Wireless Sensor Networks
Operating systems and protocols

Abdelkader Lahmadi
ENSEM – Université de Lorraine - LORIA
2013-2014

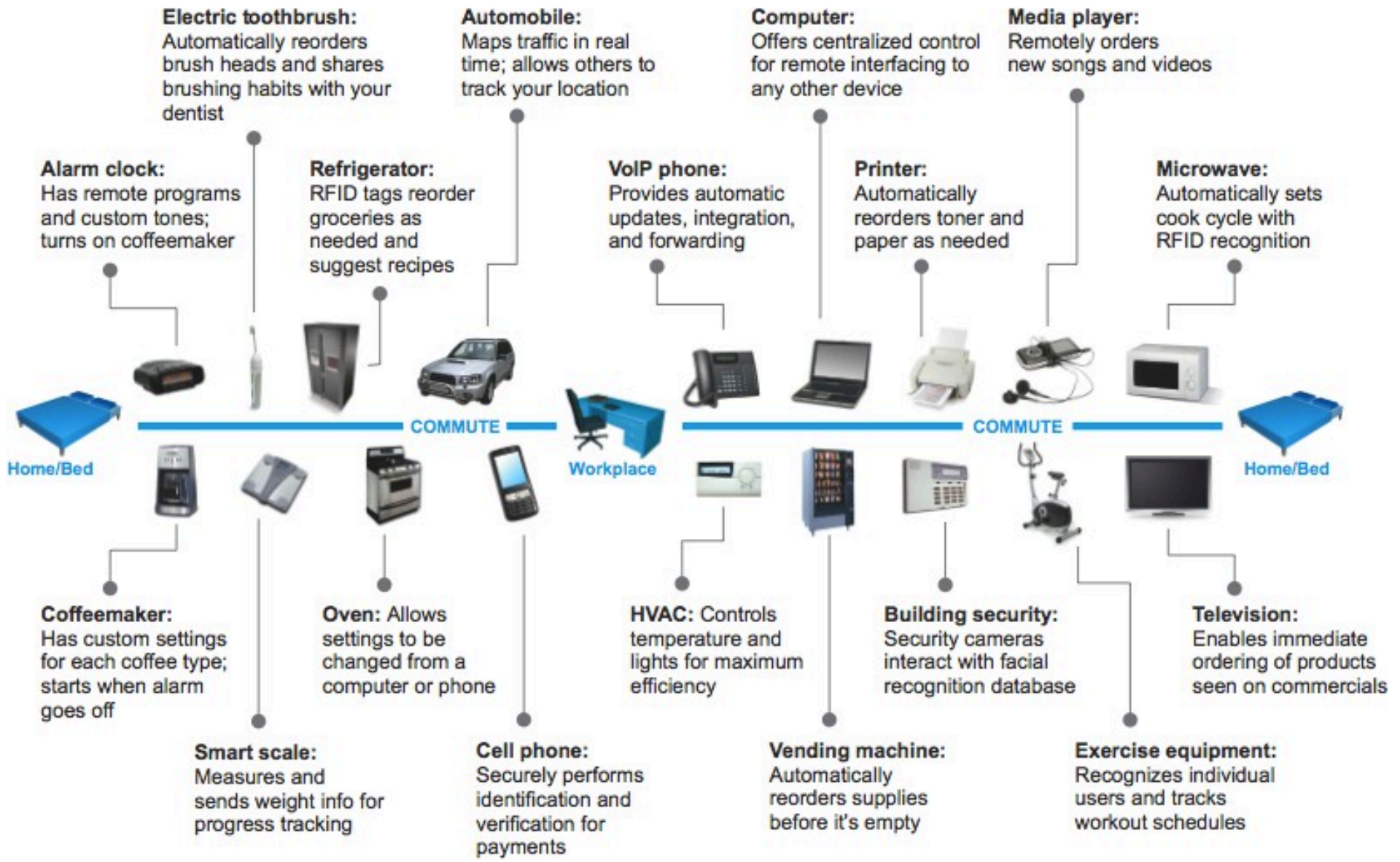
Course outline

- Introduction
- Contiki Operating system
- 6LowPAN
- RPL routing protocol

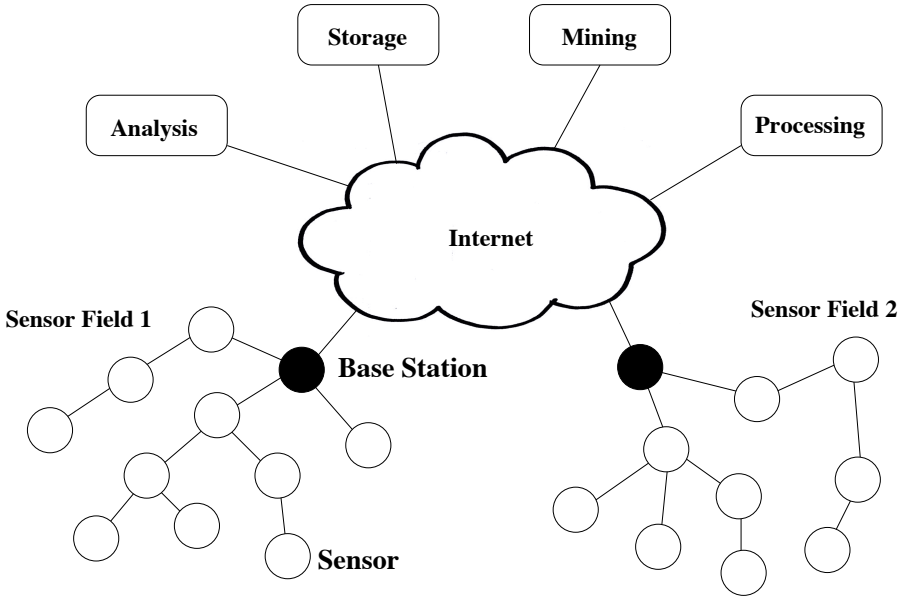
References

- Waltenegus Dargie and Christian Poellabauer, "Fundamentals of Wireless Sensor Networks: Theory and Practice" . John Wiley & Sons, August 2010.
- Gregory Pottie and William Kaiser, "Principles of Embedded Networked System Design," Cambridge University Press, 2005.
- Holger Karl and Andreas Willig, "Protocols and Architectures for Wireless Sensor Networks," John Wiley & Sons, June 2005.
- Raghavendra S. Cauligi, "Wireless Sensor Networks," Kluwer Academic Publishers, June 2004.
- Zach Shelby and Cartsen Bormann, « 6LoWPAN: The Wireless Embedded Internet», Wiley, November 2009.
- Jean-Philippe Vasseur and Adam Dunkels, « Interconnecting Smart Objects with IP: The Next Internet», Morgan Kaufmann, June 2010.

The Internet of Things



Wireless Sensor Network (WSN)



- Multiple sensors (often hundreds or thousands) form a network to cooperatively monitor large or complex physical environments
- Acquired information is wirelessly communicated to a base station (BS), which propagates the information to remote devices for storage, analysis, and processing

History of Wireless Sensor Networks

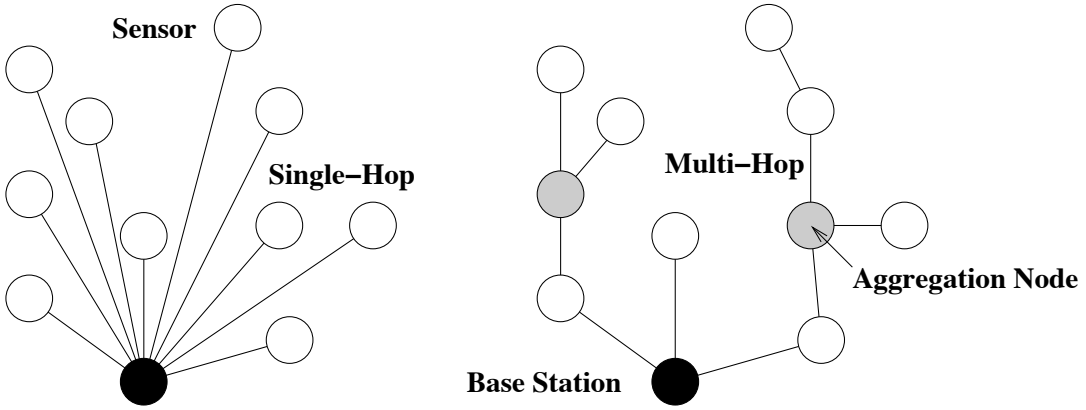
- DARPA:
  - Distributed Sensor Nets Workshop (1978)
  - Distributed Sensor Networks (DSN) program (early 1980s)
  - Sensor Information Technology (SensIT) program
- UCLA and Rockwell Science Center
  - Wireless Integrated Network Sensors (WINS)
  - Low Power Wireless Integrated Microsensor (LWIM) (1996)
- UC-Berkeley
  - Smart Dust project (1999)
  - concept of "motes": extremely small sensor nodes
- Berkeley Wireless Research Center (BWRC)
  - PicoRadio project (2000)
- MIT
  - µAMPS (micro-Adaptive Multidomain Power-aware Sensors) (2005)

WSN Communication

- Characteristics of typical WSN:
  - low data rates (comparable to dial-up modems)
  - energy-constrained sensors
- IEEE 802.11 family of standards
  - most widely used WLAN protocols for wireless communications in general
  - can be found in early sensor networks or sensors networks without stringent energy constraints
- IEEE 802.15.4 is an example for a protocol that has been designed specifically for short-range communications in WSNs
  - low data rates
  - low power consumption
  - widely used in academic and commercial WSN solutions

Single-Hop versus Multi-Hop

- Star topology:
  - every sensor communicates directly (single-hop) with the base station
  - may require large transmit powers and may be infeasible in large geographic areas
- Mesh topology
  - sensors serve as relays (forwarders) for other sensor nodes (multi-hop)
  - may reduce power consumption and allows for larger coverage
  - introduces the problem of routing



Challenges in WSNs: Energy

- Sensors typically powered through batteries
  - replace battery when depleted
  - recharge battery, e.g., using solar power
  - discard sensor node when battery depleted
- For batteries that cannot be recharged, sensor node should be able to operate during its entire mission time or until battery can be replaced
- Energy efficiency is affected by various aspects of sensor node/network design

Challenges in WSNs: Energy

- Medium access control layer:
  - contention-based strategies lead to energy-costly collisions
  - problem of idle listening
- Network layer:
  - responsible for finding energy-efficient routes
- Operating system:
  - small memory footprint and efficient task switching
- Security:
  - fast and simple algorithms for encryption, authentication, etc.
- Middleware:
  - in-network processing of sensor data can eliminate redundant data or aggregate sensor readings

Challenges in WSNs: Self-Management

- Ad-hoc deployment
  - many sensor networks are deployed “without design”
    - sensors dropped from airplanes (battlefield assessment)
    - sensors placed wherever currently needed (tracking patients in disaster zone)
    - moving sensors (robot teams exploring unknown terrain)
  - sensor node must have some or all of the following abilities
    - determine its location
    - determine identity of neighboring nodes
    - configure node parameters
    - discover route(s) to base station
    - initiate sensing responsibility

Challenges in WSNs: Self-Management

- Unattended operation
  - once deployed, WSN must operate without human intervention
  - device adapts to changes in topology, density, and traffic load
  - device adapts in response to failures

- Other terminology
  - self-organization is the ability to adapt configuration parameters based on system and environmental state
  - self-optimization is the ability to monitor and optimize the use of the limited system resources
  - self-protection is the ability recognize and protect from intrusions and attacks
  - self-healing is the ability to discover, identify, and react to network disruptions

Challenges in WSNs: Wireless Networks

- Wireless communication faces a variety of challenges
- Attenuation:
  - limits radio range

$$P_r \propto \frac{P_t}{d^2}$$

- Multi-hop communication:
  - increased latency
  - increased failure/error probability
  - complicated by use of duty cycles

Challenges in WSNs: Decentralization

- Centralized management (e.g., at the base station) of the network often not feasible to due large scale of network and energy constraints
- Therefore, decentralized (or distributed) solutions often preferred, though they may perform worse than their centralized counterparts

- Example: routing
- Centralized:
  - BS collects information from all sensor nodes
  - BS establishes "optimal" routes (e.g., in terms of energy)
  - BS informs all sensor nodes of routes
  - can be expensive, especially when the topology changes frequently
- Decentralized:
  - each sensors makes routing decisions based on limited local information
  - routes may be nonoptimal, but route establishment/management can be much cheaper

Challenges in WSNs: Design Constraints

- Many hardware and software limitations affect the overall system design

- Examples include:
  - Low processing speeds (to save energy)
  - Low storage capacities (to allow for small form factor and to save energy)
  - Lack of I/O components such as GPS receivers (reduce cost, size, energy)
  - Lack of software features such as multi-threading (reduce software complexity)

Challenges in WSNs: Security

- Sensor networks often monitor critical infrastructure or carry sensitive information, making them desirable targets for attacks

- Attacks may be facilitated by:
  - remote and unattended operation
  - wireless communication
  - lack of advanced security features due to cost, form factor, or energy

- Conventional security techniques often not feasible due to their computational, communication, and storage requirements
- As a consequence, sensor networks require new solutions for intrusion detection, encryption, key establishment and distribution, node authentication, and secrecy

Comparison

Traditional Networks	Wireless Sensor Networks
General-purpose design; serving many applications	Single-purpose design; serving one specific application
Typical primary design concerns are network performance and latencies; energy is not a primary concern	Energy is the main constraint in the design of all node and network components
Networks are designed and engineered according to plans	Deployment, network structure, and resource use are often ad-hoc (without planning)
Devices and networks operate in controlled and mild environments	Sensor networks often operate in environments with harsh conditions
Maintenance and repair are common and networks are typically easy to access	Physical access to sensor nodes is often difficult or even impossible
Component failure is addressed through maintenance and repair	Component failure is expected and addressed in the design of the network
Obtaining global network knowledge is typically feasible and centralized management is possible	Most decisions are made localized without the support of a central manager

Structural Health Monitoring

- Motivation
  - events:
    - on August 2, 2007, a highway bridge unexpectedly collapsed in Minnesota
    - nine people were killed in the event
    - potential causes: wear and tear, weather, and the weight of a nearby construction project
    - in fact, the BBC reported (August 14, 2007) that China had identified more than 6,000 bridges that were damaged or considered to be dangerous
  - these accidents motivate wireless sensor networks for monitoring bridges and similar structures

Structural Health Monitoring

- Motivation:
  - traditional inspections:
    - visual inspection ➔ everyday
      - labor-intensive, tedious, inconsistent, and subjective
    - basic inspections ➔ at least once a year
    - detailed inspection ➔ at least every five years on selected bridges
    - special inspections ➔ according to technical needs
      - the rest require sophisticated tools ➔ expensive, bulky, and power consuming

Local and Global Inspections

- ***Local inspection techniques*** focus on detecting highly localized, imperceptible fractures in a structure
  - requires:
    - a significant amount of time
    - the disruption of the normal operation of the structure
- ***Global inspection techniques*** aim to detect a damage or defect that is large enough to affect the entire structure
  - researcher have been developing and testing *wireless sensor networks as global inspection techniques*

Golden Gate Bridge (University of California)

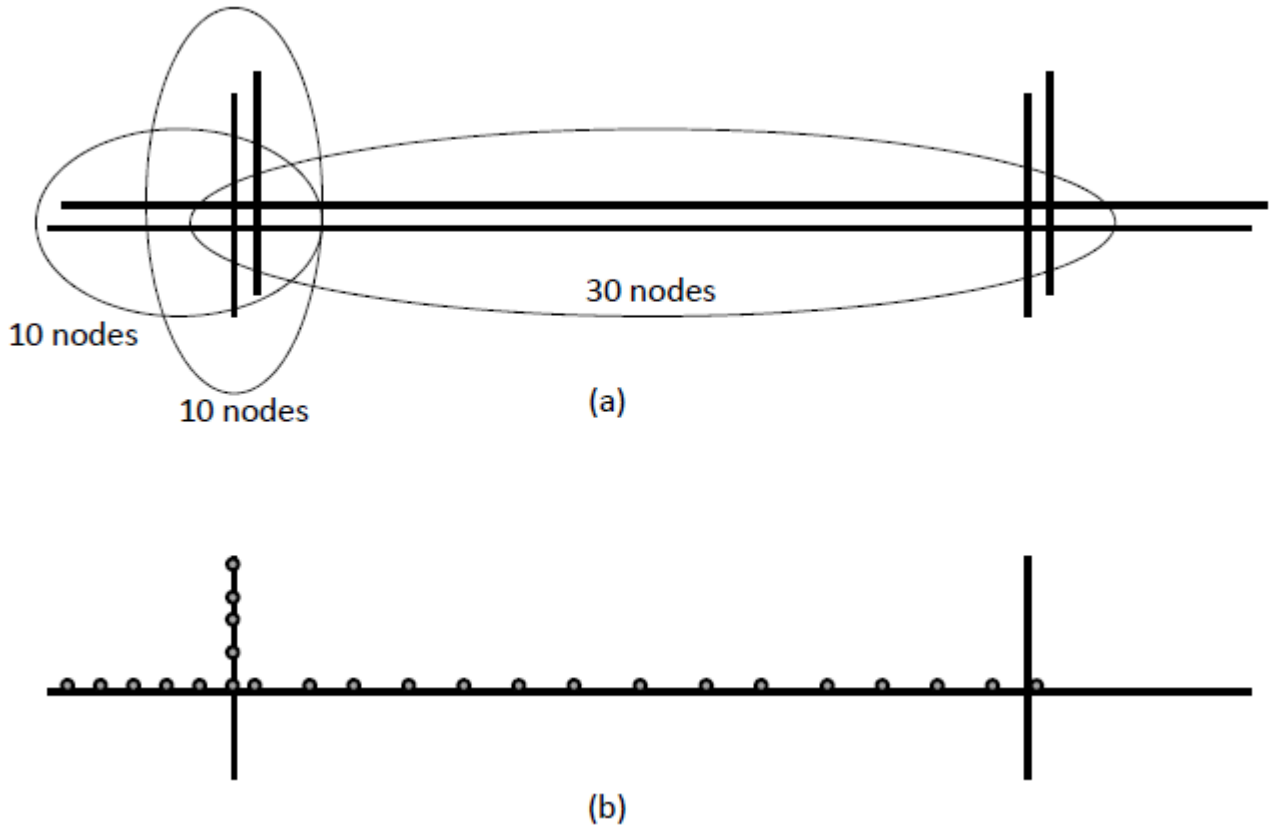


Figure 2.1 The deployment scenario on the Golden Gate Bridge

Golden Gate Bridge

- 64 wireless sensor nodes deployed on this bridge
- The network monitors ambient vibrations synchronously
  - 1 KHz rate, ≤10μs jitter, accuracy=30μG, over a 46 hop network
- The *goal* of the deployment:
  - determine the response of the structure to both ambient and extreme conditions
  - compare actual performance to design predictions
  - measure ambient structural accelerations from wind load
  - measure strong shaking from a potential earthquake
  - the installation and the monitoring was conducted without the disruption of the bridge's operation

Traffic Control

- Motivation:
  - ground transportation is a vital and a *complex* socio-economic infrastructure
  - it is *linked* with and provides *support* for *a variety of systems*, such as supply-chain, emergency response, and public health
  - the 2009 Urban Mobility Report reveals that in 2007, congestion caused urban Americans to
    - travel *4.2 billion hours more*
    - purchase *an extra 2.8 billion gallons of fuel*
  - congestion *cost is very high* - $87.2 billion; an increase of more than 50% over the previous decade

Traffic Control

- Motivation:
  - building new roads is *not* a feasible solution for many cities
    - lack of free space
    - high cost of demolition of old roads
  - *one approach*: put in place distributed systems that reduce congestions
    - *gather information* about the density, sizes, and speed of vehicles on roads
    - *infer congestions*
    - *suggest alternative routes* and emergency exits

The Sensing Task

- Inductive loops (in-road sensing devices)
  - *advantages:*
    - unaffected by weather
    - provide direct information (few ambiguity)
  - how does it work: using *Faraday's induction law*
    - a coil of wire (several meters in diameter, passes an electric current through the coil)
    - buried under the road and connected to a roadside control box
    - magnetic field strength can be induced as a result of a current and the speed and the size of passing vehicles

Magnetic Sensors

- Magnetic sensors can determine the *direction* and *speed* of a vehicle
  - a moving vehicle can disturb the distribution of the magnetic field
    - by producing its own magnetic field
    - by cutting across it
- The magnitude and direction of the disturbance depends on
  - the *speed*, *size*, *density* and *permeability* of the vehicle
- Classification of magnetic sensors:
  - *low field* (below 1μGauss)
  - *medium field* (between 1μGauss and 10μGauss)
  - *high field* (above 10μGauss)

Magnetic Sensors

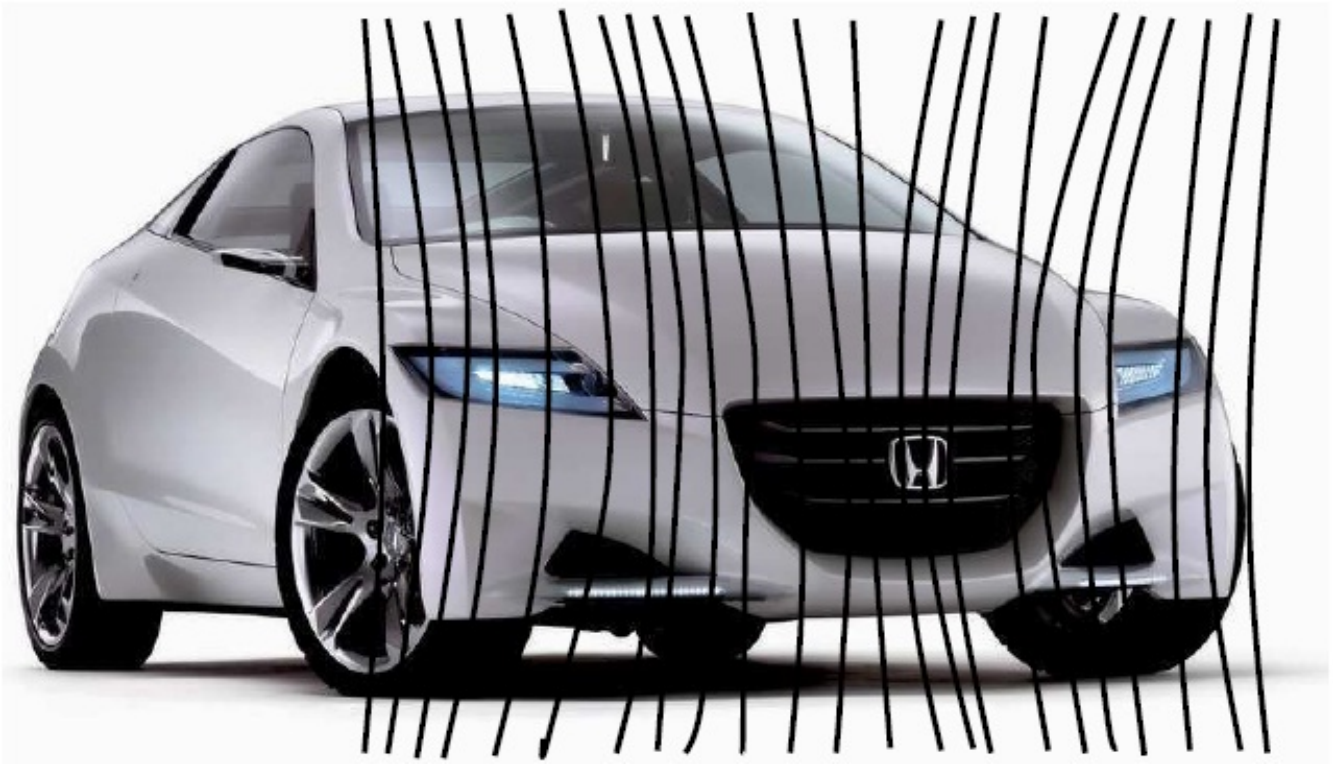


Figure 2.2 Detection of a moving vehicle with an ARM magnetic sensor (Caruso and Withanawasam 1999)

Magnetic Sensors

- Almost all road vehicles *contain* a large mass of *steel*
- The magnetic *permeability of steel is* much *higher* than the surrounding air
- *Steel* has the capacity to *concentrate* the *flux lines* of the Earth's magnetic field
- The *concentration* of magnetic flux *varies as* the *vehicle moves*; it can be *detected* from a distance of up to *15m*
- The field variation reveals a *detailed* magnetic signature
- It is possible to distinguish between different types of vehicles

Knaian (2000)

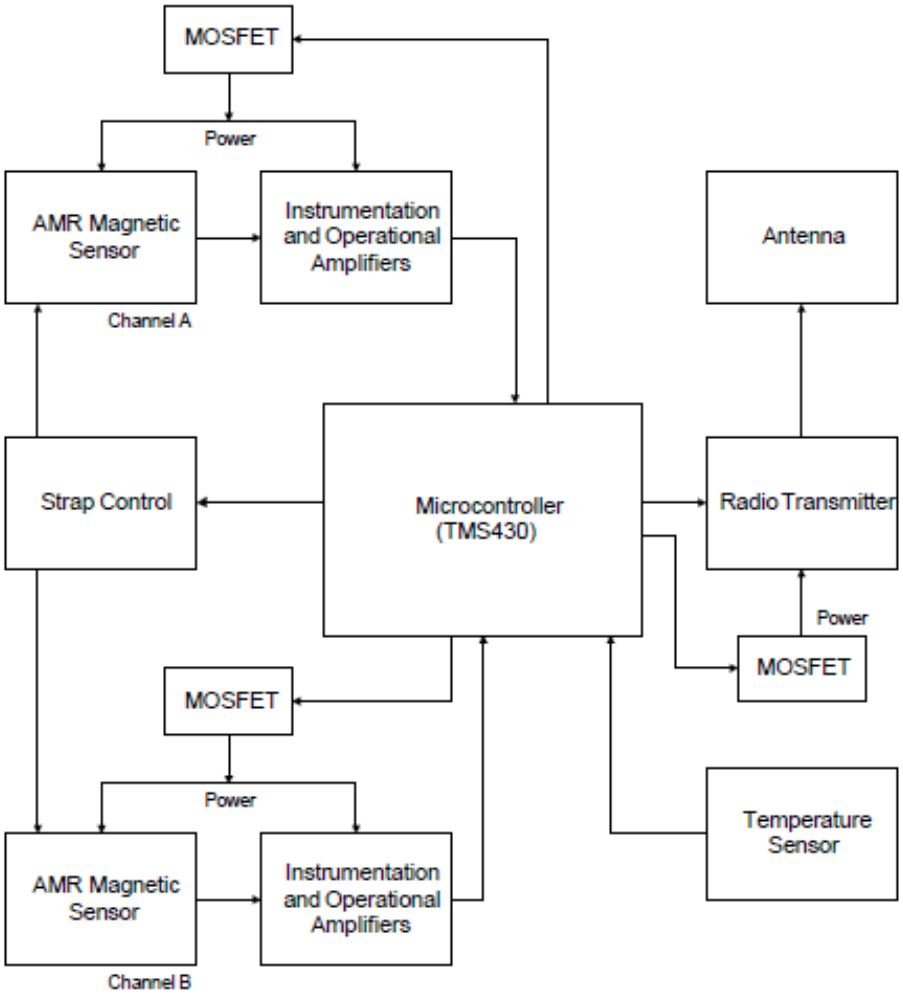


Figure 2.3 Block diagram of the MIT node for traffic monitoring (Knaian 2000)

Knaian (2000)

- Proposes wireless sensor networks for traffic monitoring in urban areas
- The node consists of
  - *two AMR magnetic sensors* to detect vehicular activities
    - by observing the disturbance in the Earth's magnetic field the vehicular creates
    - the vehicle pulls field lines away from the sensor when it approaches it
      - then towards the sensor when it drives away from it
  - *a temperature sensor* to monitor road condition (snow, ice, or water)

Knaian (2000)

- To measure the speed of a vehicle, the node waits until it detects an excursion from the predefined baseline and then starts sampling at a frequency of 2KHz
  - two AMR magnetic sensors are placed *one at the front* of the node and *the other at the back* - they are shifted in time
  - the node *waits* for the signal from the rear sensor to cross the baseline
  - then it begins to *count* the number of *samples* until the signal from the forward sensor crosses the baseline
  - from this count, it *computes* the *speed* of the passing vehicle

Arora et al. (2004)

- Deploys *90 sensor nodes* to detect the movement of vehicles and people (e.g., soldiers)
  - *78* of the nodes were *magnetic sensor nodes* that were deployed in a 60×25 square foot area
  - *12 radar sensor nodes* were overlaid on the network
- These nodes form a *self-organizing* network which connects itself to a remote computer via a base station and a long haul radio repeater

Health Care

- A wide range of health care applications have been proposed for WSN, including *monitoring patients* with:
  - Parkinson's Disease and epilepsy
  - heart patients
  - patients rehabilitating from stroke or heart attack
  - elderly people
- Health care applications do *not* function as *standalone* systems
- They are *integral parts* of a comprehensive and complex health and rescue system

Health Care

- Motivation:
  - *cost* is very high
    - according to the US Centers for Medicare and Medicaid Services (CMS):
      - the national health spending of the country in 2008 was estimated to be *$2.4 trillion USD*
      - the costs caused by heart disease and stroke are around *$394 billion*
    - this is a concern for *policy makers*, *health care providers*, *hospitals*, *insurance companies*, and *patients*
  - higher spending does *not* imply quality service or prolonged lifetime (Kulkarni and Öztürk 2007)
    - for example, in 2000, the US spent more on health care than any other country in the world – an average of $4,500 USD per person - but ranked 27th in average life expectancy
    - many countries *achieve higher* life expectancy rates *at a lower cost*

Health Care

- Motivation:
  - preventive health care - to reduce health spending and mortality rate
    - but some patients find certain practices *inconvenient, complicated,* and *interfering* with their daily life (Morris 2007)
    - many *miss* checkup visits or therapy sessions because of *a clash of schedules with* established living and working *habits*, *fear* of overexertion, or transportation *cost*

Health Care

- To deal with these problems, researchers proposed comprehensible solutions that involve the following tasks:
  - building *pervasive systems* that *provide* patients with *rich information* about diseases and their prevention mechanisms
  - seamless integration of health infrastructures with *emergency and rescue operations* as well as *transportation systems*
  - developing reliable and unobtrusive health *monitoring systems* that can be worn by patients to *reduce the task* and *presence of medical personnel*
  - *alarming* nurses and doctors *when* medical intervention is *necessary*
  - *reducing inconvenient* and *costly check-up visits* by *creating reliable links* between autonomous health monitoring systems and health institutions

Artificial Retina

- Schwiebert et al. (2001) developed a micro-sensor array that can be *implanted in the eye* as an artificial retina to assist people with visual impairments
- The system consists of *an integrated circuit* and *an array of sensors*
- An integrated circuit
  - is coated with a *biologically inert substance*
  - is *a multiplexer* with on-chip switches and pads to support a 10×10 grid of connections; it operates at 40KHz
  - has an *embedded transceiver* for wired and wireless communications
  - each *connection* in the chip interfaces a sensor through an aluminum probe surface

Artificial Retina

- An array of sensors
  - each sensor is *a micro-bump*, sufficiently *small and light*
  - the *distance* between adjacent micro-bumps is approximately *70 microns*
  - the sensors *produce electrical signals* proportional to the light reflected from an object being perceived
  - the *ganglia* and additional tissues *transform the electrical energy* into *a chemical energy*
  - the chemical energy is *transformed* into *optical signals* and *communicated to the brain* through the optical nerves
  - the *magnitude and wave shape* of the transformed energy *corresponds to* the response of *a normal retina to light stimulation*

Artificial Retina

- The system is *a full duplex system*, allowing communication in a reverse direction - the sensor array can be used for *reception* and *transmission* in a feedback loop
  - in addition to the transformation of electrical signals into optical signals
  - *neurological signals* from the ganglia can *be picked up* by the micro-sensors and *transmitted* out of the sensing system *to an external signal processor*
- Two types of wireless communications are foreseen

