer network among coordinators
    - Ad-hoc topology



MAC layer - IEEE 802.15.4

- **Addresses**
  - Each node has a unique 64-bit address
    - First 24 bits = Organizational Unique Identifier (OUI) allocated to manufacturer by IEEE
    - 40 remaining bits assigned by manufacturer
    - used mainly before joining a PAN
  - Nodes can also use short, 16-bit, addresses to reduce overhead (limited frame size)
    - Short addresses have a limited scope (can only be used within a single PAN)
    - Device outside PAN can be reached with their short address + destination PAN ID (32-bit total)

MAC layer - IEEE 802.15.4

- **Frame format**

preamble, len, ...

	Frame control	Sequence number	Dest. PAN ID	Dest. Address	Source PAN ID	Source Address	Auxiliary Security header	Payload	FCS
bytes	2	1	0/2	0/2/8	0/2	0/2/8	0/../21	variable	2

	Frame type	Security enabled	Frame pending	Ack. request	Intra PAN	Reserved	Dest. address. mode	Reserved	Source address. mode
bits	3	1	1	1	1	3	2	2	2

- Beacon (0)
- Data (1)
- Acknowledgment (2)
- MAC command (3)
- Reserved (4-7)

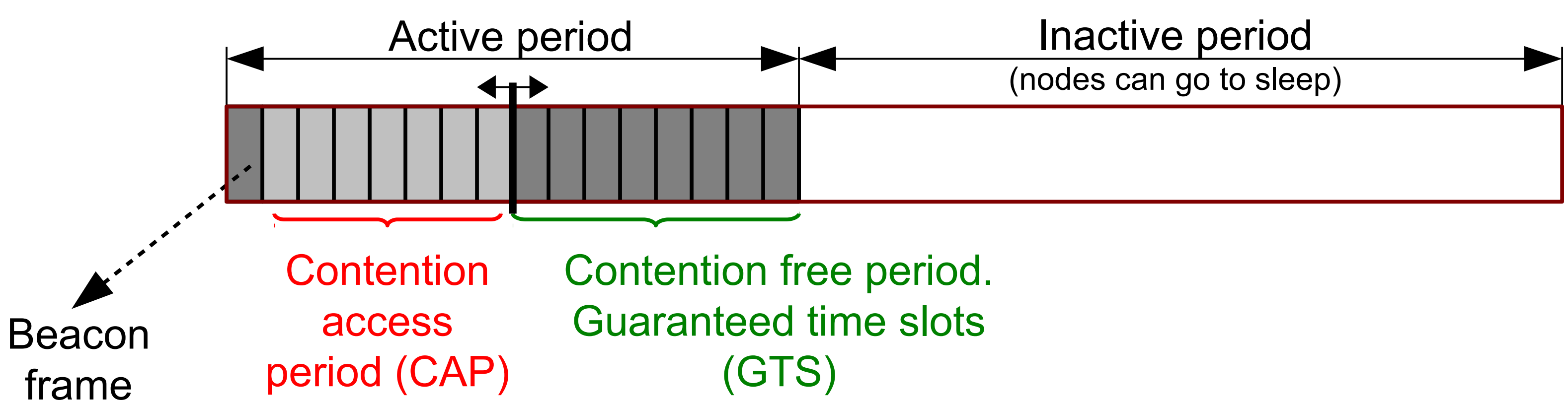
- No PAN ID / addr. Data (0)
  → only for Ack
- Reserved (1)
- Short address (2)
- Long address (3)

MAC layer - IEEE 802.15.4

- **Coordinator Role**
  - Manages list of associated devices
    - Association request / response MAC command frames
    - uses 64-bits addresses only
  - Allocates short addresses to devices
    - through association request/response
  - Manages the transmission of beacon frames
  - Allocates guaranteed time slot (GTS) in beaconed mode

MAC layer - IEEE 802.15.4

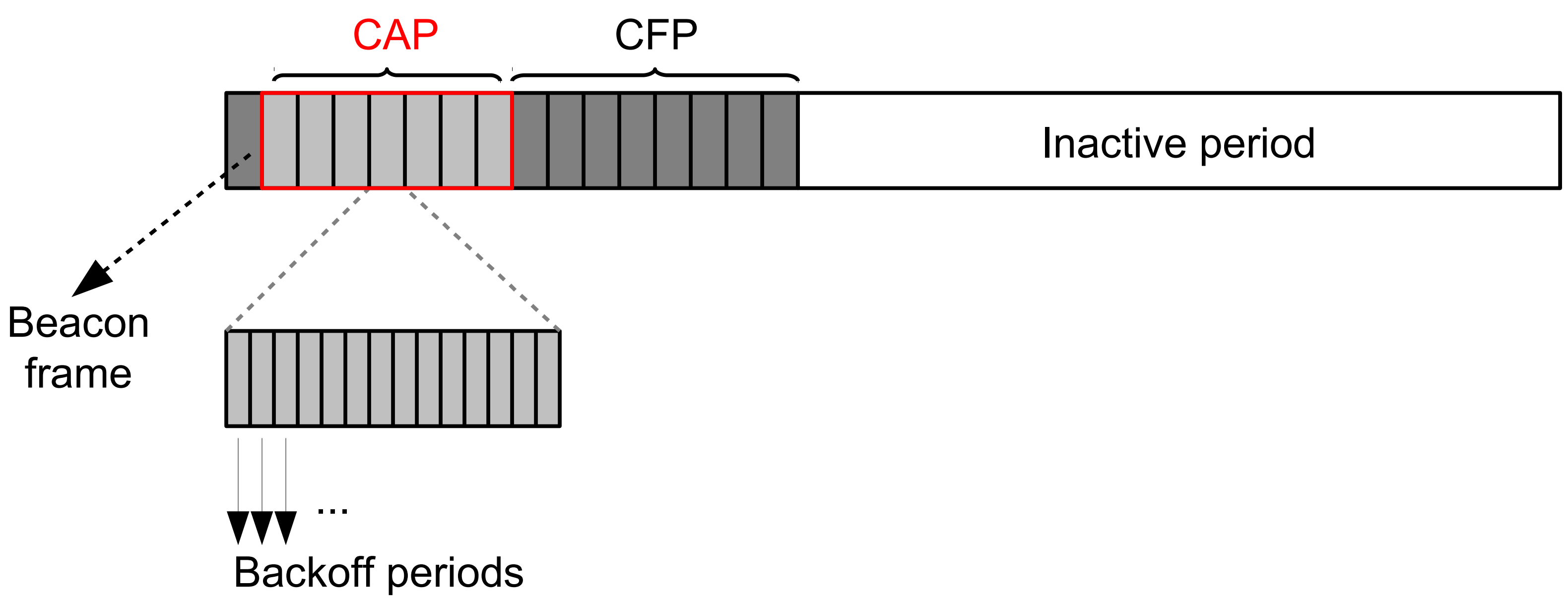
- **Beacon-mode, Superframe**
  - In *beacon-mode* channel access is organized by a coordinator
  - Beacon frame sent on a regular basis marks start of *superframe*



  - Active period : 16 slots. First slot = beacon.
    Other slots for CAP and GTS (separation configurable).
  - Coordinator active during whole active period.

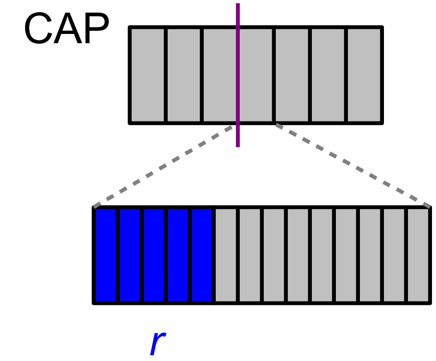
MAC layer - IEEE 802.15.4

- **Media access during CAP** (contention access period)
  - Slotted CSMA/CA protocol (synchronization with beacon)
  - Each CAP time slot is divided into backoff periods

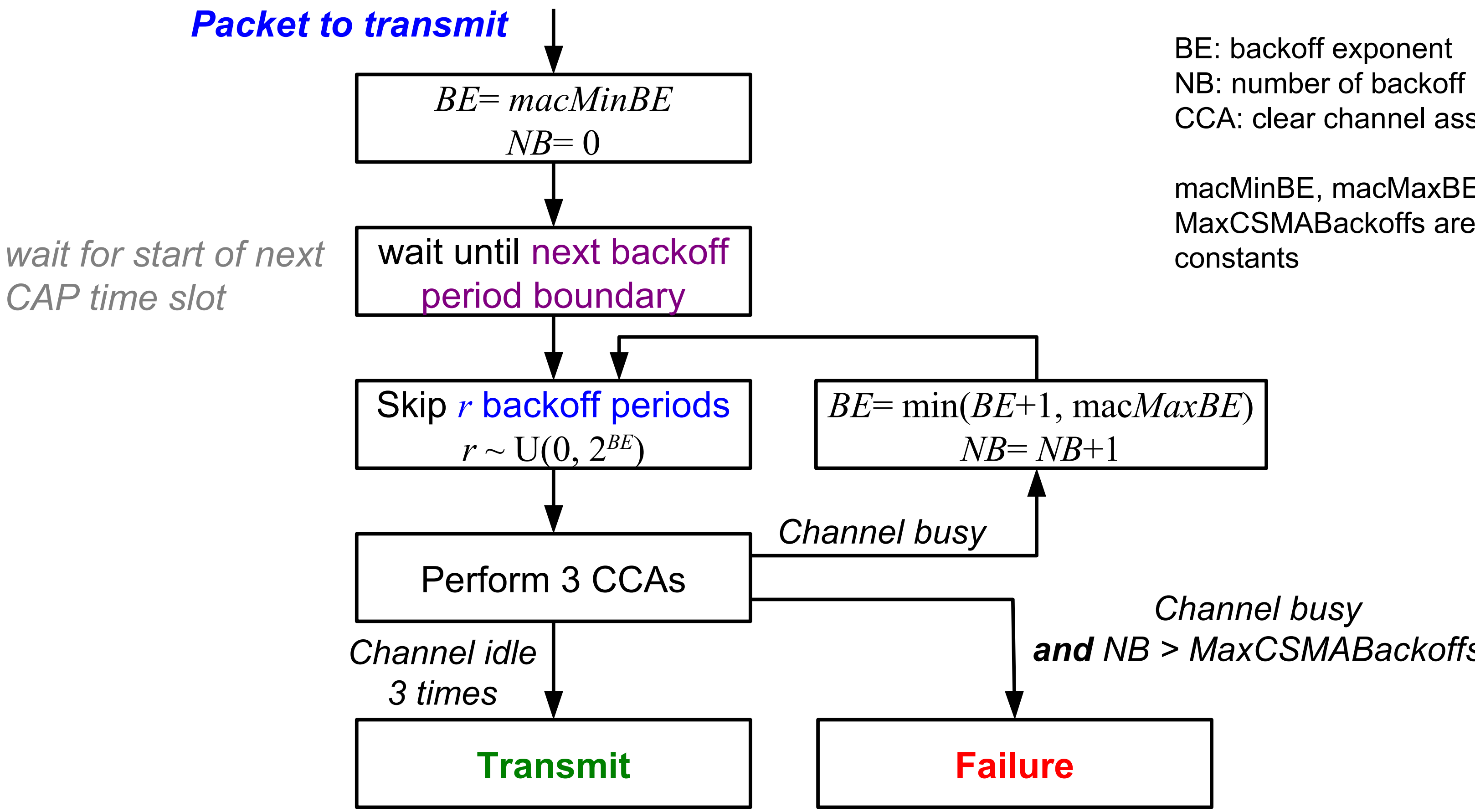


1 backoff period=20 symbols
(i.e. 320μs @ 2.4GHz)

MAC layer - IEEE 802.15.4



- **Media access during CAP**
  - Slotted CSMA/CA algorithm

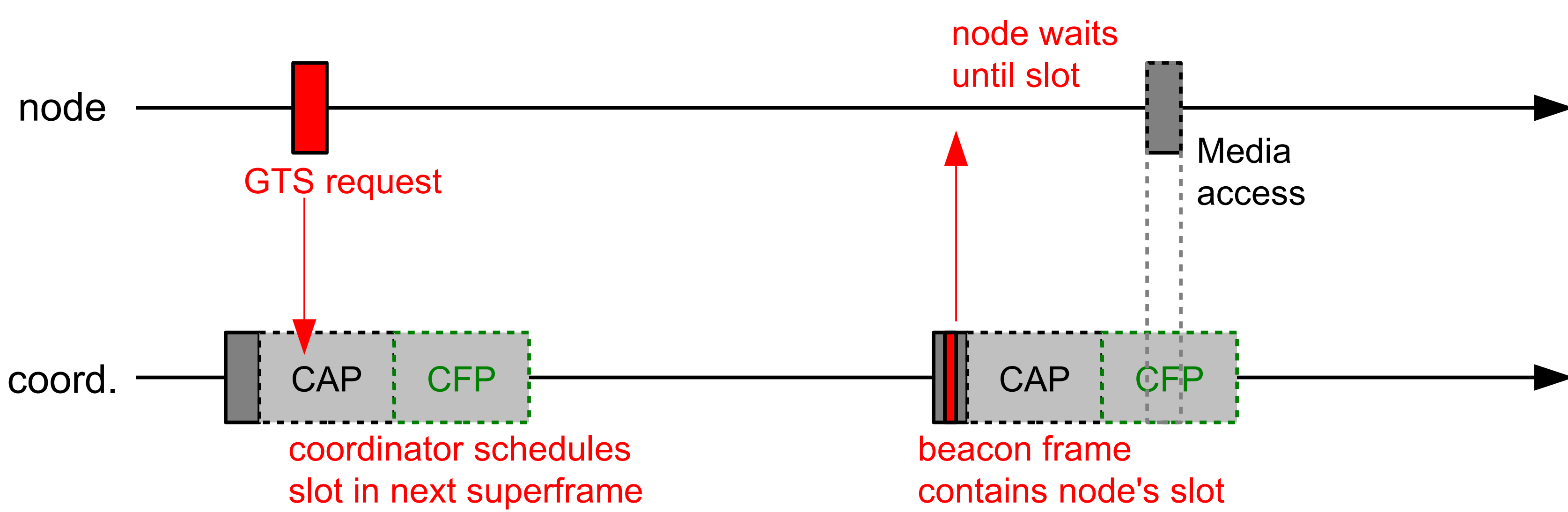


BE: backoff exponent
NB: number of backoff
CCA: clear channel assessment

macMinBE, macMaxBE and MaxCSMABackoffs are protocol constants

MAC layer - IEEE 802.15.4

- **Media access during CFP** (contention free period)
  - This is TDMA ! Beacon used for synchronization.
  - Before accessing the media during the CFP period, a node must request a time slot from the coordinator.
    - GTS request contains number of slots requested, direction (transmit or receive), and indicates whether it is an allocation or deallocation request
    - Beacon frame contains the list of short addresses that are allowed to use their GTS during the CFP period of the *superframe*.

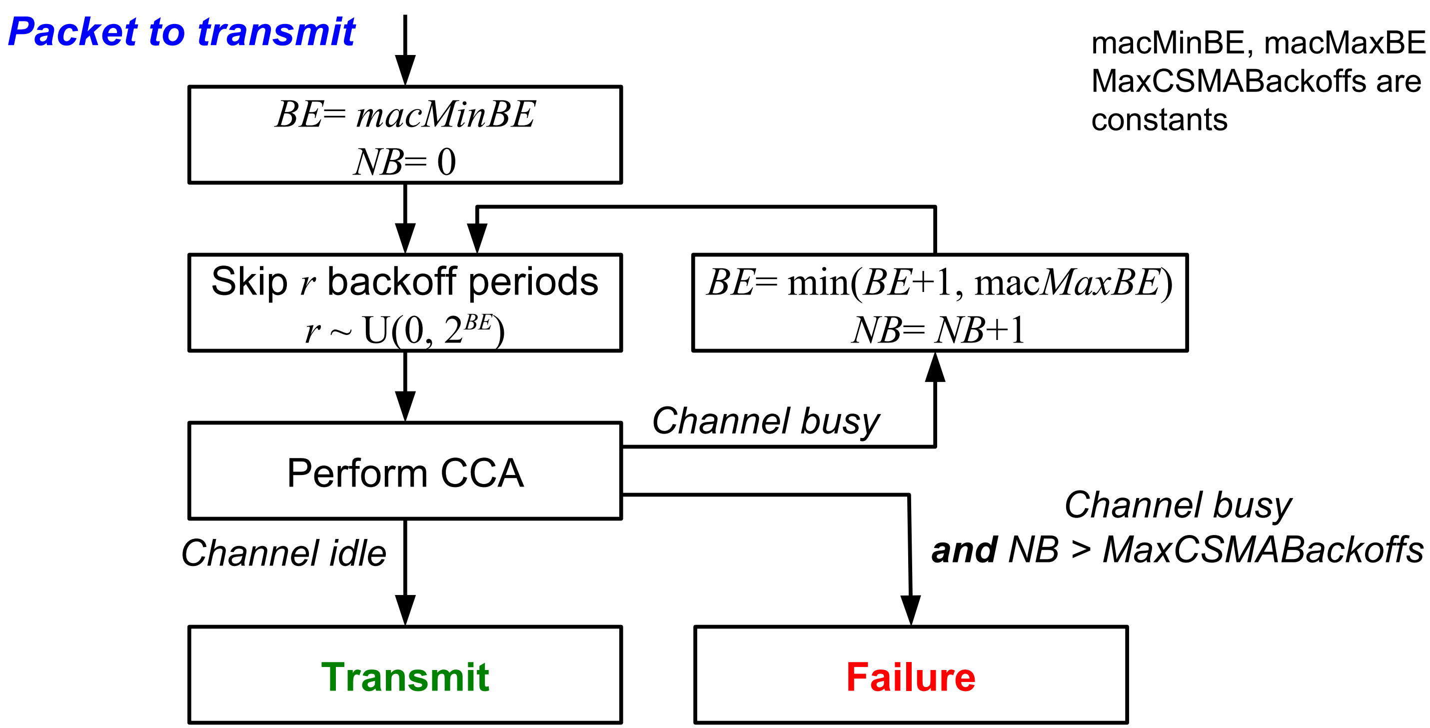


MAC layer - IEEE 802.15.4

- **Media - non-beaconed mode**
  - Used when no beacon used – no synchronization
  - Unslotted CSMA/CA algorithm

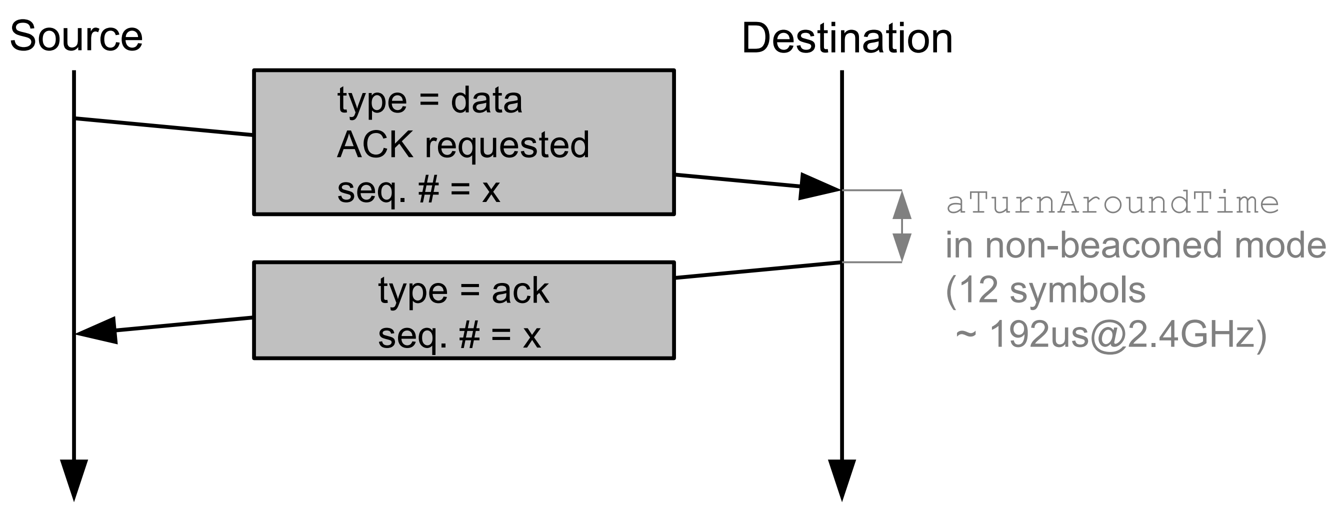
BE: backoff exponent
NB: number of backoff
CCA: clear channel assessment

macMinBE, macMaxBE and MaxCSMABackoffs are protocol constants



MAC layer - IEEE 802.15.4

- **Automatic ACK and retransmissions**
  - A sender can request the receiver to send an ACK frame upon reception
    - tight timing requirements → need help of hardware
    - ACK frame has same seq. # than received frame
    - ACK frames and broadcast must not request ACK

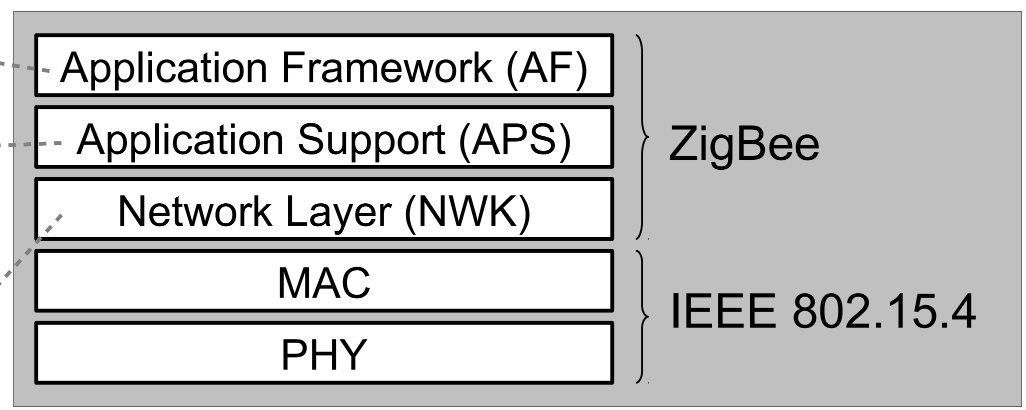


MAC layer - IEEE 802.15.4

- **Relation with ZigBee**
  - Proprietary specification for wireless communication based on top of IEEE 802.15.4 (membership of ZigBee Alliance required).
  - ZigBee Stack