Artificial Retina

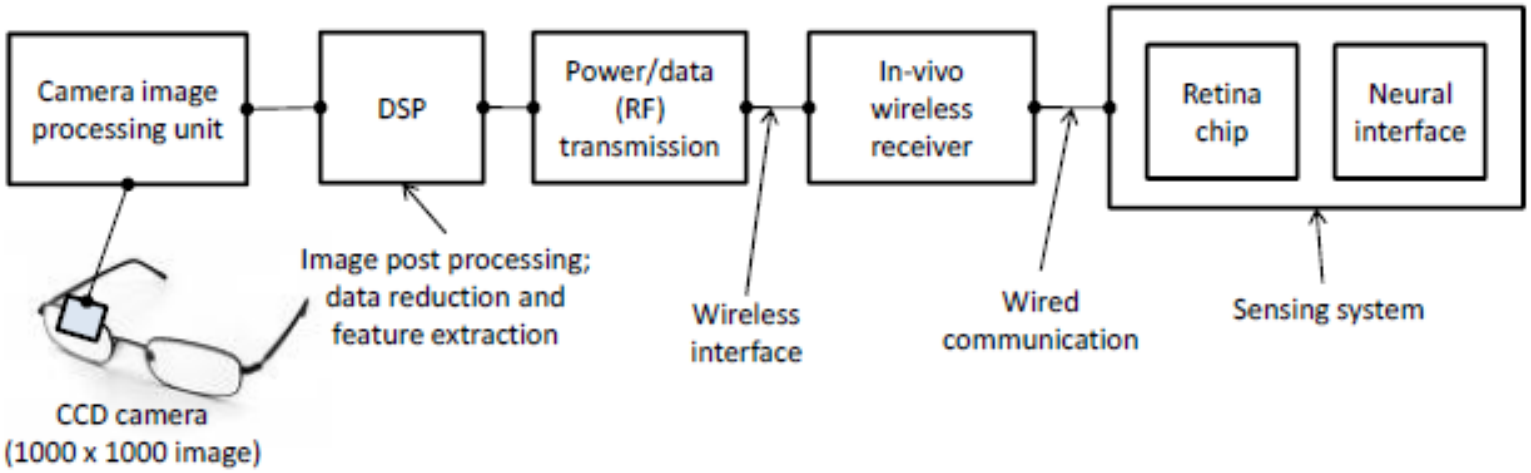


Figure 2.4 The processing components of the artificial retina (Schwiebert et al. 2001)

Artificial Retina

- Figure 2.4 illustrates the signal processing steps of the artificial retina
  - a *camera* embedded in a pair of spectacles *directs its output to* a real-time *DSP*
  - *DSP* - *data reduction and processing*
  - the camera can be combined with a laser pointer for automatic focusing
  - *the output* of the DSP *is compressed* and *transmitted* through a wireless link to the *implanted sensor array*
  - the sensor array decodes the image and produces a corresponding electrical signal

Pipeline Monitoring

- *Objective:* monitoring gas, water and oil pipelines
- Motivation:
  - management of pipelines presents *a formidable challenge*
    - long length, high value, high risk
    - difficult access conditions
    - requires continuous and unobtrusive monitoring
  - *leakages* can occur due to excessive deformations
    - earthquakes
    - landslides or collisions with an external force
    - corrosion, wear, material flaws
    - intentional damage to the structure

Pipeline Monitoring

- To detect leakages, it is vital to *understand the characteristics* of the substance *the pipelines transport*
  - fluid pipelines generate a hot-spot at the location of the leak
  - gas pipelines generate a cold-spot due to the gas pressure relaxation
  - fluid travels at a higher propagation velocity in metal pipelines than in a Polyvinyl Chloride (PVC)
  - a large number of commercially available sensors to detect and localize thermal anomalies
    - fiber optics sensors
    - temperature sensors and
    - acoustic sensors

PipeNet

- Motivation:
  - sewerage systems convey *domestic sewage*, *rainwater runoff*, and *industrial wastewater* to sewerage treatment plants
  - *historically*, these systems are designed to *discharge* their content to *nearby streams and rivers*
  - subsequently, *combined sewer* overflows are among *the major* sources of water quality *impairment*
  - nearly 770 large cities in the US, mainly older communities, have combined sewer systems (Stoianov et al. 2007)

PipeNet

- The PipeNet prototype has been developed to monitor water pipelines in urban areas

- The task is to monitor:
  - *hydraulic* and *water quality* by measuring pressure and pH
  - the *water level* in combined sewer systems
    - sewer collectors and combined sewer outflows

Three different settings

- First setting:
  - pressure and pH sensors are installed on a 12 inch cast-iron pipe
  - pressure sensor is a modified version of the OEM piezoresistive silicon sensor
  - pressure data is collected every 5 minutes at a rate of 100 Hz for a period of 5s
  - a pH sensor is a glass electrode with an Ag/AgCl reference cell
  - pH data is collected every 5 minute for a period of 10s at a rate of 100 Hz
  - the sensor nodes use a Bluetooth transceiver for wireless communication

Three different settings

- Second setting:
  - a pressure sensor measures the pressure in 8 inch cast iron pipe
  - the data are collected every 5 minutes for a period of 5 s at a sampling rate of 300 Hz
  - for this setting the raw data was transmitted to a remote gateway

Three different settings

- Third setting:
  - the water level of a combined sewer outflow collector is monitored
  - two pressure transducers (low-power device, < 10 mW) were placed at the bottom of the collector
  - an ultrasonic sensor (high-power device, < 550 mW) was placed on top of the collector
  - efficient power consumption:
    - pressure sensors are employed for periodic monitoring
    - when the difference of pressure sensors and the ultrasonic sensor exceeds a certain threshold; or
    - when the water level exceeds the weir height
    - the ultrasonic sensor is required to verify the readings from the pressure sensors

Node Architecture

- Wireless sensor nodes are the essential building blocks in a wireless sensor network
  - *sensing*, *processing*, and *communication*
  - *stores* and *executes* the communication protocols as well as data processing algorithms
- The node consists of *sensing*, *processing*, *communication*, and *power subsystems*
  - trade-off between flexibility and efficiency – both in terms of energy and performance

Node Architecture

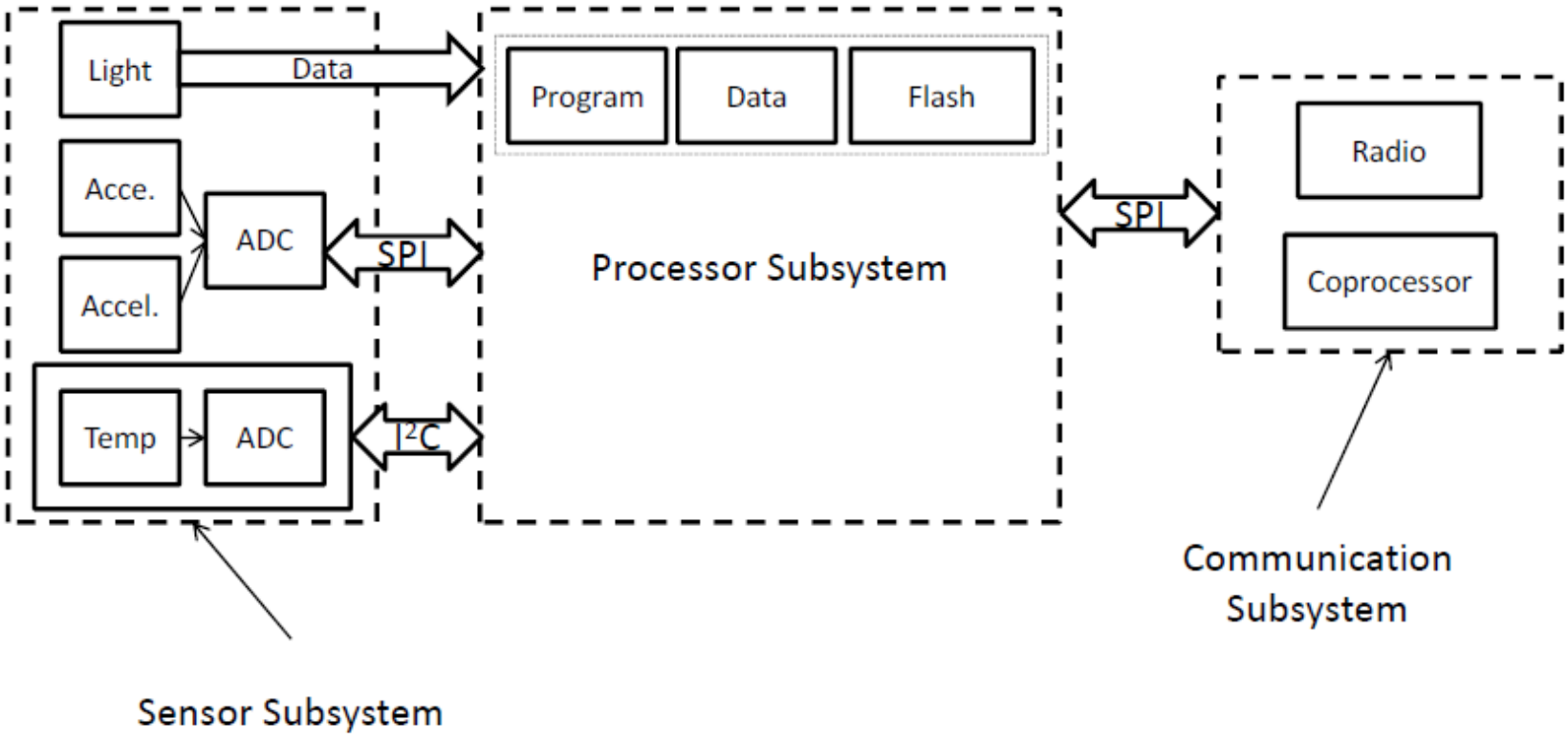
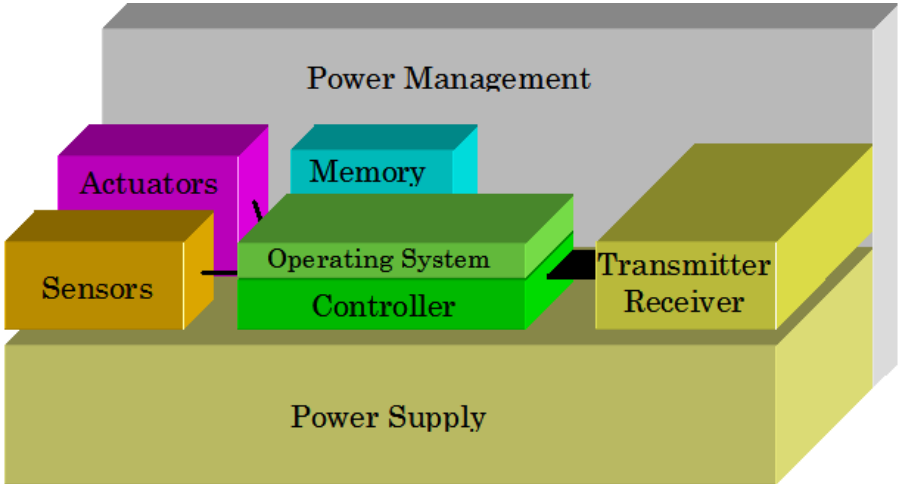


Figure 3.1 Architecture of a wireless sensor node

Device Architecture

- Microcontroller and program code
- Power supply
  - Power management
  - Renewable energy?
- Memory (RAM, FLASH)
- Sensors
- Actuators
- Communication
- Input/output
- Part of a larger system?

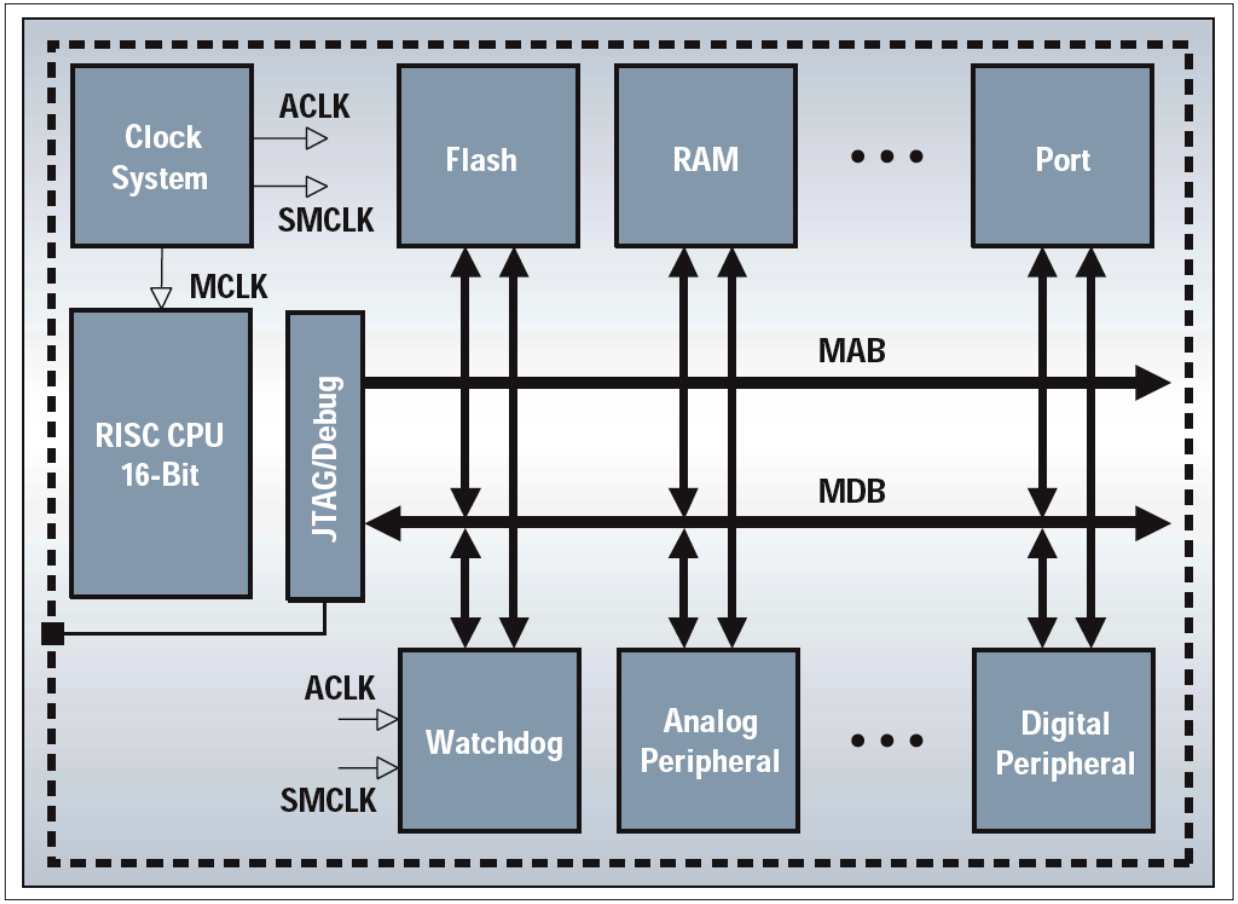


Microcontroller

- Main processing units of embedded devices
- Special purpose and highly integrated
  - Integrated RAM, ROM, I/O, peripherals
  - Extremely good power to performance ratio
  - Cheap, typically 0.25 - 10.00 USD
- Executes programs including embedded system control, measurement & communications
  - Usually time-critical requiring guarantees
  - Real-time performance a common requirement
    - Pre-emptive scheduled tasks
    - Queues and semaphores

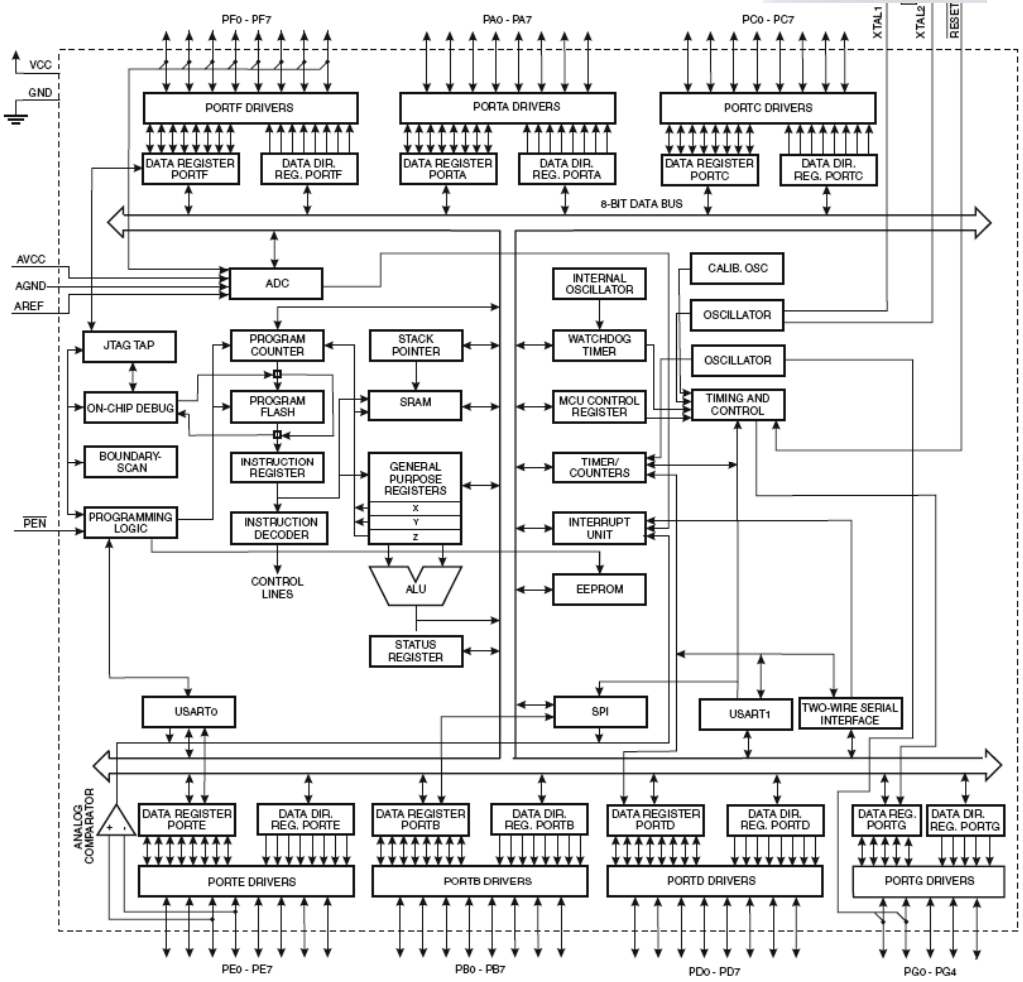
Example: MSP430

- Texas Instruments mixed-signal uC
- 16-bit RISC
- ROM: 1-60 kB
- RAM: Up to 10 kB
- Analogue
  - 12 bit ADC & DAC
  - LCD driver
- Digital
  - USART x 2
  - DMA controller
  - Timers



Example: Atmel AVR

- Atmel AVR family
- 8-bit RISC
- RAM: Up to 4 kB
- ROM: Up to 128 kB
- Analogue
  - ADC
  - PWM
- Digital
  - USARTs
  - Timers



Memory

- Random access memory (RAM)
  - Included on-board in microcontrollers
  - Often the most valuable resource
- Read-only memory (ROM)
  - Usually actually implemented with NOR flash memory
- Flash
  - Eraseable programmable memory
  - Can be read/written in blocks
  - Slow during the write process
  - Consumes power of course!
- External memory
  - External memory supported by some microcontrollers
  - Serial flash always supported

Common Bus Interfaces

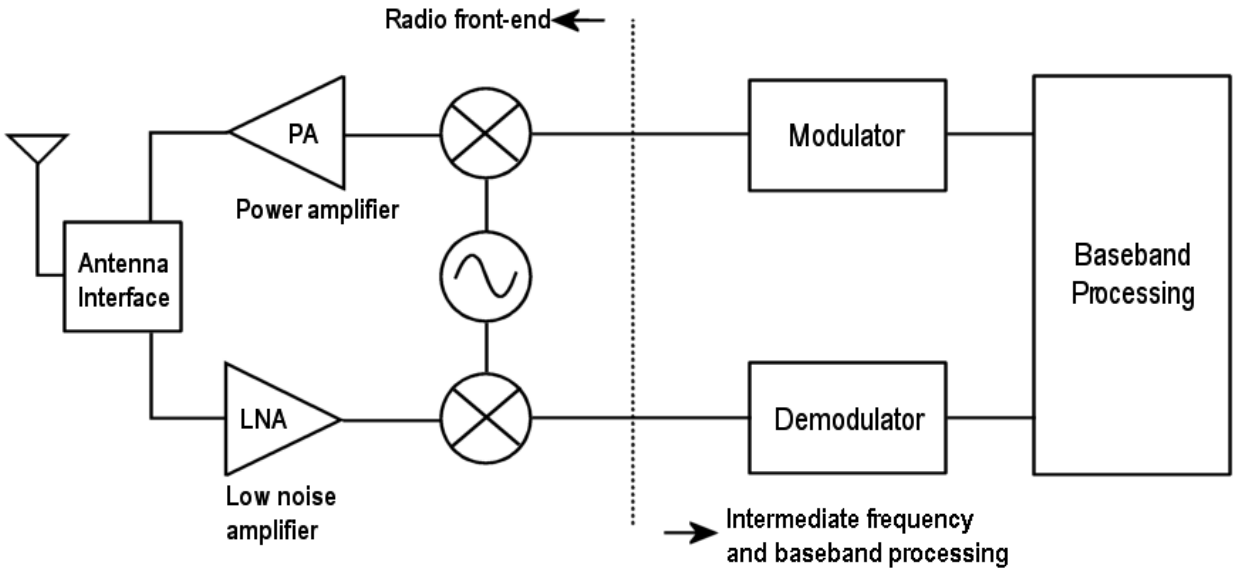
- Digital and analogue I/O
  - Accessed by port and pin number (e.g. P1.3)
  - Some pins are also connected to interrupts
- UART
  - Asynchronous serial bus
  - After level translation it is an RS232 bus
  - Usually kbps up to 1 mbps
- SPI (serial peripheral interface)
  - Synchronous serial bus
  - Reliable with speeds of several Mbps
- I^2C (inter-integrated circuit) bus
  - 2-wire bus with data and clock
- Parallel bus
  - Implemented with X-bit width
  - X-bit address and clock signals

Communications

- Embedded devices are autonomous but most often part of a larger system
- Thus communications interfaces are very important in the embedded world
- Wired interfaces
  - Serial: RS232, RS485
  - LAN: Ethernet
  - Industrial: Modbus, Profibus, Lontalk, CAN
- Wireless interfaces
  - **Low-power: IEEE 802.15.4 (ZigBee, ISA100, Wireless HART)**
  - WLAN: WiFi
  - WAN: GPRS, WiMax

Transceivers

- Modern embedded communications chips are transceivers: they combine half-duplex transmission and reception.
- Transceivers integrate varying functionality, from a bare analogue interface to the whole digital baseband and key MAC functions.



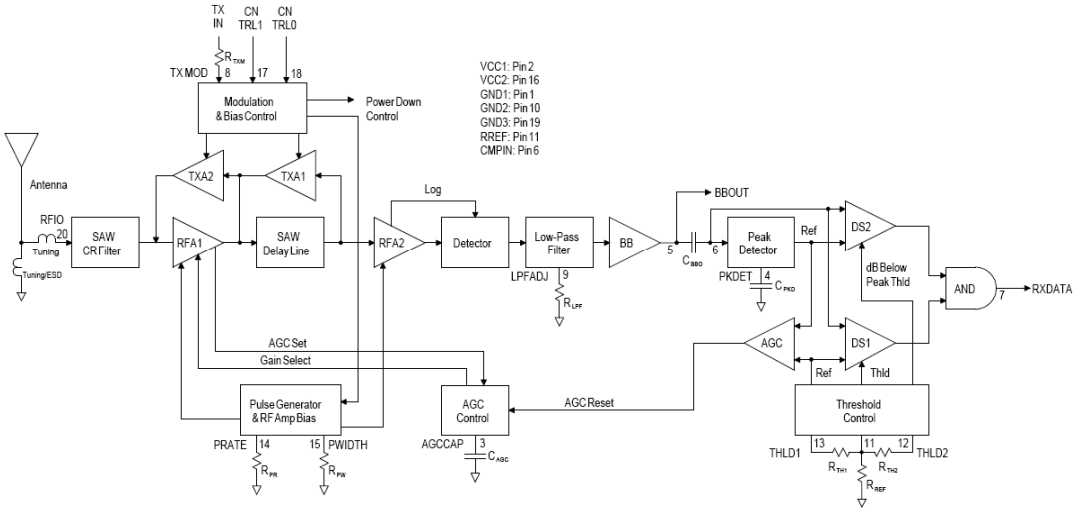
Important Characteristics

- Level of digital integration
- Power consumption and efficiency
  - Transition speeds and consumption
  - Levels of sleep
- Carrier frequency and data rate
- Modulation
- Error coding capabilities
- Noise figure and receiver sensitivity
- Received signal strength indicator (RSSI)
- Support for upper layers
- Data and control interface characteristics

Example: RFM TR1000

- Proprietary radio at 916 MHz
- OOK and ASK modulation
- 30 kbps (OOK) or 115.2 kbps (ASK) operation
- Signal strength indicator
- Provides bit interface
- Not included:
  - Synchronization
  - Framing
  - Encoding
  - Decoding

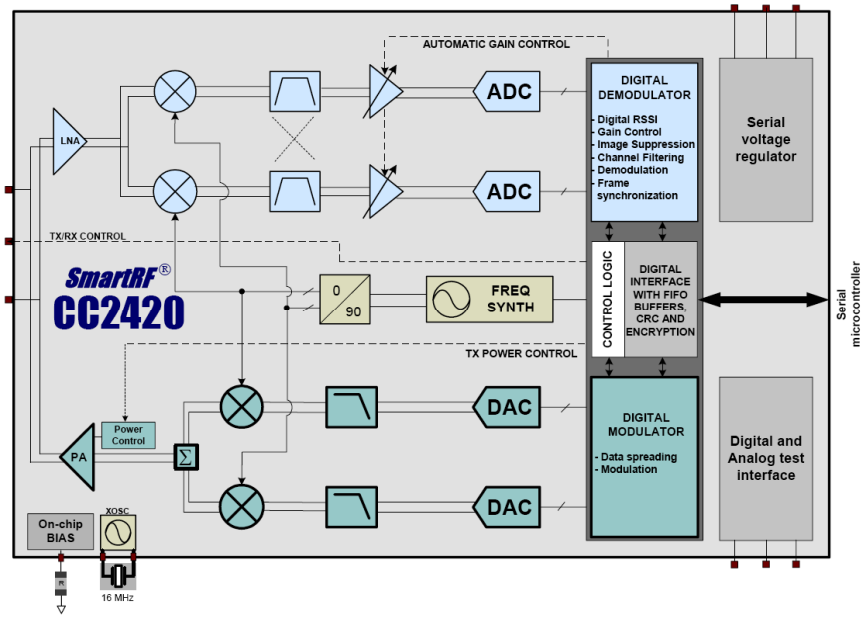
Sleep	Tx	Rx
0.7 uA	12 mA	3.8 mA



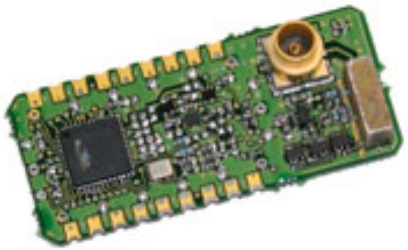
Example: CC2420

- IEEE 802.15.4 compliant radio
- 2.4 GHz band using DSSS at 250 kbps
- Integrated voltage regulator
- Integrated digital baseband and MAC functions
  - Clear channel assessment
  - Energy detection (RSSI)
  - Synchronization
  - Framing
  - Encryption/authentication
  - Retransmission (CSMA)

Sleep	Idle	Tx	Rx
20 uA	426 uA	8.5 – 17.4 mA	18.8 mA



Example: CC2430

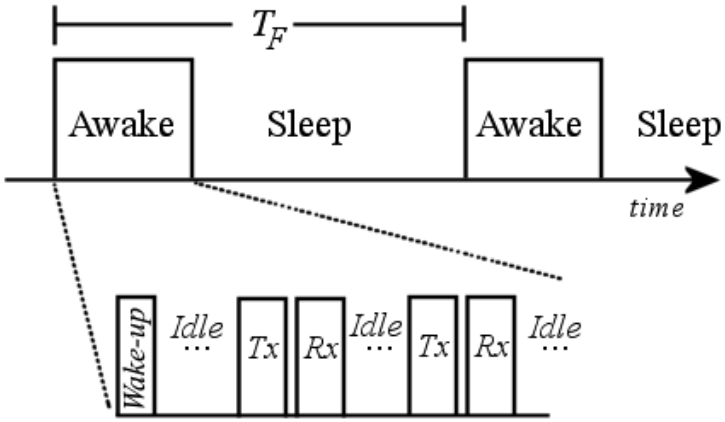


- System-on-a-chip solution
- Integrated 8051 microcontroller
  - 32 MHz Clock Speed
  - ADC, DAC, IOs, 2 UARTs etc.
  - 8 kB of RAM, up to 128 kB of ROM
- Integrated IEEE 802.15.4 radio, like the CC2420
- Power consumption 10-12 mA higher than the CC2420, coming from the 8051 microcontroller
- Saves cost, only about 1 EUR more expensive than the CC2420
- Internal DMA makes radio and UART performance better than with a uC + CC2420 solution

Power Consumption

- Radio power consumption critical to consider
- Power output level
  - Limited savings effect
  - Optimal power difficult
  - Must be considered globally
- Transition times
  - Each transition costs
  - Power equal to RX mode
  - Should be accounted for

Output Power (mW)	Power Used (mW)
0.003	15.30
0.032	17.82
0.100	20.16
0.200	22.50
0.316	25.02
0.501	27.36
0.794	29.70
1.000	31.32



Power Consumption

A simple approximation for power consumption:

$$P_{avg} = \frac{1}{T_F}\left\{P_{Rx}T_{wk-up} + P_{Rx}(N_{Tx}T_{Tx-up} + N_{Rx}T_{Rx-up}) + P_{Tx}T_{Tx} + P_{Rx}T_{Rx} + P_{idle}T_{idle} + P_{sleep}T_{sleep}\right\}$$

T_{wk-up} = Time that takes to go from sleep state to awake state
T_{Tx-up} = Transmitter setup time, i.e. time it takes for the transmitter to be ready
T_{Tx} = Time in the *Tx* state
T_{Rx-up} = Receiver setup time, i.e. time it takes for the receiver to be ready
T_{Rx} = Time in the *Rx* state
T_{idle} = Time in the idle state
T_{sleep} = Time in the sleep state
N_{Tx} = Average number of times per frame that the transmitter is used
N_{Rx} = Average number of times per frame that the receiver is used
T_F = Duration of the time frame
P_{Tx} = Power used in the *Tx* state
P_{Rx} = Power used in the *Rx* state
P_{idle} = Power used in the idle state
P_{sleep} = Power used in the sleep state
P_{avg} = Average power used by the transceiver

Sensors & Actuators

- Sensors measure real-world phenomena and convert them to electrical form
  - Analogue sensors require an ADC
  - Digital sensors use e.g. I2C or SPI interfaces
  - Human interface can also be a sensor (button)
- IEEE 1451 standard becoming important
  - Defines standard interfaces and auto-configuration
  - Also some protocol specifications
- Actuators convert an electrical signal to some action
  - Analogue and digital interfaces both common
  - A motor servo is a good example

Operating Systems

Operating Systems

- An operating System is
  - a thin software layer
  - resides between the hardware and the application layer
  - provides basic programming abstractions to application developers
- Its *main task* is to enable applications to interact with hardware resources

Operating Systems

- Operating systems are classified as: single-task/ multitasking and single-user/multiuser operating systems
  - multi-tasking OS - the overhead of concurrent processing because of the limited resources
  - single task OS - tasks should have a short duration
- The choice of a particular OS depends on several factors; typically *functional* and *non-functional* aspects

Data Types

- Interactions between the different subsystems take place through:
  - well-formulated protocols
  - data types
- Complex data types have strong expression power but consume resources - struct and enum
- Simple data types are resource efficient but have limited expression capability - C programming language

Scheduling

- Two scheduling mechanisms:
  - queuing-based scheduling
    - FIFO - the simplest and has minimum system overhead, but treats tasks unfairly
    - sorted queue - e.g., shortest job first (SJF) - incurs system overhead (to estimate execution duration)
  - round-robin scheduling
    - a time sharing scheduling technique
    - several tasks can be processed concurrently

Scheduling

- Regardless of how tasks are executed, a scheduler can be either
  - a non-preemptive scheduler - a task is executed to the end, may not be interrupted by another task
  - or preemptive scheduler - a task of higher priority may interrupt a task of low priority