Mux/demux
- end-points

Bindings
(unidirect. connection)
- end-to-end ack
- remove dup. pkts

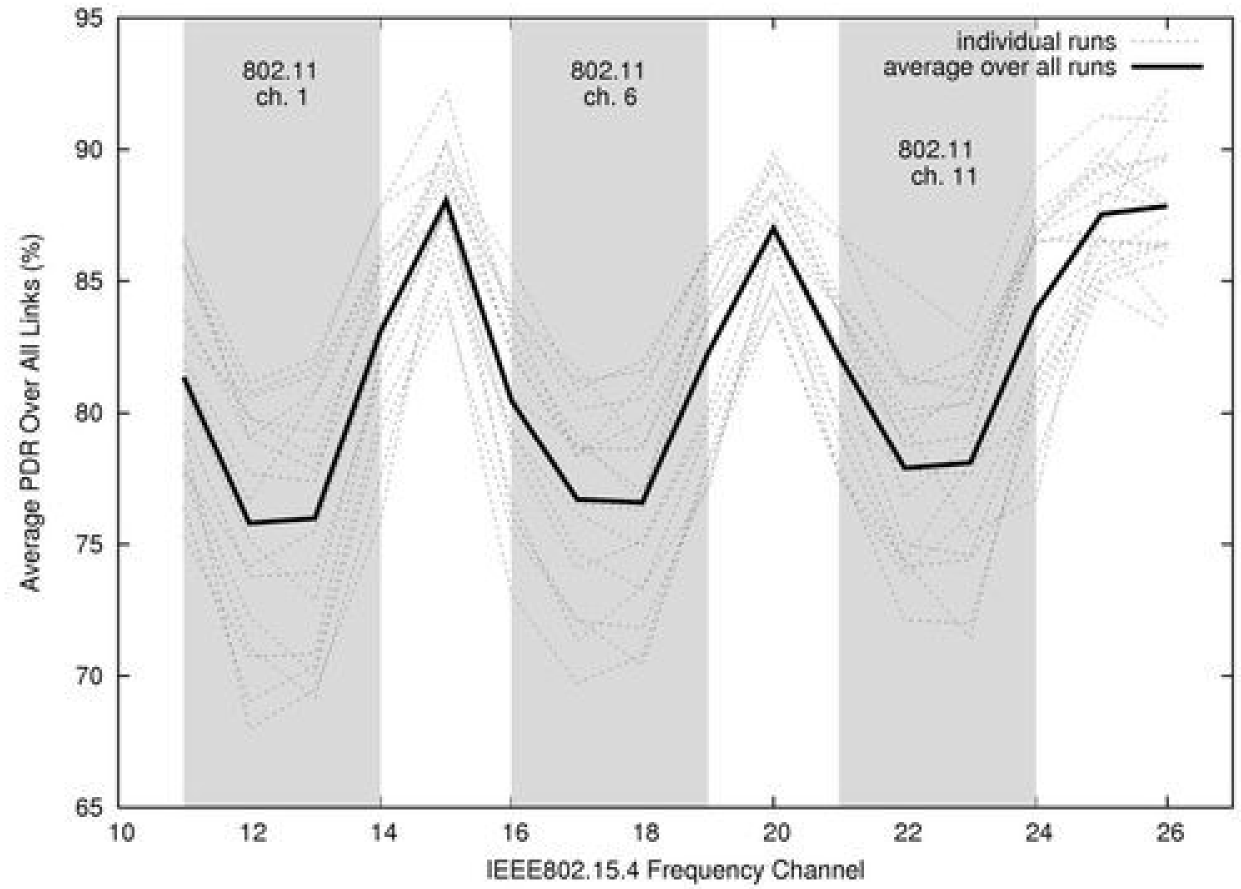


Addressing and routing
- broadcast (flooding) and unicast
- mesh routing similar to AODV

MAC layer - IEEE 802.15.4

- **Evolution**
  - 802.15.4a (2007) – new PHYs (e.g. UWB)
  - 802.15.4e (2012) – amendment for industrial applications: channel hopping MAC (TSCH – *Time Slotted Channel Hopping*)

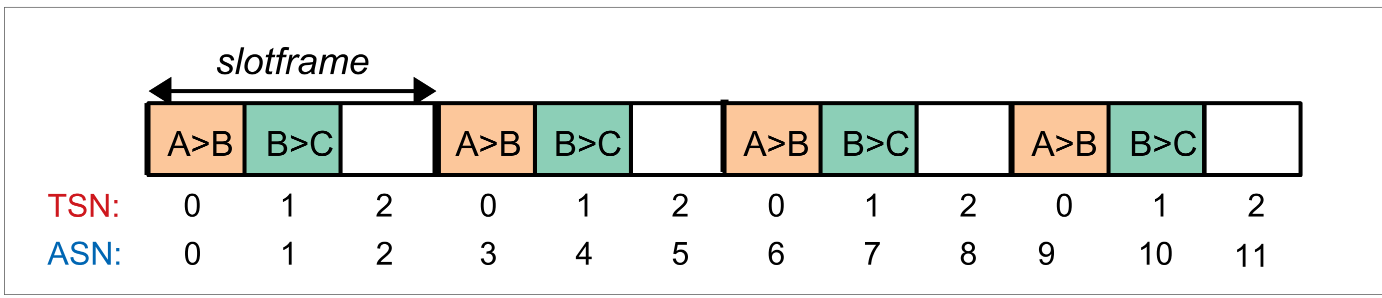
MAC layer - IEEE 802.15.4



Source: Ortiz and Culler, http://wsn.eecs.berkeley.edu/connectivity/

MAC layer - IEEE 802.15.4

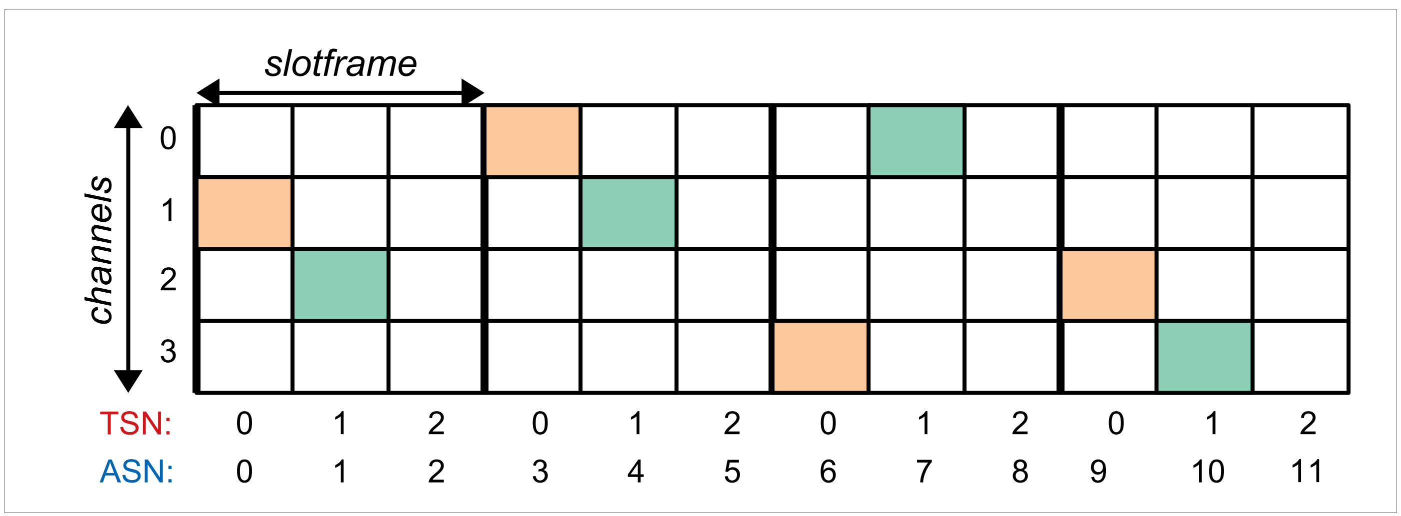
- **Time Slotted Channel Hopping (TSCH)**
  - Absolute Slot Number (ASN): identifier of slot, counted from start of network (or marked by coordinator)
  - Time Slot Number (TSN): identifier of slot within *slotframe*
  - **Time-Division Multiplexing**



MAC layer - IEEE 802.15.4

- **Time Slotted Channel Hopping (TSCH)**
  - Absolute Slot Number (ASN): identifier of slot, counted from start of network (or marked by coordinator)
  - Time Slot Number (TSN): identifier of slot within *slotframe*
  - **Channel Hopping**

ch = (ASN + offset) ***mod*** numChannels



Example: offset = 1, numChannels = 4

MAC layer - IEEE 802.15.4

- **Time Slotted Channel Hopping (TSCH)**
  - Shuffle channels : pseudo-randomly shuffled set of all channels

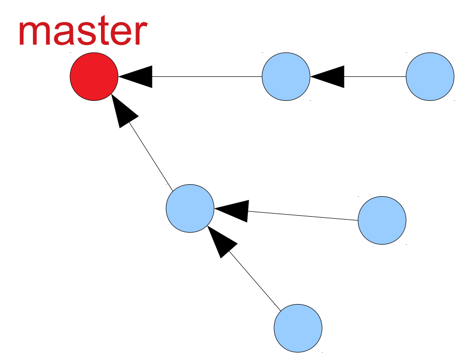
    ch = hoppingSequence **[(** ASN **+** offset **)** ***mod*** **||**hoppingSequence**||]**

  - Details
    - Default hopping sequence obtained by shuffling set of channels with sequence obtained from a *Linear Feedback Shift Register* (LFSR) with generator polynomial $(x^9 + x^5 + 1)$ and starting seed 255.
  - Example at 2.4GHz, channels = [11..26]
    - shuffling sequence = [15, 14, 12, 8, 0, 0, 1, 3, 7, 15, 14, 13, 11, 7, 15, 15]
    - shuffled sequence = [16, 17, 23, 18, 26, 15, 25, 22, 19, 11, 12, 13, 24, 14, 20, 21]

see example section **6.2.10 Channel hopping** of IEEE802.15.4-2015 standard and **example code** at https://gist.github.com/twatteyne/2e22ee3c1a802b685695

MAC layer - IEEE 802.15.4

- **TSCH - How to maintain time synchronization ?**
  - Nodes clocks drift from each other
    - due to manufacturing tolerance, changes in T°C and voltage
  - **Global synchronization**
    - Single clock source/master
    - Each node has a clock parent/source
    - Use e.g. routing to determine clock parent



  - **Local synchronization**
    - Estimate deviation from parent's clock
    - Based on message exchanges
    - Use of Enh-ACK frames

See also :

- ***Adaptive Synchronization in IEEE802.15.4e Networks***, D. Stanislowski et al, IEEE Transactions on Industrial Informatics, 2013
- ***Adaptive synchronization in multi-hop TSCH networks***, T. Chang et al, Computer Networks, 2015

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

Network Layer

- **Introduction**
  - Objective: layer 3 provides multi-hop paths
  - Variety of approaches: lots of different proposals.

  - Challenges
    - not always-ON links
    - quality of links : interferences / collisions → need for specific metrics ?
    - memory constraints : full routing tables ?
    - energy constraints : higher energy consumption in relays → need for specific metrics ?
    - node stability (exhausted energy) / mobility

Network Layer

- **Introduction**
  - Study MANET (Mobile Ad-hoc Network) Routing Protocol
    - Table-driven / proactive protocols
    - On-demand / reactive protocols
  - Examples of standardized protocols
    - Optimized Link-State Routing (OLSR)
    - Ad-hoc On-demand Distance Vector (AODV)
      - many derivatives : uAODV, LOAD, LOADng, ZigBee routing protocol, ...
    - Routing Protocol for Low-Power Lossy Networks (RPL)
  - Many other approaches not discussed in this course
    - e.g. gradient routing, opportunistic routing, geographic routing, ...

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
  - AODV
  - RPL
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

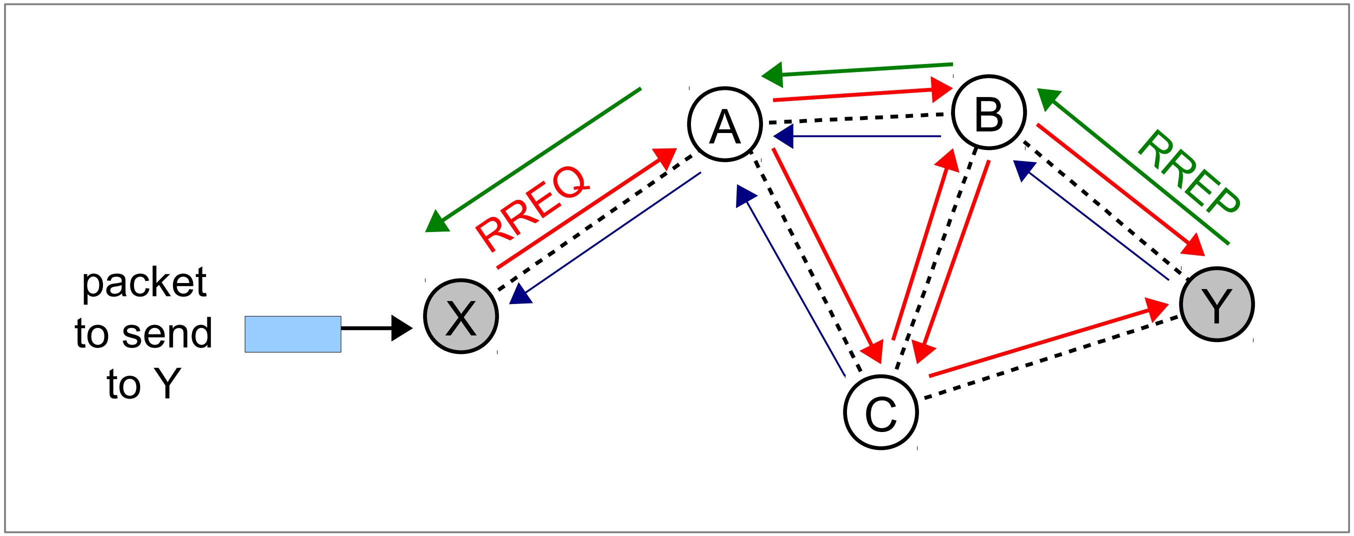
Network Layer - AODV

- **AODV Routing**
  - *Ad-hoc On-demand Distance Vector*, RFC3561 (July 2003)
  - Mesh networks routing protocol
  - Works above IP, using UDP port 654
  - Can be used on wired and wireless networks
  - 4 types of messages
    - Route Request (RREQ)
    - Route Reply (RREP)
    - Route Error (RERR)
    - Route-Reply Acknowledgment (RREP-ACK)

  - Not specific to WSNs !
  - Implementations available in RTOS such as TinyOS and Contiki

Network Layer - AODV

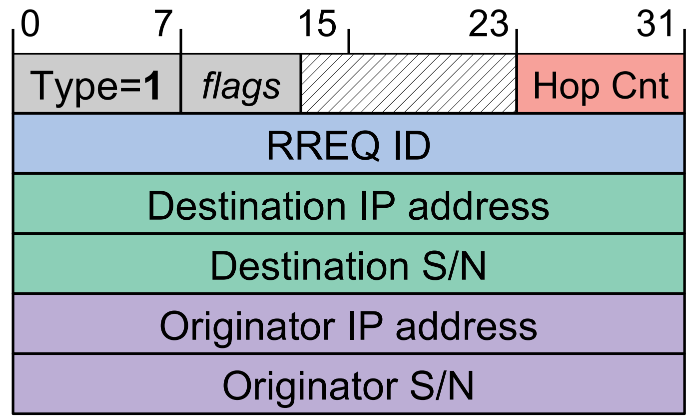
- **Intuition**



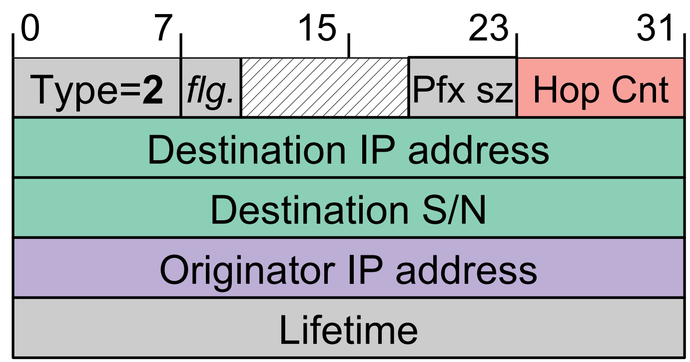
- X has no route to Y → on-demand route to Y
- X floods RREQ towards Y
- Reverse routes towards X maintained by nodes
- Y unicasts RREP towards X
  (thanks to reverse routes)
- Distance Vector approach: lowest hop-count

Network Layer - AODV

- **Message formats**
  - **Route Request (RREQ)**
    - flags JRGDU
    - D flag : destination only
    - U flag : unknown dst. S/N



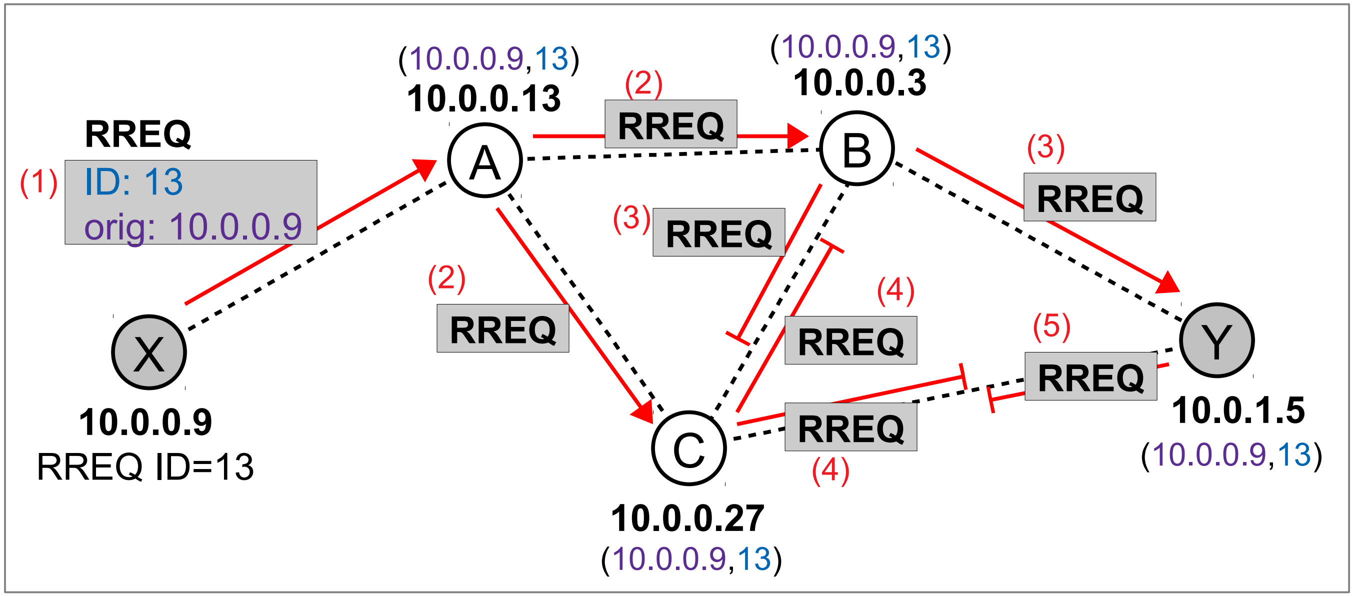
  - **Route Reply (RREP)**
    - flags RA
    - A flag : RREP-ACK required
    - prefix size in case of route aggregation ()



S/N = sequence number

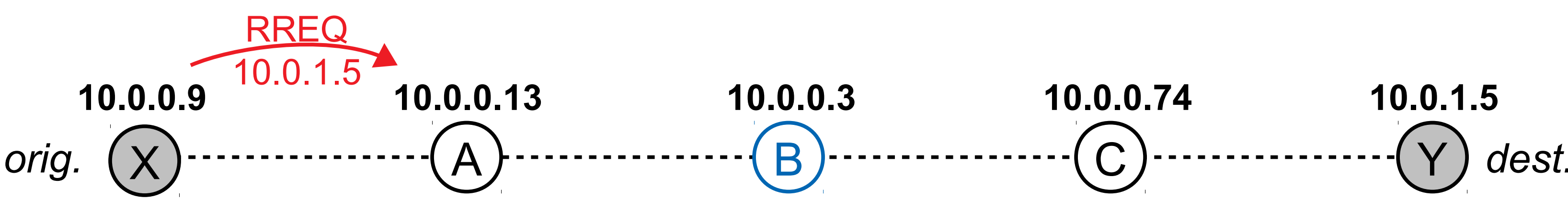
Network Layer - AODV

- **On-Demand Flooding of Route Request**
  - RREQ sent in broadcast (dst. IP address = `255.255.255.255`)
  - Increment RREQ ID before each new request
  - Couple (RREQ ID, Orig. address) used as identifier during RREQ flooding
    - Each node buffers previous RREQ ; filters already received RREQ



Network Layer - AODV

- **Routing Table**
  - Usual fields: dest. IP prefix, output interface, next-hop
  - Additional fields: sequence number (S/N), lifetime, hop count, flags, precursors



Routing Table of B

Dest.	Output interface	Next-hop	Dst S/N	Hop Count	Expiration time	Flags	Precursors
10.0.0.9	en0	10.0.0.13	173	2	500	valid	10.0.0.74
10.0.0.13	en0	10.0.0.13	unknown	1	600	valid	---
10.0.1.5	en0	10.0.1.74	47	3	900	valid	10.0.0.9

Typ. a single interface

Used for *route freshness*

current time upon insertion + lifetime

nodes likely to use local node as next-hop along a route (based on replies sent)

Network Layer - AODV

- **Where do routes come from ?**
  - **Forward routes** towards destination, learned through RREP
    - valid dest. S/N taken from RREP
  - **Reverse routes** towards the originator, learned via RREQ
    - valid orig. S/N taken from RREQ
  - **Routes to previous hops**, learned through RREQ and RREP
    - these routes do not have a valid S/N

- **Updating routes**
  - Let *R* the existing route, *Q* the new route
  - Route *R* is updated if

    (R.SN < Q.SN)

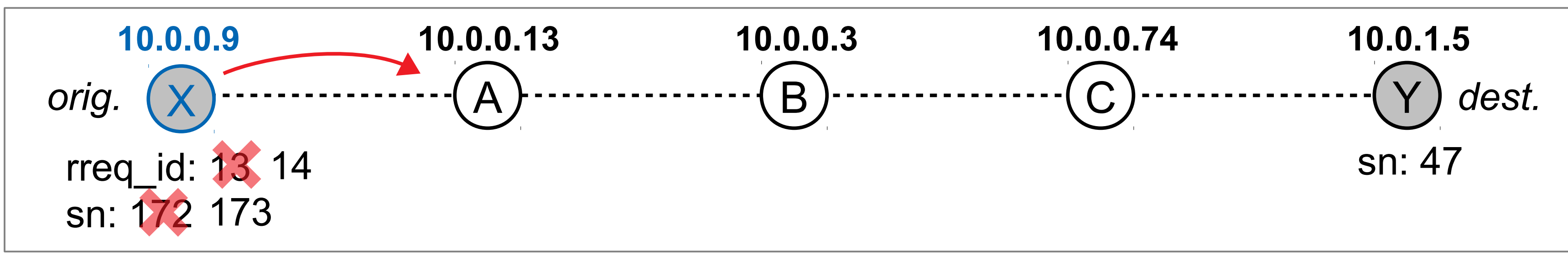
    or if

    (R.SN = Q.SN)

    and (R.hop_count > Q.hop_count)

Network Layer - AODV

- **RREQ Generation -** Originator
  - increments RREQ ID
  - increments local S/N
  - send RREQ, **broadcast**

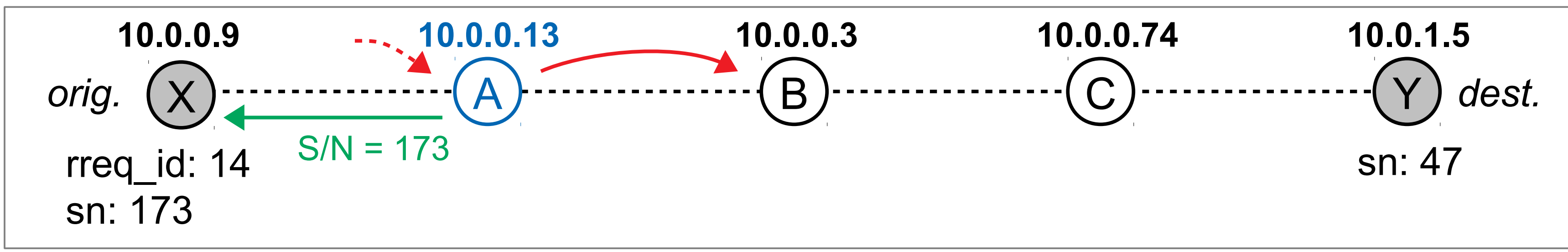


IP src: 10.0.0.9
dst: 255.255.255.255

RREQ hop_count: 0
rreq_id: 14
dst_addr: 10.0.1.5
dest_sn: 47 → last known dest. S/N (or unknown)
orig_addr: 10.0.0.9
orig_sn: 173

Network Layer - AODV

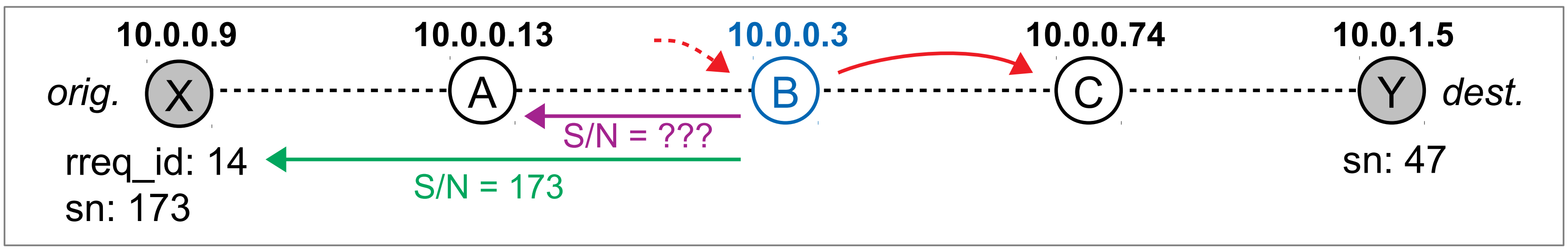
- **RREQ Propagation -** Intermediate node
  - update / create route to originator (*reverse route*)
  - increase hop count ; propagate



IP	src: 10.0.0.13 dst: 255.255.255.255
RREQ	hop_count: **1** rreq_id: 14 dst_addr: 10.0.1.5 dest_sn: 47 orig_addr: 10.0.0.9 orig_sn: 173

Network Layer - AODV

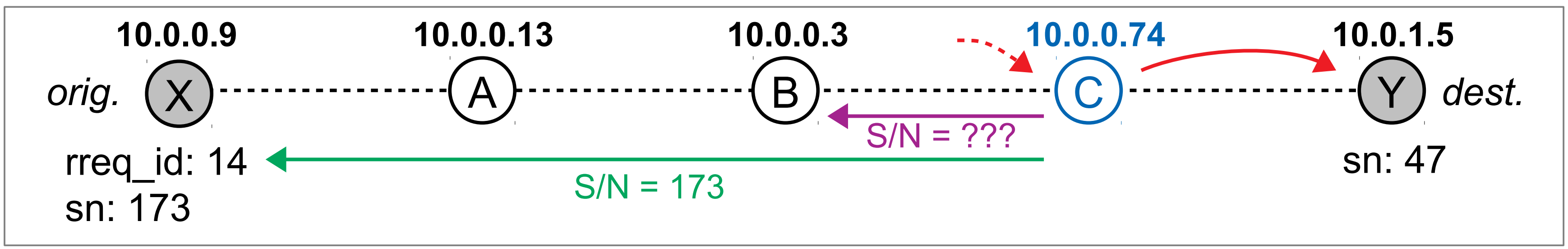
- **RREQ Propagation -** Intermediate node
  - update / create route to previous hop (reverse route)
  - update / create route to originator (*reverse route*)
  - increase hop count ; propagate



IP	src: 10.0.0.3
	dst: 255.255.255.255
RREQ	hop_count: **2**
	rreq_id: 14
	dst_addr: 10.0.1.5
	dest_sn: 47
	orig_addr: 10.0.0.9
	orig_sn: 173

Network Layer - AODV

- **RREQ Propagation -** Intermediate node
  - update / create route to previous hop (reverse route)
  - update / create route to originator (*reverse route*)
  - increase hop count ; propagate

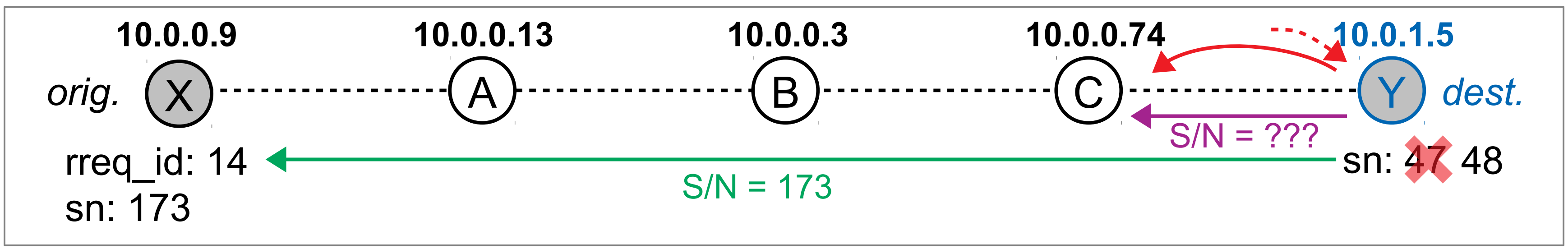


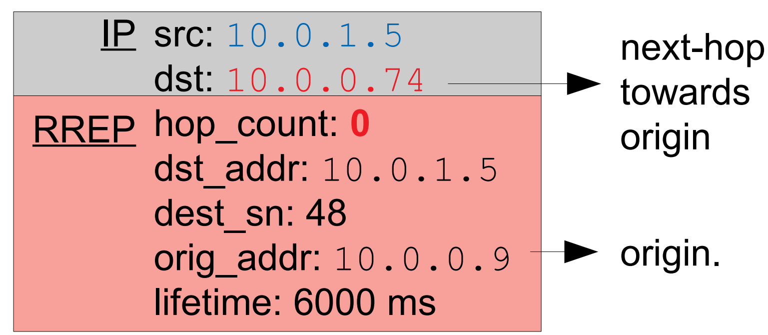
<u>IP</u> src: `10.0.0.74`
dst: `255.255.255.255`

<u>RREQ</u> hop_count: **3**
rreq_id: 14
dst_addr: `10.0.1.5`
dest_sn: 47
orig_addr: `10.0.0.9`
orig_sn: 173

Network Layer - AODV

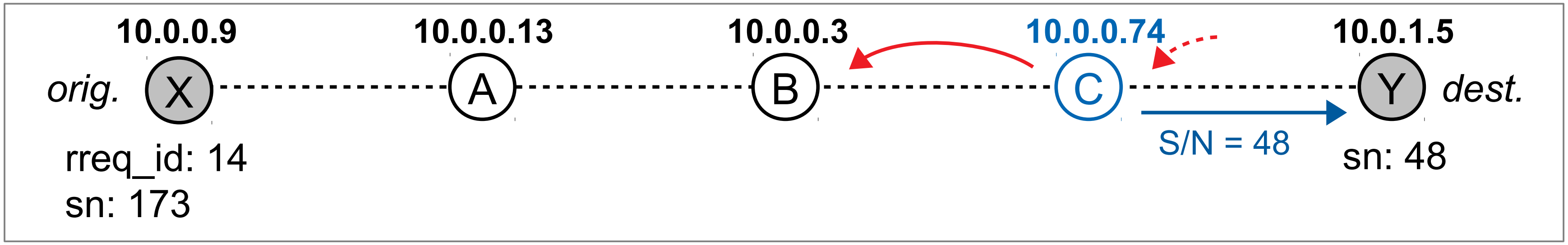
- **RREP Generation -** Destination node
  - update / create route to previous hop (reverse route)
  - update / create route to originator (*reverse route*)
  - increment S/N (if received RREQ has same dest. S/N)
  - send RREP, **unicast** (lookup route towards originator)





Network Layer - AODV

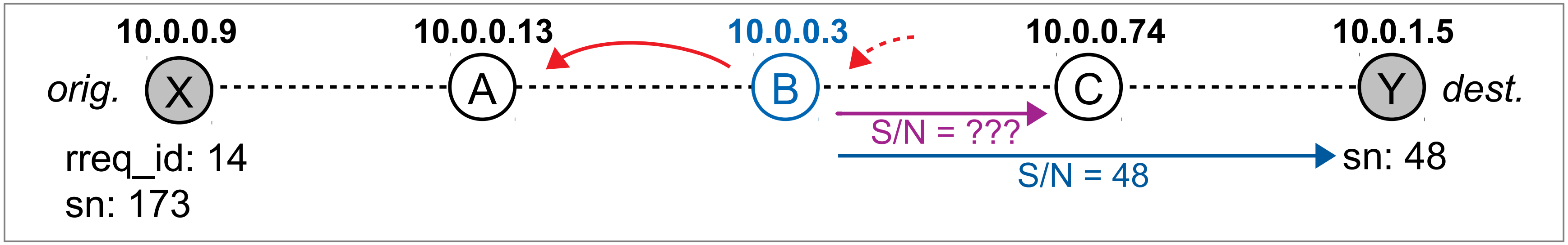
- **RREP Propagation -** Intermediate node
  - update / create route to destination (forward *route*)
  - increase hop count
  - propagate to originator, unicast



IP	src: 10.0.0.74
	dst: 10.0.0.3
RREP	hop_count: **1**
	dst_addr: 10.0.1.5
	dest_sn: 48
	orig_addr: 10.0.0.9
	lifetime: 6000 ms

Network Layer - AODV

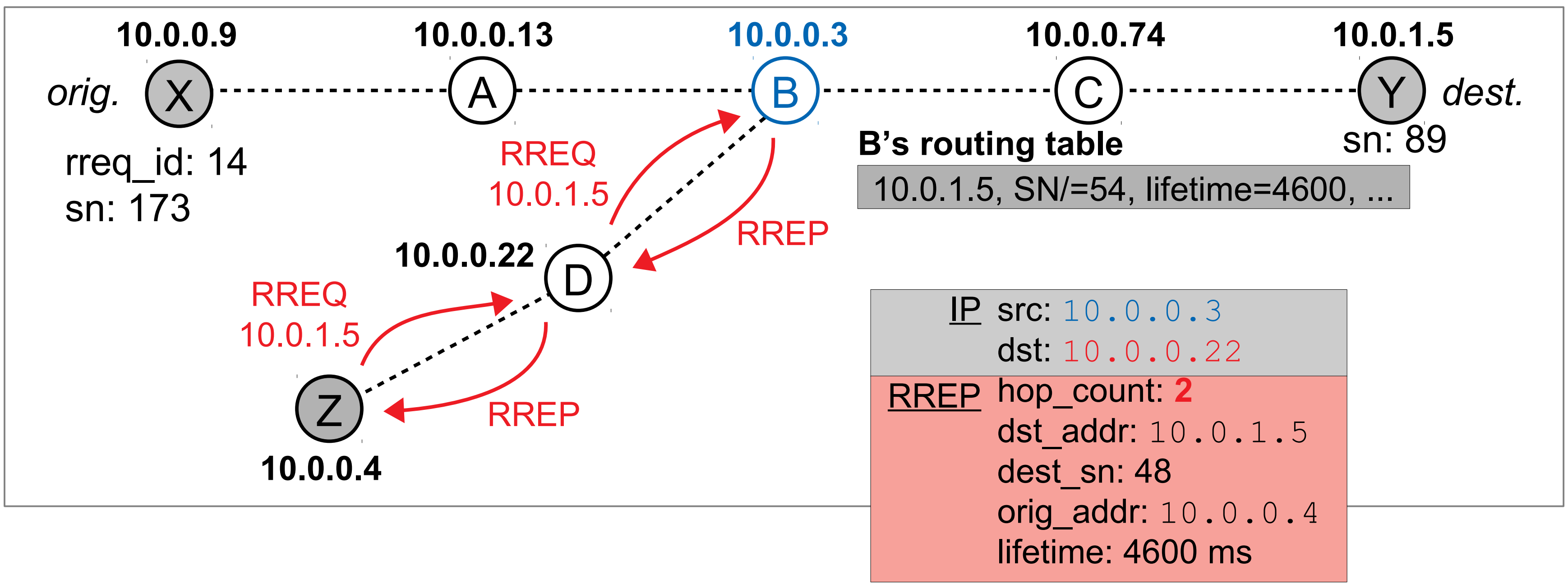
- **RREP Propagation -** Intermediate node
  - update / create route to prev. hop (reverse route)
  - update / create route to destination (forward *route*)
  - increase hop count
  - propagate to originator, unicast



IP	src: 10.0.0.3 dst: 10.0.0.13
RREP	hop_count: **2** dst_addr: 10.0.1.5 dest_sn: 48 orig_addr: 10.0.0.9 lifetime: 6000 ms

Network Layer - AODV

- **RREP Generation –** Intermediate node
  - update / create reverse routes as usual
  - can send an RREP if **active route towards destination** and **route S/N >= destination S/N in RREQ**



Network Layer - AODV

- **Additional details**
  - Expanding ring search
    - RREQ scope limited by IP TTL
    - Initial TTL = TTL_START (e.g. 1)
    - If no RREP received, increase TTL by TTL_INCREMENT (e.g. 2) and send new RREQ until some TTL_THRESHOLD is reached
  - Link failures
    - Hello messages (RREP) are sent regularly by a node on an active route towards its predecessor hop
    - If no Hello is received within a specific time, the route is considered as invalid and a recovery procedure must start
  - Gratuitous RREP

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
  - AODV
  - RPL
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

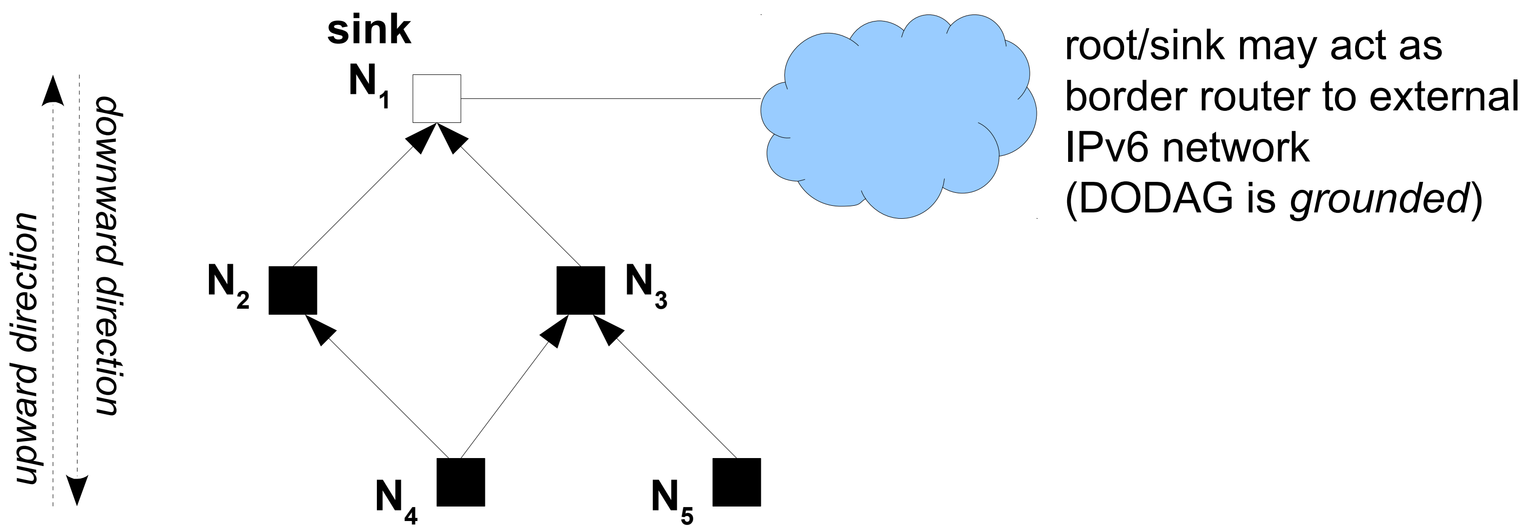
RPL
IPv6 Routing Protocol for Low-Power and Lossy Networks

- **Introduction**
  - IETF RFC6550 (March 2012)
  - Mainly for collect application : traffic from nodes to sink
  - Proactive: routes established before they are needed
  - Distance-vector
  - Versatile: different link/path metrics (hop count, energy, link quality and constraints)
  - Builds a virtual tree-like topology: DODAG (*Destination Oriented Directed Acyclic Graph*)
  - Some principles inspired from *Collection Tree Protocol* (CTP)
    - *adaptive beaconing* (trickle) and *path validation*

Reference: ***Collection Tree Protocol***, O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, ACM SenSys, 2009.

RPL

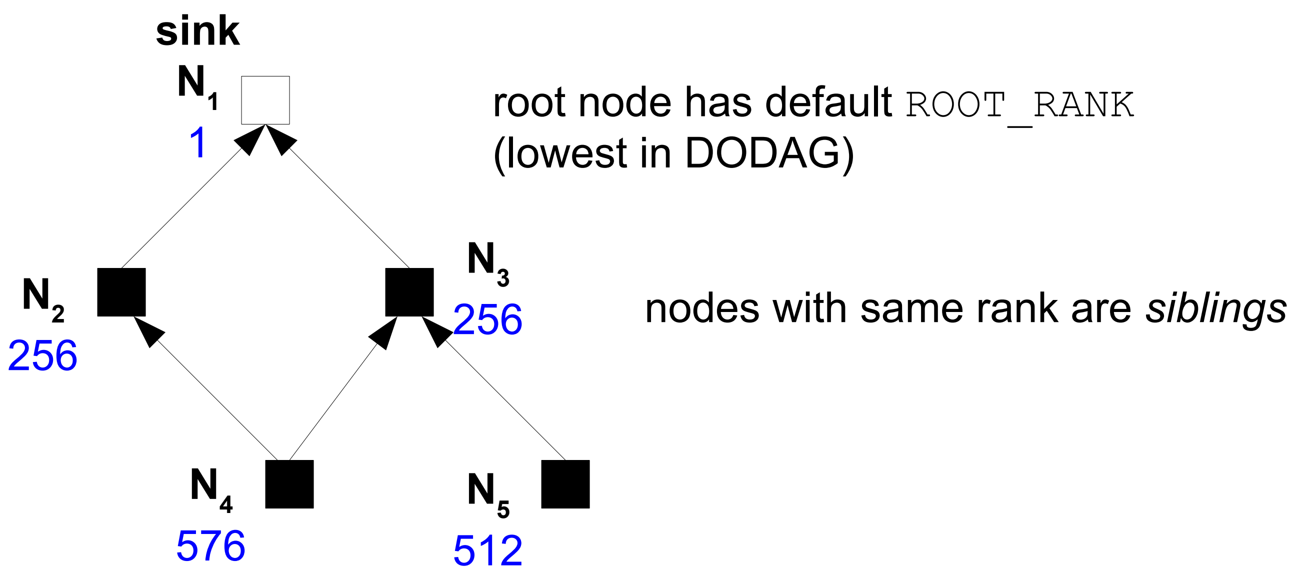
- **DODAG**
  - Destination Oriented Directed Acyclic Graph
  - DAG with a single root (traffic sink)



  - DODAG may change accross time → version number
  - DODAG has unique identifier (`DODAGID`), typically one IPv6 address of the root

RPL

- **Rank and parents**
  - Rank: integer value assigned to each node
  - Role of rank: determine routing position in DODAG based on objective function (OF)
  - Increasing monotonically : $\text{rank}(x) > \text{rank}(\text{parent}(x))$



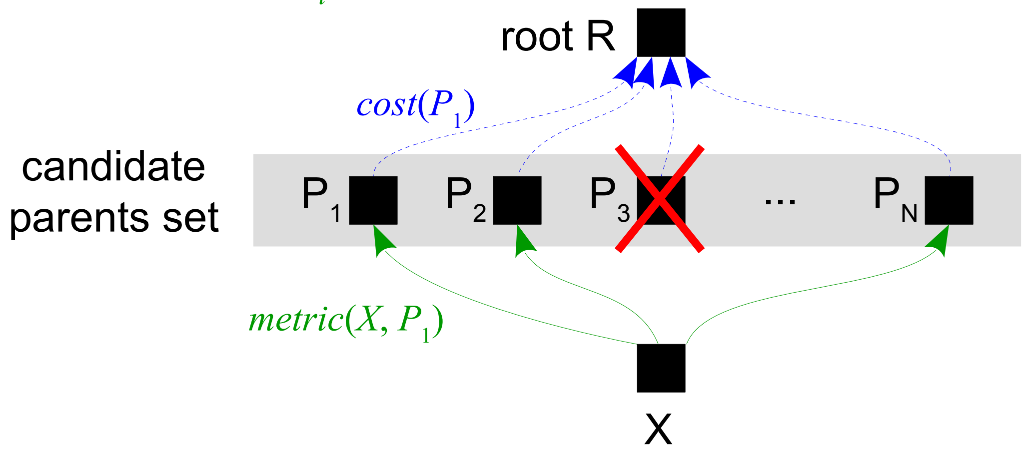
  - Rank computed by node when it selects parent(s)

RPL

- **Objective function (OF)**
  - Role
    - orders, selects candidate parents
    - computes node rank
  - Routing protocols requirements
    - Traditional routing protocols : simple, implicit OF, e.g. *minimize path cost*
    - WSNs : more versatile, e.g. min. # of transmissions AND avoid energy constrained nodes
  - OF standardized so far
    - OF0 - *Objective Function Zero* (RFC6552, 3/2012)
    - MRHOF - *Minimum Rank with Hysteresis Objective Function* (RFC6719, 9/2012)

RPL

- **Objective function "MRHOF"**
  - Simplified : assumes additive metric
  - Compute cost to root through each candidate parent
    - $cost(P_i)$ sent by P_i to X in RPL message
    - filter parents : too high cost / link metric ; constraint mismatch
    - $metric(X, P_i)$ locally computed by X



$$cost(P_i) = cost(P_i) + metric(X, P_i)$$

    - pick parent with smallest cost
    - <u>hysteresis</u> : change parent only if $|cost - old\_cost| > threshold$

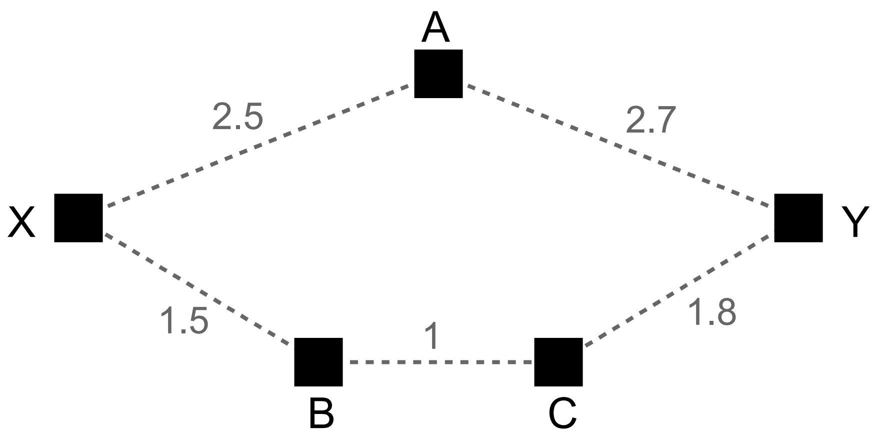
MRHOF: Minimum Rank with Hysteresis Objective Function

RPL

- **Metrics and attributes**
  - Metrics
    - Hop-Count
    - Link throughput
    - Link latency
    - Expected Transmission Count (ETX)
    - Link Quality Level (LQL) : unknown / high / medium /low
  - Attributes (used in constraints)
    - Link Color Attribute : define link classes
    - Node State and Attribute : e.g. node with limited CPU/memory resources
    - Node Energy (NE) : mains / battery / harvesting