Stacks & System Calls

- *Stacks*
  - a data structure that temporarily stores data objects in memory by piling one upon another
  - objects are accessed using last-in-first-out (LIFO)
- *System Calls*
  - decouple the concern of accessing hardware resources from implementation details
  - whenever users wish to access a hardware resource, they invoke these operations without the need to concern themselves how the hardware is accessed

Handling Interrupts

- An interrupt is an asynchronous signal generated by
  - a hardware device
  - several system events
  - OS itself
- An interrupt causes:
  - the processor to interrupt executing the present instruction
  - to call for an appropriate interrupt handler
- Interrupt signals can have different priority levels, a high priority interrupt can interrupt a low level interrupt
- Interrupt mask: let programs choose whether or not they wish to be interrupted

Multi-threading

- A *thread* is the path taken by a processor or a program during its execution
- *Multi-threading* - a task is divided into several logical pieces
  - scheduled independent from each other
  - executed concurrently
- Two advantages of a multi-threaded OS:
  1. tasks do not block other tasks
  2. short-duration tasks can be executed along with long-duration tasks

Multi-threading

- Threads cannot be created endlessly
  - the creation of threads *slows down* the processor
  - no sufficient resources to divide
- The OS can keep the number of threads to a *manageable size* using a thread pool

Thread-based vs. Event-based Programming

- Decision whether to use threads or events programming:
  - need for separate stacks
  - need to estimate maximum size for saving context information
- *Thread-based programs* use multiple threads of control within:
  - a single program
  - a single address space

Thread-based vs. Event-based Programming

- *Advantage:*
  - a thread blocked can be suspended while other tasks are executed in different threads
- *Disadvantages:*
  - must carefully protect shared data structures with locks
  - use condition variables to coordinate the execution of threads

Thread-based vs. Event-based Programming

- In *event-based programming*: use events and event handlers
  - event-handlers register with the OS scheduler to be notified when a named event occurs
  - a loop function:
    - polls for events
    - calls the appropriate event-handlers when events occur

- An event is processed to completion
  - unless its handler reaches at a blocking operation (callback and returns control to the scheduler)

Memory Allocation

- The memory unit is a precious resource
- Reading and writing to memory is costly
- How and for how long a memory is allocated for a piece of program determines the speed of task execution

Memory Allocation

- Memory can be allocated to a program:
  - *statically* - a frugal approach, but the requirement of memory *must be known* in advance
    - memory is used efficiently
    - runtime adaptation is not allowed
  - *dynamically* - the requirement of memory is *not known* in advance (on a transient basis)
    - enables flexibility in programming
    - but produces a considerable management overhead

System Overhead

- An operating system executes program code - requires its own share of resources
- The resources consumed by the OS are the system's overhead, it depends on
  - the size of the operating system
  - the type of services that the OS provides to the higher-level services and applications

System Overhead

- The resources of wireless sensor nodes have to be shared by programs that carry out:
  - sensing
  - data aggregation
  - self-organization
  - network management
  - network communication

Dynamic Reprogramming

- Once a wireless sensor network is deployed, it may be necessary to reprogram some part of the application or the operating system for the following reasons:
  1. the network may not perform optimally
  2. both the application requirements and the network's operating environment can change over time
  3. may be necessary to detect and fix bugs

Dynamic Reprogramming

- Manual replacement may not be feasible - develop an operating system to provide dynamic reprogramming support, which depends on
  - clear separation between the application and the OS
  - the OS can receive software updates and assemble and store it in memory
  - OS should make sure that this is indeed an updated version
  - OS can remove the piece of software that should be updated and install and configure the new version
  - all these consume resources and may cause their own bugs

Dynamic Reprogramming

- Software reprogramming (update) requires robust *code dissemination protocols*:
  - splitting and compressing the code
  - ensuring code consistency and version controlling
  - providing a robust dissemination strategy to deliver the code over a wireless link

TinyOS (Gay et al. 2007)

- TinyOS is *the most widely used*, *richly documented*, and *tool-assisted* runtime environment in WSN
  - static memory allocation
  - event-based system

- TinyOS's architecture consists of
  - a scheduler
  - a set of components, which are classified into
    - configuration components - "wiring" (how models are connected with each other)
    - modules - the basic building blocks of a TinyOS program

TinyOS (Gay et al. 2007)

- A component is made up of
  - a frame
  - command handlers
  - event handlers
  - a set of non-preemptive tasks

- A component is similar to an object in object-based programming languages:
  - it encapsulates state and interacts through well-defined interfaces
  - an interface that can define commands, event handlers, and tasks

TinyOS (Gay et al. 2007)

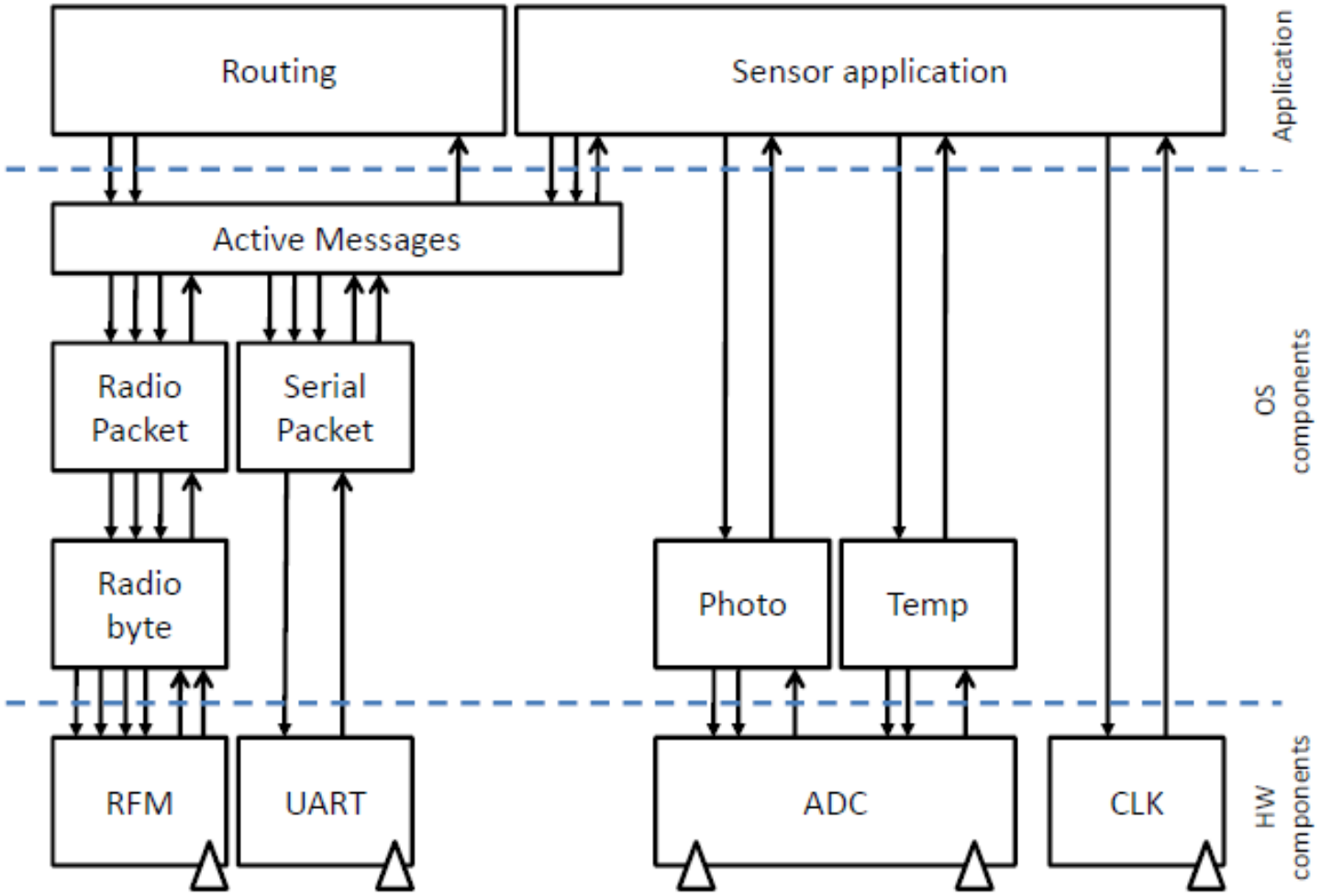


Figure 4.1 Logical distinction between low-level and high-level components (Hill et al. 2000)

TinyOS (Gay et al. 2007)

- Components are structured hierarchically and communicate with each other through commands and events:
  - higher-level components issue commands to lower-level components
  - lower-level components signal events to higher-level components
- In Figure 4.1, two components at the highest level communicate asynchronously through active messages
  - routing component - establishing and maintaining the network
  - sensor application - responsible for sensing and processing

TinyOS (Gay et al. 2007)

- The logical structure of components and component configurations

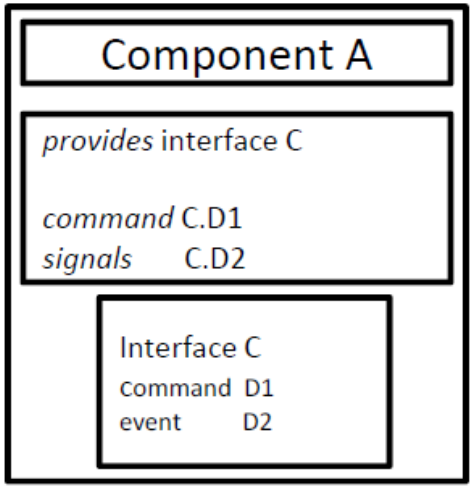


Figure 4.2

A TinyOS component providing an interface

In Figure 4.2, *Component A* declares its service by providing *interface C*, which in turn provides *command D1* and signals *event D2*.

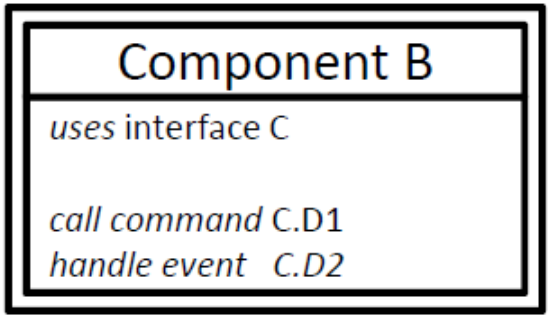


Figure 4.3

A TinyOS components that uses an interface

In Figure 4.3, *Component B* expresses interest in *interface C* by declaring a call to *command D1* and by providing an event handler to process *event D2*.

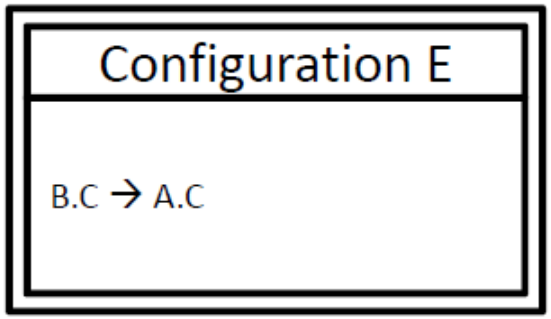


Figure 4.4

A TinyOS configuration that wires an interface provider and an interface user

In Figure 4.4, a binding between *Component A* and *Component B* is established through the *Configuration E*.

Tasks, Commands and Events

- The fundamental building blocks of a TinyOS runtime environment: *tasks, commands,* and *events*
  - enabling effective communication between the components of a single frame
- *Tasks :*
  - *monolithic processes* - should execute to completion - they cannot be preempted by other tasks, though they can be interrupted by events
  - possible to allocate a single stack to store context information
  - call lower level commands; signal higher level events; and post (schedule) other tasks
  - scheduled based on FIFO principle (in TinyOS)

Tasks, Commands and Events

- *Commands:*
  - non-blocking requests made by higher-level components to lower-level components
  - split-phase operation:
    - a function call returns immediately
    - the called function notifies the caller when the task is completed
- *Events:*
  - events are processed by the event handler
  - event handlers are called when hardware events occur
  - an event handler may react to the occurrence of an event in different ways
    - deposit information into its frame, post tasks, signal higher level events, or call lower level commands

Contiki


PÅL SVERRE HAGEN
ANDERS BAASMO CHRISTIANSEN
GUSTAF SKARSGÅRD
ODD-MAGNUS WILLIAMSON
TOBIAS SANTELMANN
JAKOB OFTEBRO
AGNES KITTELSEN
FRA REGISSØRENE AV
MAX MANUS
KON TIKI
24.08

What is Contiki?

- Contiki is an open-source operating system/protocol stack for networked embedded systems
- Small memory footprint
- Highly portable and reasonably compact
- Many platforms : ESB, Tmote Sky, etc
- Protocol stack configuration customizable
- Used in both academia and industry: Cisco and Atmel have joined the Contiki project
- Originally created by Adam Dunkels, developer of the uIP stack
- http://www.sics.se/contiki

Contiki processes

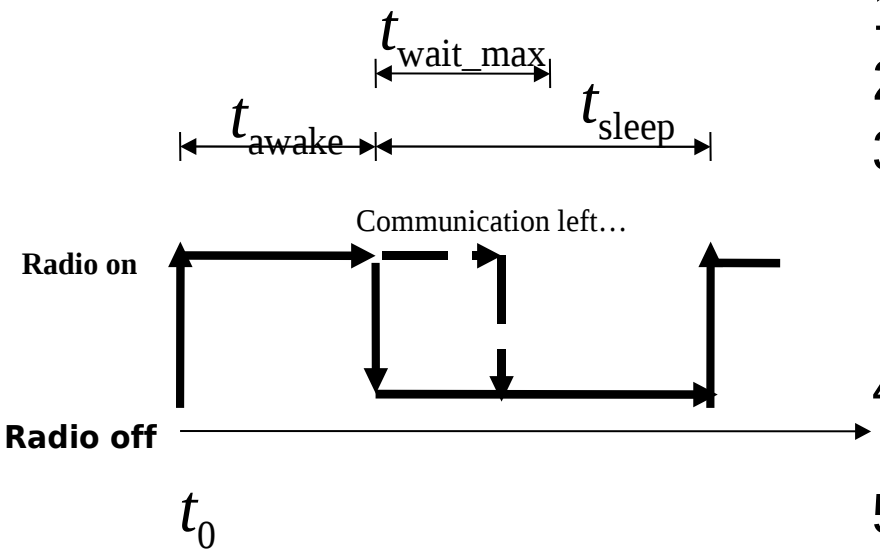
- Contiki core is event-driven
  - Interrupts and HW drivers generate events
  - Events are dispatched to event handlers by the Contiki core
  - Event handlers must return control to core as soon as possible
  - Co-operative multitasking

- Basic processes are implemented using protothreads
  - Easier to create sequential operations
  - An abstraction to avoid complex state-machine programming
    - In more complex applications, the amount of states may be huge

Contiki execution models

- Contiki offers multiple execution models
- Protothreads: thread-like event handlers
  - Allow thread-like structures without the requirement of additional stacks
  - Limits process structure: no switch/case structures allowed
  - May not use local variables
- Multi-threading model available
  - For more powerful systems
  - Allows structured application design

Protothreads – Simplifying Event-driven Programming



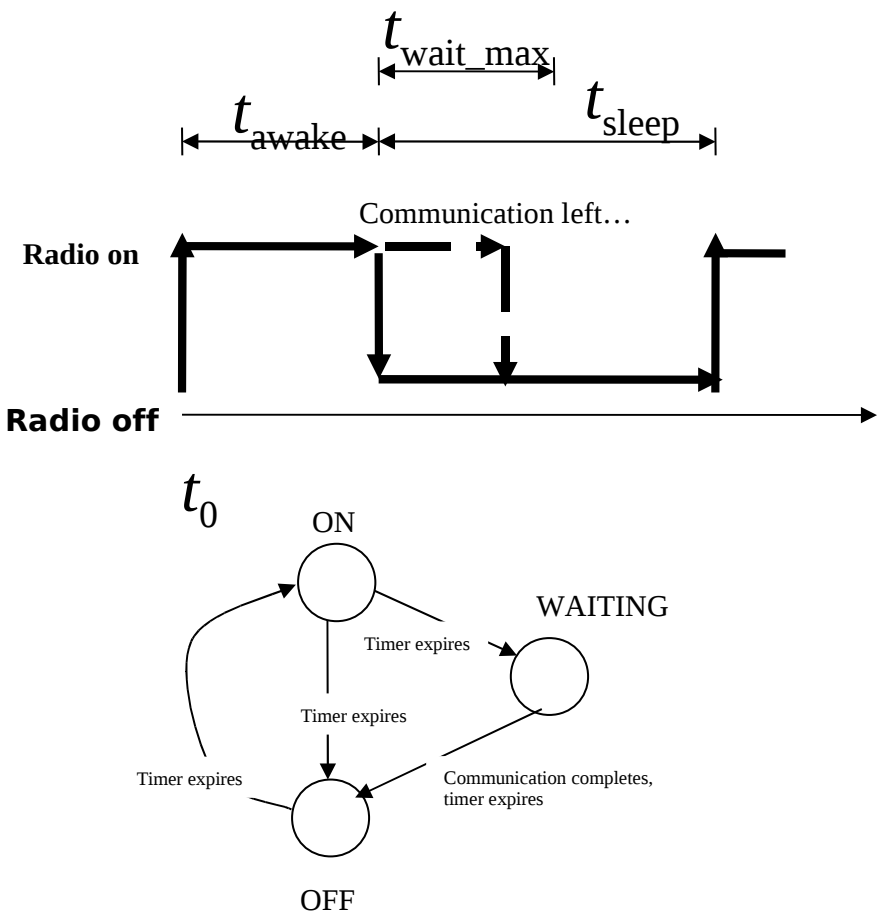
1. Turn radio on.
2. Wait until $t = t\_0 + t\_awake$.
3. If communication has not completed, wait until it has completed or $t = t\_0 + t\_awake + t\_wait\_max$.
4. Turn the radio off. Wait until $t = t\_0 + t\_awake + t\_sleep$.
5. Repeat from step 1.

No blocking wait!

Problem: with events, we cannot implement this as a five-step program!

Source: Contiki crash course (Thiemo Voig)

Event-driven state machine implementation

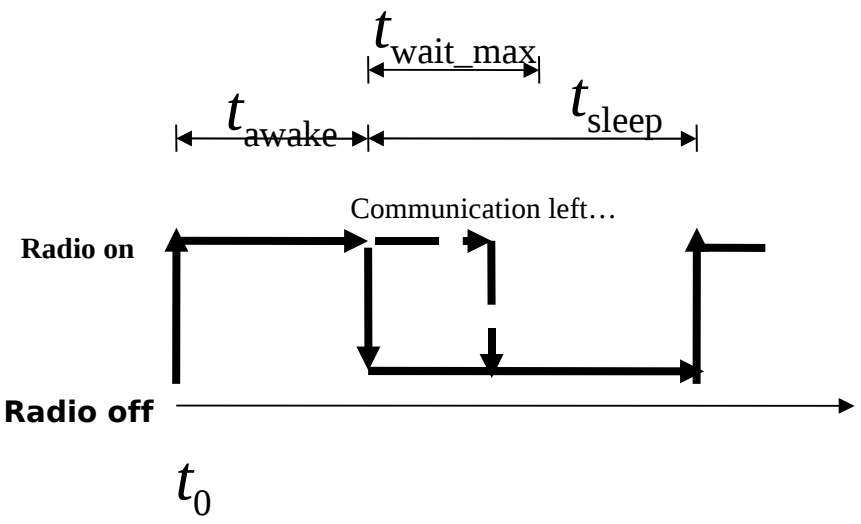



```
enum {ON, WAITING, OFF} state;

void eventhandler() {
  if(state == ON) {
    if(expired(timer)) {
      timer = t_sleep;
      if(!comm_complete()) {
        state = WAITING;
        wait_timer = t_wait_max;
      } else {
        radio_off();
        state = OFF;
      }
    }
  } else if(state == WAITING) {
    if(comm_complete() ||
       expired(wait_timer)) {
      state = OFF;
      radio_off();
    }
  } else if(state == OFF) {
    if(expired(timer)) {
      radio_on();
      state = ON;
      timer = t_awake;
    }
  }
```

Source: Contiki crash course (Thiemo Voig)

Protothreads-based implementation




```
int protothread(struct pt *pt) {
  PT_BEGIN(pt);
  while(1) {
    radio_on();
    timer = t_awake;
    PT_WAIT_UNTIL(pt, expired(timer));
    timer = t_sleep;
    if(!comm_complete()) {
      wait_timer = t_wait_max;
      PT_WAIT_UNTIL(pt, comm_complete()
                  || expired(wait_timer));
    }
    radio off();
    PT_WAIT_UNTIL(pt, expired(timer));
  }
  PT_END(pt);
}
```

- Code uses structured programming (**if** and **while**), mechanisms evident from code

  → Protothreads make Contiki code **nice**

Source: Contiki crash course (Thiemo Voig)

Contiki processes: An example

```
/* Declare the process */
PROCESS(hello_world_process, "Hello world");

/* Make the process start when the module is loaded */

AUTOSTART_PROCESSES(&hello_world_process);

/* Define the process code */

PROCESS_THREAD(hello_world_process, ev, data) {
  PROCESS_BEGIN();              /* Must always come first */
  printf("Hello, world!\n");    /* Initialization code goes here */
  while(1) {                    /* Loop for ever */
    PROCESS_WAIT_EVENT();       /* Wait for something to happen */
  }
  PROCESS_END();                /* Must always come last */
}
```

Contiki processes: Notes

- A process may not use switch-case constructs
  - A limitation of the protothread model
  - Complex state structures and switches should be subroutines
- A process may not declare local variables
  - Variables will lose their values at any event waiting call
  - All variables required by the main process must be static
- Effects on application design
  - The main process thread should only contain sequences between event waits
  - All operations should be done in subroutines

Contiki events

- `Process_wait_event();`

Waits for an event to be posted to the process

- `Process_wait_event_until(condition c);`

Waits for an event to be posted to the process, with an extra condition. Often used: wait until timer has expired

```
Process_wait_event_until(etimer_expired(&timer));
```

- `process_post(&process, eventno, evdata);`
  - Process will be invoked later
- `process_post_synch(&process, evno, evdata);`
  - Process will be invoked now
  - Must not be called from an interrupt (device driver)
- Using events

```
PROCESS_THREAD(rf_test_process, ev, data) {
  while(1) {
    PROCESS_WAIT_EVENT();
    if (ev == EVENT_PRINT) printf("%s", data);
  }
}
```

Contiki timers

- Contiki has two main timer types; etimer and rtimer
- Etimer: generates timed events

```
Declarations:
static struct etimer et;
In main process:
  while(1) {
    etimer_set(&et, CLOCK_SECOND);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    etimer_reset(&et);
  }
```

- Rtimer: uses callback function
  - Callback executed after specified time

```
rtimer_set(&rt, time, 0 , &callback_function, void *argument);
```

Contiki Protocol Stacks

- Contiki has 2 different protocol stacks: uIP and Rime
- uIP provides a full TCP/IP stack
  - For interfaces that allow protocol overhead
  - Ethernet devices
  - Serial line IP
  - Includes IPv4 and IPv6/6LoWPAN support
- Rime provides compressed header support
  - Application may use MAC layer only
- Protocol stacks may be interconnected
  - uIP data can be transmitted over Rime and vice versa

The Rime protocol stack

- Separate modules for protocol parsing and state machines
  - Rime contains the protocol operation modules
  - Chameleon contains protocol parsing modules

- Rime startup: an example
  - Configure Rime to use sicslowmac over cc2430 rf
  - Startup is done in platform main function: platform/sensinode/contiki-sensinode-main.c

```
rime_init(sicslowmac_init(&cc2430_rf_driver));

set_rime_addr(); //this function reads MAC from flash and places
                 //it to Rime address
```

Rime: Receiving

- Setting up Rime receiving: broadcast
  - Set up a callback function

```
Declarations:
static struct broadcast_conn bc;
static const struct broadcast_callbacks broadcast_callbacks =
    {recv_bc};

The callback definition:
static void
recv_bc(struct broadcast_conn *c, rimeaddr_t *from);

In main process:
broadcast_open(&bc, 128, &broadcast_callbacks);
```

- Unicast receive in a similar manner

Rime: Sending

- Sending broadcast data using Rime

```
Declarations:
static struct broadcast_conn bc;
In main process:
packetbuf_copyfrom("Hello everyone", 14);
broadcast_send(&bc);
```

- Sending unicast data using Rime

```
Declarations:
static struct unicast_conn uc;
In your function:
rimeaddr_t *addr;
addr.u8[0] = first_address_byte;
addr.u8[1] = second_address_byte;
packetbuf_copyfrom("Hello you", 9);
unicast_send(&uc, &addr);
```

Simulators

- COOJA: extensible Java-based simulator
  - Cross-level: Contiki nodes (deployable code), Java nodes, emulated MSP430 nodes
- MSPSim: sensor node emulator for MSP430-based nodes:
  - Tmote sky, ESB
  - Enables cycle counting, debuuging, power profiling, etc
  - Integrated in COOJA or standalone
  - COOJA/MSPSim enables also interoperability testing for MSP-based platforms (e.g IPv6 interop testing)

Demo

Evaluation

OS	Programming Paradigm	Building Blocks	Scheduling	Memory Allocation	System Calls
TinyOS	Event-based (split-phase operation, active messages)	Components, interfaces and tasks	FIFO	Static	Not available
SOS	Event-based (Active messages)	Modules and messages	FIFO	Dynamic	Not available
Contiki	Predominantly event-based, but it provides an optional multi-threading support	Services, service interface stubs and service layer	FIFO, poll handlers with priority scheduling	Dynamic	Runtime libraries
LiteOS	Thread-based (based on thread pool)	Applications are independent entities	Priority-based scheduling with an optional Round-robin support	Dynamic	A host of system calls available to the user (file, process, environment, debugging and device commands)

Table 4.1 Comparison of functional aspects of existing operating systems

Evaluation

OS	Minimum System overhead	Separation of Concern	Dynamic reprogramming	Portability
TinyOS	332 Bytes	There is no clean distinction between the OS and the application. At compilation time a particular configuration produces a monolithic, executable code.	Requires external software support	High
SOS	ca. 1163 Byte	Replaceable modules are compiled to produce an executable code. There is no clean distinction between the OS and the application.	Supported	Midium to low
Contiki	ca. 810 Byte	Modules are compiled to produce a reprogrammable and executable code, but there is no separation of concern between the application and the OS.	Supported	Medium
LiteOS	Not available	Application are separate entities; they are developed independent of the OS	Supported	Low

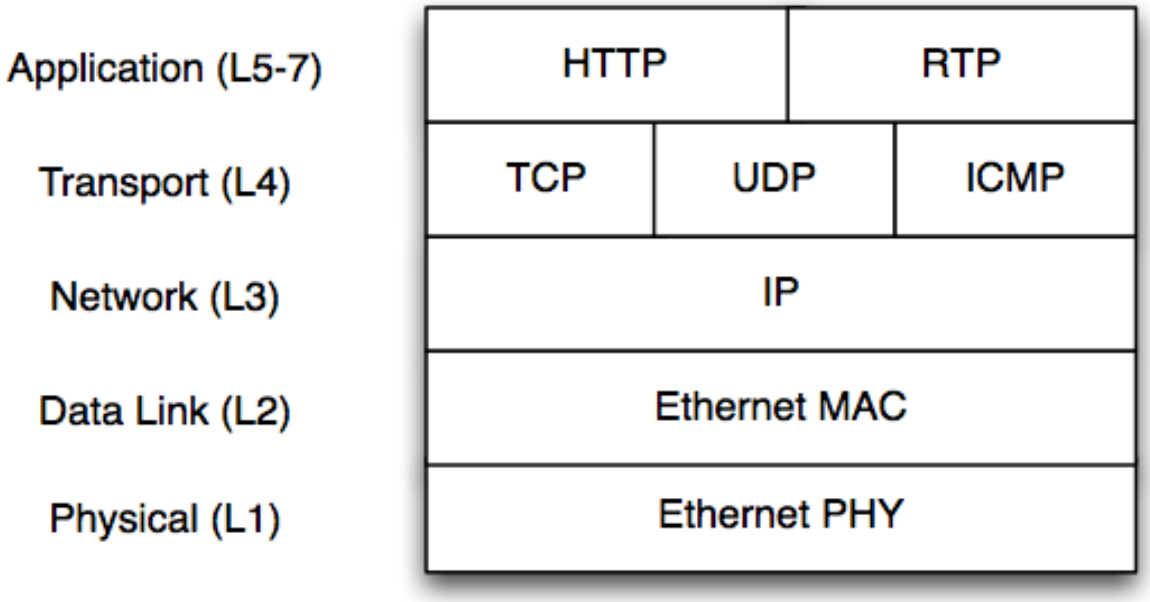
Table 4.2 Comparison of nonfunctional aspects of existing operating systems

The Internet Architecture & Protocols

The Internet

- A global, publicly accessible, series of interconnected computer networks (made up of hosts and clients) using the packet-switched Internet Protocol
- Consists of millions of small network domains
- ICANN, the Internet Corporation for Assigned Names and Numbers
  - Unique identifiers, domain names, IP addresses, protocol ports etc.
  - Only a coordinator, not a governing body
- These days an Internet Governance Forum (IGF) has been formed to discuss global governance
- Internet-related protocols are standardized by the Internet Engineering Task Force (IETF)

IP Protocol Stack



Internet Architecture

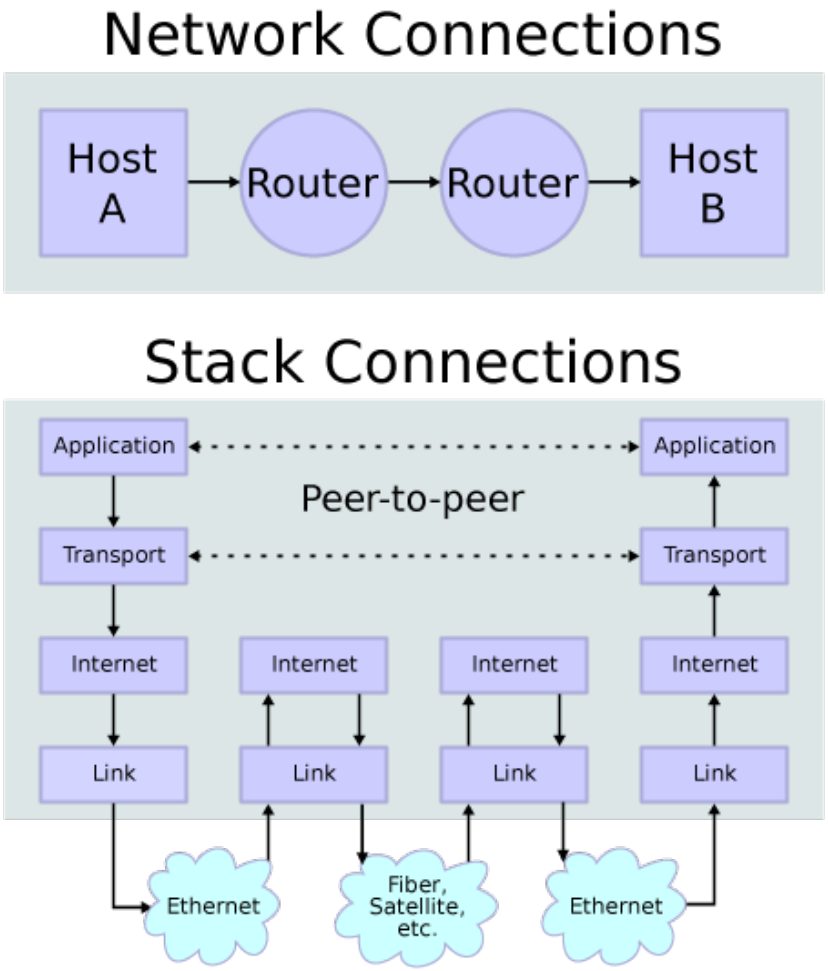


Image source: (Wikipeida) GFDL

Internet Protocol v6

- IPv6 (RFC 2460) = the next generation Internet Protocol
  - Complete redesign of IP addressing
  - Hierarchical 128-bit address with decoupled host identifier
  - Stateless auto-configuration
  - Simple routing and address management
- Majority of traffic not yet IPv6 but...
  - Most PC operating systems already have IPv6
  - Governments are starting to require IPv6
  - Most routers already have IPv6 support
  - So the IPv6 transition is coming
    - 1400% annual growth in IPv6 traffic (2009)