RPL

- **Expected Transmission Count (ETX)**
  - Average number of transmissions required to transmit successfully (including retransmissions)
  - path ETX = sum of link ETX



ETX(X-A-Y) = 5.2
ETX(X-B-C-Y) = 4.3

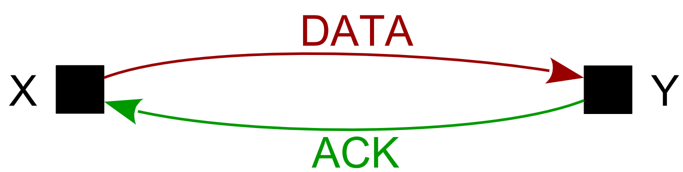
HopCount(X-A-Y) = 2
HopCount(X-B-C-Y) = 3

  - ETX computation = responsibility of link layer
    - could be computed at each transmission attempt
    - could rely on probes sent in both directions

Reference: ***A High-Throughput Path Metric for Multi-Hop Wireless Routing***
De Couto, D., Aguayo, D., Bicket, J., and R. Morris, ACM MobiCom, 2003

RPL

- **ETX computation**
  - Successful transmission implies success of DATA frame and of ACK frame



$$p_{\text{success}} = p_{\text{DATA}}\, p_{\text{ACK}}$$

  - Probes sent in both directions at regular interval
    - Success probability of forward and reverse probes can be computed (respectively p_f and p_r)
    - A sends probes to B ; B counts received probes ;
      B computes p_f = count / sent
  - Note: in RPL, approach is different (to avoid sending probes)

Reference: ***A High-Throughput Path Metric for Multi-Hop Wireless Routing***
De Couto, D., Aguayo, D., Bicket, J., and R. Morris, ACM MobiCom, 2003

RPL

- **ETX computation**
  - What is the probability that k transmissions are required ?
    - Single frame transmission = event of a Bernoulli trial (either success or failure)
    - Probability of single event

$$p = p_f \cdot p_r$$

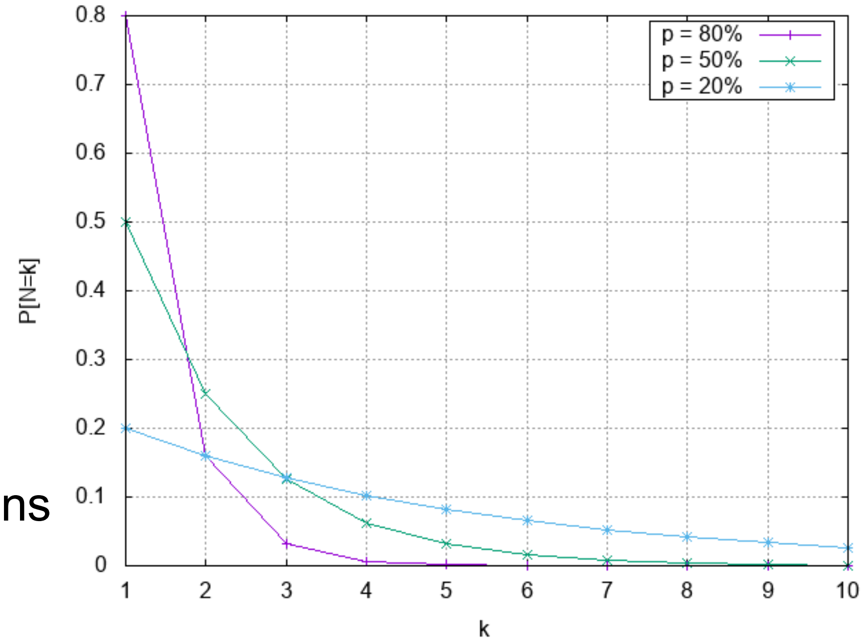
    - Probability that k transmissions required to obtain a success (geometric distribution)

$$P[N=k] = (1-p)^{k-1} p$$

    - Expected number of transmissions

$$E[N] = \sum_{k=0}^{+\infty} k \cdot P[N=k]$$

$$= \ldots = \frac{1}{p} = ETX$$



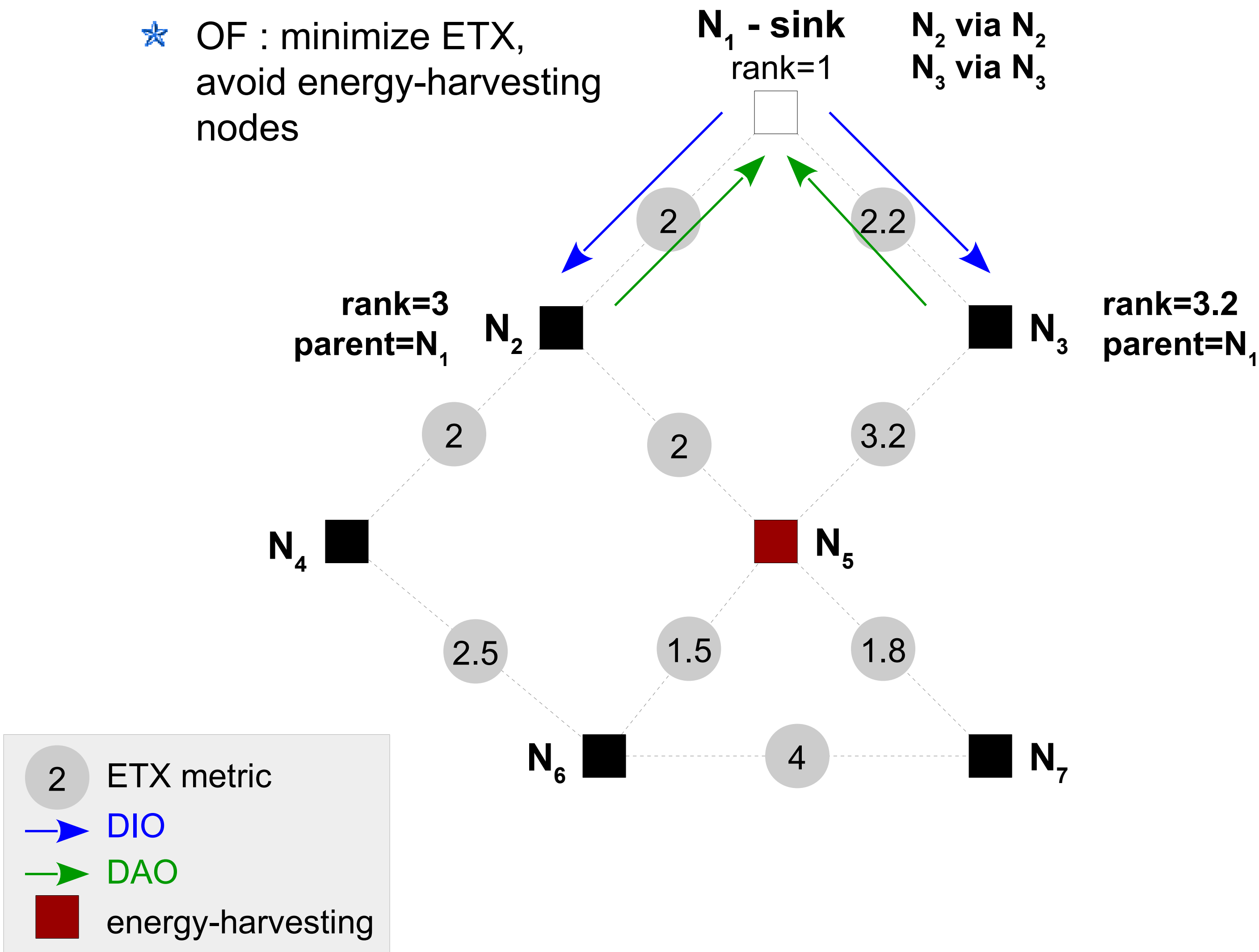
Reference: ***A High-Throughput Path Metric for Multi-Hop Wireless Routing***
De Couto, D., Aguayo, D., Bicket, J., and R. Morris, ACM MobiCom, 2003

RPL

- **Messages**
  - ICMPv6 messages
  - DIO – *DODAG Information Object*
    - sent by nodes already in DODAG ; initially only the root
    - destination address = `FF02::1A` (*all-RPL-nodes multicast*)
    - announce DODAG version, parameters, Instance ID, rank of sender, metrics and constraints
  - DIS – *DODAG Information Solicitation*
  - DAO – *DODAG Advertisement Object*
    - used to build downward routes (from root to nodes)
    - always sent upwards (towards the root)
    - different modes, e.g. storing / non-storing
  - DAO-ACK

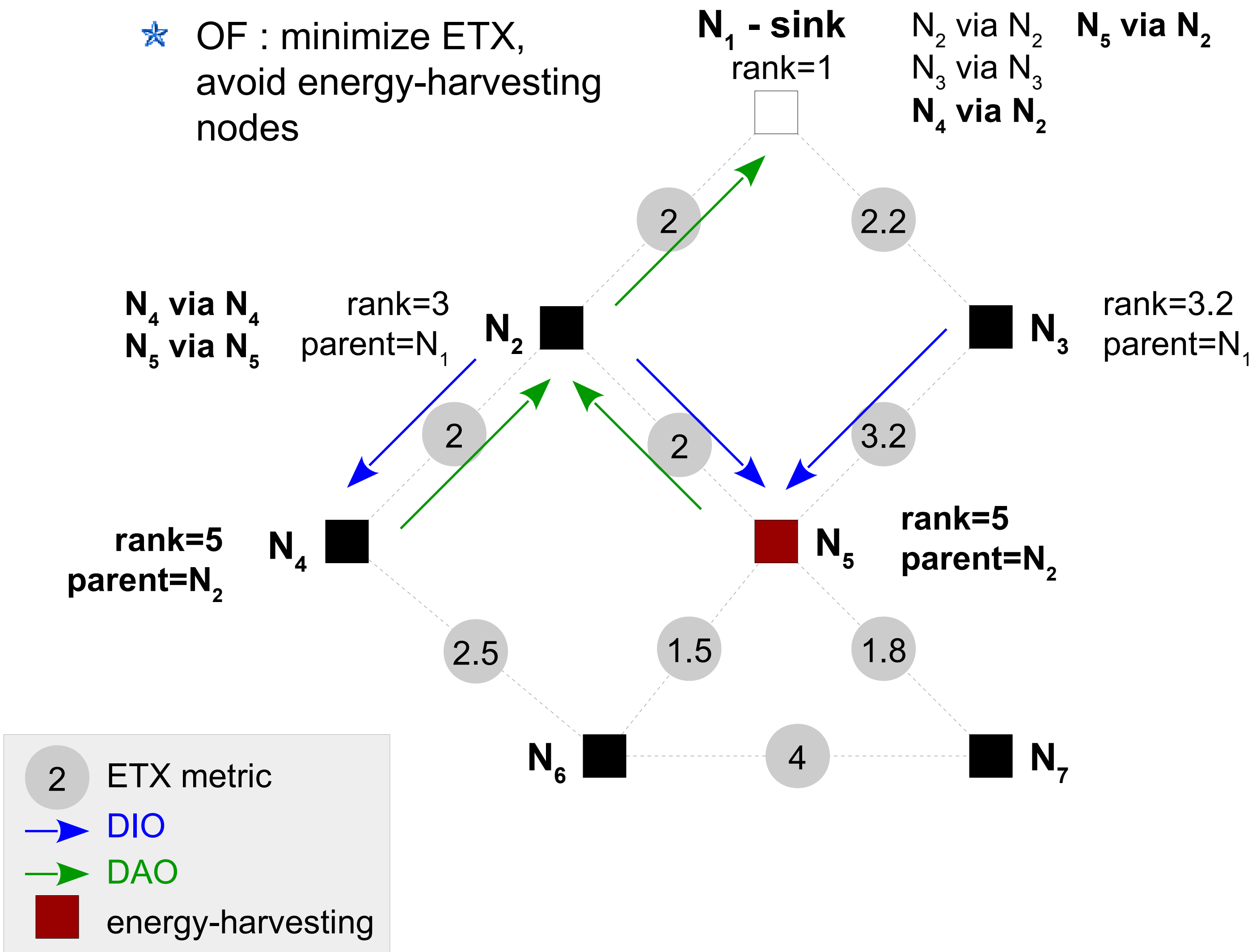
RPL

- **Example**
  - OF : minimize ETX, avoid energy-harvesting nodes



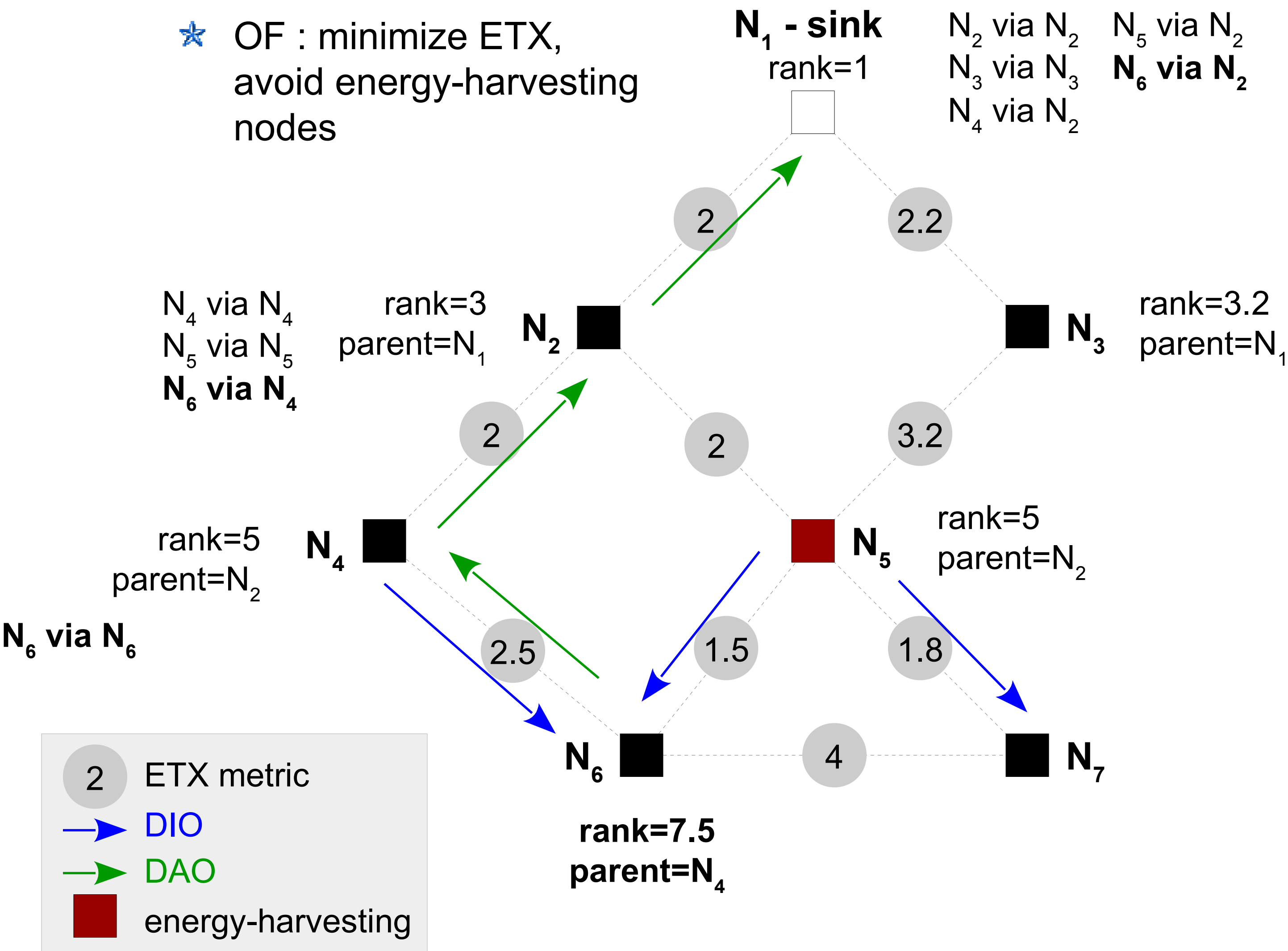
RPL

- **Example**
  - OF : minimize ETX, avoid energy-harvesting nodes



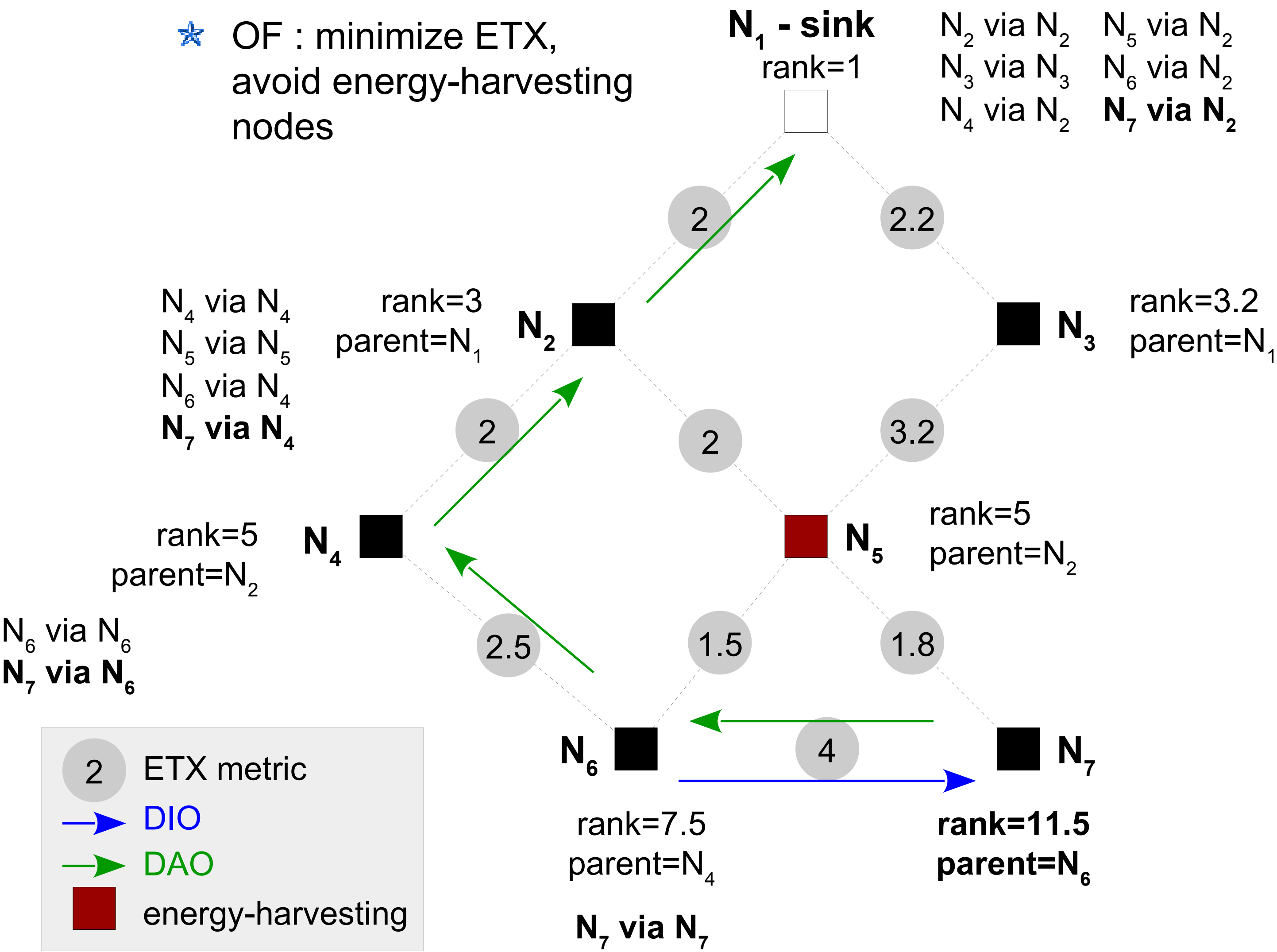
RPL

- **Example**
  - OF : minimize ETX, avoid energy-harvesting nodes



RPL

- **Example**
  - OF : minimize ETX, avoid energy-harvesting nodes



RPL – Adaptive beaconing

- **Trickle Algorithm**
  - Idea
    - when nodes need to share information, avoid sending too many times the same information
  - Principle
    - when node detects inconsistency : send more frequently
    - while no inconsistency : send less and less frequently
    - notion of inconsistency is context dependent. Here, let's say it is a version number
  - IETF RFC6206 (March 2011)
    - based on a paper by Levis et al (NSDI 2004)
    - "*To trickle*" → "*couler goutte-à-goutte*"
    - used in RPL for sending DIO

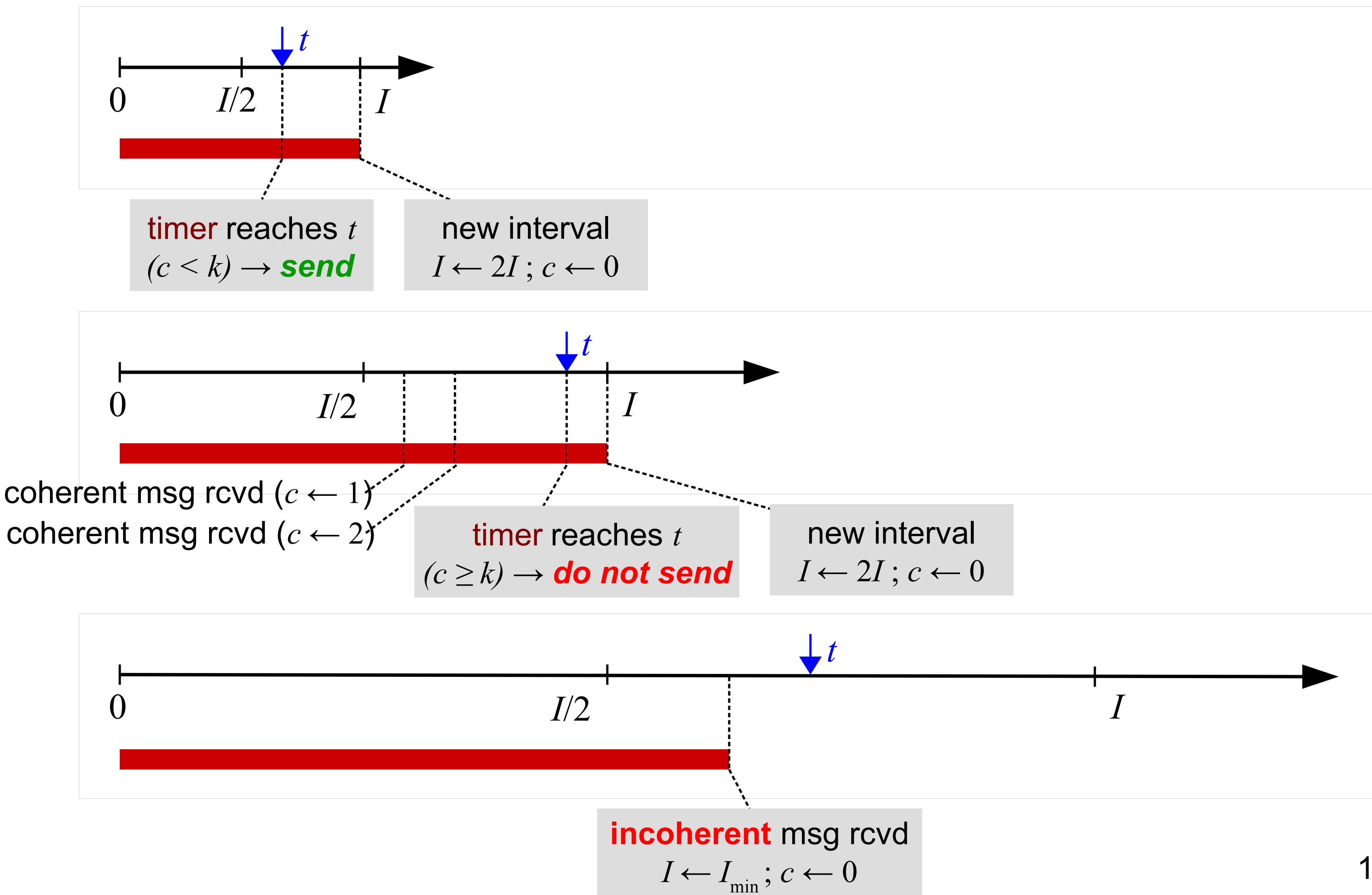
Source :