IPv4 vs. IPv6 Addressing

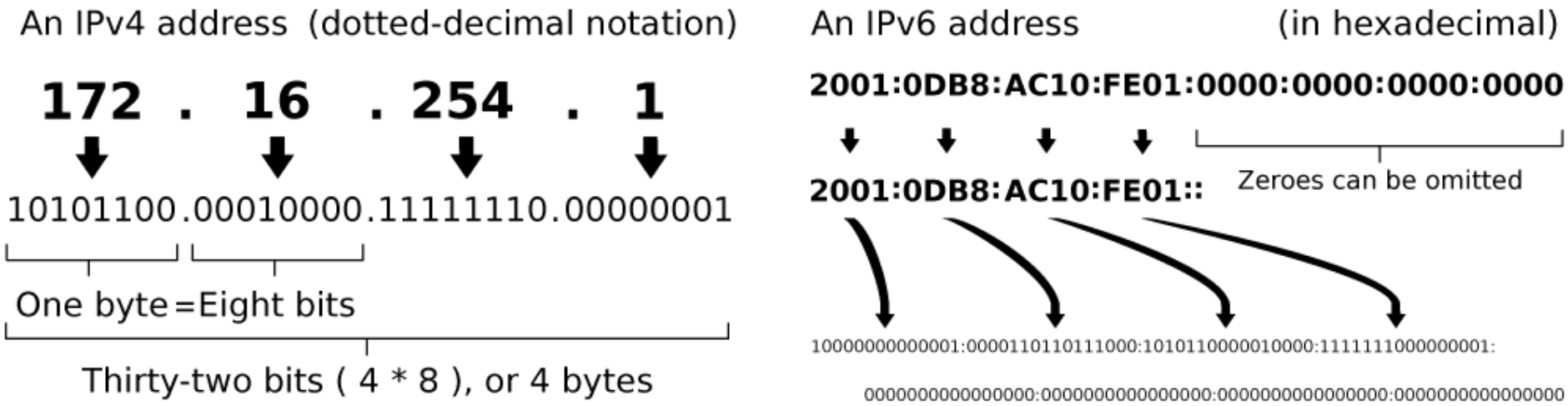
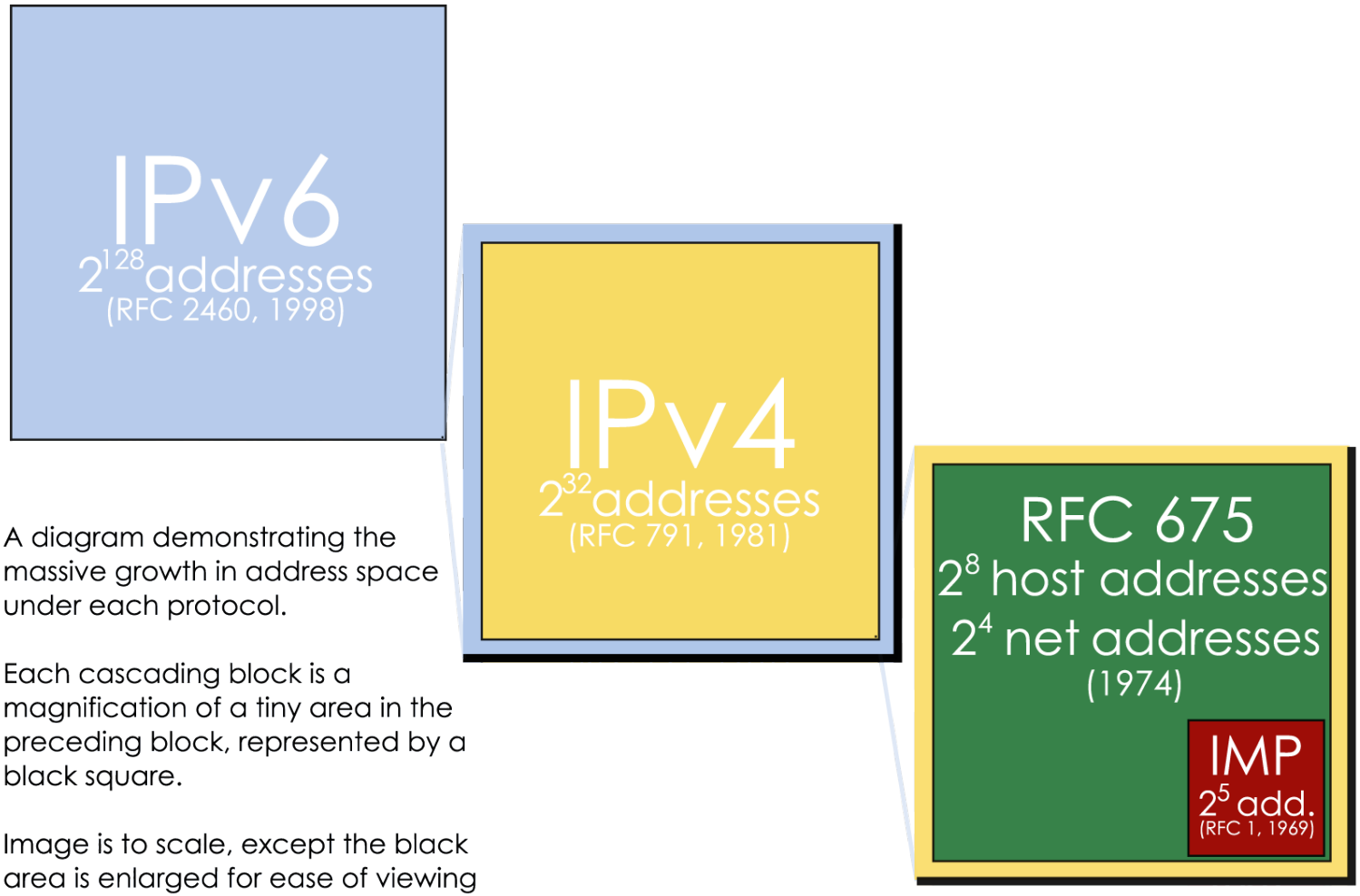


Image source: Indeterminant (Wikipeida) GFDL

Address Space Comparison



A diagram demonstrating the massive growth in address space under each protocol.

Each cascading block is a magnification of a tiny area in the preceding block, represented by a black square.

Image is to scale, except the black area is enlarged for ease of viewing

IPv4 vs. IPv6 Header

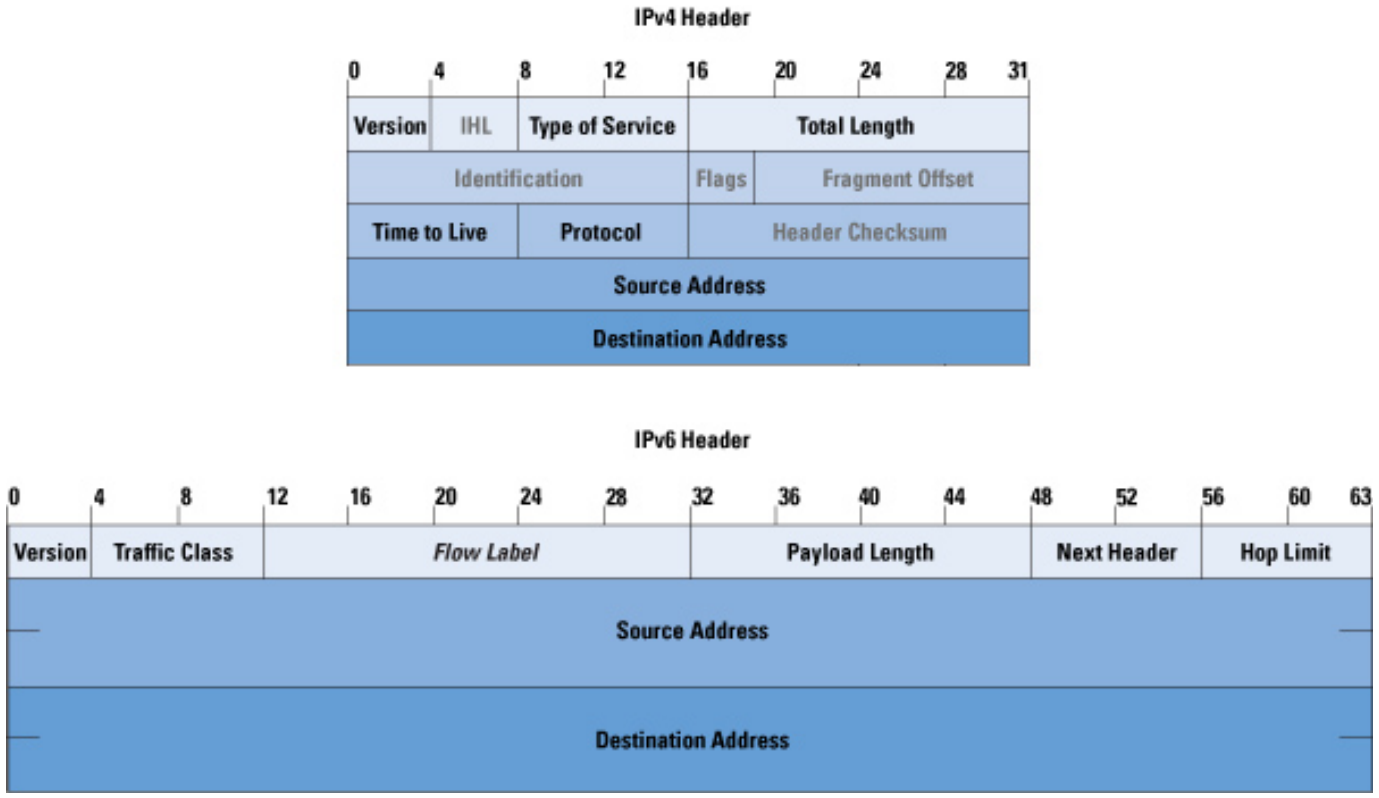
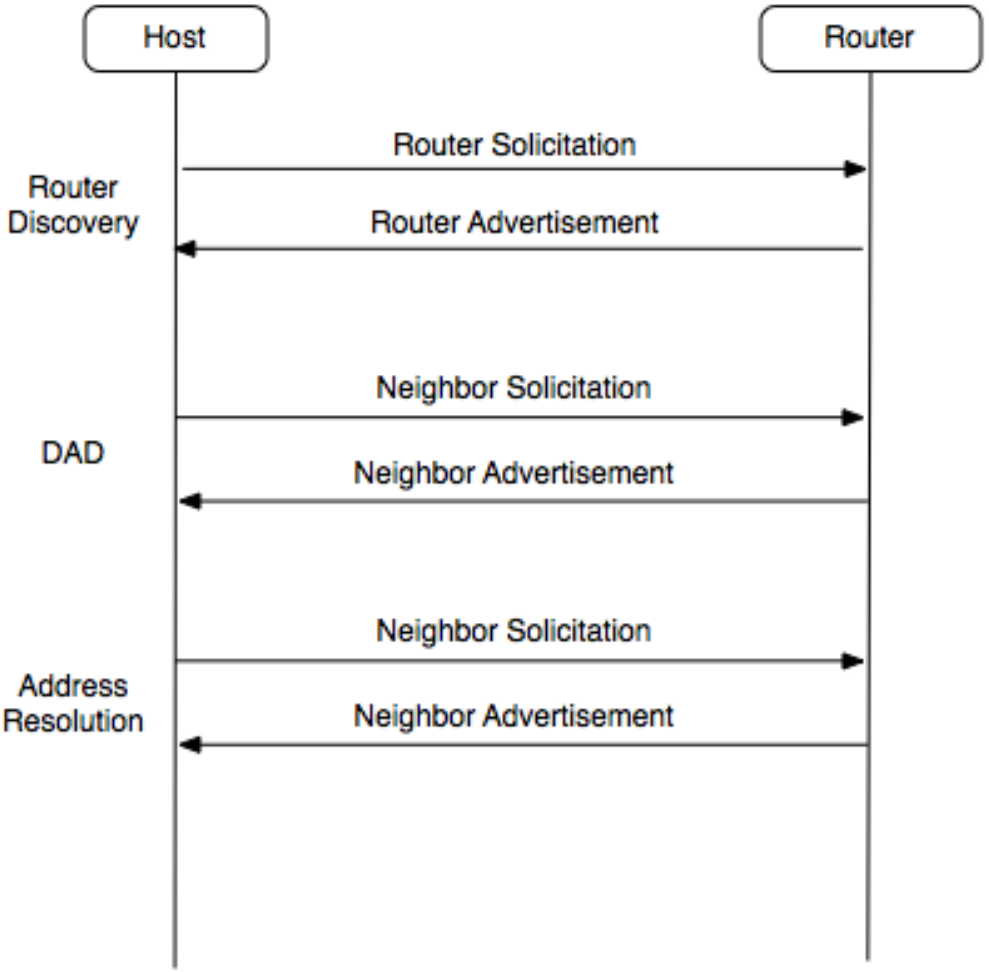


Image source: Bino1000, Mkim (Wikipeida) GFDL

IPv6 Neighbor Discovery

- IPv6 is the format - ND is the brains
  - “One-hop routing protocol” defined in RFC4861
- Defines the interface between neighbors
- Finding Neighbors
  - Neighbor Solicitation / Neighbor Acknowledgement
- Finding Routers
  - Router Solicitation / Router Advertisement
- Address resolution using NS/NA
- Detecting Duplicate Addresses using NS/NA
- Neighbor Unreachability Detection using NS/NA
- DHCPv6 may be used in conjunction with ND

IPv6 Neighbor Discovery

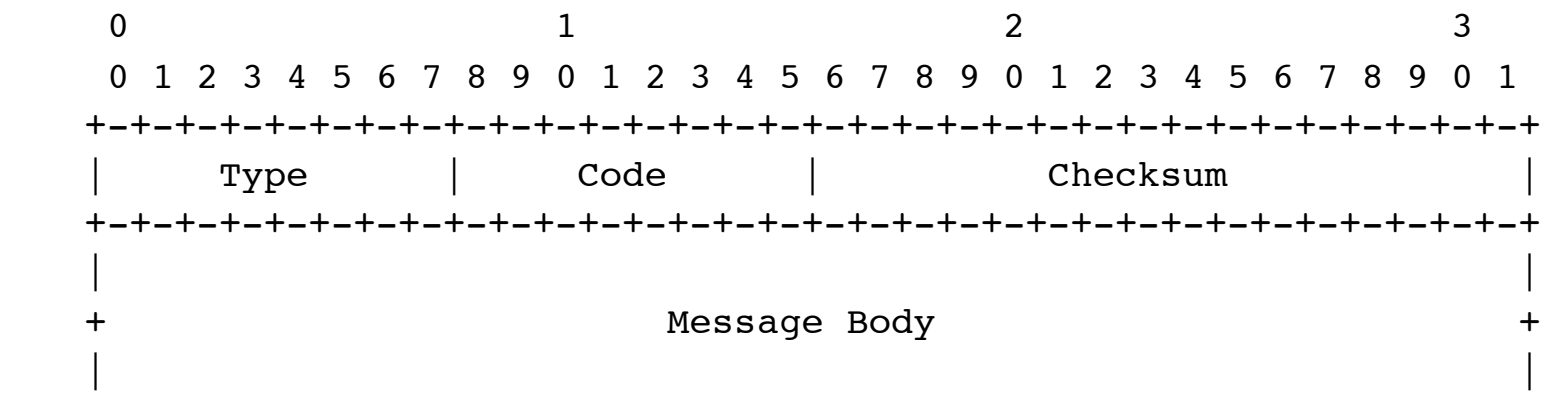


ICMPv6

- The Internet Control Message Protocol (ICMPv6)
  - Defined by RFC2463
  - Used for control messaging between IPv6 nodes
- ICMPv6 Error Messages
  - Destination Unreachable Message
  - Packet Too Big Message
  - Time Exceeded Message
  - Parameter Problem Message
- ICMPv6 Informational Messages
  - Echo Request Message
  - Echo Reply Message

ICMPv6

The ICMPv6 messages have the following general format:



The type field indicates the type of the message. Its value determines the format of the remaining data.

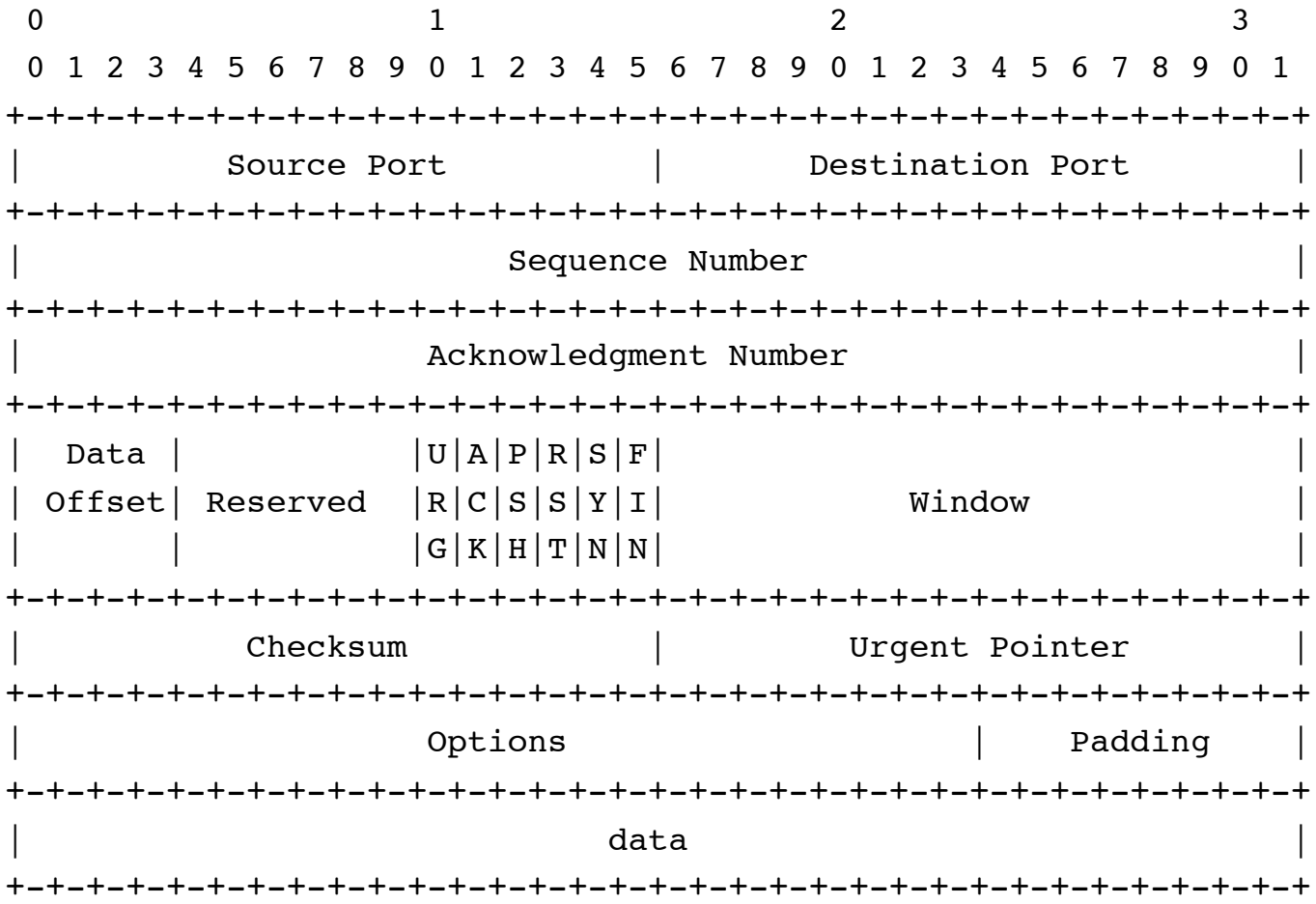
The code field depends on the message type. It is used to create an additional level of message granularity.

The checksum field is used to detect data corruption in the ICMPv6 message and parts of the IPv6 header.

TCP

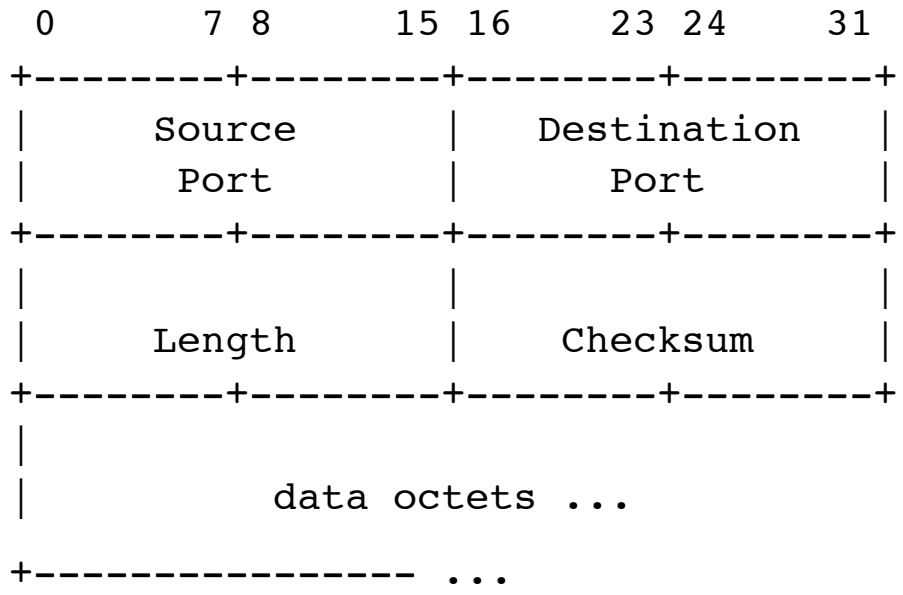
- The Transmission Control Protocol (TCP) (RFC 793)
  - A reliable, ordered transport for a stream of bytes
  - TCP is connection oriented, forming a pairing between 2 hosts using a 3-way handshake
  - Positive ack windowing is used with flow control
  - Congestion control mechanism critical for the Internet
- TCP is not suitable for every application
  - Support for unicast communications only
  - Reacts badly to e.g. wireless packet loss
  - Not all protocols require total reliability
  - TCP connection not suitable for very short transactions

The TCP Header



UDP

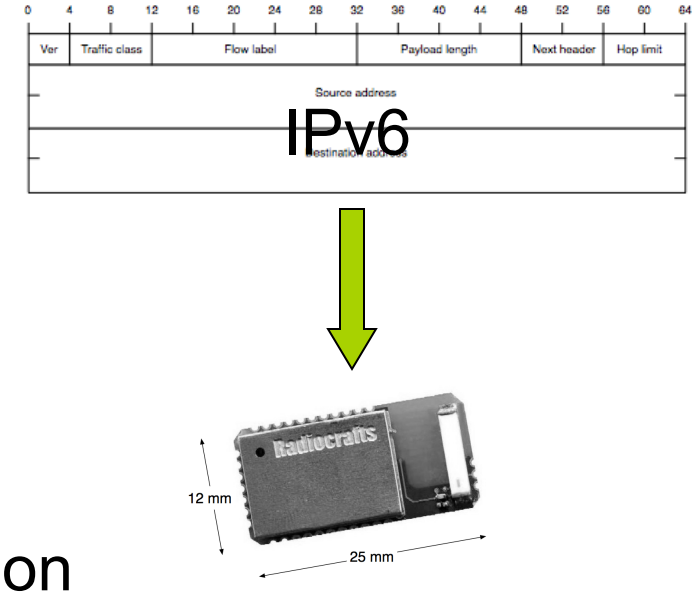
- The User Datagram Protocol (UDP) (RFC 768)
  - Used to deliver short messages over IP
  - Unreliable, connectionless protocol
  - Can be used with broadcast and multicast
  - Common in streaming and VoIP, DNS and network tools



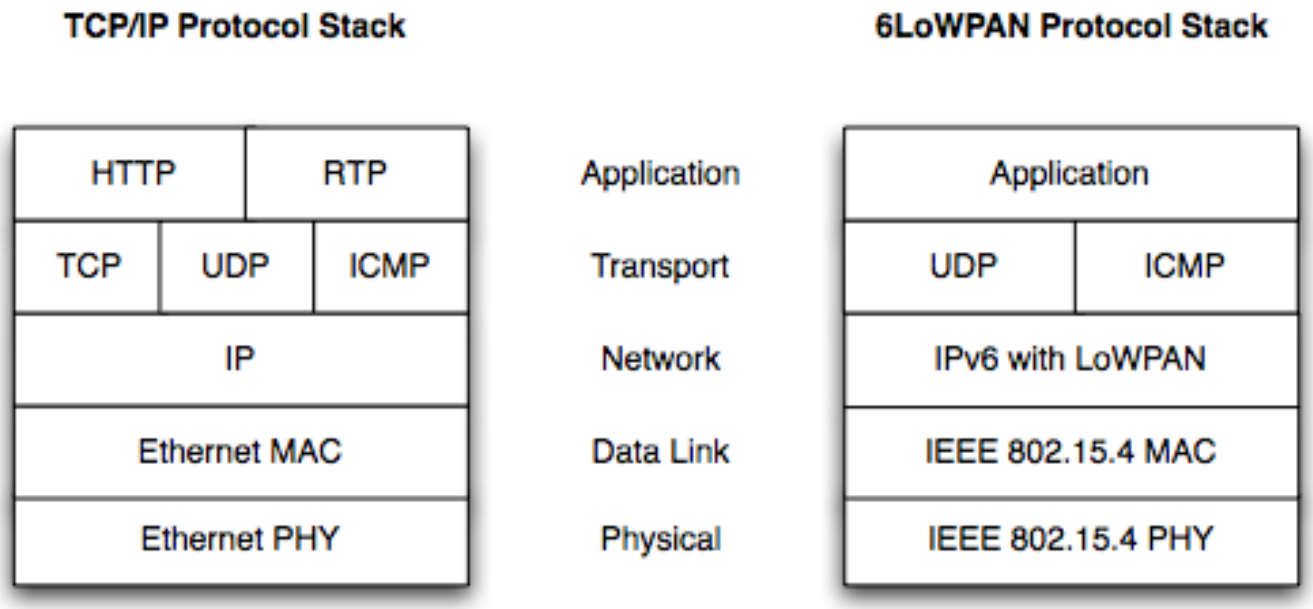
Introduction to 6LoWPAN

What is 6LoWPAN?

- IPv6 over Low-Power wireless Area Networks
- Defined by IETF standards
  - RFC 4919, 4944
  - draft-ietf-6lowpan-hc and -nd
  - draft-ietf-roll-rpl
- Stateless header compression
- Enables a standard socket API
- Minimal use of code and memory
- Direct end-to-end Internet integration
  - Multiple topology options

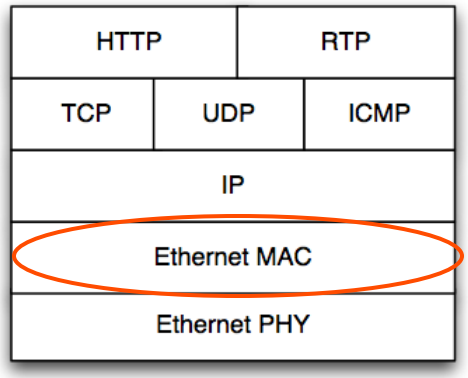


Protocol Stack



The Link-Layer and IP

- The Internet Protocol interconnects heterogeneous links
- Key link-layer features to support IP:
  - Framing
  - Addressing
  - Error checking
  - Length indication
  - Broadcast and unicast
- RFC3819 discusses IP subnetwork design
- 6LoWPAN enables IPv6 over very constrained links
  - Limited frame size and bandwidth
  - Wireless mesh topologies and sleeping nodes
  - No native multicast support

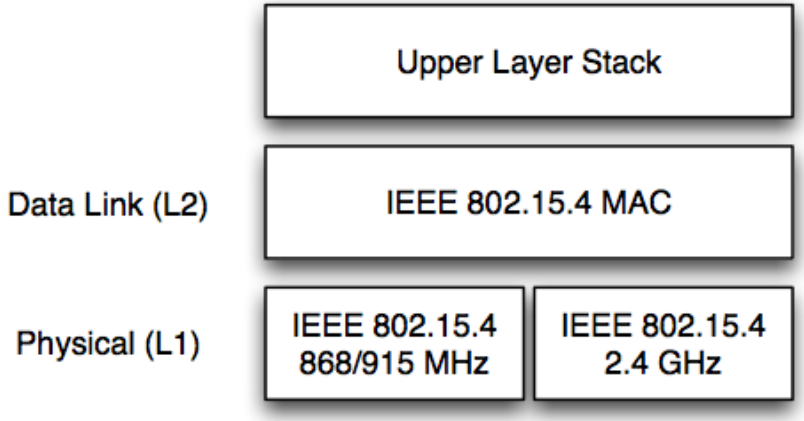


Medium Access Control

- The sharing of a radio by multiple independent devices
- There are multiple ways to share a radio
  - Frequency Division Multiple Access
  - Time Division Multiple Access
  - Carrier Sense Multiple Access
  - Code Division Multiple Access
  - Hybrids of the above
- MAC algorithms also take care of
  - Acknowledgements for packets
  - Link topology and addressing
  - Error checking and link security

IEEE 802.15.4

- Important standard for home networking, industrial control and building automation
- Three PHY modes
  - 20 kbps at 868 MHz
  - 40 kbps at 915 MHz
  - 250 kbps at 2.4 GHz (DSSS)
- Beaconless mode
  - Simple CSMA algorithm
- Beacon mode with superframe
  - Hybrid TDMA-CSMA algorithm
- Up to 64k nodes with 16-bit addresses
- Extensions to the standard
  - IEEE 802.15.4a, 802.15.4e, 802.15.5



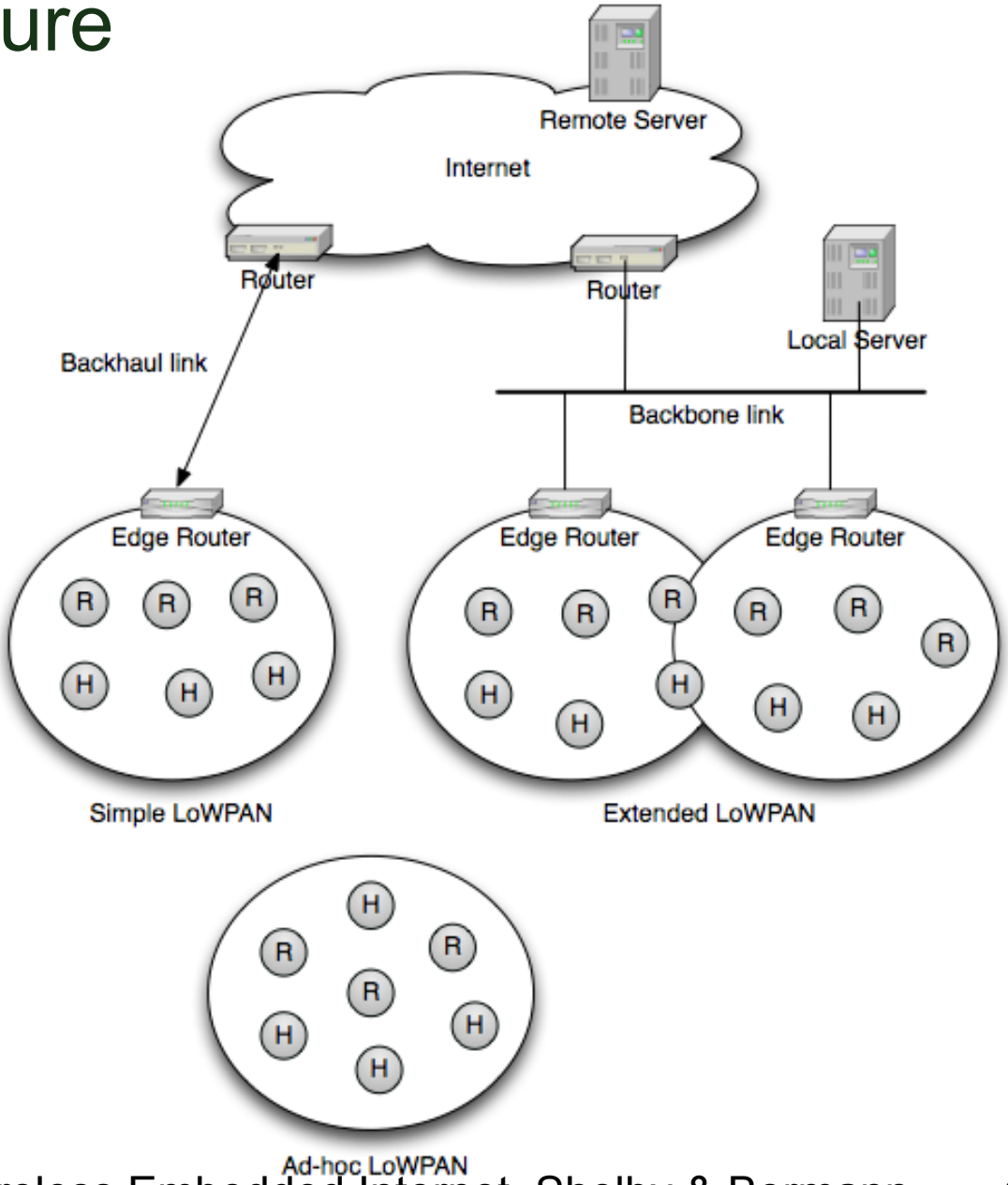
Other Link-Layers for 6LoWPAN

- Sub-GHz Industrial, Scientific and Medical band radios
  - Typically 10-50 kbps data rates, longer range than 2.4 GHz
  - Usually use CSMA-style medium access control
  - Example: CC1110 from Texas Instruments
- Power-Line Communications
  - Some PLC solutions behave like an 802.15.4 channel
  - Example: A technology from Watteco provides an 802.15.4 emulation mode, allowing the use of 6LoWPAN
- Z-Wave
  - A home-automation low-power radio technology

Features

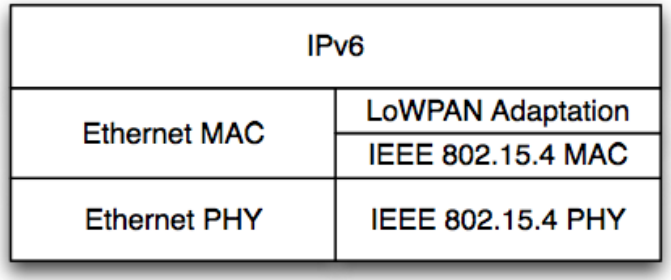
- Support for e.g. 64-bit and 16-bit 802.15.4 addressing
- Useful with low-power link layers such as IEEE 802.15.4, narrowband ISM and power-line communications
- Efficient header compression
  - IPv6 base and extension headers, UDP header
- Network autoconfiguration using neighbor discovery
- Unicast, multicast and broadcast support
  - Multicast is compressed and mapped to broadcast
- Fragmentation
  - 1280 byte IPv6 MTU -> 127 byte 802.15.4 frames
- Support for IP routing (e.g. IETF RPL)
- Support for use of link-layer mesh (e.g. 802.15.5)

Architecture



Architecture

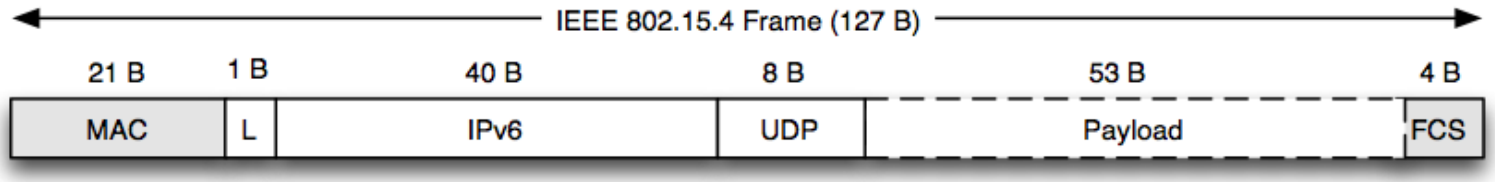
- LoWPANs are stub networks
- Simple LoWPAN
  - Single Edge Router
- Extended LoWPAN
  - Multiple Edge Routers with common backbone link
- Ad-hoc LoWPAN
  - No route outside the LoWPAN
- Internet Integration issues
  - Maximum transmission unit
  - Application protocols
  - IPv4 interconnectivity
  - Firewalls and NATs
  - Security



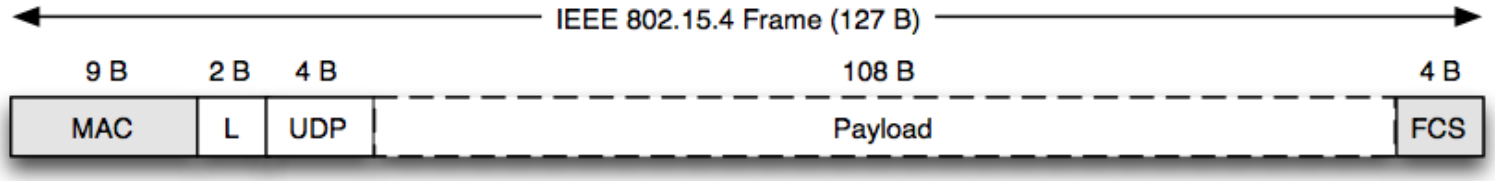
IPv6-LoWPAN Router Stack

6LoWPAN Headers

- Orthogonal header format for efficiency
- Stateless header compression



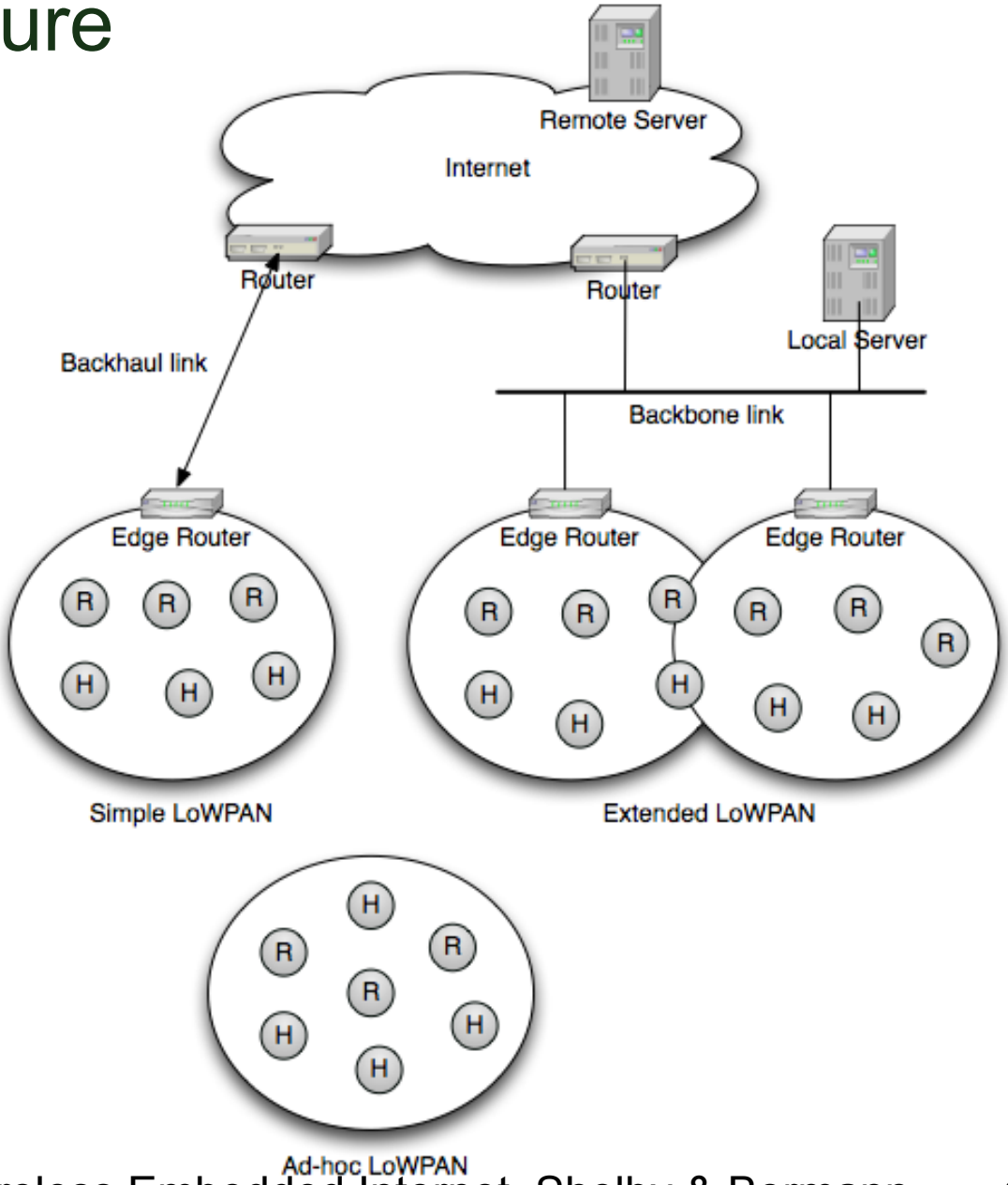
Full UDP/IPv6 (64-bit addressing)



Minimal UDP/6LoWPAN (16-bit addressing)

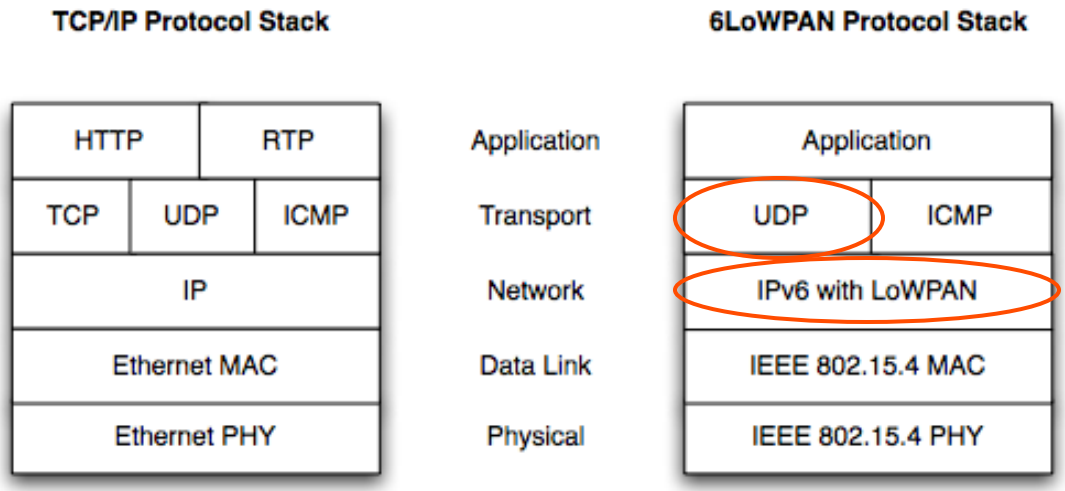
The 6LoWPAN Format

Architecture



The 6LoWPAN Format

- 6LoWPAN is an adaptation header format
  - Enables the use of IPv6 over low-power wireless links
  - IPv6 header compression
  - UDP header compression
- Format initially defined in RFC4944
- Updated by draft-ietf-6lowpan-hc (work in progress)



The 6LoWPAN Format

- 6LoWPAN makes use of IPv6 address compression
- RFC4944 Features:
  - Basic LoWPAN header format
  - HC1 (IPv6 header) and HC2 (UDP header) compression formats
  - Fragmentation & reassembly
  - Mesh header feature (depreciation planned)
  - Multicast mapping to 16-bit address space
- draft-ietf-6lowpan-hc Features:
  - New HC (IPv6 header) and NHC (Next-header) compression
  - Support for global address compression (with contexts)
  - Support for IPv6 option header compression
  - Support for compact multicast address compression

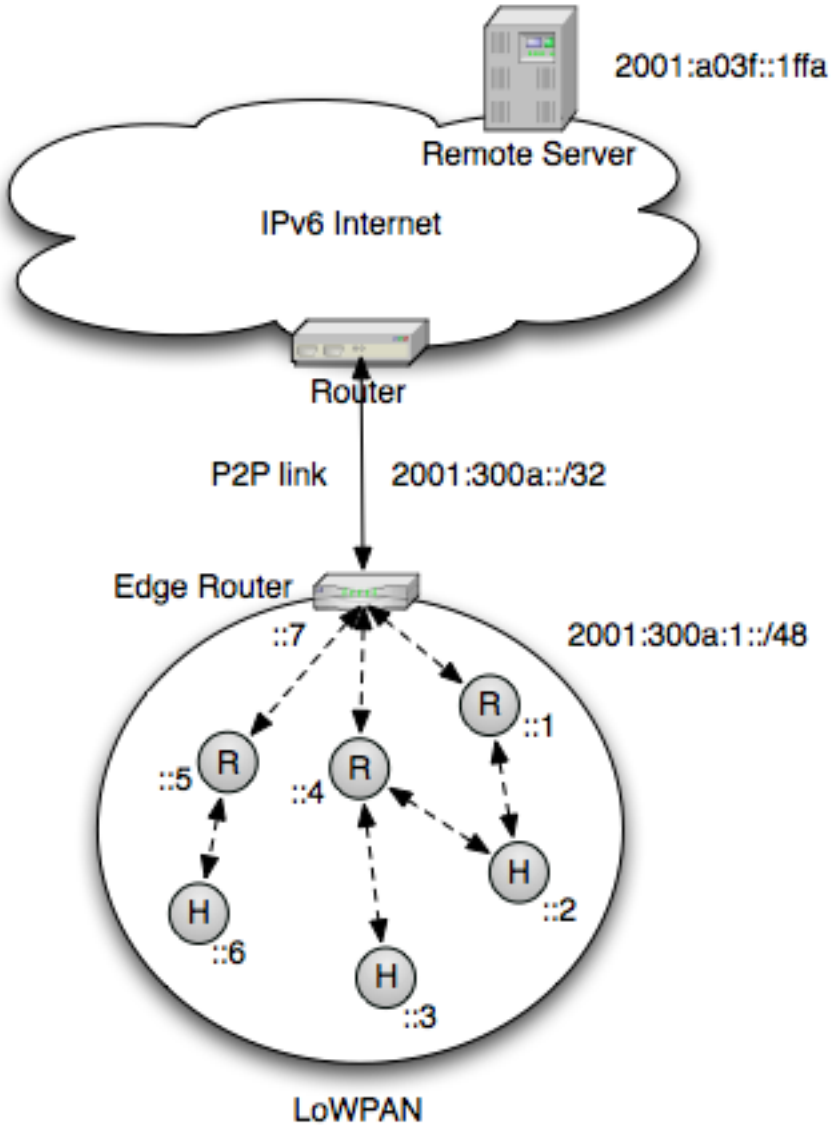
IPv6 Addressing

- 128-bit IPv6 address = 64-bit prefix + 64-bit Interface ID (IID)
- The 64-bit prefix is hierarchical
  - Identifies the network you are on and where it is globally
- The 64-bit IID identifies the network interface
  - Must be unique for that network
  - Typically is formed statelessly from the interface MAC address
    - Called Stateless Address Autoconfiguration (RFC2462)
- There are different kinds of IPv6 addresses
  - Loopback (0::1) and Unspecified (0::0)
  - Unicast with global (e.g. 2001::) or link-local (FE80::) scope
  - Multicast addresses (starts with FF::)
  - Anycast addresses (special-purpose unicast address)

6LoWPAN Addressing

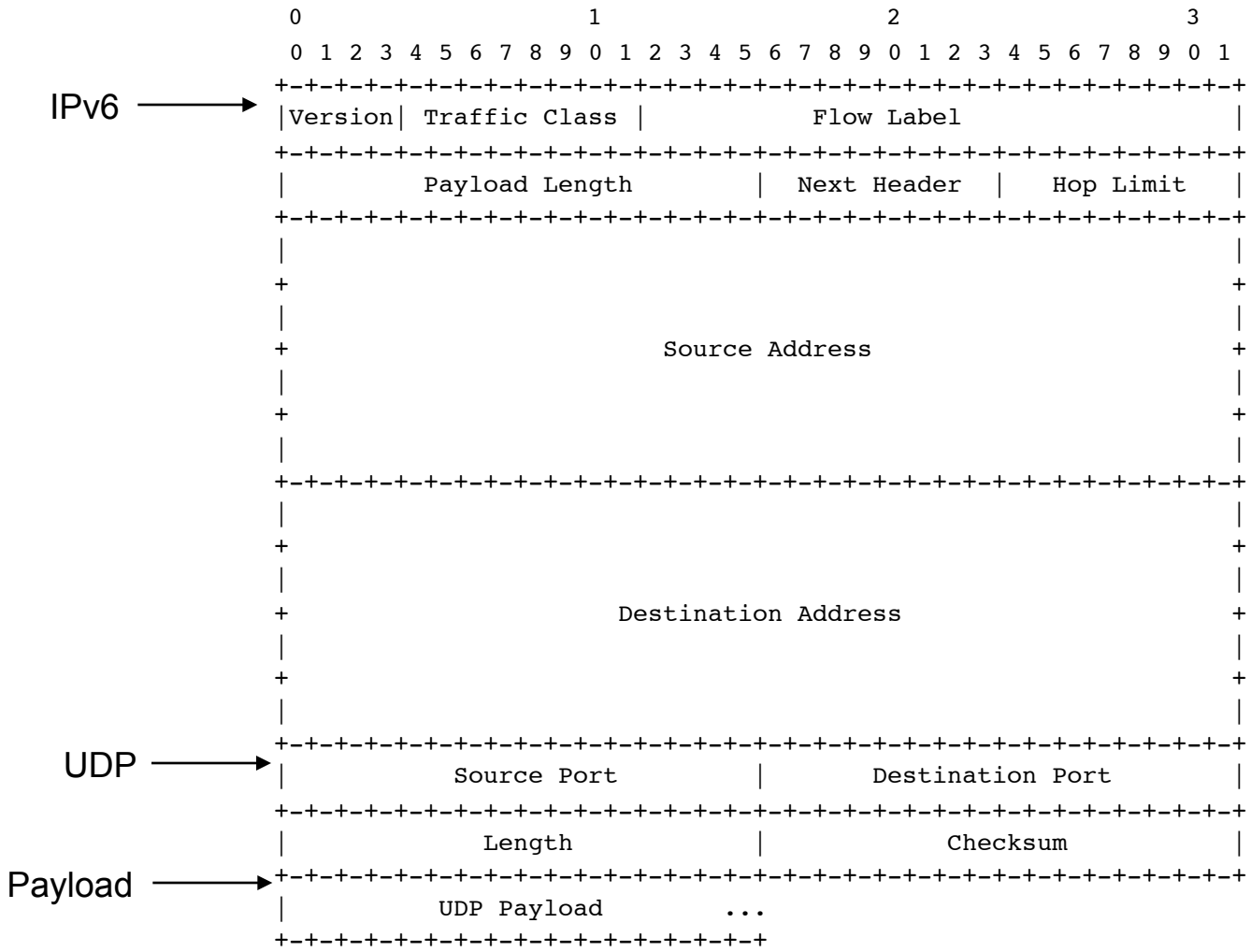
- IPv6 addresses are compressed in 6LoWPAN
- A LoWPAN works on the principle of
  - flat address spaces (wireless network is one IPv6 subnet)
  - with unique MAC addresses (e.g. 64-bit or 16-bit)
- 6LoWPAN compresses IPv6 addresses by
  - Eliding the IPv6 prefix
    - Global prefix known by all nodes in network
    - Link-local prefix indicated by header compression format
  - Compressing the IID
    - Elided for link-local communication
    - Compressed for multihop dst/src addresses
  - Compressing with a well-known “context”
  - Multicast addresses are compressed

Addressing Example

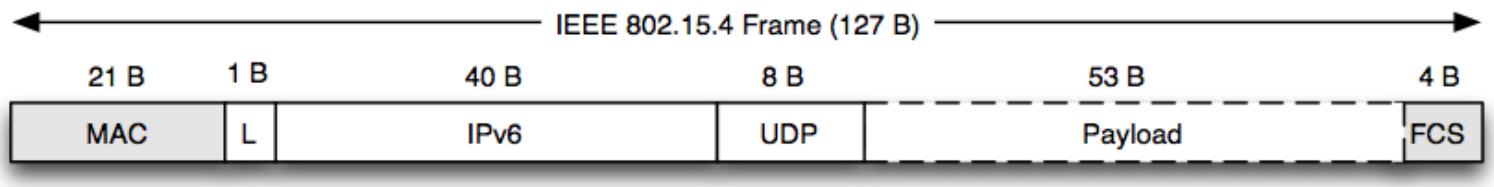


UDP/IPv6 Headers

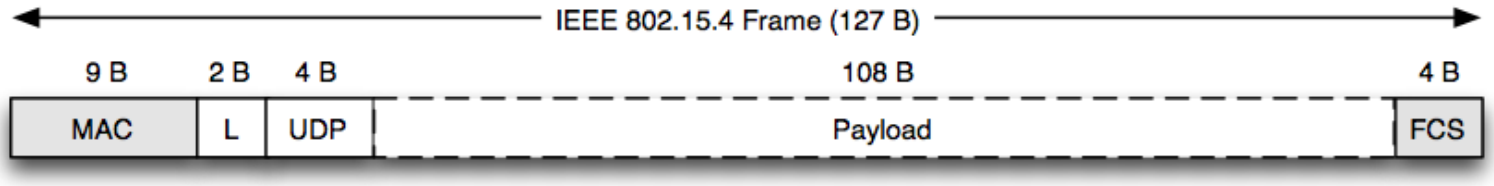
48 Bytes!



Header Comparison



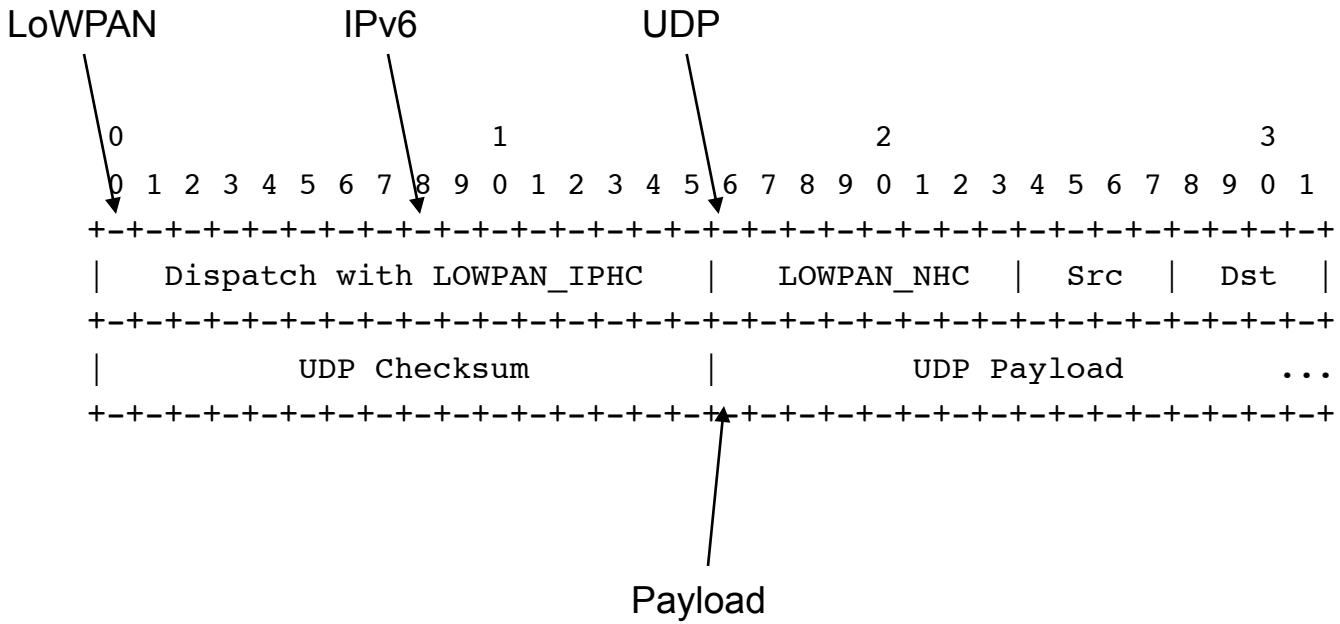
Full UDP/IPv6 (64-bit addressing)



Minimal UDP/6LoWPAN (16-bit addressing)

LoWPAN UDP/IPv6 Headers

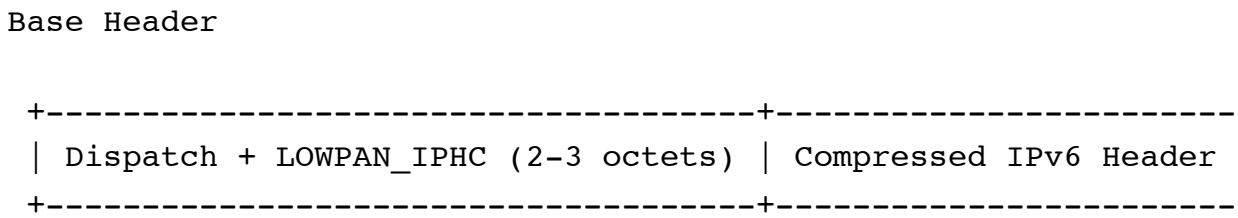
6 Bytes!



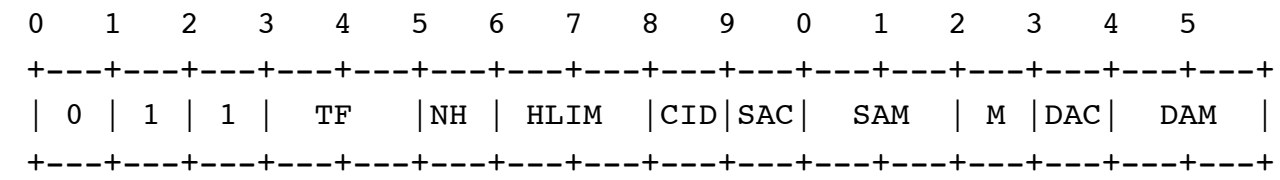
draft-ietf-6lowpan-hc

IP Header Compression (IPHC)

Base Header



LOWPAN_IPHC Encoding

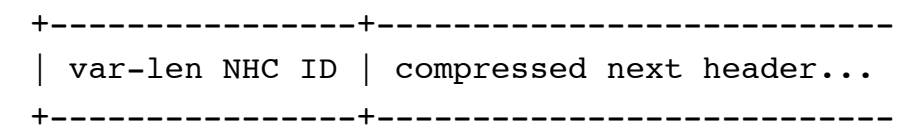


TF = Traffic Class, Flow Label
NH = Next Header Flag
HLIM = Hop Limit
CID = Context Identifier Extension
SAC = Source Address Compression
SAM = Source Address Mode
M = Multicast Compression
DAC = Destination Address Compression
DAM = Destination Address Mode

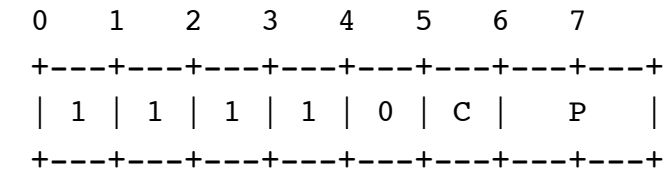
draft-ietf-6lowpan-hc

Next-header Compression (NHC)

NHC Format



UDP NHC Encoding



C = Checksum Compression

P = UDP Port Compression

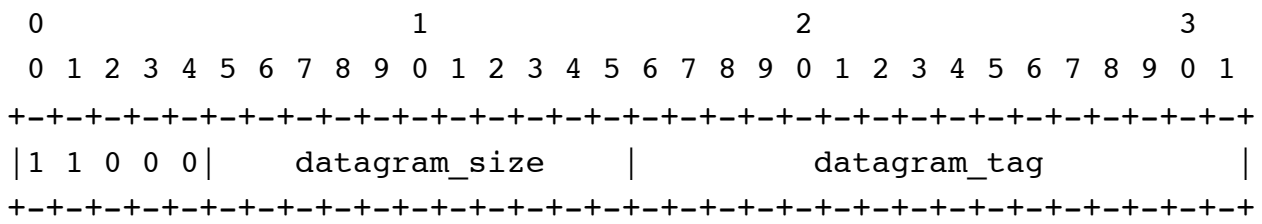
draft-ietf-6lowpan-hc

Fragmentation

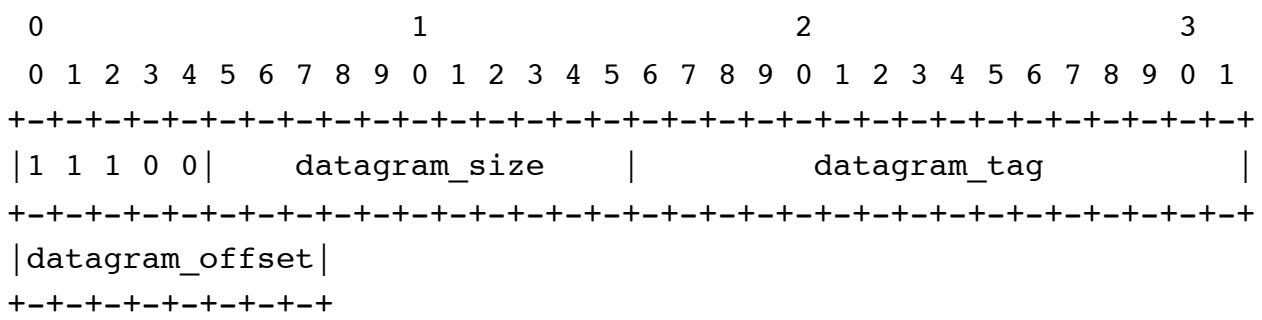
- IPv6 requires underlying links to support Minimum Transmission Units (MTUs) of at least 1280 bytes
- IEEE 802.15.4 leaves approximately 80-100 bytes of payload!
- RFC4944 defines fragmentation and reassembly of IPv6
- The performance of large IPv6 packets fragmented over low-power wireless mesh networks is poor!
  - Lost fragments cause whole packet to be retransmitted
  - Low-bandwidth and delay of the wireless channel
  - 6LoWPAN application protocols should avoid fragmentation
  - Compression should be used on existing IP application protocols when used over 6LoWPAN if possible
- Fragment recovery is currently under IETF consideration

Fragmentation

Initial Fragment



Following Fragments



Bootstrapping

6LoWPAN Setup & Operation

- Autoconfiguration is important in embedded networks
- In order for a 6LoWPAN network to start functioning:
  - 1. Link-layer connectivity between nodes (commissioning)
  - 2. Network layer address configuration, discovery of neighbors, registrations (bootstrapping)
  - 3. Routing algorithm sets up paths (route initialization)
  - 4. Continuous maintenance of 1-3

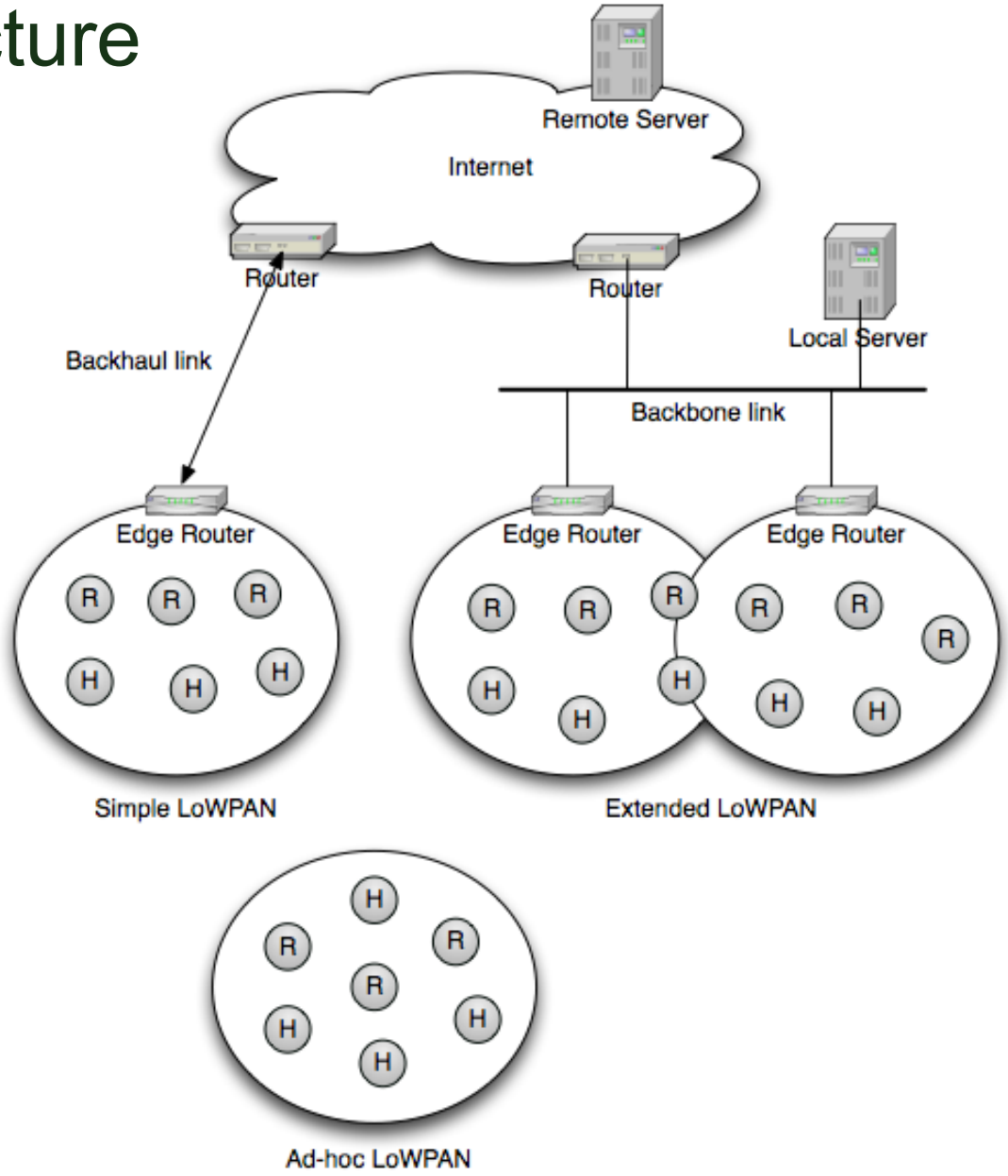
Link-layer Commissioning

- In order for nodes to communicate with each other, they need to have compatible physical and link-layer settings.
- Example IEEE 802.15.4 settings:
  - Channel, modulation, data-rate (Channels 11-26 at 2.4 GHz)
    - Usually a default channel is used, and channels are scanned to find a router for use by Neighbor Discovery
  - Addressing mode (64-bit or 16-bit)
    - Typically 64-bit is a default, and 16-bit used if address available
  - MAC mode (beaconless or super-frame)
    - Beaconless mode is easiest for commissioning (no settings needed)
  - Security (on or off, encryption key)
    - In order to perform secure commissioning a default key should already be installed in the nodes