RPL – Adaptive beaconing

- **Trickle Algorithm**
  - Parameters
    - [I_{min} , I_{max}] - Range of possible transmit intervals (in seconds)
    - k – amount of redundancy required (an integer value)
  - Variables
    - counter c *(for coherent messages received)*
    - timer expiration t
    - current transmit interval I (s.t. $I_{min} \leq I \leq I_{max}$)
  - Rules
    - new interval : $c \leftarrow 0$; select t in [$I/2$, I]
    - when consistent message received : $c \leftarrow c + 1$
    - when timer expires ; if ($c < k$) send message
    - when interval expires ($t = I$) ; $I \leftarrow min(2I , I_{max})$; new interval
    - when inconsistent message received : $I \leftarrow I_{min}$; new interval

RPL – Adaptive beaconing

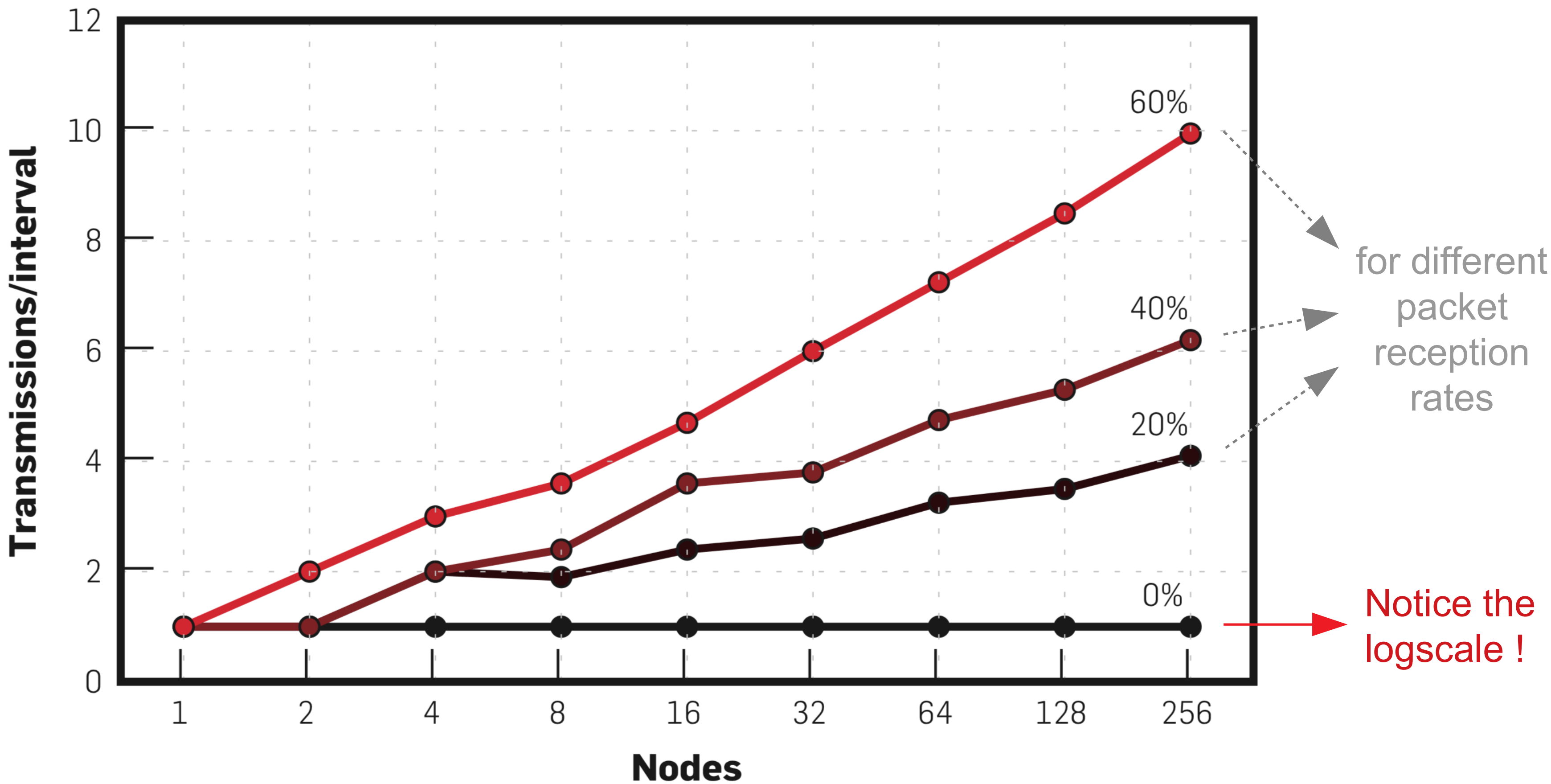
- **Trickle Algorithm** $k = 2$ (redundancy)



RPL – Adaptive beaconing

- **Trickle Algorithm**

Number of transmissions per interval grows in $O(\log(n))$ where n is the number of nodes



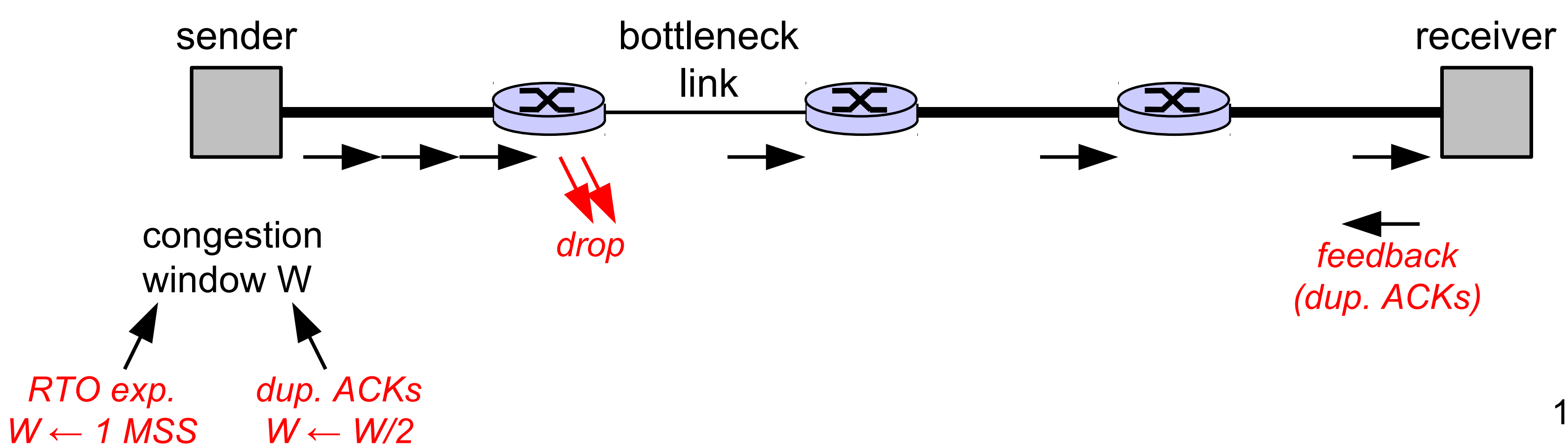
Source: ***The Emergence of a Networking Primitive in Wireless Sensor Networks***, Ph. Levis et al, Communications of the ACM, 2008

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS

Transport Layer

- **TCP for WSNs ?**
  - TCP was designed for wireline networks with low packet losses due to errors and high packet losses due to congestion
    - packet loss interpreted as congestion → reduction of congestion window → reduction of sending rate
    - end-to-end principle → consider intermediate nodes are dumb and cannot take part in retransmission



Transport Layer

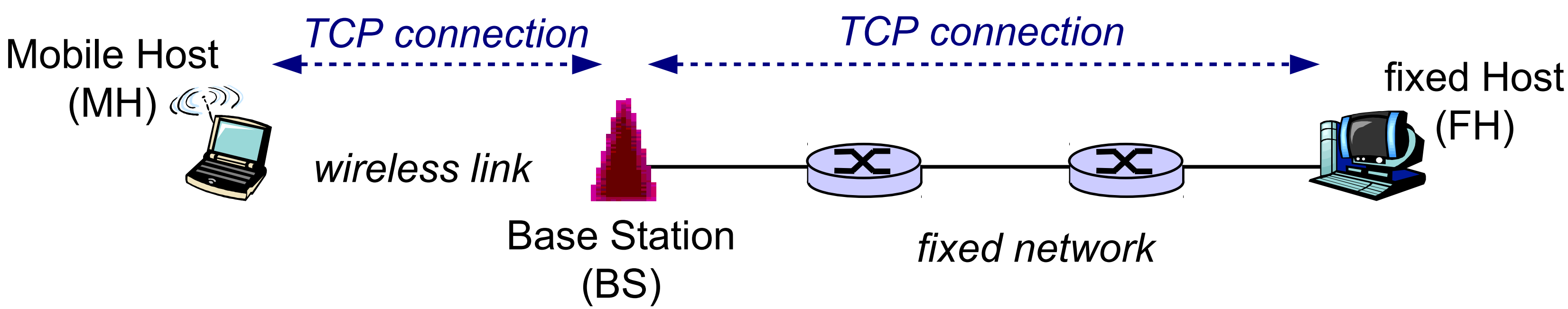
- **TCP issues**
  - Does not work well with wireless networks due to higher Bit Error Rate (BER)
    - packet losses due to bit errors ~ 5-10% and higher
    - packet lost → retransmission and reduction of congestion window although the network is not congested !
    - packets may be retransmitted by link layer BUT increases RTT → increases RTO → TCP reacts slowly to congestion changes
    - unfairness is increased : longer paths (more hops) have a higher probability of error → higher perf. degradation → receive smaller BW share
  - Moreover, TCP's end-to-end retransmissions are harmful in an energy constrained WSNs as several nodes might need to retransmit along a multi-hop path !

Transport Layer

- **Proposed Solutions**
  - Improvement to TCP with wireless links
    - Split connections: I-TCP, Split-TCP
    - Link-layer solution: Snoop
  - Improvement to TCP in WSNs
    - Distributed TCP Caching (DTC)
  - Other transport protocols
    - e.g. CoAP = HTTP over UDP

Transport Layer

- **Split-Connection (I-TCP)**
  - Main idea:
    - TCP connection is split at base station (BS)
    - two connections established, BS copies from one to the other



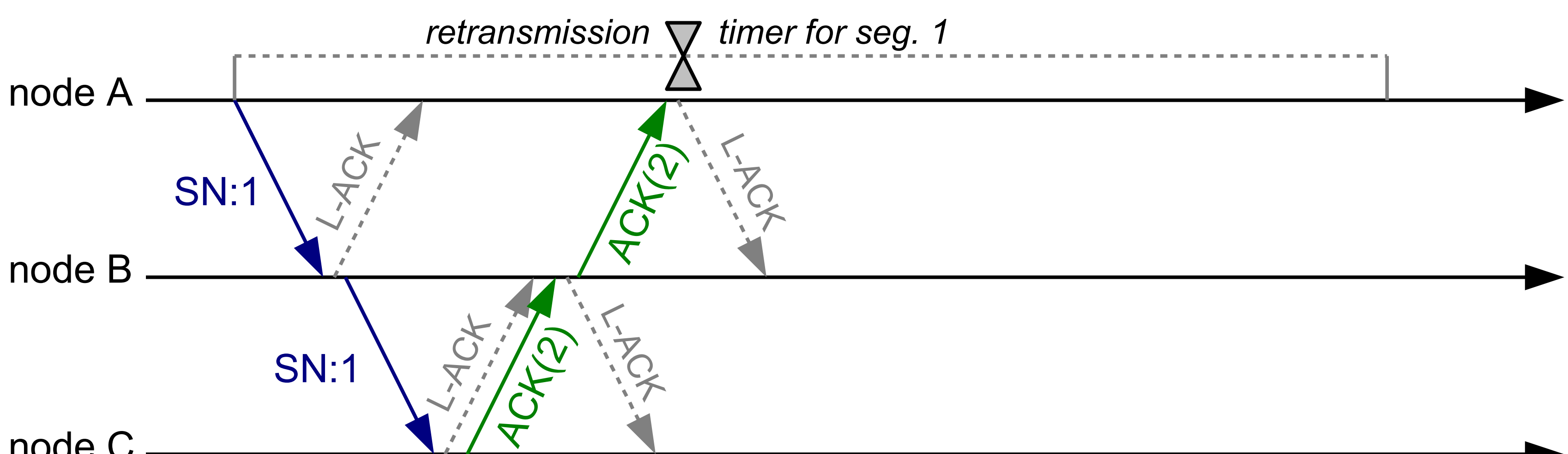
Reference: **I-TCP: Indirect TCP for Mobile Hosts**, A. Bakre and B. R. Badrinath, In Proceeding of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95), 1995

Transport Layer

- **Split-Connection (I-TCP)**
  - Benefits
    - Relies on exact same TCP protocol
    - Smaller RTT for each connection → faster recovery (remember: W+= 1MSS/RTT)
    - Wireless losses are hidden from connection between BS and FH
    - Connection between MH and BS can used fine-tuned transport protocol
  - Drawbacks
    - Violates TCP's end-to-end principle (bouhouhou)
    - Additional memory and CPU resources required on BS (buffer for both connections, connection control *2, copies from one connection to the other)
    - Handoff is more complex (requires state to be moved from former BS to new BS)

Transport Layer

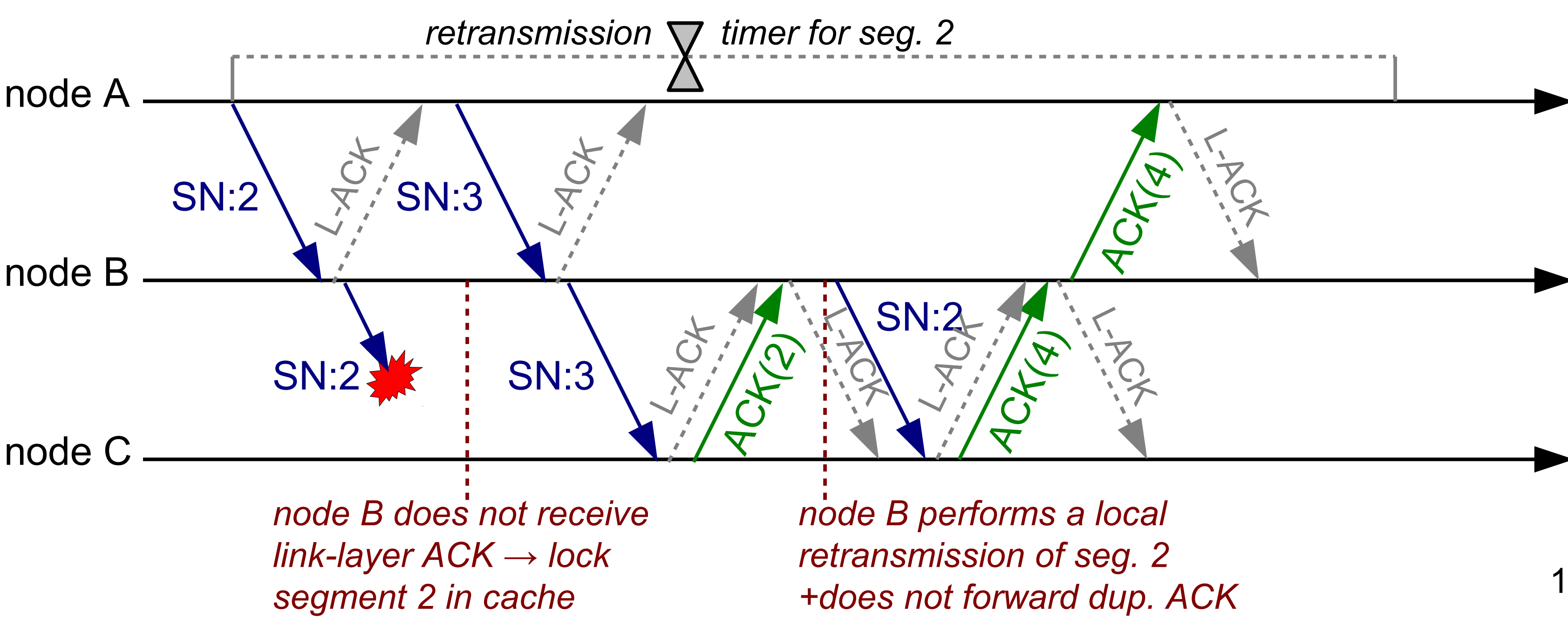
- **Distributed TCP Caching (DTC)**
  - Idea:
    - involve intermediate nodes along multi-hop path
    - try to perform local retransmissions as much as possible
    - rely on link-layer positive acknowledgements to select what must be cached



Reference: “***Distributed TCP Caching for Wireless Sensor Networks***”, A. Dunkels et al, 2004

Transport Layer

- **Distributed TCP Caching (DTC)**
  - Principle of operations
    - every node must be able to cache a few TCP segments
    - cache high seq. # segments
    - lock un-ack'd (link-layer) segments in cache
    - soft-state in intermediate nodes (age cached segments)



Transport Layer

- **Distributed TCP Caching (DTC)**
  - Benefits:
    - retransmission occur closer to destination → retransmission does not involve every node along multi-hop path
  - Drawbacks:
    - intermediate nodes need additional memory to cache segments and additional computation
    - difficult to predict improvement in presence of small cache and large sending window
      - need simulations / experimental measurements

Application Layer

- **Constrained Application Protocol (CoAP)**
  - RFC7252 (June 2014)
  - Replacement for HTTP
  - Uses UDP instead of TCP
  - Default port 5683

  - Compact messages: binary headers instead of ASCII headers
  - Asynchronous communications possible

  - Supports same methods as HTTP (GET, POST, PUT, DELETE) → can be used for RESTful applications

  - New URI scheme: `coap://`...

CoAP

- **Message format**

32 bits

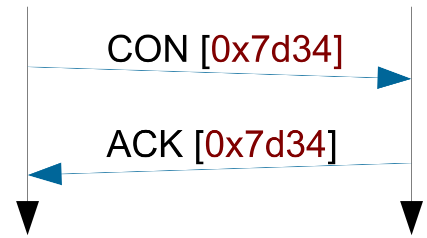
<table>
<tr><td rowspan="2">Vers. (01)</td><td rowspan="2">Type</td><td rowspan="2">Token length</td><td colspan="2">Code</td><td rowspan="2">Message ID</td></tr>
<tr><td>class</td><td>detail</td></tr>
<tr><td colspan="6">Token
(optional, 0 to 8 bytes)</td></tr>
<tr><td colspan="6">Options
(optional, variable length)</td></tr>
<tr><td colspan="3">Payload Marker (FF)
(if payload present)</td><td colspan="3" rowspan="2">Payload</td></tr>
<tr><td colspan="3"></td></tr>
</table>

payload length inferred from datagram length

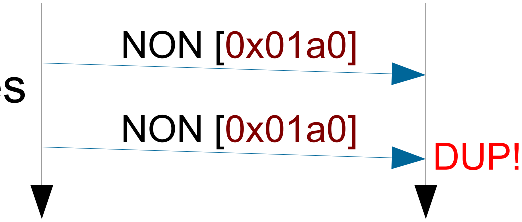
Type: CONfirmable (0), NON-confirmable (1), ACKnowledgment (2), ReSeT (3)
Code: noted c.dd (where c=class, dd=detail)
  class: request (0), successful response (2), client error response (4),
    server error response (5)
  requests: GET (0.01), POS (0.02), PUT (0.03), DELETE (0.04)
Message ID: used to detect duplicates and to match acknowledgments
Token: used to correlate requests and responses
  (token generated by client, echoed by server)

CoAP

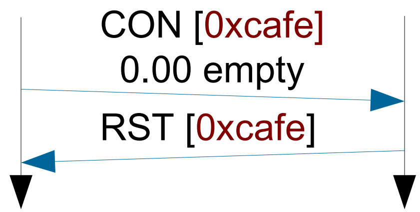
- **Simple reliable message transmission**
  - Retransmissions
    - for CONfirmable messages only
    - acknowledgments (*stop-and-wait*)
    - use of Message ID to match message/ACK
    - retransmissions with exponential back-off



  - Duplicate detection
    - for CONfirmable and NON-confirmable messages
    - use of Message ID as sequence number

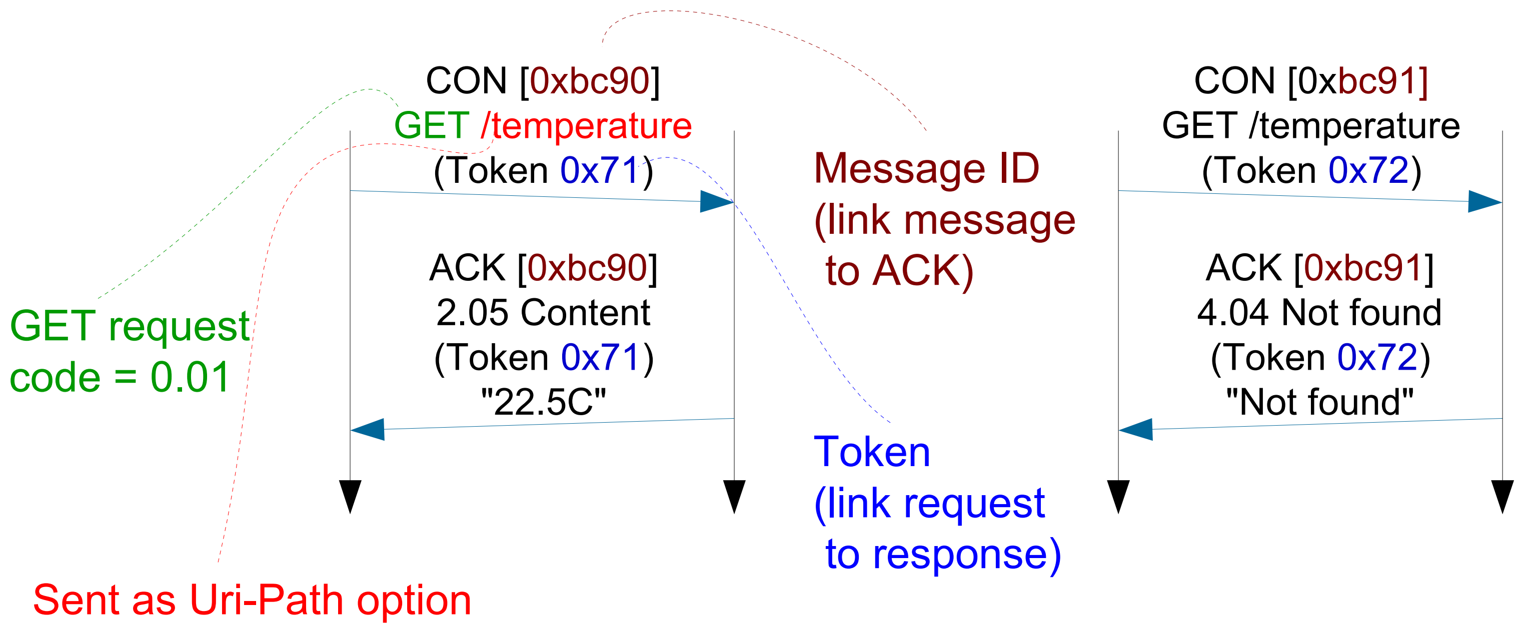


  - ReSeT messages
    - sent when received CON / NON message cannot be processed
    - testing *liveness* of remote endpoint: send empty CON message, remote endpoint should reply with ReSeT