6LoWPAN Neighbor Discovery

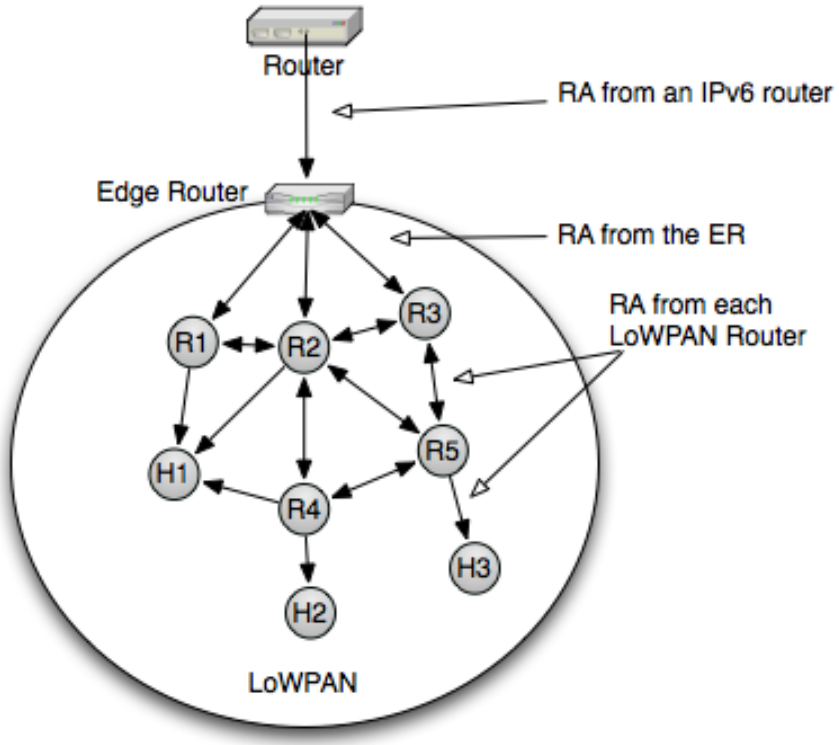
- Standard ND for IPv6 is not appropriate for 6LoWPAN:
  - Assumption of a single link for an IPv6 subnet prefix
  - Assumption that nodes are always on
  - Heavy use of multicast traffic (broadcast/flood in 6LoWPAN)
  - No efficient multihop support over e.g. 802.15.4
- 6LoWPAN Neighbor Discovery provides:
  - An appropriate link and subnet model for low-power wireless
  - Minimized node-initiated control traffic
  - Node Registration (NR) and Confirmation (NC)
  - Duplicate Address Detection (DAD) and recovery
  - Support for extended Edge Router infrastructures
- ND for 6LoWPAN has been specified in draft-ietf-6lowpan-nd (work in progress)

Architecture



Prefix Dissemination

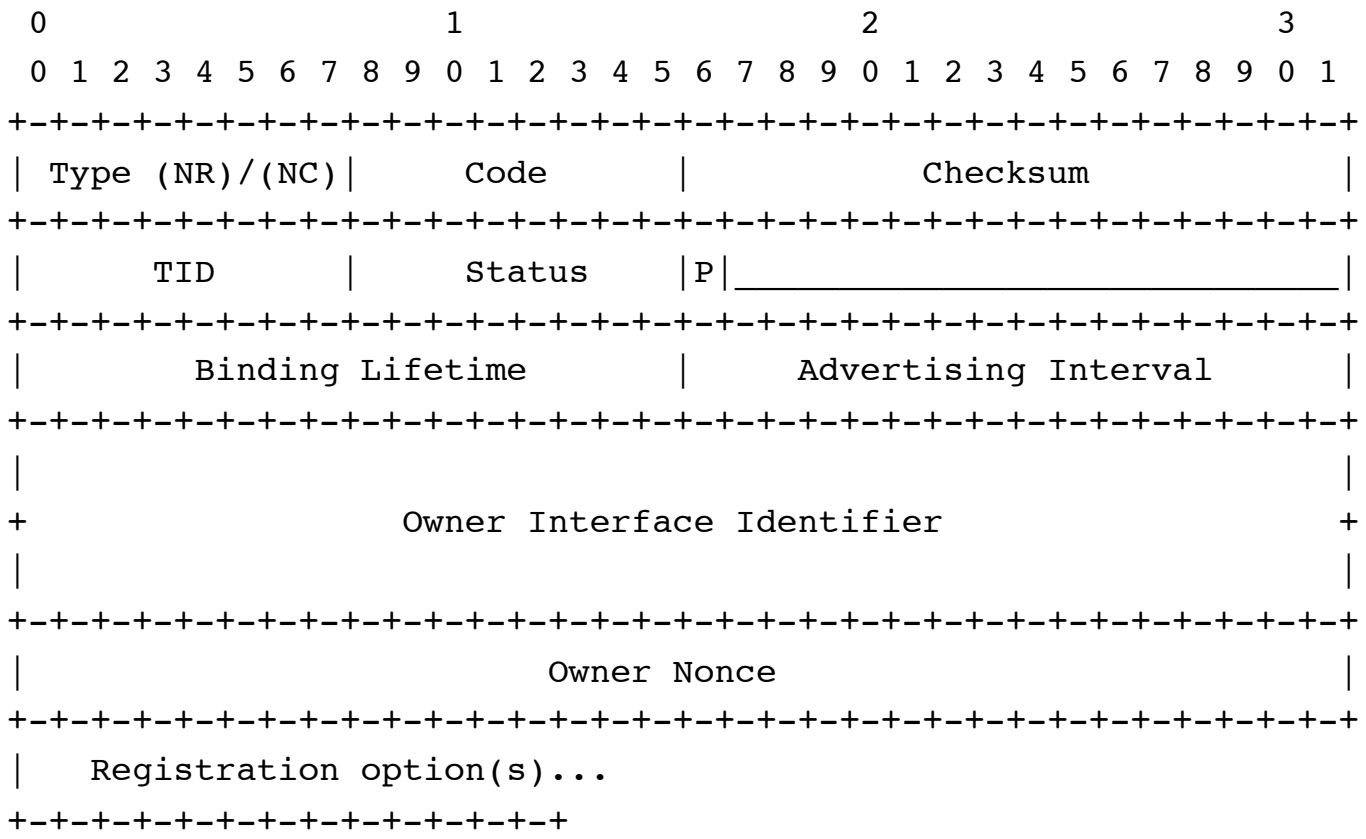
- In normal IPv6 networks RAs are sent to a link based on the information (prefix etc.) configured for that router interface
- In ND for 6LoWPAN RAs are also used to automatically disseminate router information across multiple hops



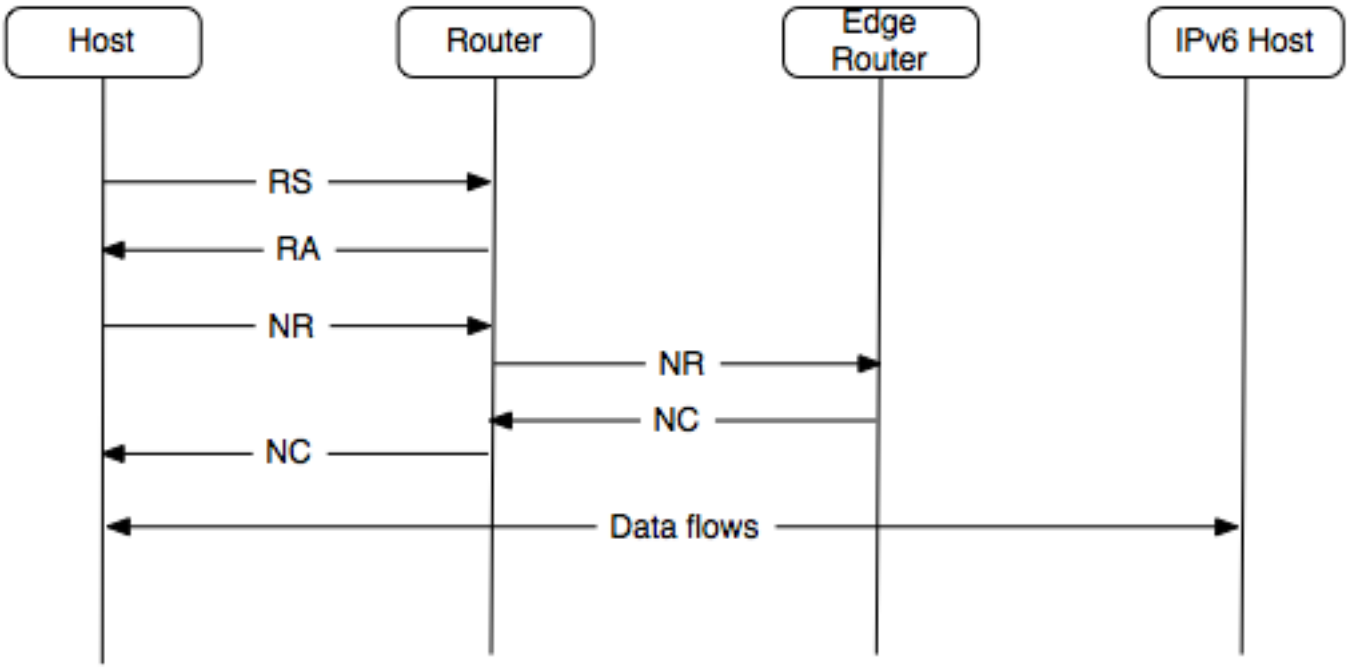
Node Registration

- 6LoWPAN-ND Optimizes only the **host-router** interface
  - RFC4861 = signaling between all neighbors (distributed)
- Nodes register with their neighboring routers
  - Exchange of NR/NC messages
  - Binding table of registered nodes kept by the router
- Node registration exchange enables
  - Host/router unreachability detection
  - Address resolution (a priori)
  - Duplicate address detection
- Registrations are soft bindings
  - Periodically refreshed with a new NR message

NR/NC Format

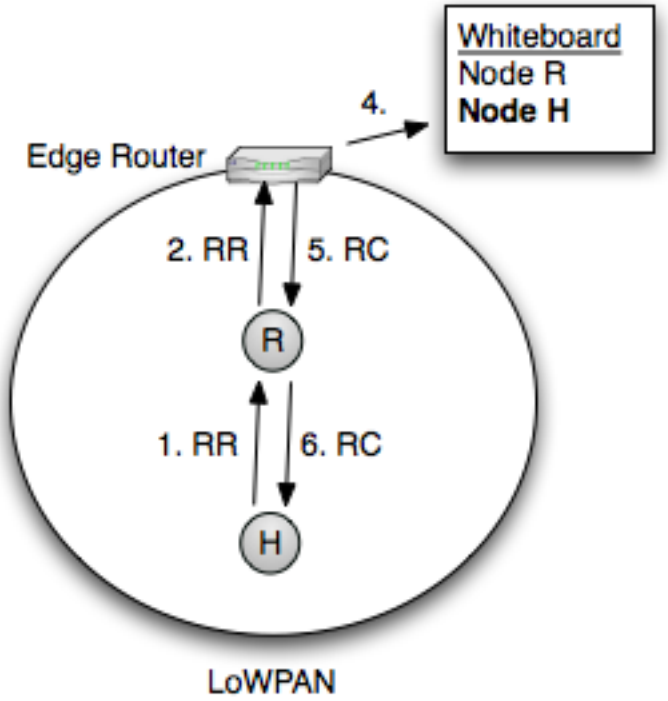

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type (NR)/(NC)
Code
Checksum
TID
Status
P
Binding Lifetime
Advertising Interval
Owner Interface Identifier
Owner Nonce
Registration option(s)...

Typical 6LoWPAN-ND Exchange



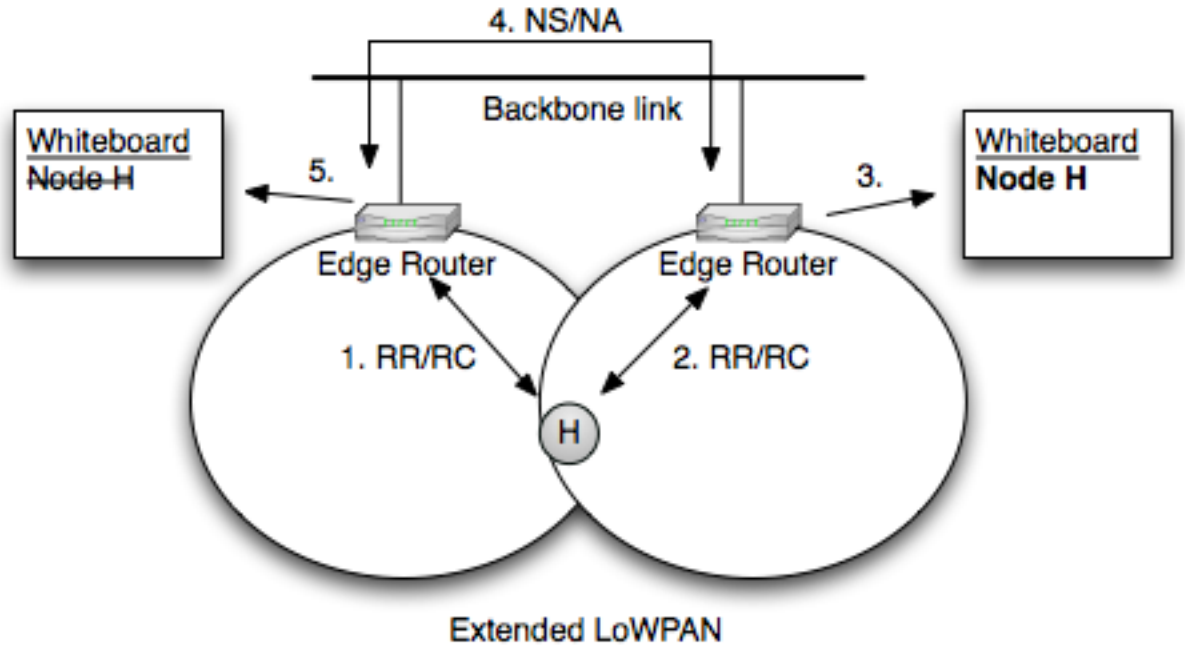
The Whiteboard

- The whiteboard is used in the LoWPAN for:
  - Duplicate address detection for the LoWPAN (= prefix)
  - Dealing with mobility (Extended LoWPANs)
  - Short address generation
  - Locating nodes



Extended LoWPANs

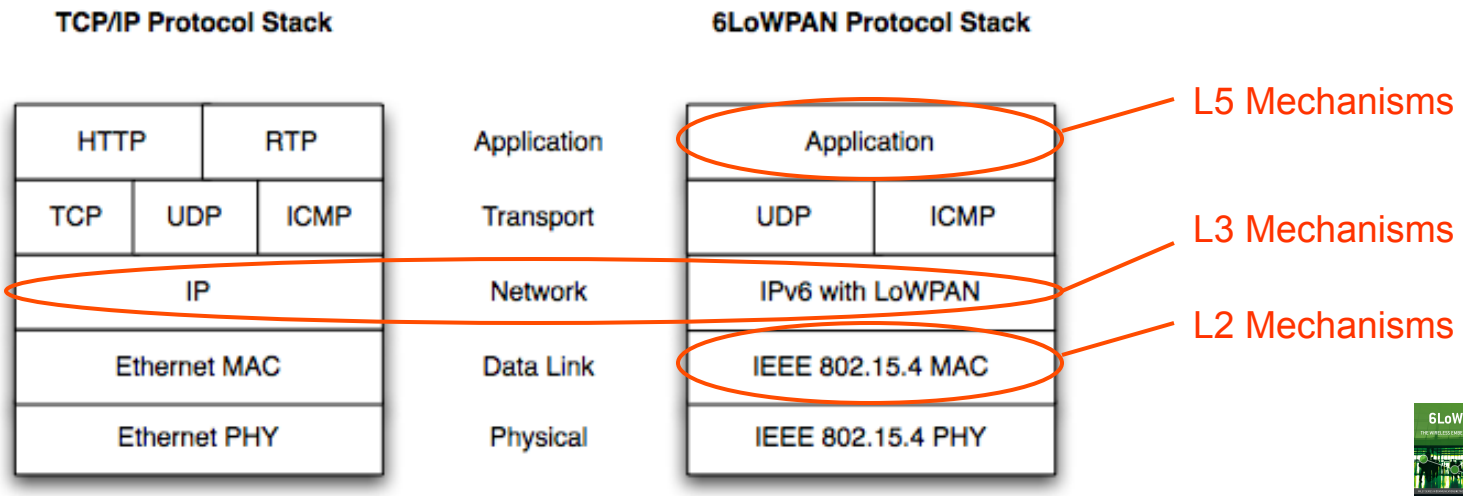
- Extended LoWPANs consist of two or more LoWPANs:
  - Which share the same IPv6 prefix
  - Which are connected together by a backbone link
- Whiteboards are synchronized over the backbone link



Security

Security for 6LoWPAN

- Security is important in wireless embedded networks
  - Wireless radios are easily overheard
  - Autonomous devices with limited processing power
- A system usually has three main security goals
  - Confidentiality
  - Integrity
  - Availability
- See the threat model for Internet security in RFC3552



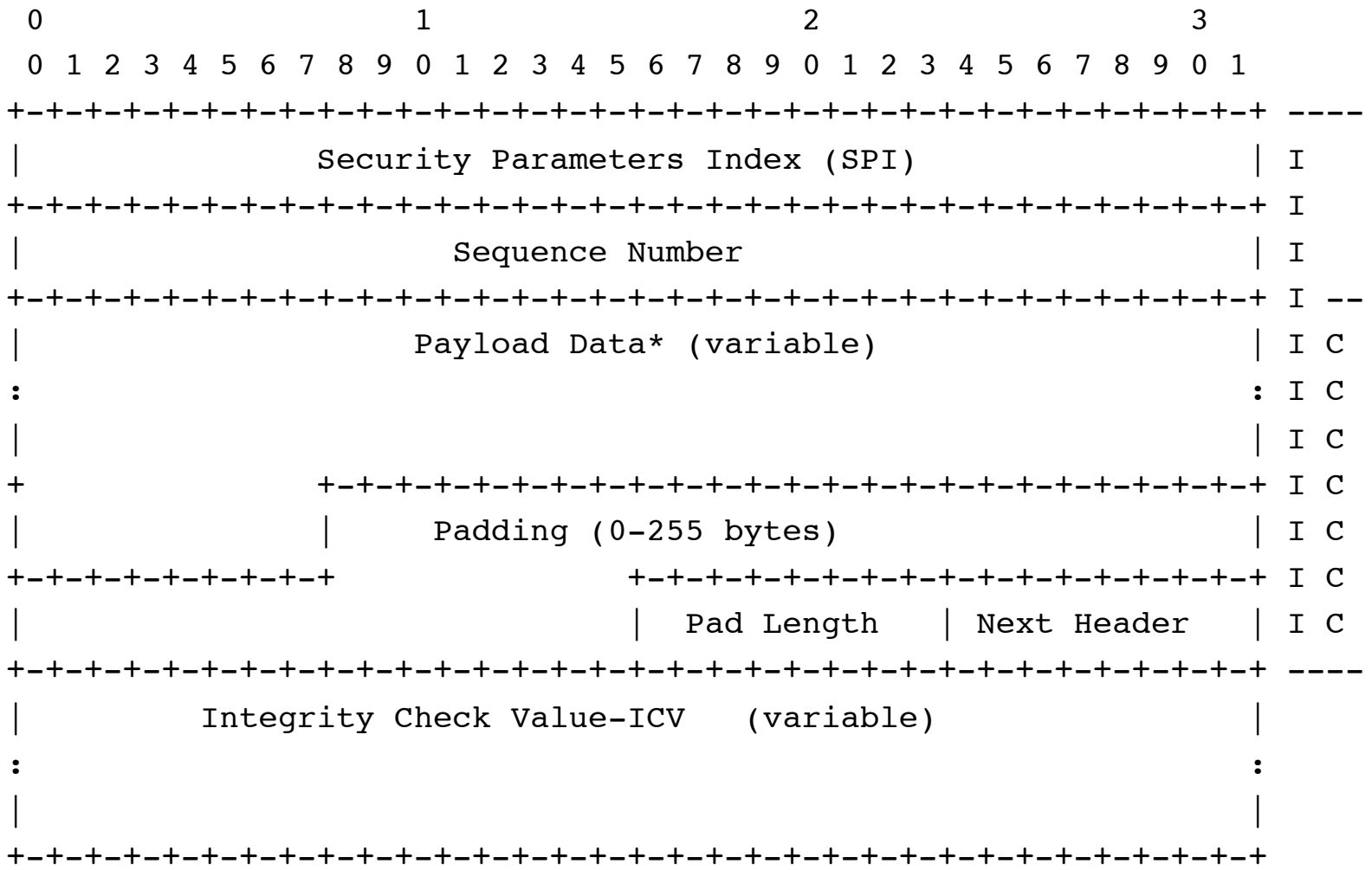
Layer-2 Mechanisms

- Internet security is usually thought of as end-to-end
- In wireless networks the channel itself is very vulnerable
  - The channel is easy to overhear
  - Nodes and packets are easy to spoof
- The goals of security at the data-link layer
  - Protect the wireless network against attackers
  - Increase robustness against attacks
- IEEE 802.15.4 provides built-in encryption
  - Based on the 128-bit Advanced Encryption Standard (AES)
  - Counter with CBC-MAC mode (CCM)
    - Provides both encryption and an integrity check
  - Most chips include an AES-128 hardware engine

Layer-3 Mechanisms

- End-to-end security can be provided by IP
  - Protects the entire path between two end-points
- The IPsec standard [RFC4301] defined IP security
- Two packet formats are defined:
  - Authentication Header (AH) in [RFC4302]
    - Integrity protection and authentication only
  - Encapsulating Security Payload (ESP) [RFC4303]
    - Also encrypts for confidentiality
- ESP is most widely used
- A mode of ESP defines using AES/CCM [RFC4309]
  - Suitable for use with 6LoWPAN nodes
  - The same L2 IEEE 802.15.4 hardware engine can be applied!

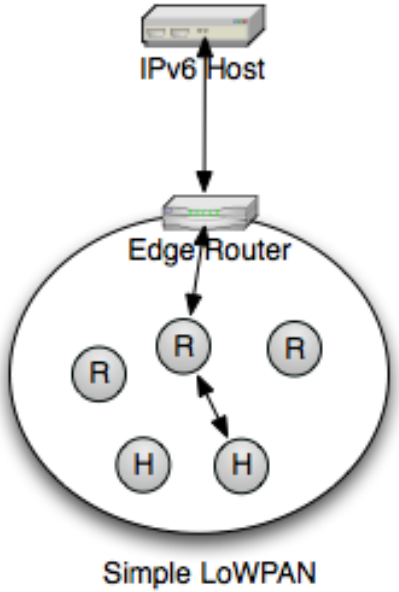
ESP Format

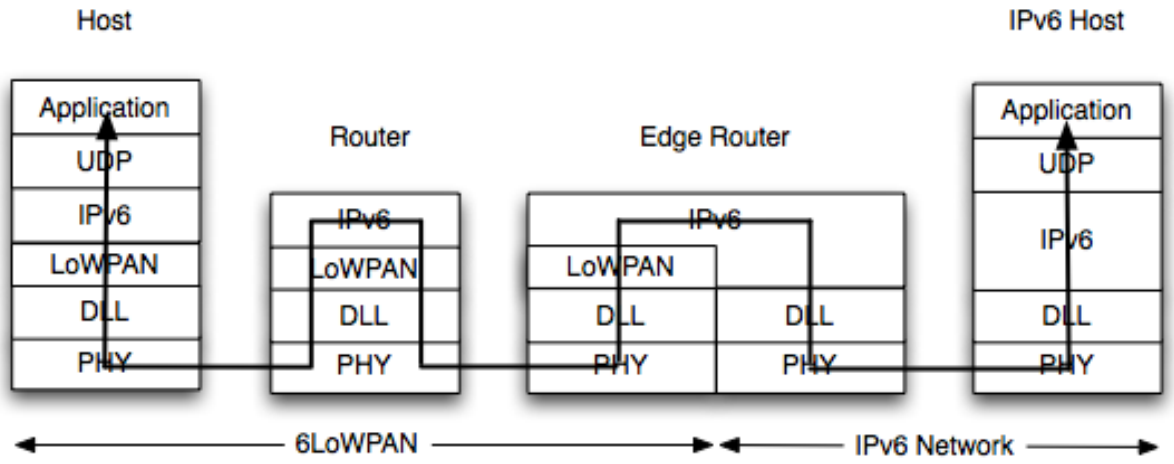

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Security Parameters Index (SPI)
I
Sequence Number
I
Payload Data* (variable)
I C
Padding (0-255 bytes)
I C
Pad Length
Next Header
I C
Integrity Check Value-ICV (variable)

RPL: IPv6 routing Protocol for Low power and Lossy Networks

6LoWPAN Routing

- Here we consider IP routing (at layer 3)
- Routing in a LoWPAN
  - Single-interface routing
  - Flat address space (exact-match)
  - Stub network (no transit routing)





Types of Routing Protocols

- Algorithm classes
  - Distance-vector

    Links are associated with cost, used to find the shortest route. Each router along the path store local next-hop information about its route table.
  - Link-state

    Each node aquires complete information about the network, typically by flooding. Each node calculated a shortest-path tree calculated to each destination.
- Types of Signaling
  - Proactive

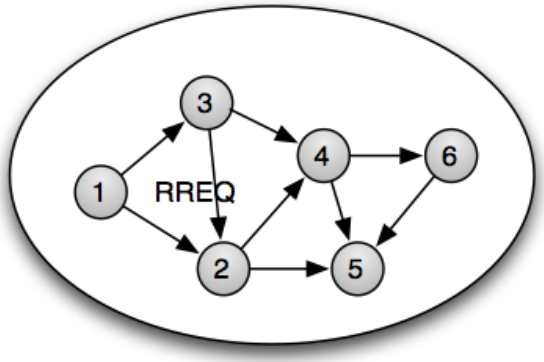
    Routing information aquired before it is needed.
  - Reactive

    Routing information discovered dynamically when needed.
- Route metrics are an important factor

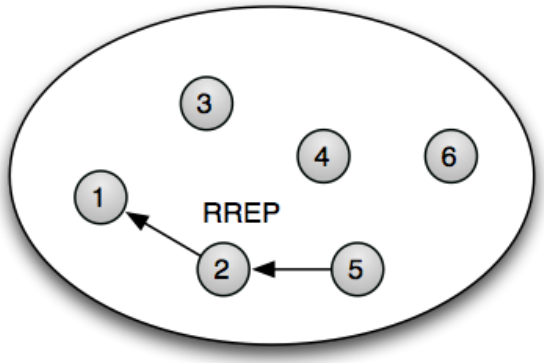
Protocols for 6LoWPAN

- IP is agnostic to the routing protocol used
  - It forwards based on route table entries
- Thus 6LoWPAN is routing protocol agnostic
- Special consideration for routing over LoWPANs
  - Single interface routing, flat topology
  - Low-power and lossy wireless technologies
  - Specific data flows for embedded applications
- MANET protocols useful in some ad-hoc cases
  - e.g. AODV, DYMO
- New IETF working group formed
  - Routing over low-power and lossy networks (ROLL)
  - Deloped specifically for embedded applications

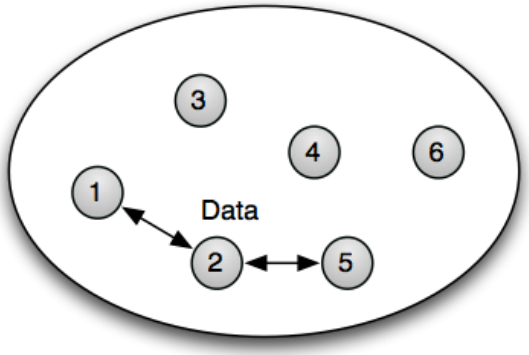
Reactive MANET Protocols



1. RREQ for node 5 broadcast over multiple hops.



2. RREP unicast back to node 1, creates route entries.



3. Route entries in 1, 2 and 5 enable forwarding.

Survey of existing routing protocol

Protocol	State	Loss	Control	Link Cost	Node Cost
OSPF/IS-IS	fail	fail	fail	pass	fail
OLSRv2	fail	?	?	pass	pass
TBRPF	fail	pass	fail	pass	?
RIP	pass	fail	pass	?	fail
AODV	pass	fail	pass	fail	fail
DYMO	pass	?	pass	?	?
DSR	fail	pass	pass	fail	fail

- Routing state: limited memory resources of low-power nodes.
- Loss Response: what happens in response to link failures
- Control cost: constraints on control traffic
- Link and Node cost: link and node properties are considered when choosing routes

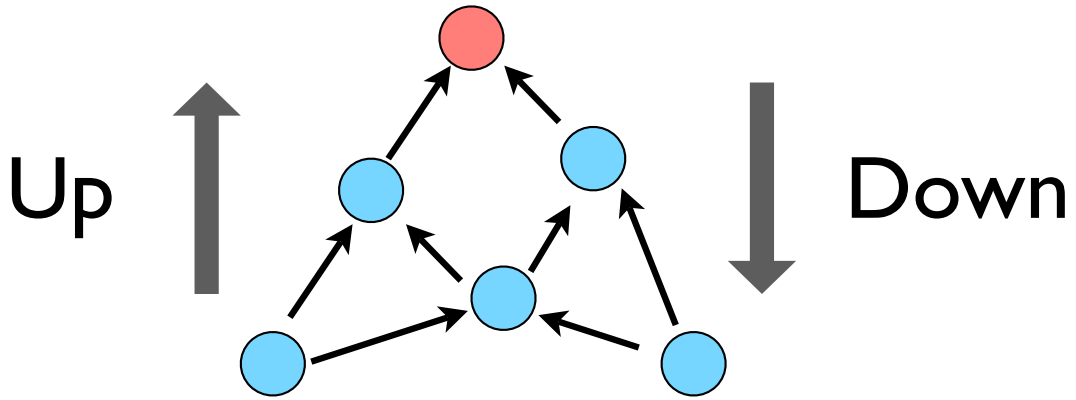
Source: Slides ROLL-IETF 72

IETF ROLL

- Routing Over Low power and Lossy networks (ROLL)
  - Working group at the IETF
- Standardizing a routing algorithm for embedded apps
- Application specific requirements
  - Home automation
  - Commercial building automation
  - Industrial automation
  - Urban environments
- Analyzed all existing protocols
- Solution must work over IPv6 and 6LoWPAN
- Protocol in-progress called RPL “Ripple”
  - Proactive distance-vector approach
  - See draft-ietf-roll-rpl for detailed information

RPL : IPv6 routing protocol for LLN

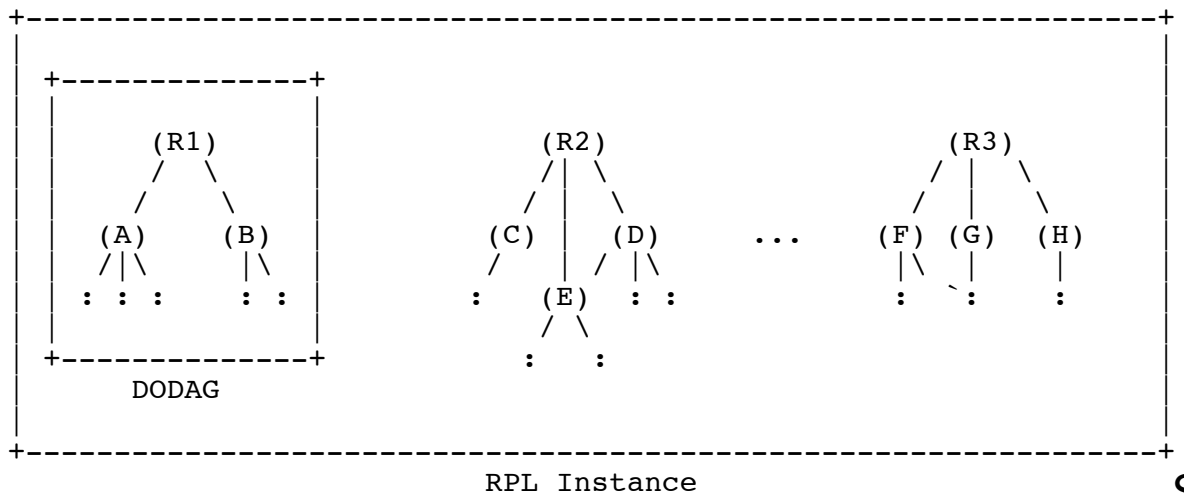
- Assumption : most traffic flows through few nodes
  - many-to-one : MP2P
  - One-to-many : P2MP



- Approach : build DAG(s) rooted at these nodes
  - Up towards the DAG root for many-to-one
  - Down away from the DAG root for one-to-many
  - Use the DAG to detect and avoid loops
  - Allow point-to-point via up and down

Definitions

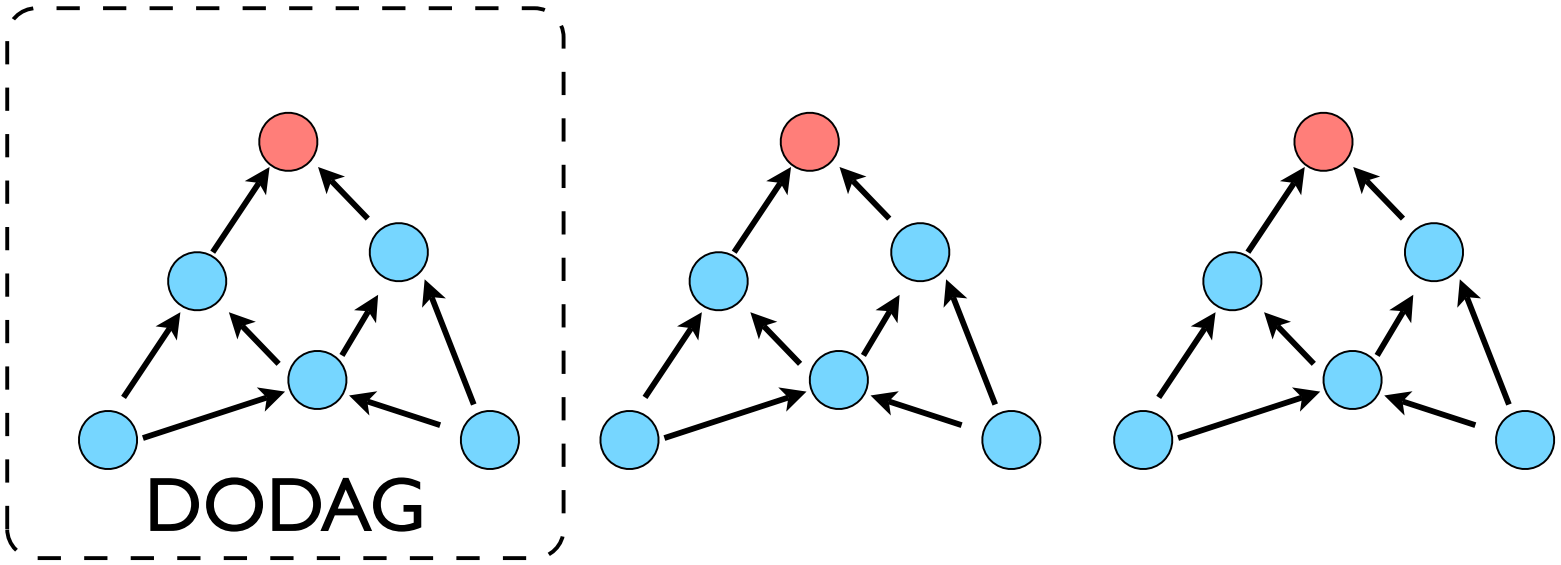
- Instance
  - Defines the optimization objective Function when forming paths towards roots
    - Link properties : (reliability, latency)
    - Node properties : (Powered or not)
  - Objective : optimize paths based on one or more metrics
  - Scope : LLN network
  - Composed of one or more disjoint DODAGs sharing the same Objective Function



Source: draft-RPL

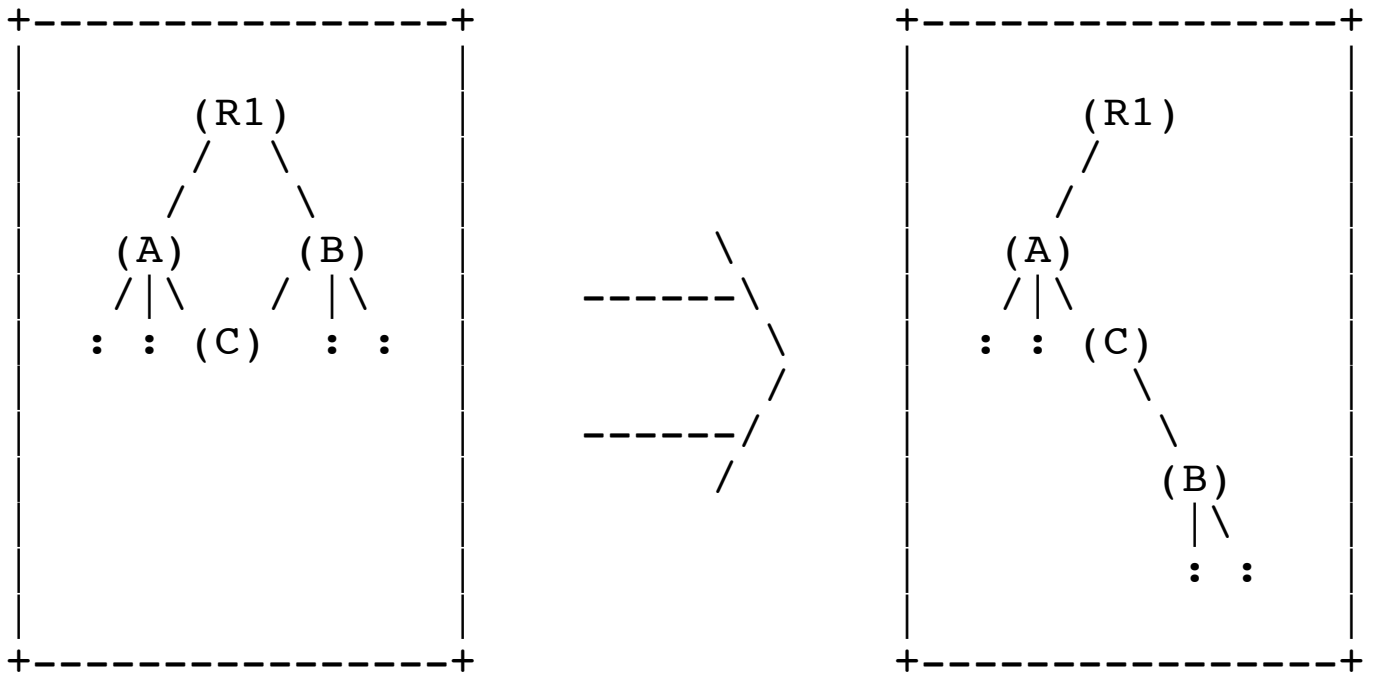
Definitions

- DODAG
  - Defines a disjoint DAG that forms paths to a single logical root
  - Scope : within an Instance



Definitions

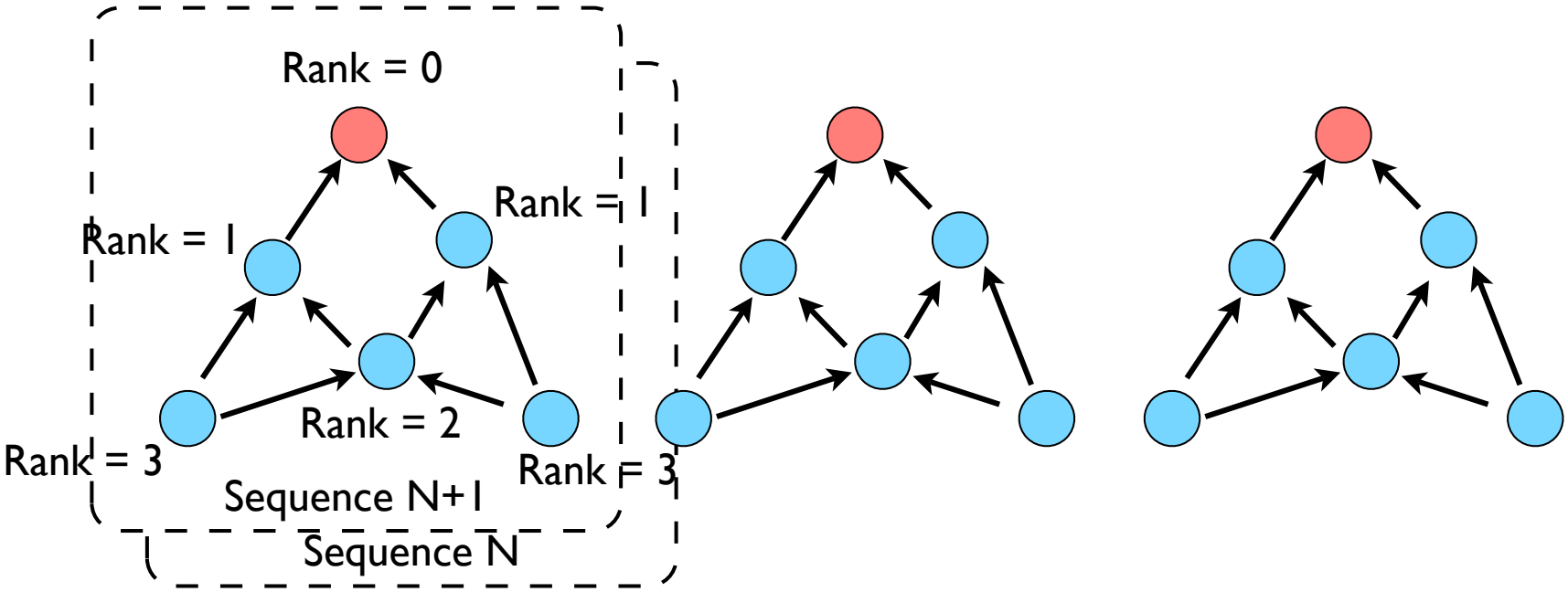
- DODAG version
  - A DODAG constructed using a particular sequence
  - May be a different DODAG topology
  - Triggered by the root
  - Scope : within a DODAG



Source: draft-RPL

Definitions

- Node Rank
  - Defines a node's relative position from the root within a DODAG version
  - Scope : within a DODAG



Definitions

- Instance: defines optimization objective for the network
- DODAG: a disjoint DAG within an instance
- DODAG version : a DODAG built with a particular sequence number
- Rank: a node position within a DODAG version
- Objective Function: identifies metrics, constraints, and objectives

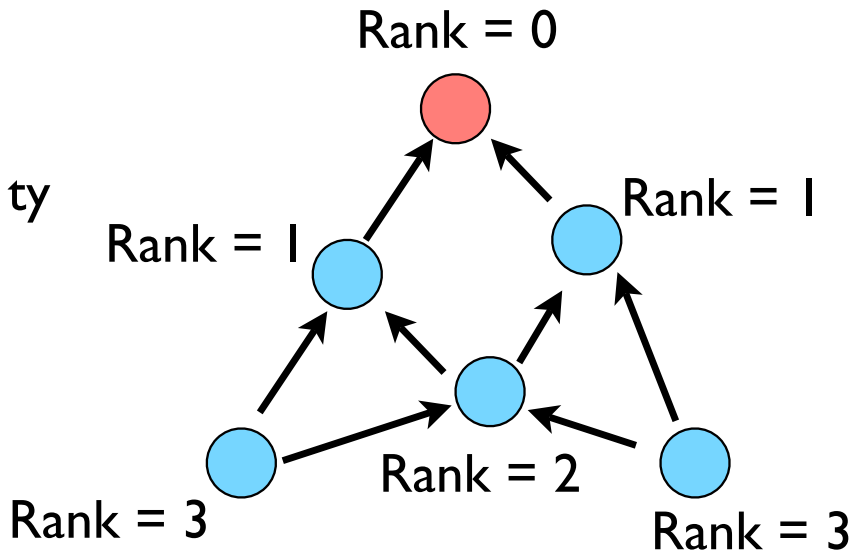
Metric vs. Rank

- Metric is used to achieve an optimization goal
  - Announced by the root
- Rank: path calculation according to objective metric
  - Scalar that represents relative position within a DAG
  - Strictly increasing from the root
  - Topological constraint to avoid and detect loops
  - Allows sibling: in addition to parents and children

Rank(Node) = **Rank(parent)**+(RANK_MAX_INC – RANK_MIN_INC) *
(1 – **linkQuality(parent)**)^2 + RANK_MIN_INC

DAG construction

- Distance-Vector
  - Advertise path cost to root
  - Choose parents that minimize path cost
  - But be careful about loops and count-to-infinity
- A rank assigned to every node
  - decreases towards root



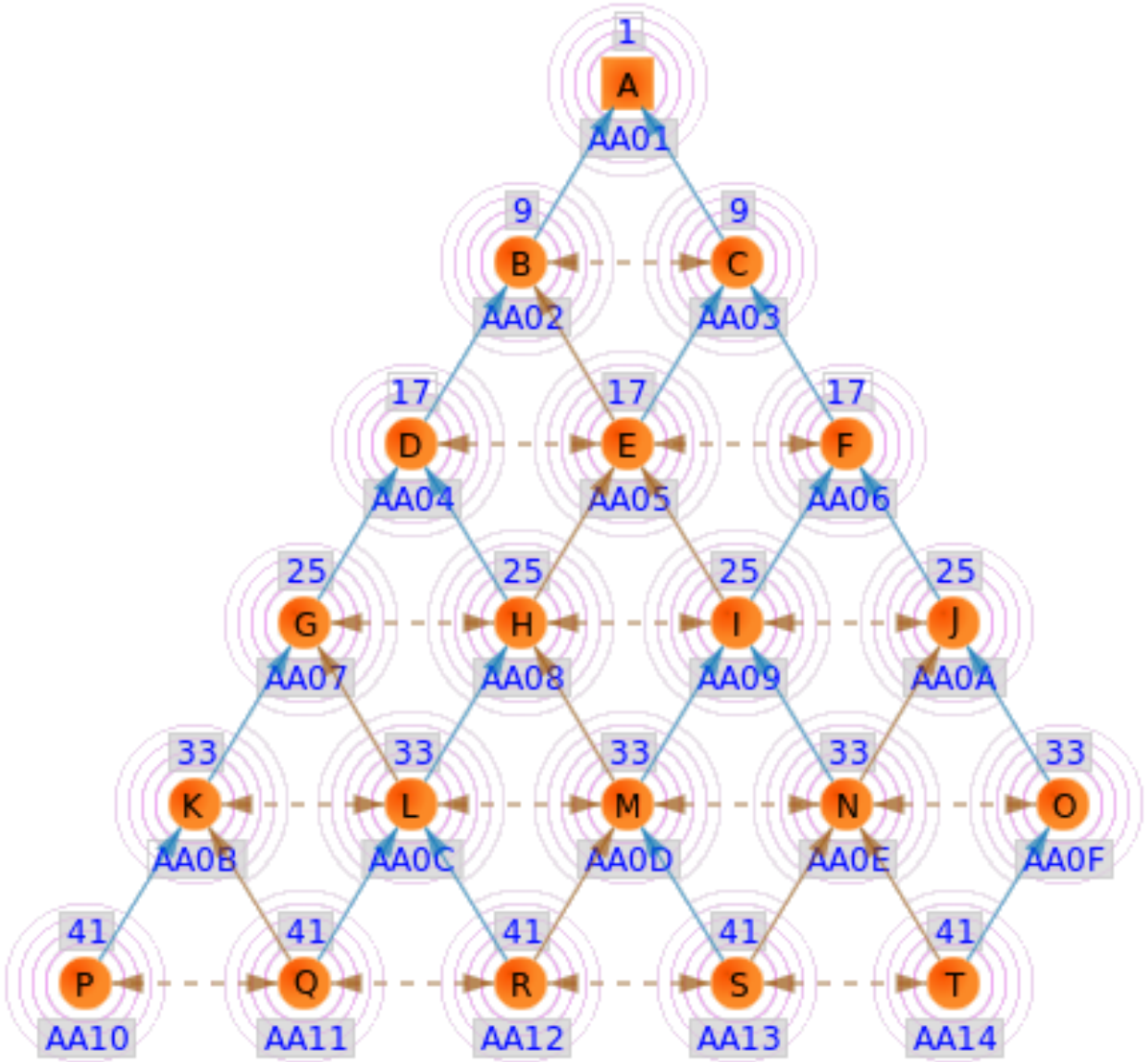
DODAG : distributed algorithm operation

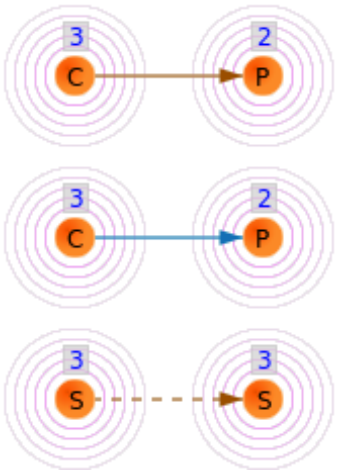
- Some node configured to be DODAG roots
- Nodes advertise their presence using a link-local multicast DIO
- Nodes listen for DIO and join a new DODAG
  - Select DODAG parents
  - Maintain an existing DODAG
- Nodes provision routing table for destinations specified by DIO via their DODAG parents

Route construction

- Up routes towards nodes of increasing rank
  - DODAG parents
- Down routes towards nodes of increasing rank
  - Nodes inform parents of their presence and reachability to descendants
  - Record route/source route for nodes that cannot maintain any down routes.

Example : triangular pattern





Parent link

Preferred Parent link

Sibling link

Forwarding rules

- All routes
  - up/down along DODAG
- When going up
  - Always forward to lower rank when possible
  - May forward to sibling if no lower rank exists
- When going down
  - Forward based on down routes

Protocol mechanisms

- Control messages
  - ICMPv6 RPL control message
  - DODAG Information Solicitation (DIS)
  - DOADAG Information Object (DIO)
  - Destination Advertisement Object (DAO)
  - RPL Options
- DODAG repair
- Loop detection and recovery

Conveyance

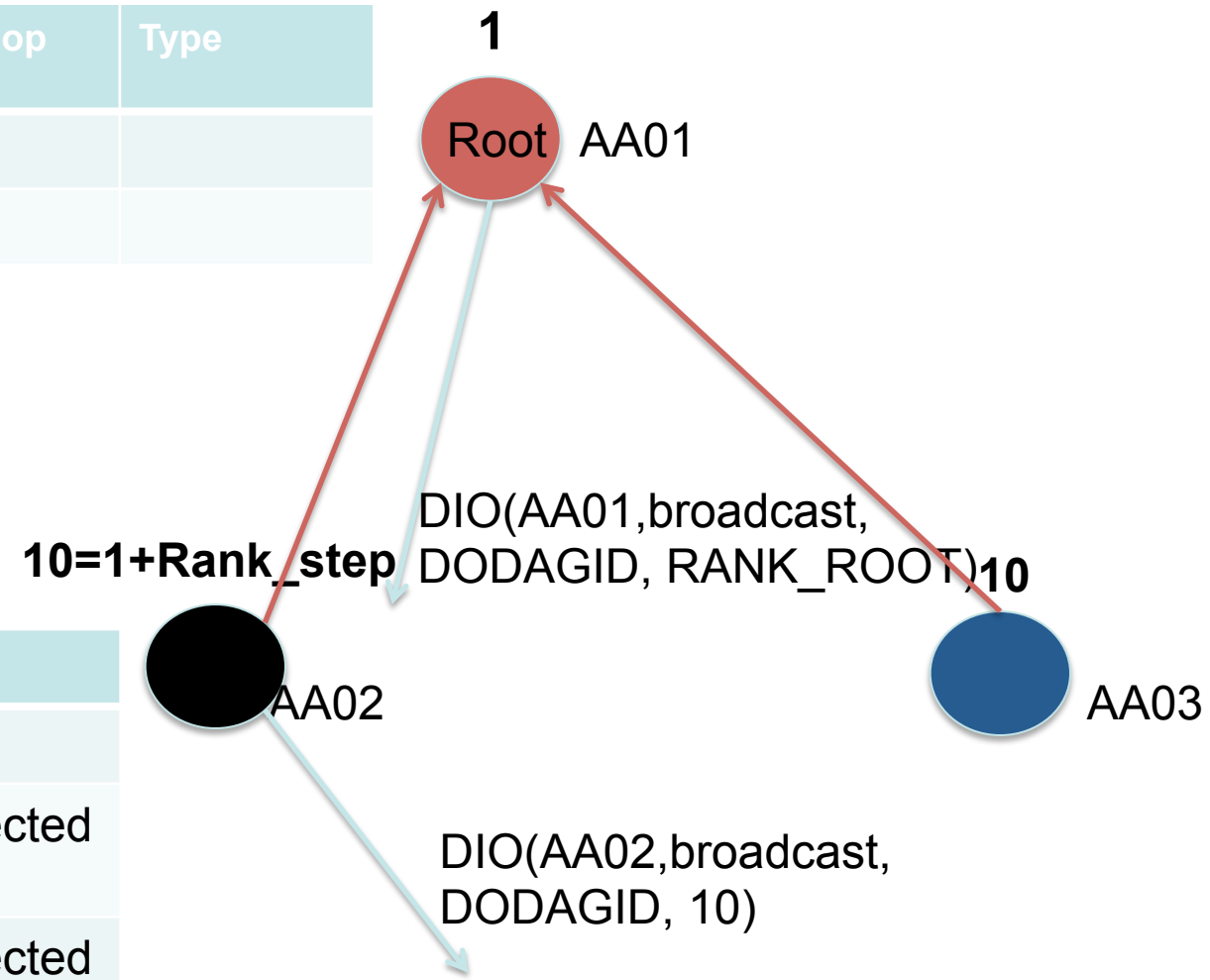
- New ICMPv6 type for RPL : ICMPv6 code 155
- Source address : link local
  - Except DAO in non-storing mode
- Destination address : link local unicast or all-RPL-nodes multicast address
- Code field : type of RPL message
  - 0x0 : DODAG Information Solicitation
  - 0x01 : DODAG Information Object
  - 0x03 : Destination Advertisement Object

DODAG Information Object: DIO

- Discover a RPL instance : nodes listen for DIOs
- Learn RPL instance configuration parameters
- Select a DODAG parent set
  - Join a DODAG
- Maintain the DODAG : DIO transmission
  - Trickle timer or DIS message
- DODAGID (128 bits ipv6 address of the DODAG root)
- Indicates the DODAG rank of the node sending the DIO message
- May contain options: metric, routing information, DODAG configuration, prefix information

Upward routes: MP2P traffic

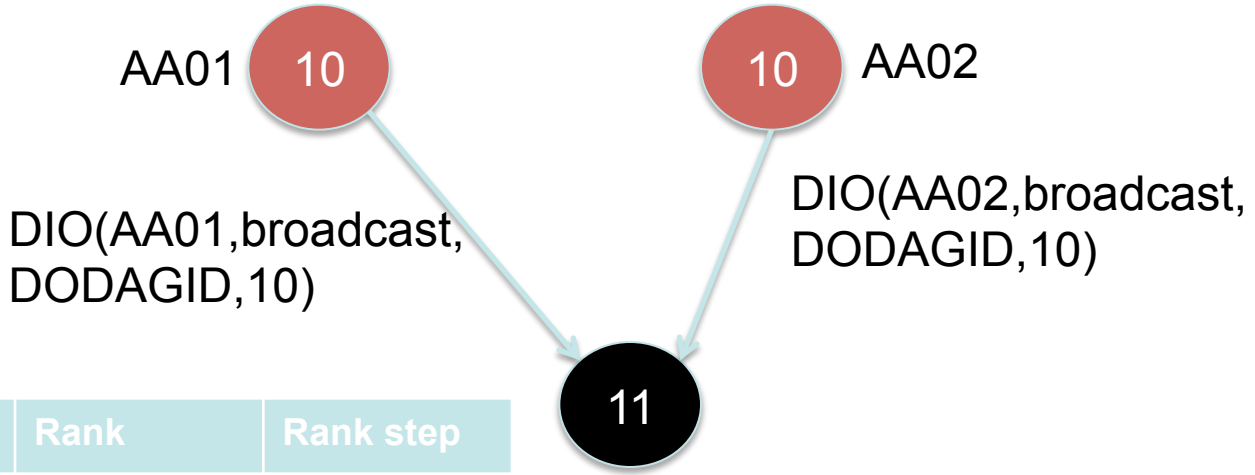
destination	Next Hop	Type



destination	Next Hop	Type
default	A001	DIO
AA03	AA03	connected
AA01	AA01	connected

Parent selection

- Select a lower rank node which minimize the node rank



	Parent	Rank	Rank step
	AA01	10	2
Best	AA02	10	1

DIO transmission triggers

- DIOs are transmitted using a Trickle timer
  - $\mathbf{I_{min}}$:learned from received DIO
  - $\mathbf{I_{max}}$: learned from received DIO
  - k: redundancy constant
- Consistent transmission
  - DIO from sender with lesser rank
    - No changes in parents set
    - Preferred parent
    - Rank
- Inconsistent transmission
  - Inconsistency detection when forwarding a packet
  - Receive a multicast DIS (Destination Information solicitation)
  - Join a new DODAG version

Trickle algorithm

- Dynamic timers
- Establishing consistency in a wireless network
  - Quickly
  - Low overhead when consistent
  - Logarithmic cost with density
  - Very little RAM requirements
  - No topology assumptions
- No synchronisation needed between nodes

P. Levis, T. clausen, J. Hui, O. Gnawali, Ko. J, "Trickle algorithme", draft IETF 2011"

Consistency

- Routing tree maintenance
  - Invariant: Next hop has lower rank (RPL)
- Network configuration
  - Invariant: all have the most recent configuration
- Network monitoring
  - Invariant: data is up-to-date
- Neighbour discovery
  - Invariant: node is in all neighbour's lists

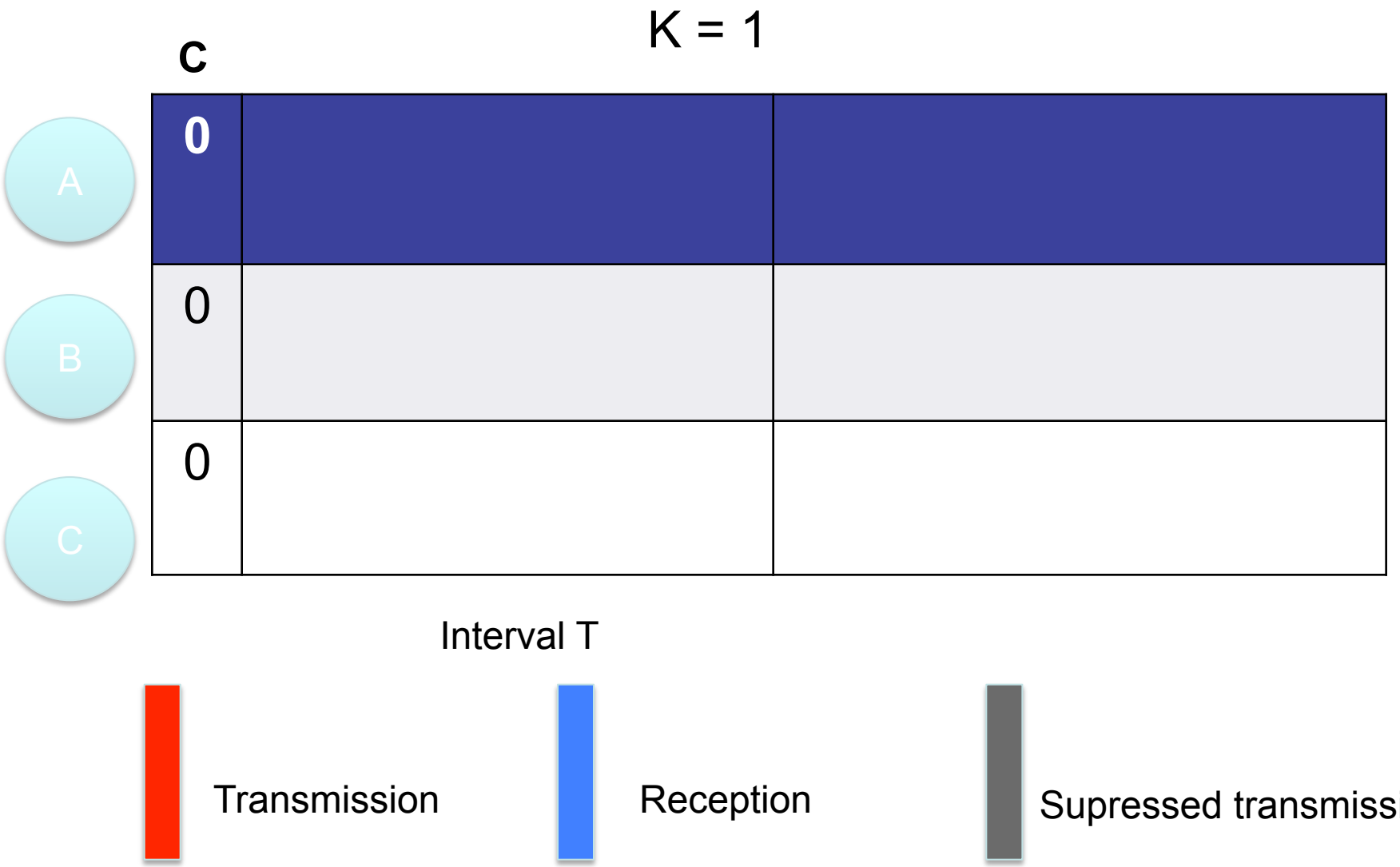
Overview

- Trickle operates over time intervals
- In each interval, node optionally transmits
  - If it has not heard transmissions that are consistent with its own
- Dynamically scales interval lengths to have fast updates yet low cost when consistent
  - When heard inconsistent transmission, sets interval to I_{min}: inconsistent network
  - Otherwise double the interval length: consistent network