CoAP

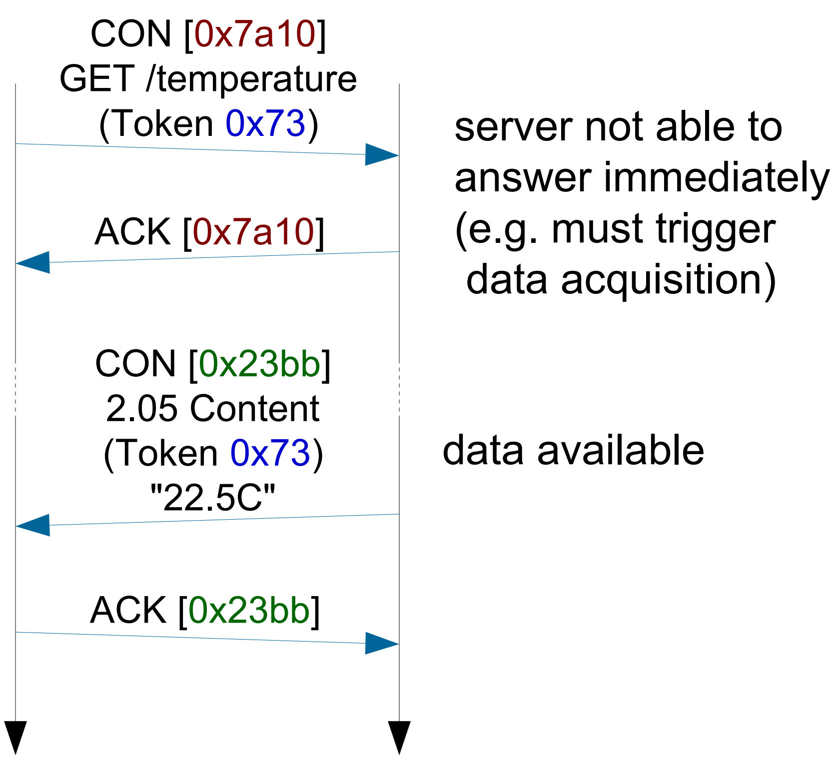
- **Example exchanges**
  - CONfirmable requests with piggybacked responses (ACK and response in same message)



source: RFC7252

CoAP

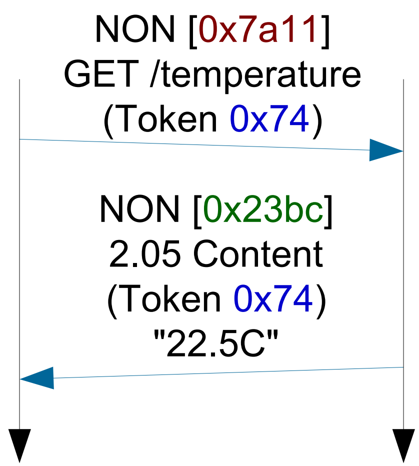
- **Example exchanges**
  - CONfirmable request with separate response



source: RFC7252

CoAP

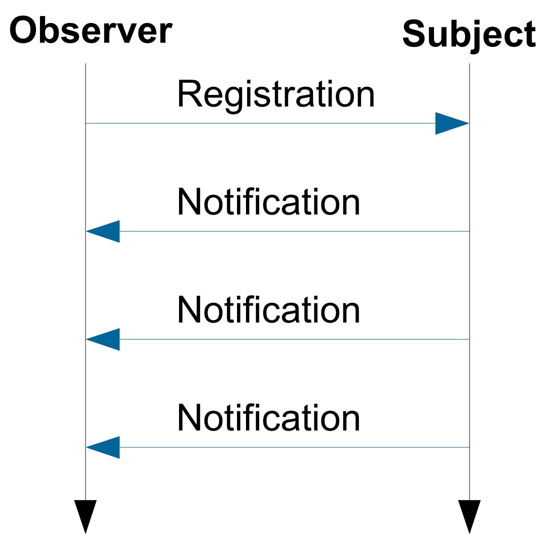
- **Example exchanges**
  - NON-confirmable request



source: RFC7252

CoAP – Observing resources

- **Observer design pattern**
  - RFC7641, September 2015
  - Asynchronous mechanism to notify registered observers of a change in resource



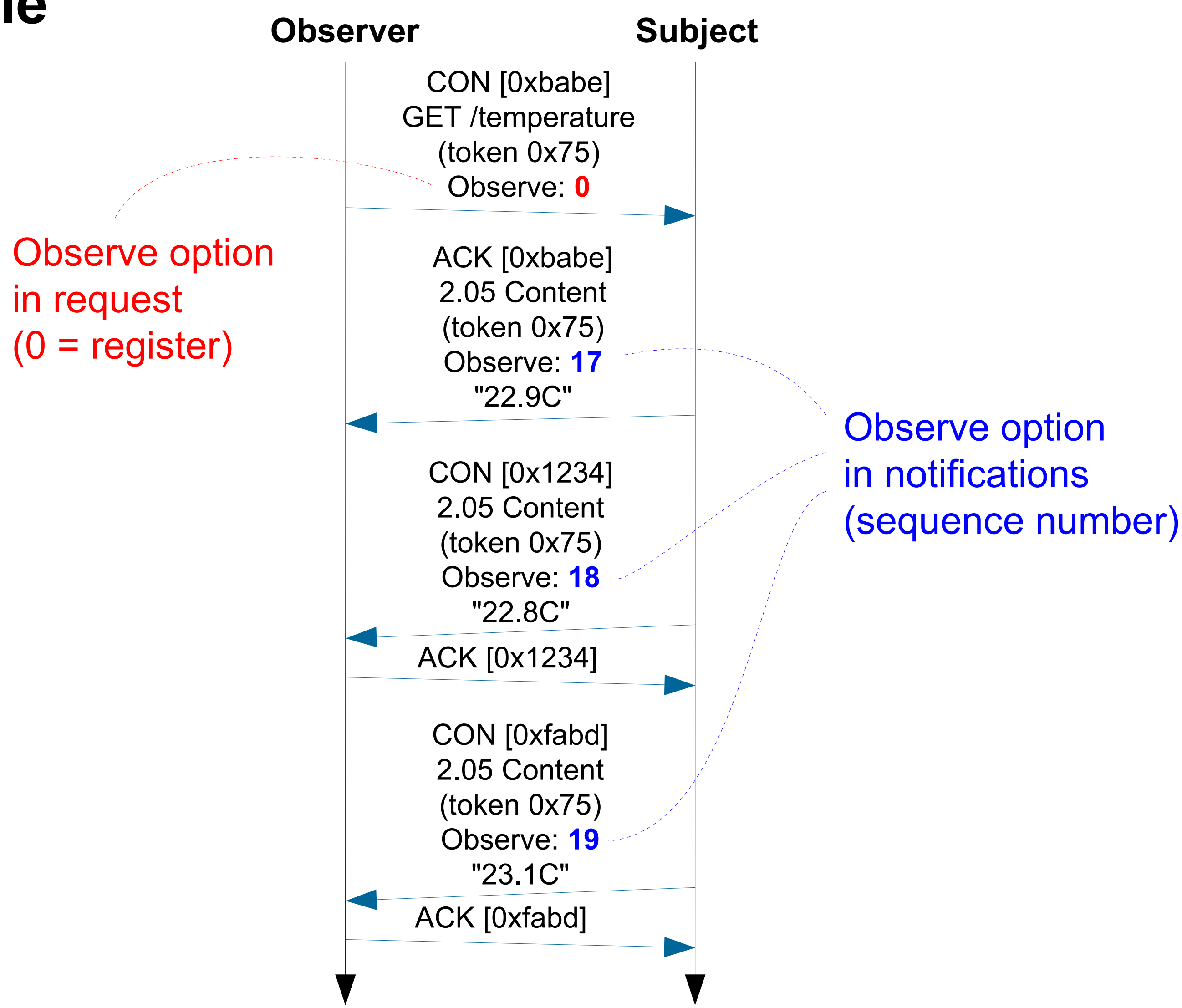
  - Subject (server) might support complex conditions for notifications such as `coap://server/temperature/critical?above=42`

CoAP – Observing resources

- **Observe option**
  - Number = 6
  - Name = Observe
  - Format = uint
  - Length = 0-3 bytes
  - Possible values for requests
    - Register (0) : adds the requestor to the list of observers
    - Deregister (1) : removes the requestor from the list of observers
  - Values for responses
    - Sequence number used to detect reordering
    - based on 24-bit "serial number arithmetic" (RFC1982)

CoAP – Observing resources

- **Example**



CoAP - Implementations

- **Copper (Cu : CoAP user-agent)**
  - CoAP protocol handler for Firefox
  - http://people.inf.ethz.ch/mkovatsc/copper.php
- **Californium (Cf : CoAP framework)**
  - Java implementation (client and server)
  - https://github.com/eclipse/californium
- **libcoap**
  - C implementation for constrained devices
  - http://libcoap.sourceforge.net

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
  - uIP
  - 6LoWPAN
- 4.8 Programming Model / RTOS

uIP – A lightweight IP stack

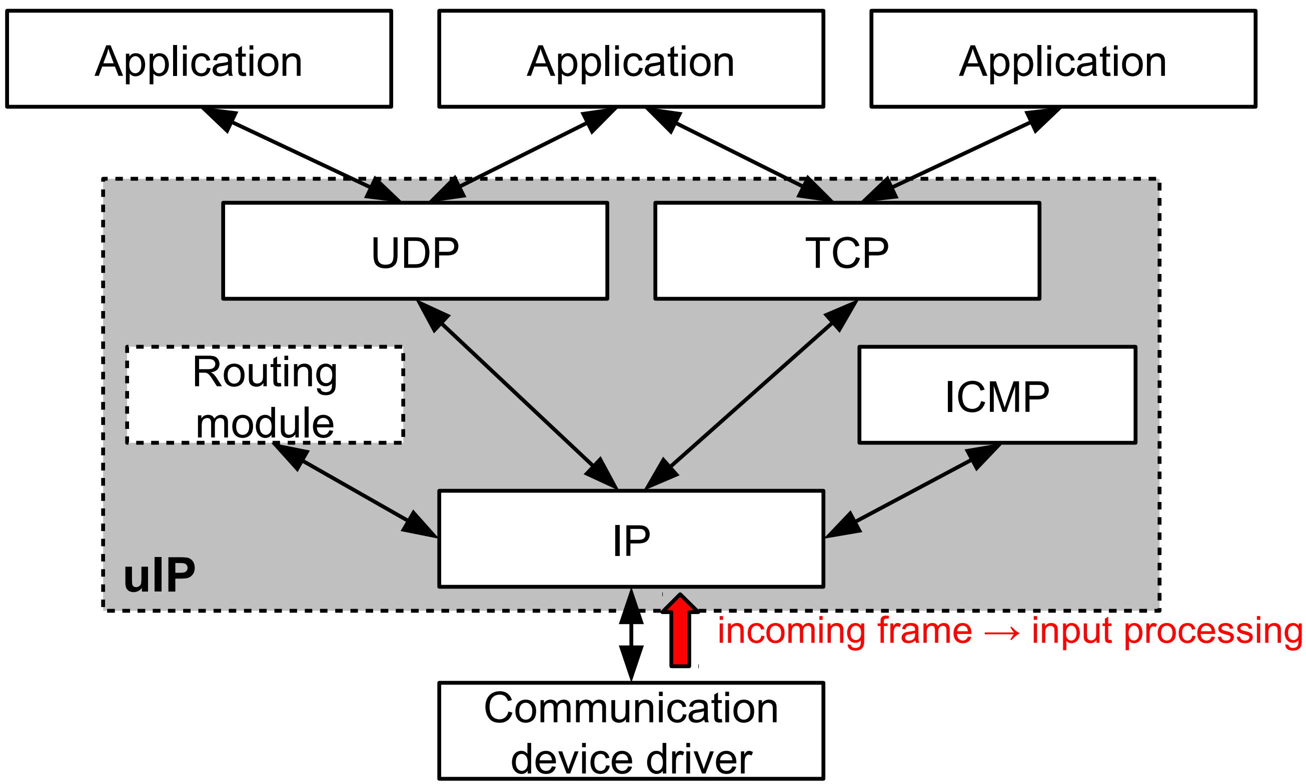
- **Introduction**
  - Long belief: IP too complex and heavyweight to be used in small networked embedded systems
    - MCU has limited memory/processing resources
  - Most control networks use proprietary protocols
    - CAN, Profibus, Modbus, X10, LonTalk, ...
    - Specific gateway required for Internet connectivity
  - Belief changed in 2001 with 1st version of a working prototype of a complete IP stack for an 8-bit MCU
    - memory footprint[1] as low as 10 KB flash / 1 KB RAM
    - includes IP, ICMP, UDP and TCP

(1) exact size depends on platform and requirement for complete standard compliance.

Reference: **"Full TCP/IP for 8-Bit Architectures"**, Adam Dunkels, Mobisys, 2003

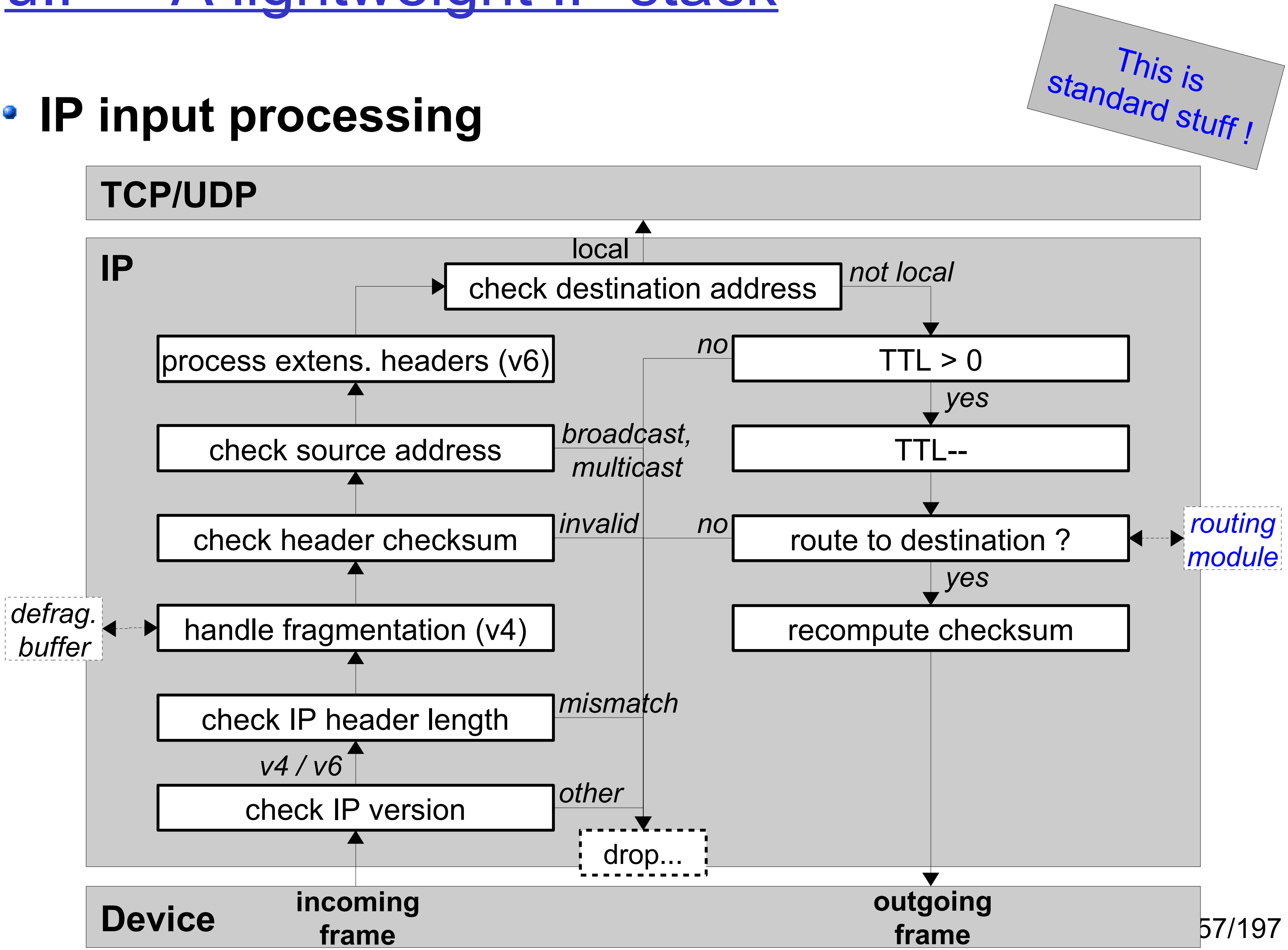
uIP – A lightweight IP stack

- **Principles of operation**
  - Processes frames from comm. device driver
  - Processes packets from application(s)
  - Does periodic processing (e.g. timers for retransmissions)



uIP – A lightweight IP stack

- **IP input processing**

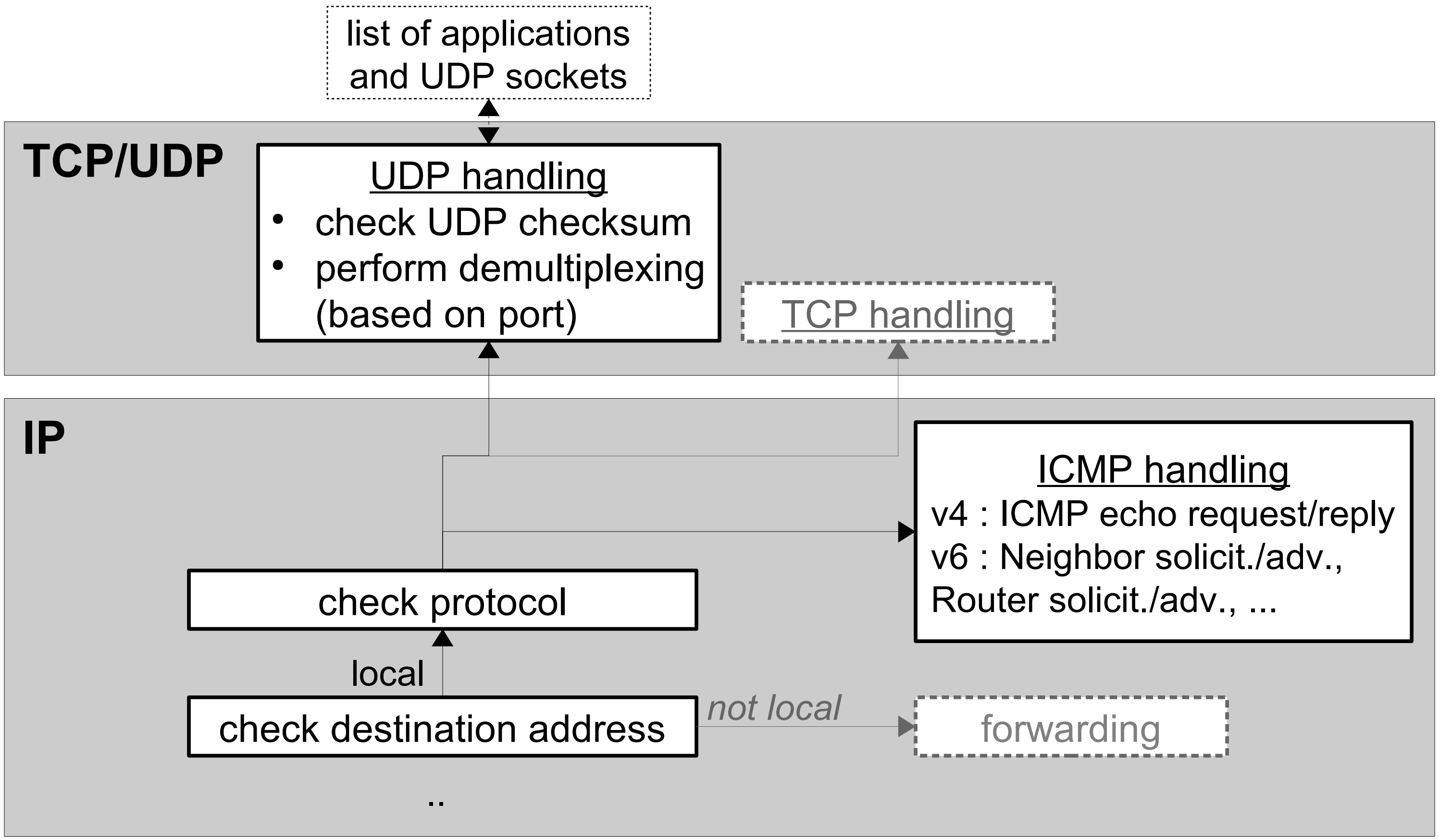


uIP – A lightweight IP stack

- **Routing module**
  - **uIP** does not mandate a particular routing mechanism
  - for every IP datagram to be forwarded, route lookup is delegated to **routing module**
  - **uIP** can therefore be associated with any possible routing mechanism (specified at compile time)
  - forwarding table can be destination table, prefix table/tree, hash table, cache with recent results of on-demand routing, ...

uIP – A lightweight IP stack

- **Above layer protocols**



uIP – A lightweight IP stack

Differs from BSD UNIX stack !

- **TCP input processing**
  - check TCP checksum
  - check (src/dst IP, src/dst port) against list of active connections
    - check seq. # against expected seq. #. If they differ, **drop segment** (not enough memory to buffer un-ordered segments + sender will retransmit later).
    - update RTT estimate
    - act according to state of TCP FSM
  - if no corresponding active connection and segment has SYN flag (alone), check against list of listening ports
    - remember sender's initial sequence number
    - remember sender's MSS option (if present)
    - send a TCP segment with SYN and ACK flags

uIP – A lightweight IP stack

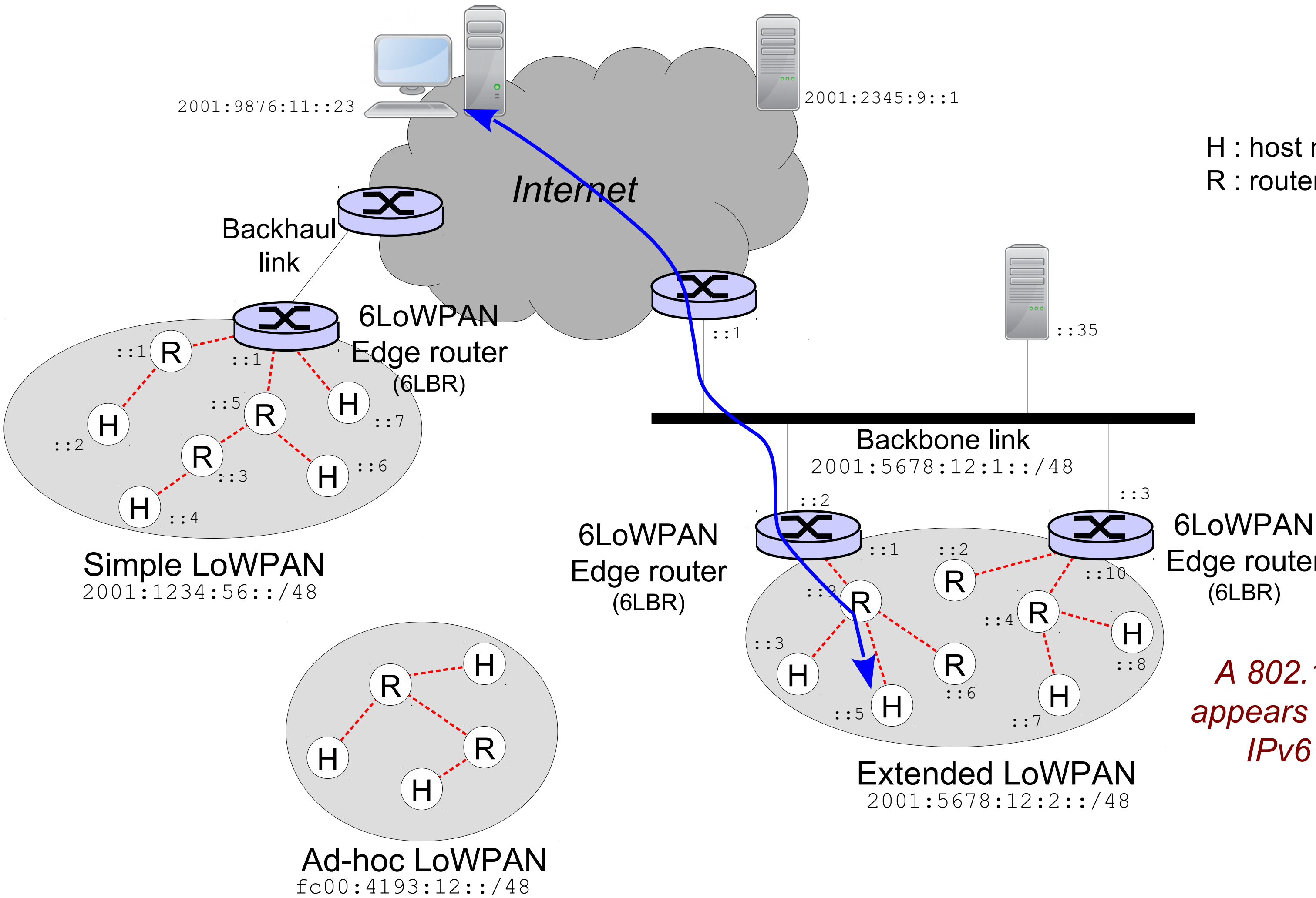
- **TCP processing**
  - Recall : TCP's sliding window mechanism allows to pipeline multiple packets at a time and improve efficiency
    - Throughput ≈ W/RTT
  - Drawback : a sliding window requires a lot of memory !
  - **uIP** does not use a sliding window
    - will send a single unack'd segment at a time (stop and wait)
    - does not affect interoperability or standards compliance
    - affects efficiency (max. achievable throughput ~1MSS/RTT)

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
  - uIP
  - 6LoWPAN
- 4.8 Programming Model / RTOS

6LoWPAN

- **Architecture**



A 802.15.4 PAN appears as a single IPv6 subnet

Inspired from ***6LoWPAN : The Wireless Embedded Internet***, Z. Shelby and C. Bormann, Wiley, 2009

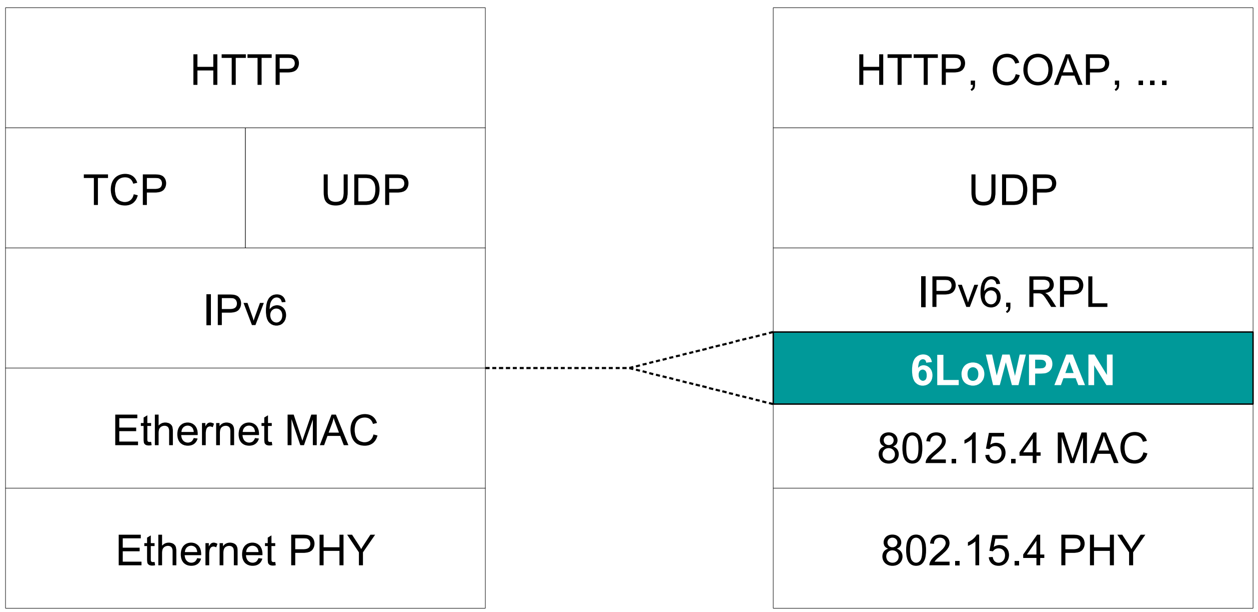
6LoWPAN

- **How to efficiently support IPv6 on WSNs ?**
  - More generally on *Low-Power Lossy Networks*[1]
  - Saving power → duty-cycle
    - IP assumes always ON links
  - Limited data rate ~ 100kbps[2]
  - Limited frame size ~ 100 bytes[2]
    - IPv4 assumes MTU ≥ 576 bytes
    - IPv6 assumes MTU ≥ 1280 bytes
  - Low reliability
    - wireless links, node failures, duty-cycling
  - Multi-hop network
    - IPv6 neighbor discovery assumes link = single broadcast domain

(1) Wireless, Power Line Control (PLC), ...
(2) Recall IEEE 802.15.4 : data rate = 250kbps, frame size = 127 bytes

6LoWPAN

- **Adaptation layer**
  - IETF standard
  - RFC4944, Sept. 2007 ; RFC6282, Sept. 2011
  - focuses on IPv6 only

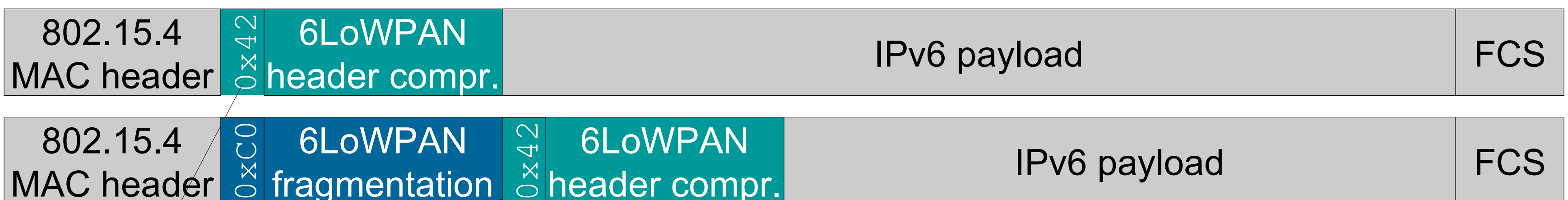


6LoWPAN

- **Services of adaptation layer**
  - Header compression
    - omit from IP/UDP headers fields that can be *inferred from the MAC header*, are *common values*, or can be *derived from a shared context*
  - Fragmentation
    - IEEE 802.15.4 frame length = 127 bytes
    - perform fragmentation and reassembly below IP layer
  - Stateless auto-configuration
    - Helps IPv6 Neighbor Discovery to operate on a non broadcast domain

6LoWPAN

- **Header format**
  - Approach = stacked sub-headers
  - Different sub-headers
    - Mesh addressing
    - Fragmentation
    - Header compression

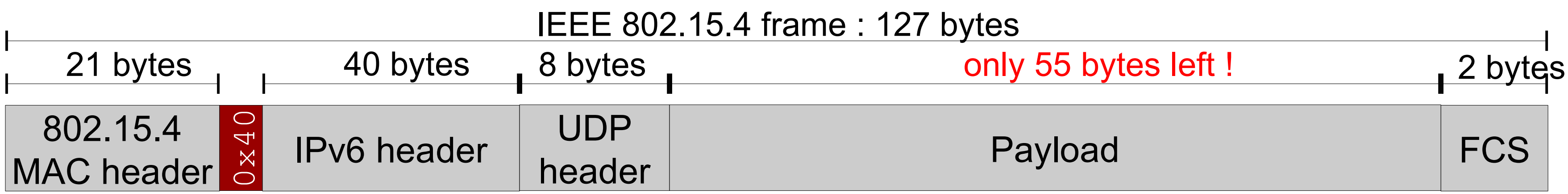


dispatch bytes

Value	Description
00000000-0011111	Not a LoWPAN frame
01000000	Uncompressed IPv6 datagram
01000010	HC1 Compressed IPv6
10000000-10111111	Mesh header
11000000-11000111	First fragmentation header
11100000-11100111	Subsequent fragmentation header

6LoWPAN

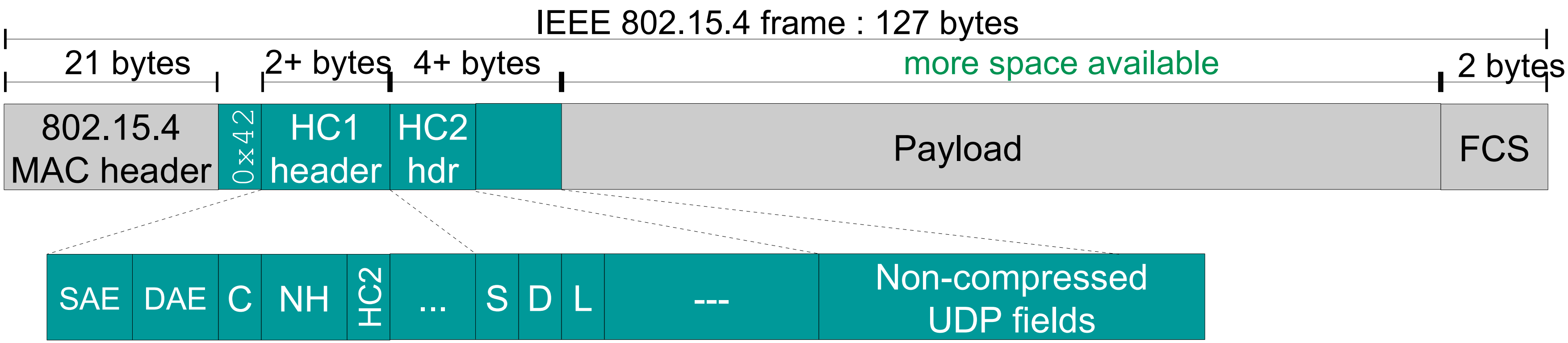
- **Header compression (HC1, HC2)**
  - IPv6 and transport (UDP/TCP/ICMP) headers take too much spacein 802.15.4 frame



  - **Principle: Optimize for the most common cases**
  - Header fields that can be derived from context are **omitted**
  - Other fields are sent unmodified
  - **Stateless compression**: nodes do not have to maintain compression staten

6LoWPAN

- **Header compression (HC1, HC2)**



- HC1 : IPv6 header compression

  SAE/DAE : *Source/Destination Address Encoding*
  C (*Traffic Class*) = 1 : traffic class and flow label are zero
  NH (*Next Header*) = 01 (UDP) ; 10 (ICMP) ; 11 (TCP) ; 00 (sent uncompressed)
  HC2=1 : HC2 header (UDP)
  hop-limit not compressed
  length derived from 802.15.4 frame length

- HC2 : UDP header compression

  S/D/L : source/destination port, length of UDP datagram = 1 : compressed
  checksum not compressed

6LoWPAN

- **Header compression (HC1, HC2)**
  - Common cases for address compression

Prefix	Interface ID

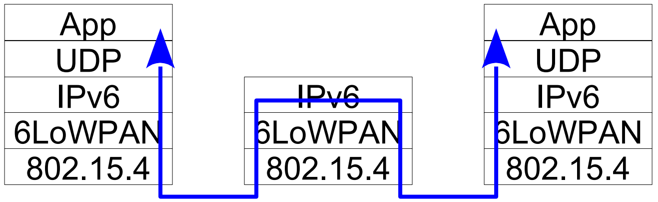
    - link-local addresses → prefix = `FE80::/64`
    - IPv6 addresses derived from MAC (EUI-64) → Interface ID already known from MAC header

SAE / DAE	Prefix	Interface ID	Size req.
00	uncompressed	uncompressed	16
01	uncompressed	derived from MAC	8
10	link-local	uncompressed	8
11	link-local	derived from MAC	0

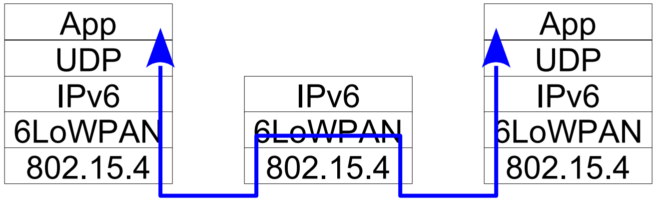
  - Other compression schemes than HC1 exist to compress the prefix part (shared context)

6LoWPAN

- **Fragmentation**
  - Used to send larger IPv6 datagrams over multiple 802.15.4 frames
  - Fragmentation header contains data required to allow reassembly
- **Routing in a 6LoWPAN**
  - Route-over
    - routing at the IP layer, using RPL
    - fragmentation and reassembly at each hop



  - Mesh-under
    - routing at the data-link layer (actually in the 6LoWPAN adaption layer)
    - requires use of mesh header
    - fragmentation and reassembly only at end hosts



Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS
  - TinyOS
  - Contiki

WSN Programming Model

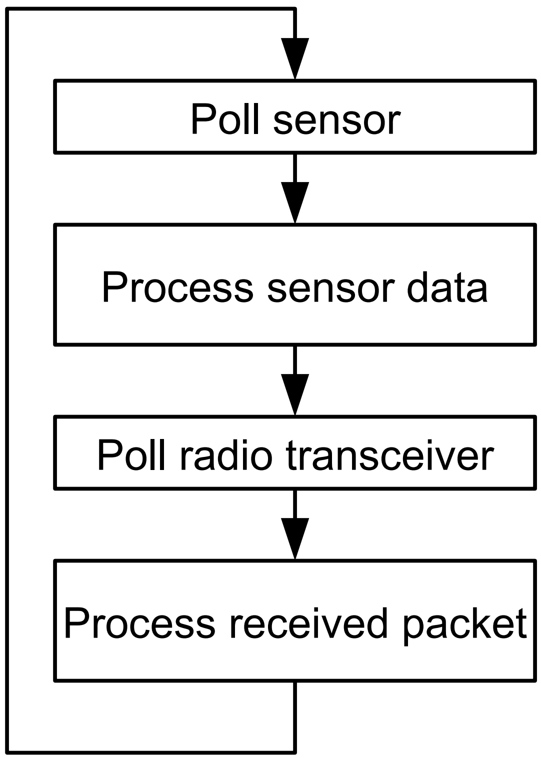
- **Objectives**
  - Understand how networked embedded systems programming differ from regular “application programming”
    - not much OS support
    - sequential vs event-driven
  - Get a touch of recent WSN-oriented RTOS
    - TinyOS, ContikiOS, OpenWSN, RIOT-OS, ...
  - For a recent survey, see e.g.
    - ***Operating Systems for Wireless Sensor Networks : A Survey***, Muhammad Omer Farooq and Thomas Kunz, MDPI Sensors 11(6), 2011

WSN Programming Model

- **Operating System support**
  - Traditional application programming heavily relies on services provided by an Operating System (OS)
    - control/protection of access to resources
    - management of resources allocation to different users/processes
    - support for concurrent execution of multiple processes
    - communication between processes
  - This is different for WSN nodes
    - Microcontrollers usually do not have resources for a full-blown OS (no MMU, limited memory, ...)
    - Concurrency requirements are different : single user, limited number of tasks
    - Scarce memory often requires static allocation or ad-hoc dynamic allocation schemes

WSN Programming Model

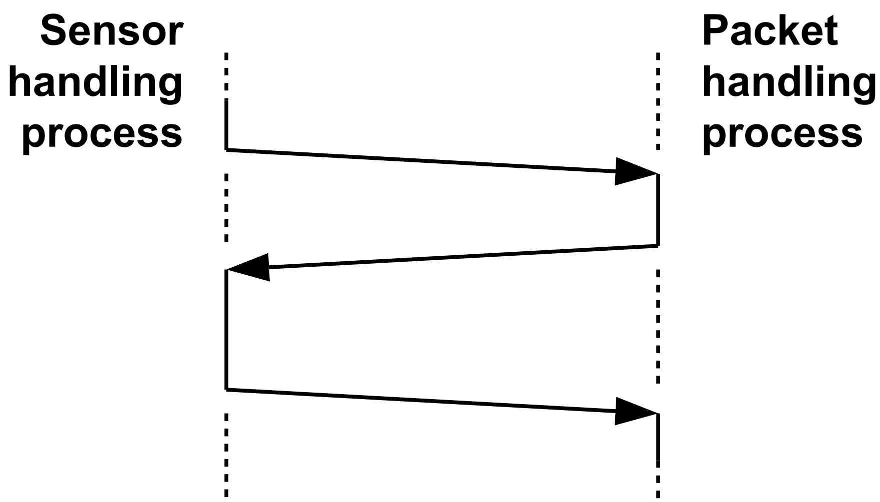
- **Sequential Programming**
  - Traditional applications are often designed around a sequential programming model
    - This model could lead to either sensor data or radio packet losses (the radio transceiver has limited buffering capability)



too many radio packets received during these phases could lead to losses

WSN Programming Model

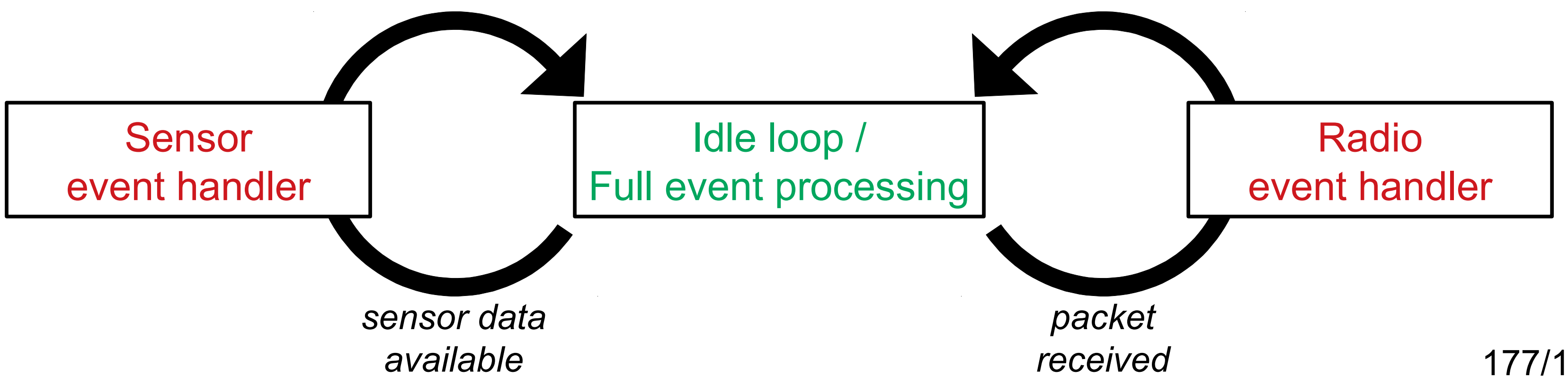
- **Process-based Concurrency**
  - OS takes care of CPU sharing and provides a “parallel” execution environment



  - Issues
    - each process has its own stack
      → not appropriate for memory constrained devices
    - context-switching induces significant overhead
      → CPU/energy consumption + risk of missing sensor data / radio packet

WSN Programming Model

- **Event-based Programming**
  - Introduce reactive nature of WSN node into the programming model
  - Operations Principles
    - Idle loop
      - do nothing / low priority processing
      - go to sleep when adequate
    - Event handlers
      - quickly process incoming events
      - e.g. sensor data available, packet has arrived, ...
      - store required information to later fully process the event



WSN Programming Model

- **Event-based Programming**
  - Operations Principles
    - An event handler can interrupt any normal code
    - An event handler cannot interrupt another event handler
      - would require costly context-switch (saving stack + registers)
    - Event handler run to completion
      - these are short pieces of code
  - There are thus **only 2 execution contexts**
    - one for time-critical event handlers (no interrupt)
    - one for “normal code” (can be interrupted)

Note: some modern micro-controllers (e.g. some ARM and MIPS MCU) offer *shadow registers* to allow for efficiently switching between only 2 contexts.

Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS
  - TinyOS
  - Contiki

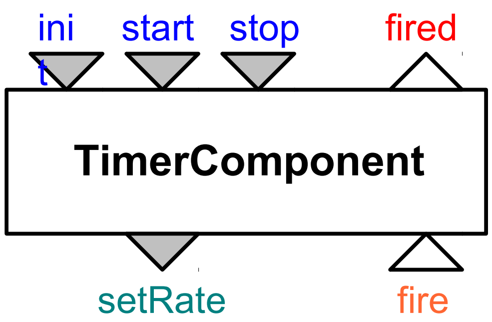
WSN Programming Model

- **Case Study : TinyOS**
  - Features
    - Framework for event-based programming (event-based paradigm with only 2 contexts)
    - Hide complexity of managing multiple communicating state-machines
    - Allow modular design (e.g. replace one state machine with another)
    - Extension to C language : nesC dialect
  - http://www.tinyos.net/

Reference: **System Architecture Directions for Networked Sensors**, J. Hill, M. Horton, R. Kling and L. Krishnamurthy, In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, 2000

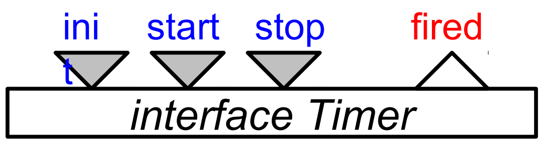
WSN Programming Model

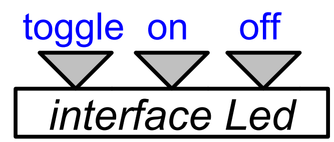
- **TinyOS : The central concept of <u>Component</u>**
  - <u>Interface</u>
    - Can handle commands
    - Can issue commands
    - Can handle events
    - Can fire events
    - Command/event handlers must run to conclusion
  - <u>Tasks</u>
    - must run to conclusion, can be interrupted by handlers
    - tasks are atomic to each other
    - tasks are triggered by event handlers
    - scheduling of tasks done with a power-aware FIFO scheduler
      - (shuts down the node when there is no task to execute)



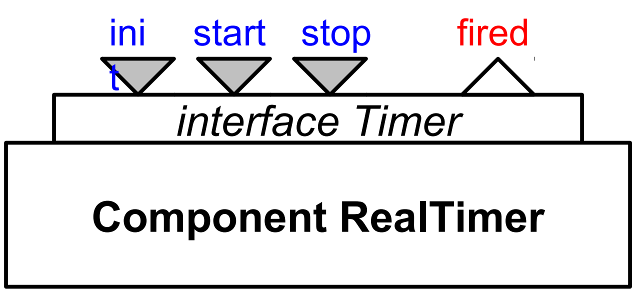
WSN Programming Model

- **TinyOS : Component interfaces**
  - Events/commands can be grouped into <u>interfaces</u>.

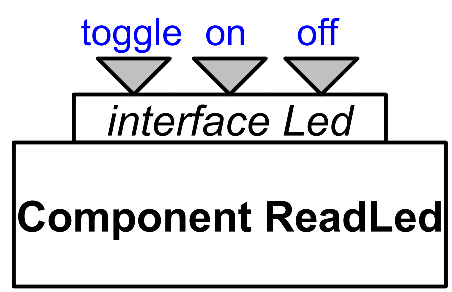




  - Components *provide* and/or *use* interfaces
    - a single component can use/provide multiple interfaces
    - components can have their interfaces wired together



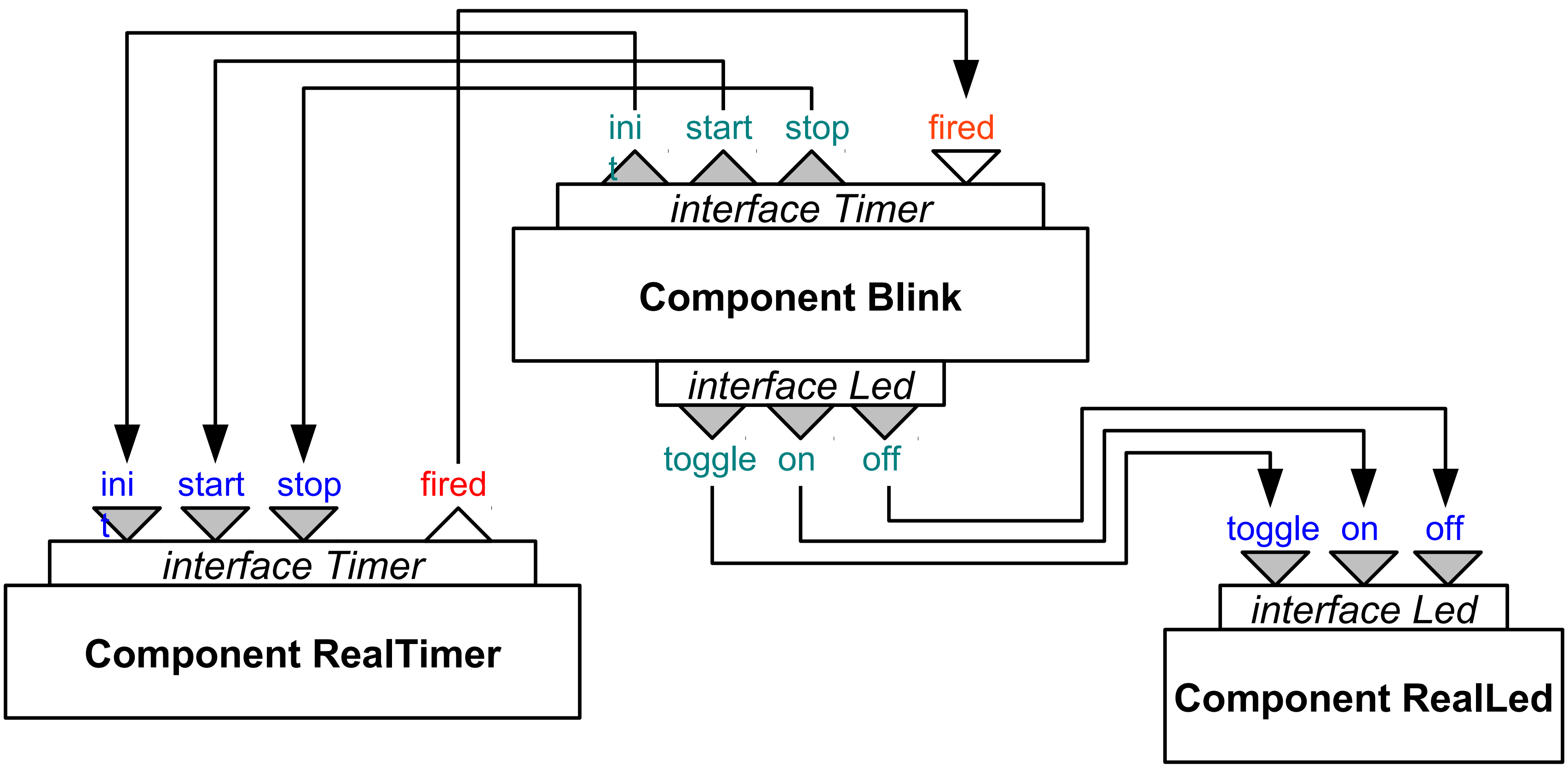
RealTimer provides interface *Timer*



RealLed provides interface *Led*

WSN Programming Model

- **TinyOS : Wiring Components**



WSN Programming Model

- **TinyOS : nesC language**
  - C language extension that allows to define components, interfaces, ... and wire them together
  - Compiled to C

```
module BlinkC {
  uses interface Timer;
  uses interface Led;
}
implementation {
  event void Timer.fired() {
    call Led.toggle();
  }
}
```

when the timer event is fired the state of the LED is toggled

```
configuration BlinkAppC {
}
implementation {
  components BlinkC, RealLedC;
  components RealTimerC;

  BlinkC.Timer → RealTimerC.Timer;
  BlinkC.Led → RealLed.Led;
}
```

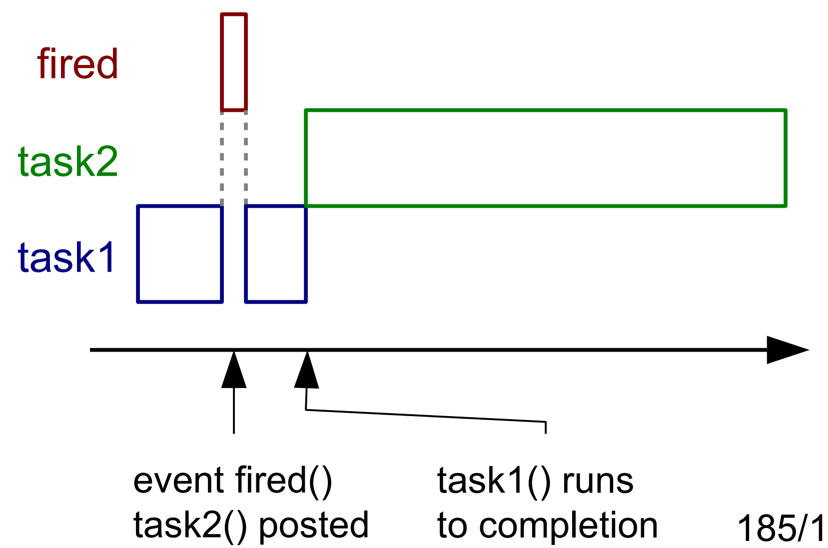
```
interface Led {
  async command void on();
  async command void off();
  async command void toggle();
}
```

```
interface Timer {
  command void init();
  command void start(uint32_t p);
  command void stop();
  event void fired();
}
```

WSN Programming Model

- **TinyOS : Execution Model**
  - Concept of task
    - code that runs to completion (can be interrupted by events)
    - tasks are *posted* by events/commands (put them in the scheduler's queue)
    - tasks handle computation that can be deferred to later (similar to interrupt bottom halves / deferred procedure calls)

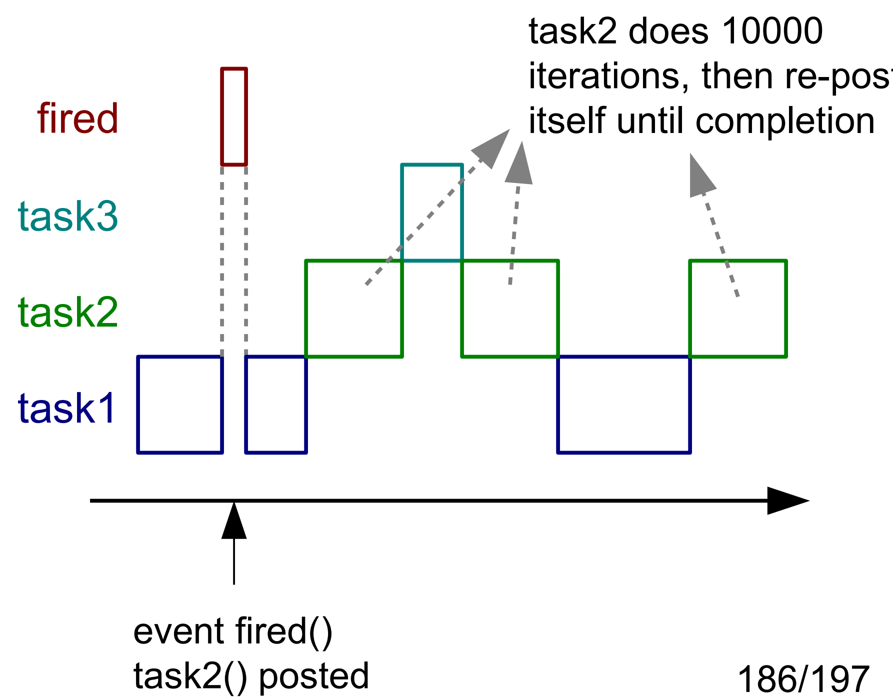
```
task void task2 {
  for (int i= 0; i < 1000000; i++) {
    /* ... heavy computation ... */
  }
}

event void fired() {
  post task2();
}
```



WSN Programming Model

- **TinyOS : Execution Model**
  - If a task does really heavy computation, it might be interesting to split it in smaller parts to give opportunity to other tasks to be executed → **responsibility of task itself**

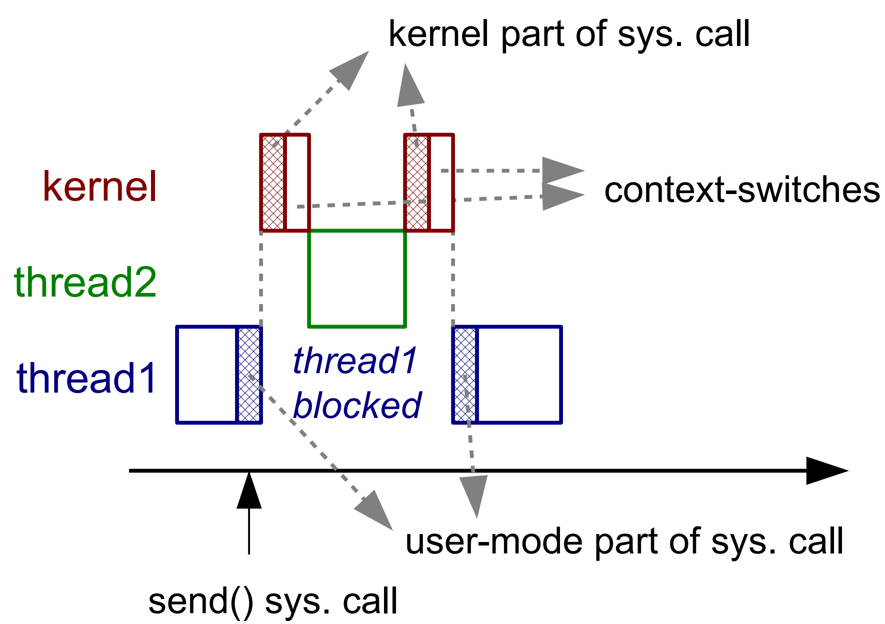
```
task void task2 {
  static int i= 0;
  int start= i;
  for (; (i < start+10000) &&
         (i < 1000000); i++) {
    /* ... heavy computation ... */
  }

  if (i >= 1000000)
    i= 0;
  } else {
    post task2();
  }
}

event void fired() {
  post task2();
}
```



WSN Programming Model

- **TinyOS : Split-Phase Programming**
  - Most operating systems support long operations through blocking system calls
    - calling process / thread put to sleep until operation done
    - context-switch required → requires CPU resources :(
    - each process / thread keeps its own stack → requires memory resources :(

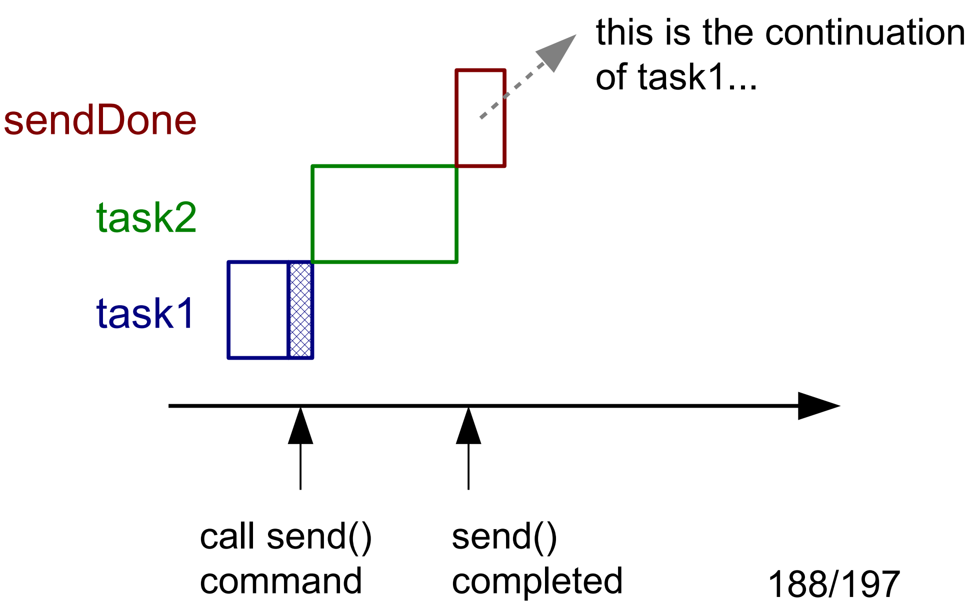
```
error_t err= send();
if (err == SUCCESS)
  ...
else
 ...
```



WSN Programming Model

- **TinyOS : Split-Phase Programming**
  - TinyOS does not provide blocking system calls (to avoid complex and costly context switches).
    - Long operations have to be implemented in **multiple phases**
    - Similar to the use of callback functions BUT much more efficient as wiring is static (known at compile time)

```
/* first phase */
send();

/* second phase */
event void sendDone(error_t err) {
  if (err == SUCCESS)
    ...
  else
    ...
}
```



Wireless Sensor Networks

- 4. 1 Motivations
- 4.2 Sensor Node
- 4.3 Network Architecture
- 4.4 MAC Layer
- 4.5 Network Layer
- 4.6 Transport and Application Layers
- 4.7 IP for WSN ?
- 4.8 Programming Model / RTOS
  - TinyOS
  - Contiki

WSN Programming Model

- **Contiki**
  - Another open-source operating system for networked embedded systems
  - Shares similarities with TinyOS in term of features
  - Uses a different approach to deal with the event-driven programming paradigm: prototthreads

  - http://www.sics.se/contiki

Reference: **Contiki - a lightweight and flexible operating system for tiny networked sensors**, A. Dunkels, B. Grönvall and Th. Voigt. In Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA, November 2004.

WSN Programming Model

- **Event-driven** versus **Multi-threading**

	Event-driven	Multi-threading
Blocking	Not possible	Possible
Preemption	Not possible	Possible
Code complexity	State machine (logic harder to grasp)	Sequential code (easier to understand)
Code compactness	Compact	Larger code overhead (thread management)
Memory efficiency	efficient (e.g. FSM’s state)	Less efficient (each thread has its own stack)
Locking	usually not a problem	Problematic (dead-locks)

WSN Programming Model

- **Contiki : Protothreads**
  - A simple mechanism to write event-driven programs as if they were based on OS multi-threading support (similar to *stackless coroutines*)
  - Rely on simple primitives
    - `PT_INIT(pt)` : initialize protothread
    - `PT_BEGIN(pt)` : delimit start of protothread
    - `PT_END(pt)` : delimit end of protothread
    - `PT_WAIT_UNTIL(pt, cond)` : wait until condition is true (implements pseudo-blocking condition)
    - `PT_YIELD(pt)` : immediately returns control to scheduler (implements unconditional blocking)
    - ...

Reference: **Protothreads: Simplifying event-driven programming of memory-constrained embedded systems**, A. Dunkels, O. Schmidt, Th. Voigt and M. Ali, In Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), November 2006

WSN Programming Model

- **Contiki : Protothreads**
  - Details: primitives implemented as C macros
    - using *labels as values*, a `gcc` feature, (http://gcc.gnu.org/onlinedocs/gcc/Labels-as-Values.html)
    - or using a `switch` statement

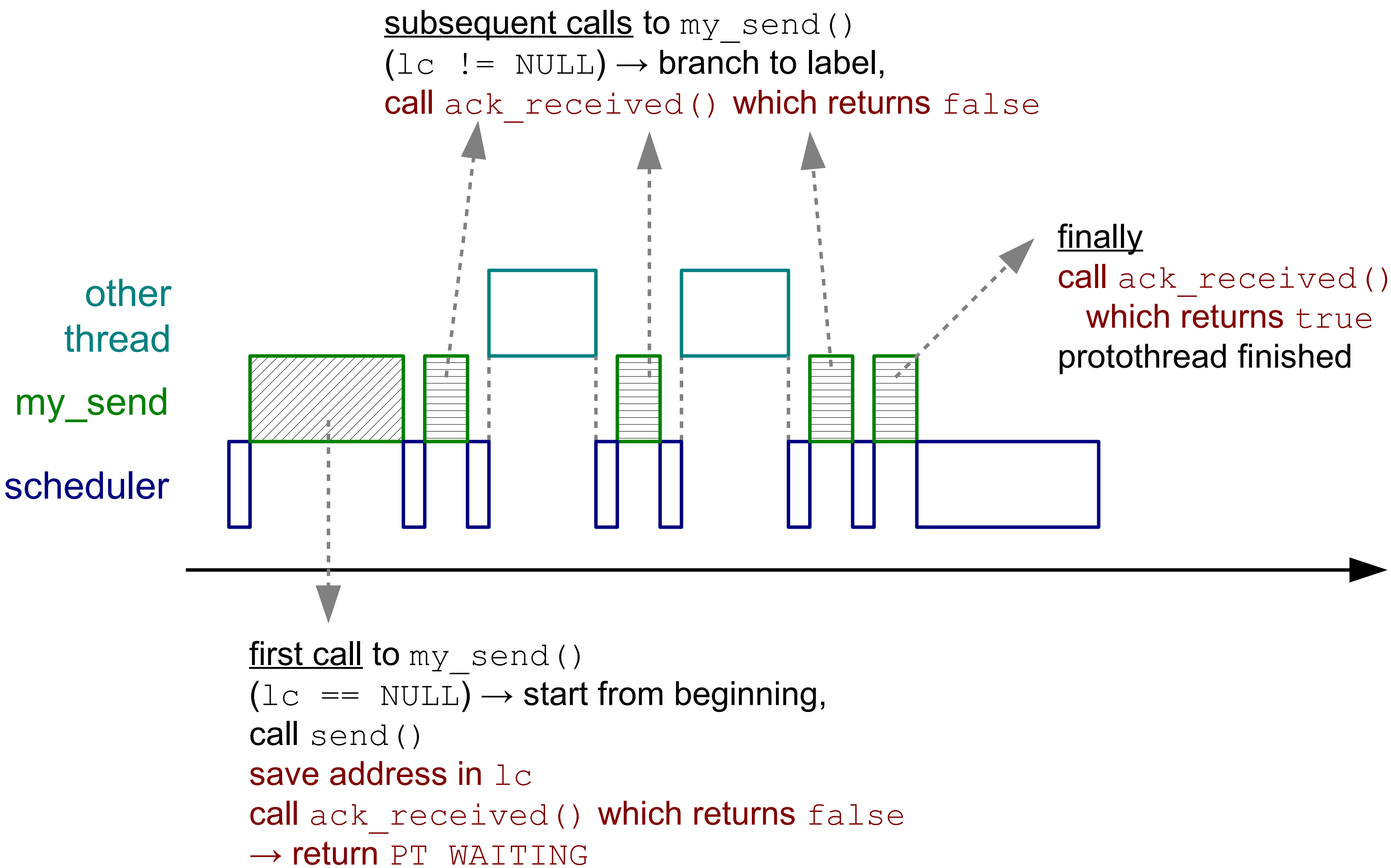
```
PT_THREAD(my_send(struct pt * pt,
                  struct msg * msg))
{
  PT_BEGIN(pt);
  send(msg);
  PT_WAIT_UNTIL(pt, ack_received());
  PT_END(pt);
}
```

translated by
C preprocessor

```
char my_send(struct pt * pt,
             struct msg * msg)
{
  if (pt->lc != NULL)
    goto *(pt->lc);
  send(msg);
  {
    __label__ r;
  r: pt->lc= &&r;
  }
  if (!(ack_received()))
    return PT_WAITING;
  return PT_ENDED;
}
```

WSN Programming Model

- **Contiki : Protothreads**



Wireless Sensor Networks

- **Conclusion**
  - WSN requirements / environments challenge many assumptions of classical IP networks
    - **Stringent resource constraints** (energy, CPU, memory)
      → duty-cycling, special MAC protocols
    - **Lossy links, frequent topology changes**
      → scalable MAC and routing protocols
      → adaptations to transport protocol (e.g. TCP)
    - **No OS support, Event-driven environment** (data, radio)
      → new programming paradigms
      → IP still feasible (uIP, and 6LOWPAN)

References

- ***Protocols and Architectures for Wireless Sensor Networks***, H. Karl and A. Willing, Wiley, 2005.
- ***Wireless Sensor Networks: A Networking Perspective***, J. Zheng and A. Jamalipour, 2009.
- ***Applicative Study of Wireless Sensor Networks***, UMONS/polytech course, S. Bette and V. Moeyeart, 2011.
- ***Interconnecting Smart Objects with IP – The Next Internet***, J.-P. Vasseur and A. Dunkels, Morgan-Kaufmann, 2010.
- ***Contiki - a lightweight and flexible operating system for tiny networked sensors***, A. Dunkels, B. Grönvall and Th. Voigt. In Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA, November 2004.
- ***System Architecture Directions for Networked Sensors***, J. Hill, M. Horton, R. Kling and L. Krishnamurthy, In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.
- ***Full TCP/IP for 8-Bit Architectures***, Adam Dunkels, In Proceedings of Mobisys, 2003.
- ***I-TCP: Indirect TCP for Mobile Hosts***, A. Bakre and B. R. Badrinath, In Proceeding of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95), 1995.
- ***An Energy-Efficient MAC Protocol for Wireless Sensor Networks***, W. Ye, J. Heidemann, and D. Estrin, Proceedings of INFOCOM 2002, IEEE Press, June 2002.

References

- ***Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks***, W. Ye, J. Heidemann, and D. Estrin, IEEE/ACM Transactions on Networking, Volume 12, issue 3, 2004.
- ***Distributed TCP Caching for Wireless Sensor Networks***, A. Dunkels, J. Alonso, Th. Voigt and H. Ritter, Proceedings of the 3rd Annual Mediterranean Ad-Hoc Networks Workshop, 2004.
- ***6LoWPAN – The Wireless Embedded Internet***, Z. Shelby and C. Bormann, Wiley Series in Communications Networking and Distributed Systems, 2009.
- ***802.15.4a-2007 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirement Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)***, IEEE, 2007.
- ***Transmission of IPv6 Packets over IEEE 802.15.4 Networks***, G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, RFC 4944, Oct. 2007.
- ***The ContikiMAC Radio Duty Cycling Protocol***, A. Dunkels, Swedish Institute of Computer Science, Tech. Rep. T2011:13, Dec. 2011.