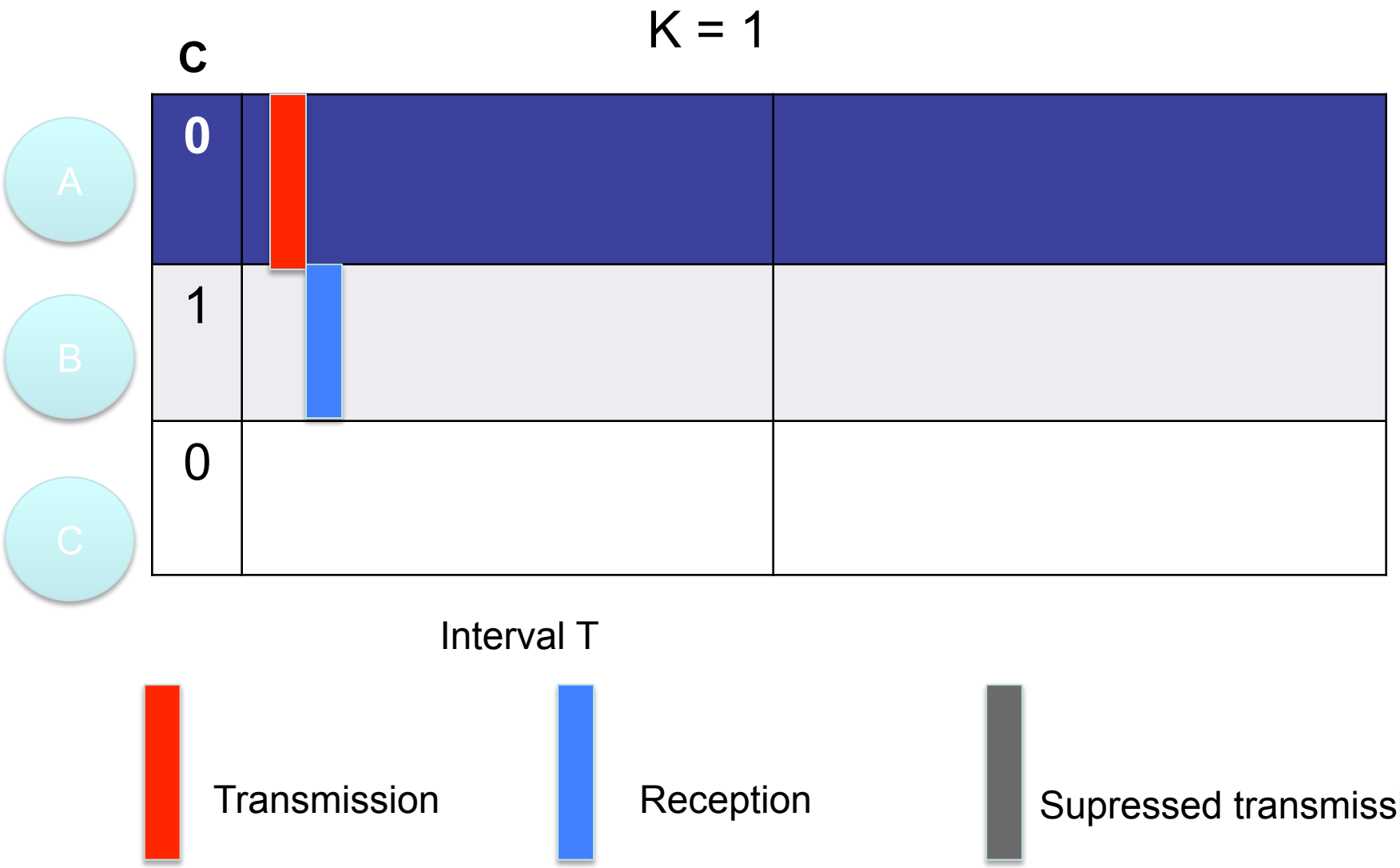
Suppression

- Motivation: don’t waste messages if all nodes agrees
- Interval length T
  - At beginning of interval, counter c=0
  - On consistent transmission, c++
- Node picks a time t in range [T/2,T]
  - At t, transmit if c < k (redundancy constant)

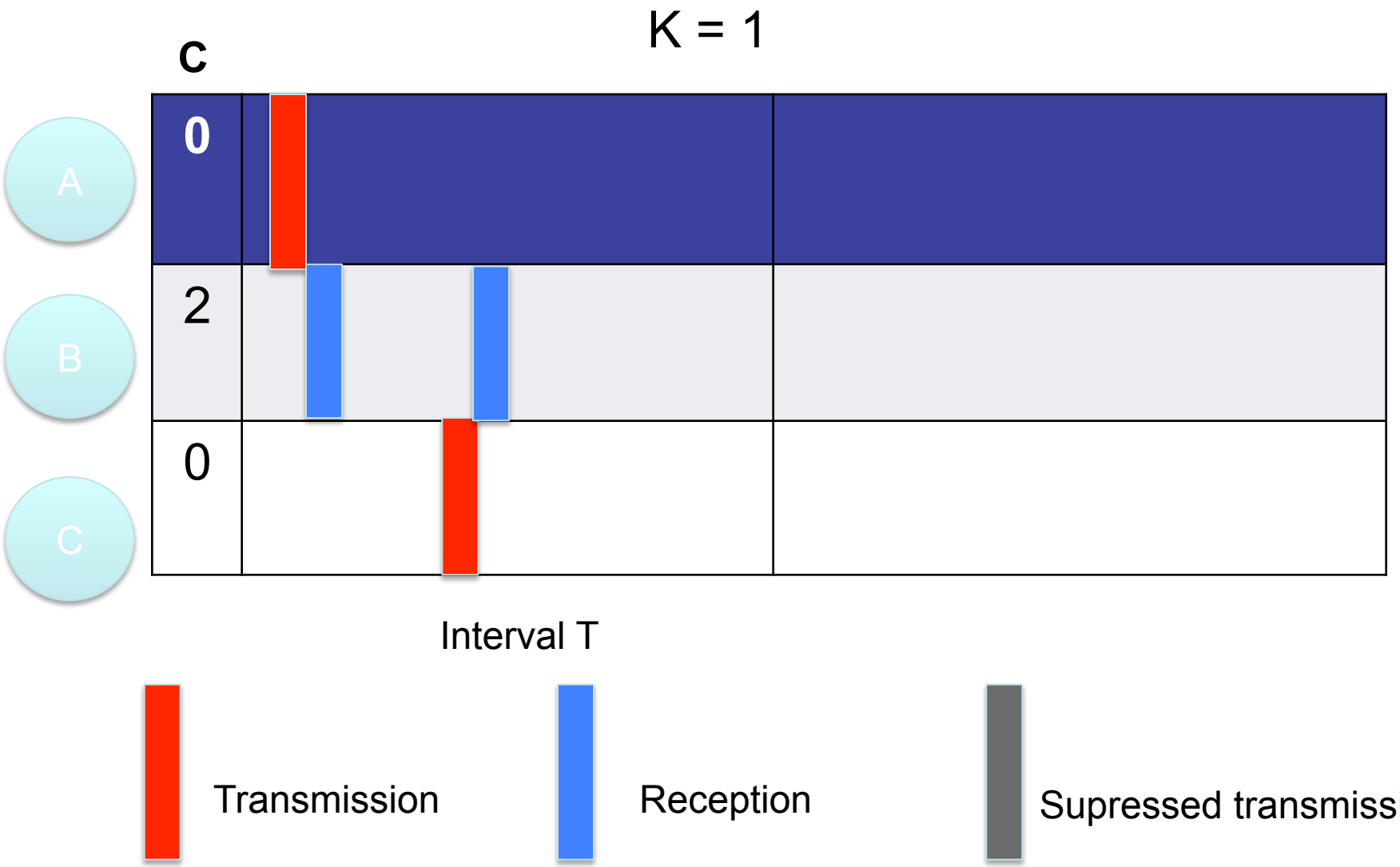
Example Execution



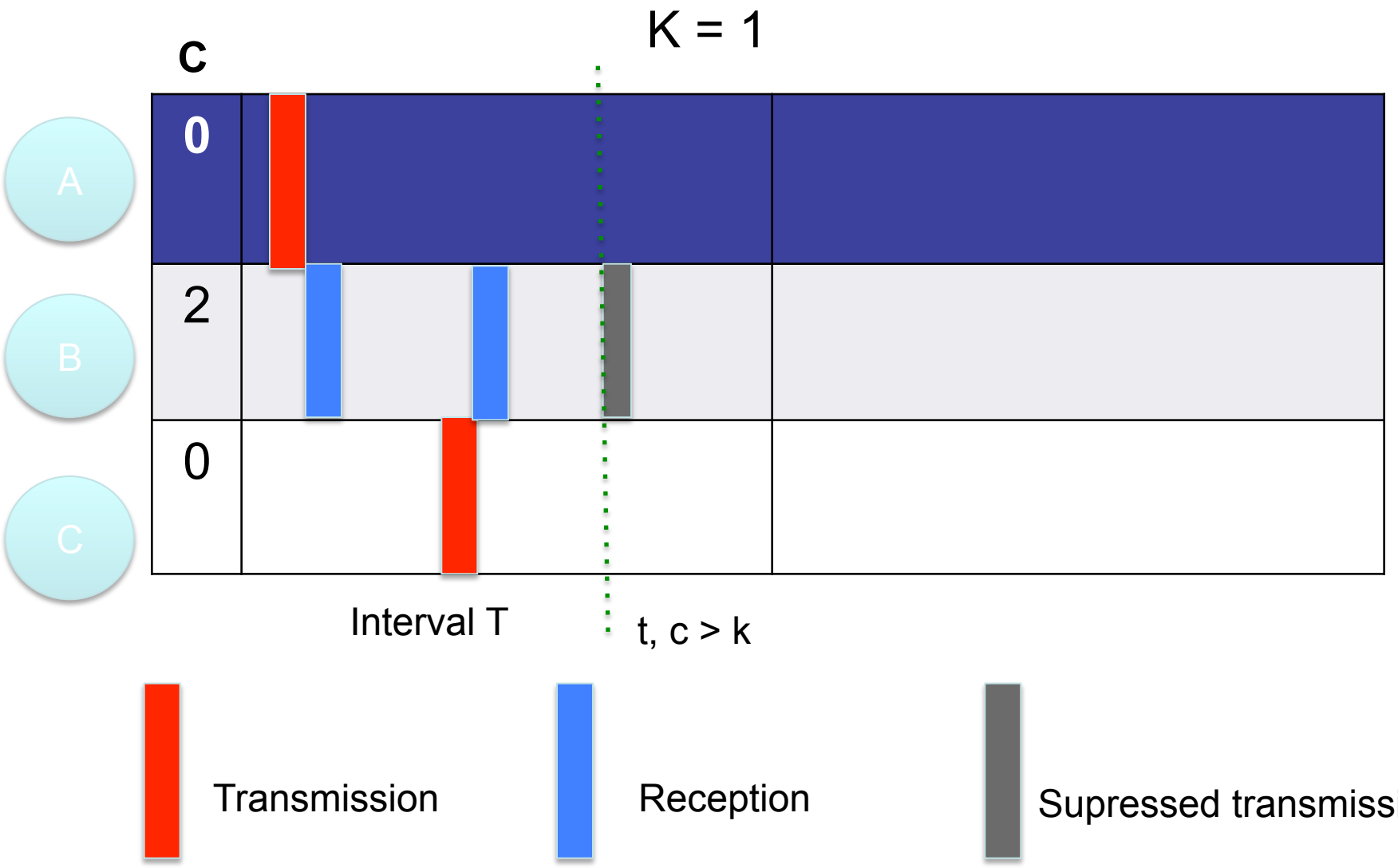
Example Execution



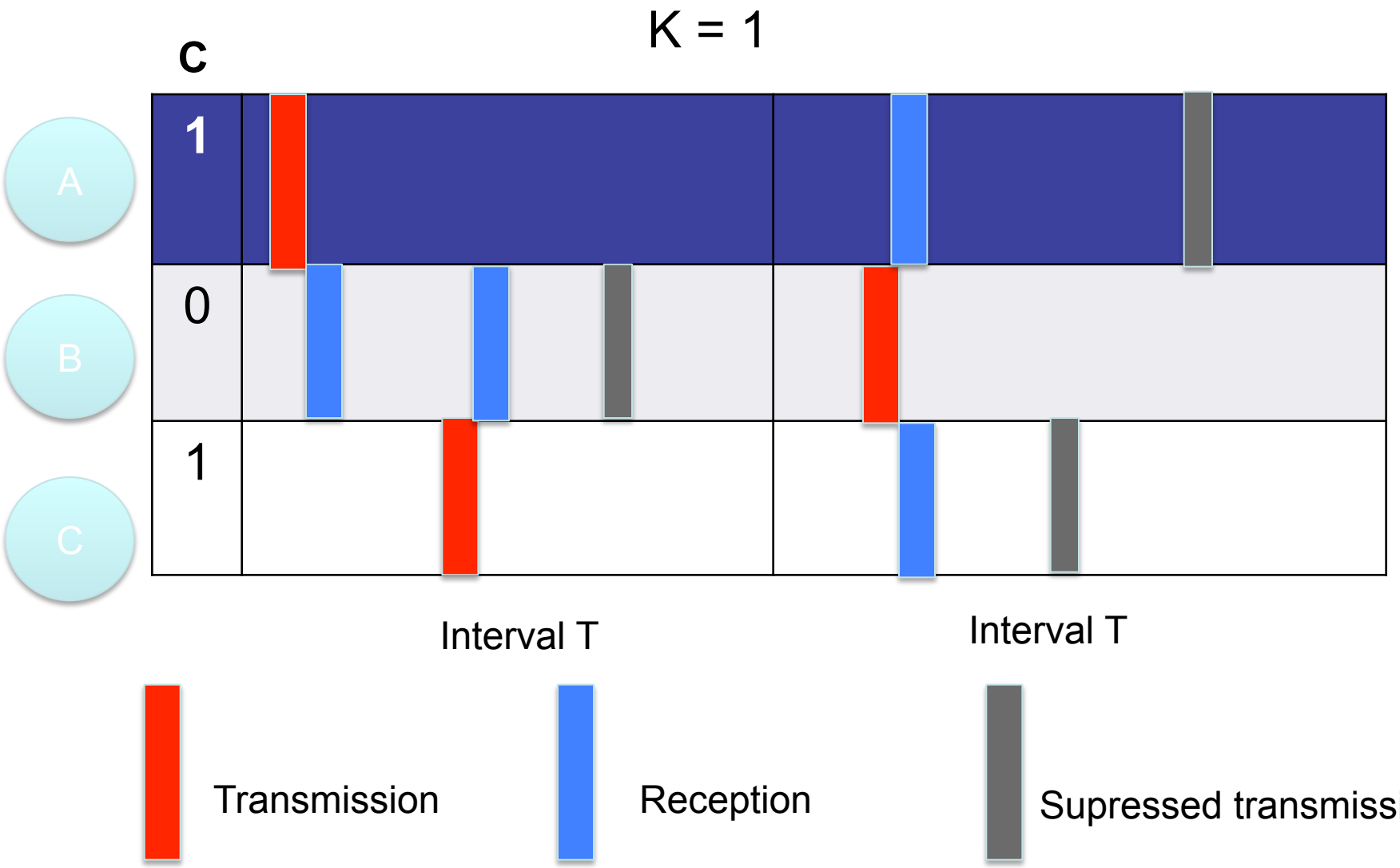
Example Execution



Example Execution



Example Execution



Trickle: further reading

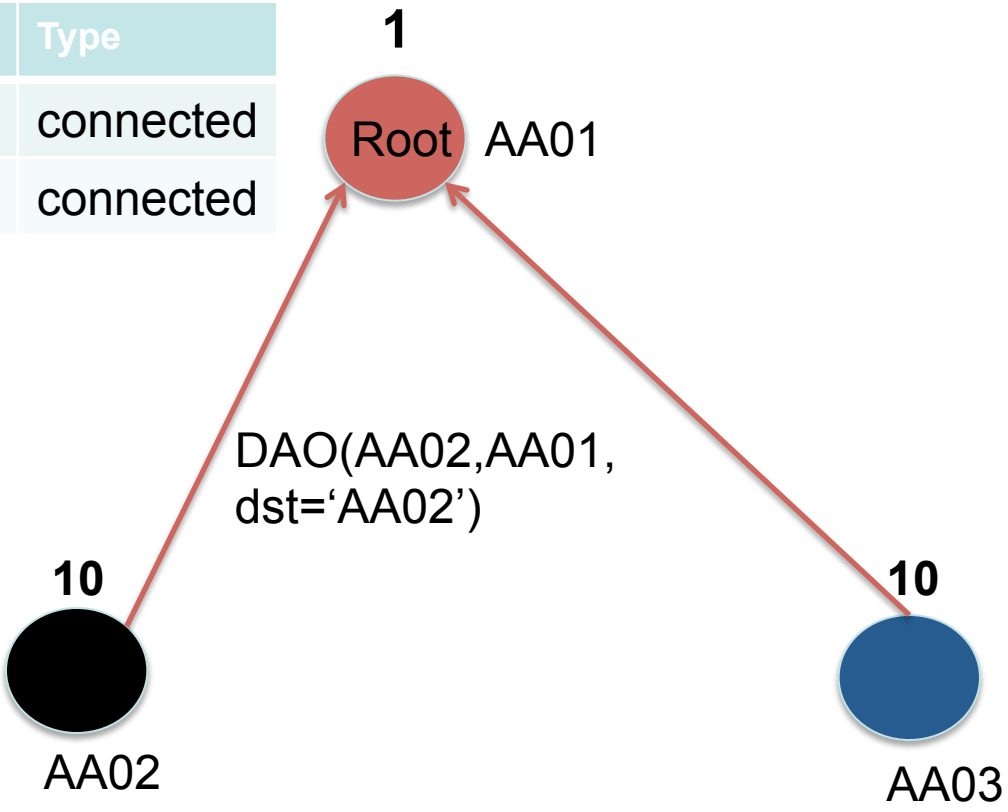
- Philip Levis, Eric Brewer, David Culler, David Gay, Samuel Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk, and Alec Woo, "The emergence of a Networking primitives in Wireless Sensor Networks", In communications of the ACM, Volumen51, Issue 7, July 2008.
- Jonathan W, Hui and David E.Culler. "IP is Dead, Long live IP for Wireless Sensor Networks", In proceedings of the 6th international Conference on Embedded Networked Sensor Systems (Sensys), 2008
- Philip Levis, Neil Patel, David Culler, and Scott Shenker, "Trickle: A self regulating algorithm for Code propagation and Maintenance in Wireless Sensor Networks", In proceedings of the First USENIX/aCM Symposium on Networked Systems Design and Implementation NSDI 2004
- Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson and Philip Levis, "Robust and Efficient Collection through Control and Data plane integration". Technical Report SING-08-02

Destination Advertisement Object: DAO

- Propagate destination information upwards along the DODAG
- Delay sending: aggregate DAO information from other nodes
- Storing mode
  - Unicast to the selected parent node
- Non-storing mode
  - Unicast to the DODAG root
- Remove a parent node: send a No-Path DAO message
- May be acknowledged by its destination
  - DAO-ACK
- May contain options: configuration information, RPL target (ipv6 prefix, multicast group)

Downward routes: P2MP traffic

destination	Next Hop	Type
AA02	AA02	connected
AA03	AA03	connected

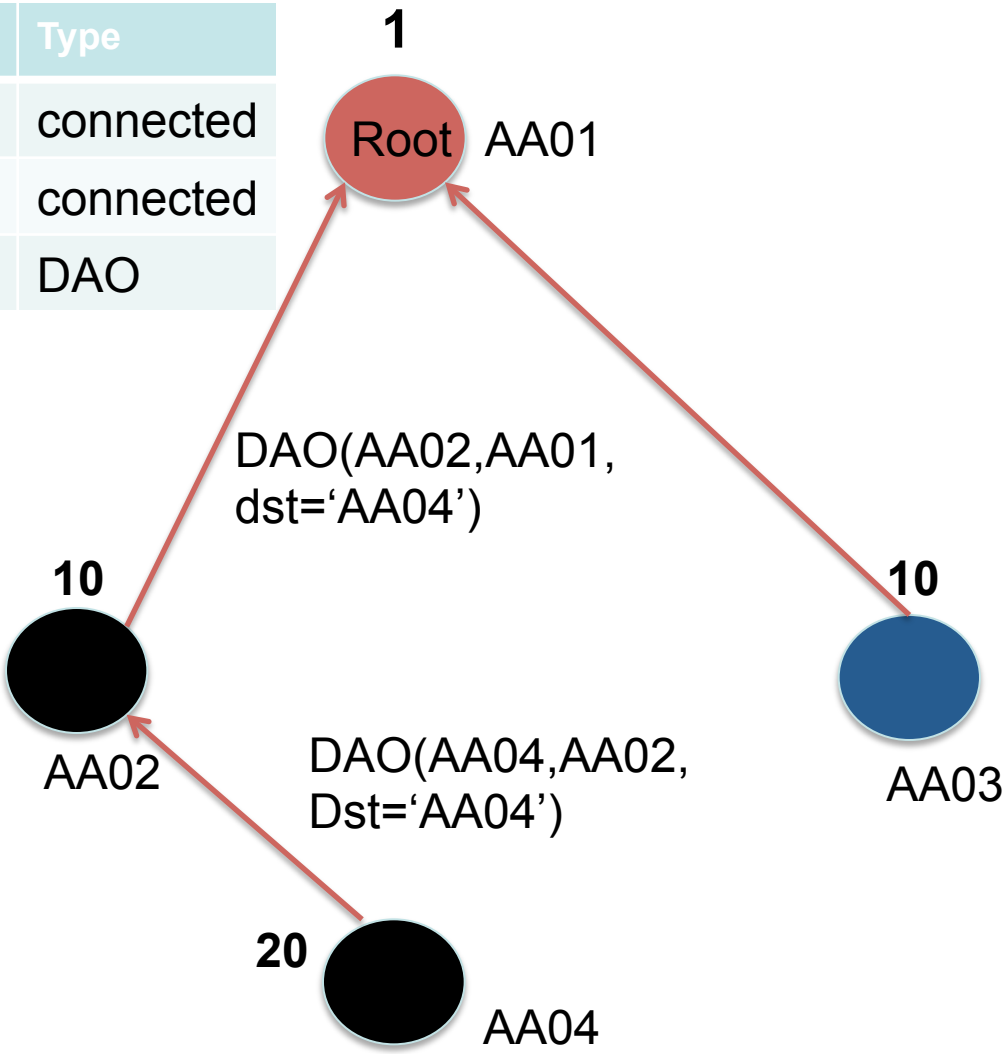


destination	Next Hop	Type
default	AA01	DIO
AA03	AA03	connected
AA01	AA01	connected

Downward routes: P2MP traffic

destination	Next Hop	Type
AA02	AA02	connected
AA03	AA03	connected
AA04	AA02	DAO

destination	Next Hop	Type
default	AA01	DIO
AA03	AA03	connected
AA01	AA01	connected
AA04	AA04	DAO

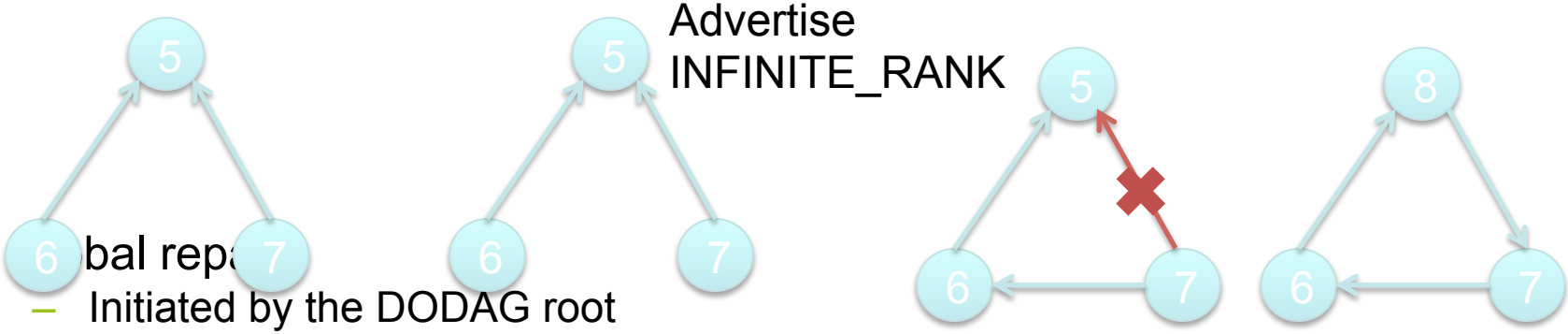


RPL-simulator

Demo : run a triangular pattern nodes simulation

DODAG Loops

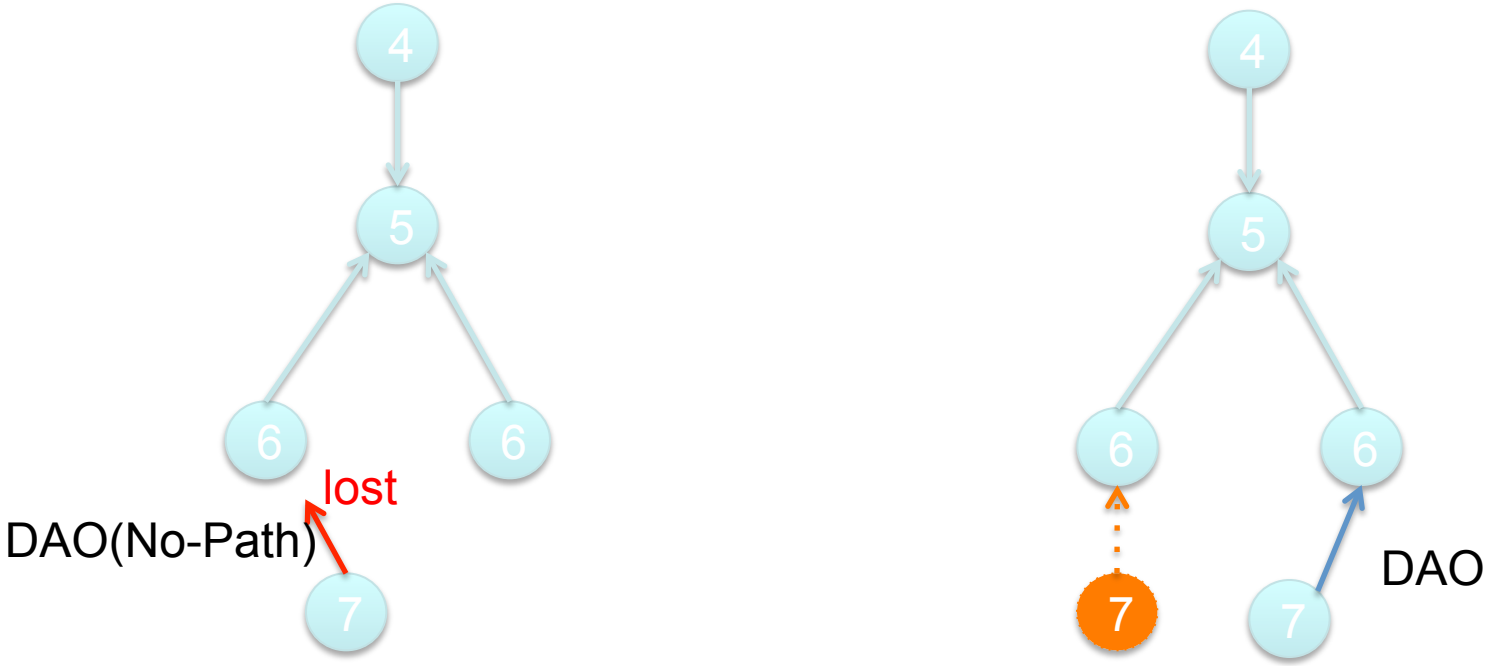
- Loops may occur when node increases Rank



- bal repa
  - Initiated by the DODAG root
  - Create a new DODAG version
- Local repair
  - Detach/float/merge within a DODAG version
  - Reset the trickle timer
  - If no parents exist with lower rank, MUST poison route by advertising infinite cost

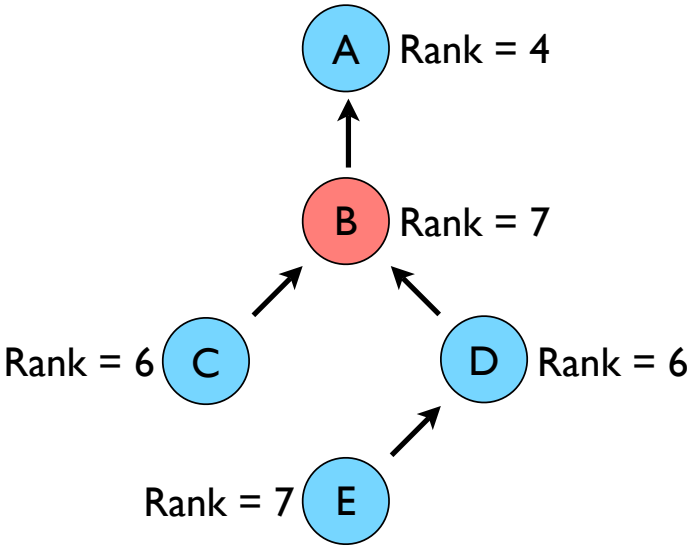
DAO Loops

- DAO loop may occur when a parent has an inconsistent route to a child
  - A No-Path DAO message was missed/lost



Loop detection and recovery

- Rank-based data-path validation
- Up routes must strictly decrease in rank
- Down routes must strictly increase in rank
- Generalized to inconsistency detection and repair

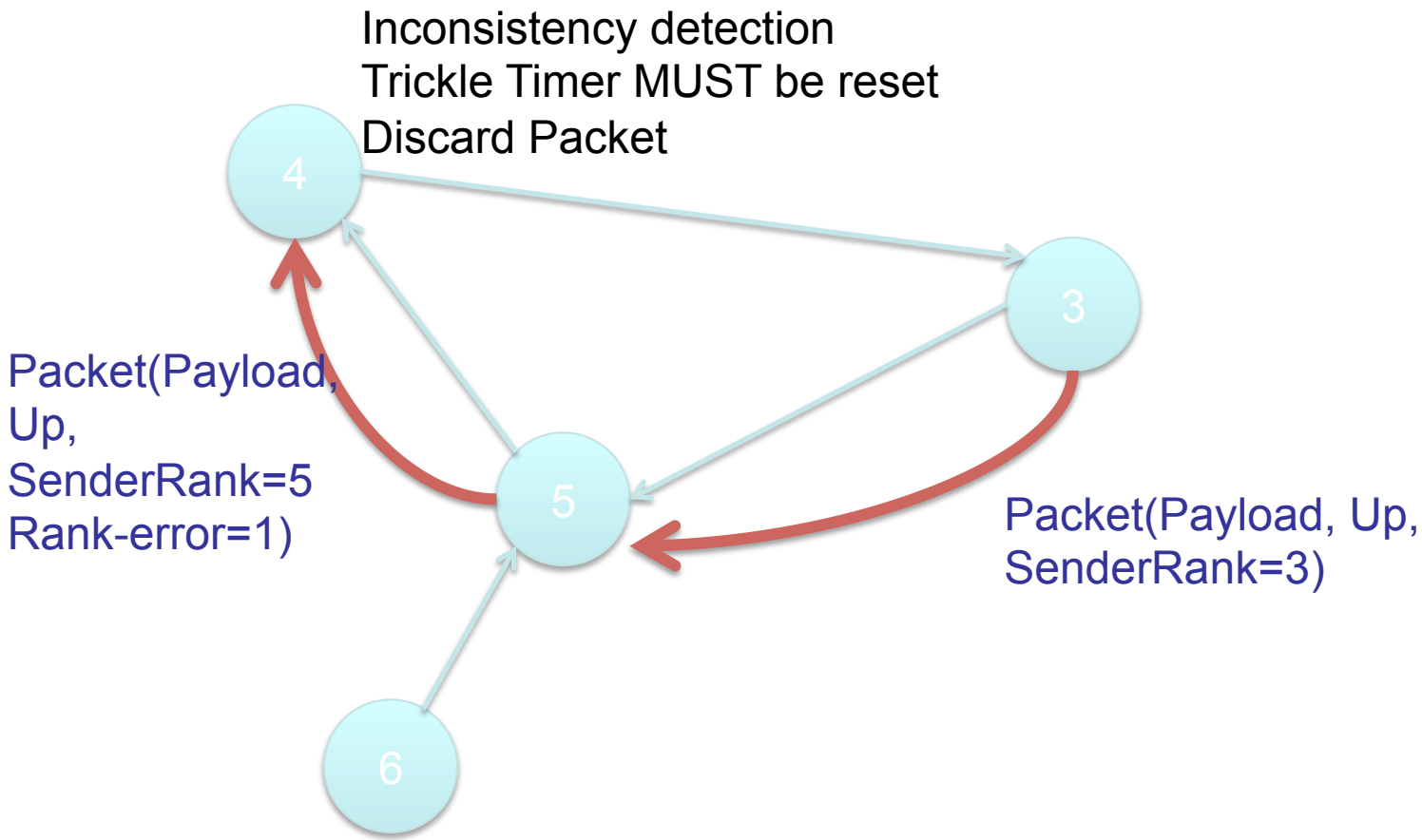


Loop detection and recovery

- Include routing info in data path to validate DAG (RPL option)
  - IPv6 Hop-by-Hop options
- Data path verification: there is no need to solve routing in the absence of data traffic
  - Instance ID: identify instance to route along
  - Up/down bit: progress up or down
  - Rank error bit: rank error detection with respect to up/down bit
  - Forwarding error: no route to destination
  - Sender rank: hop by hop rank

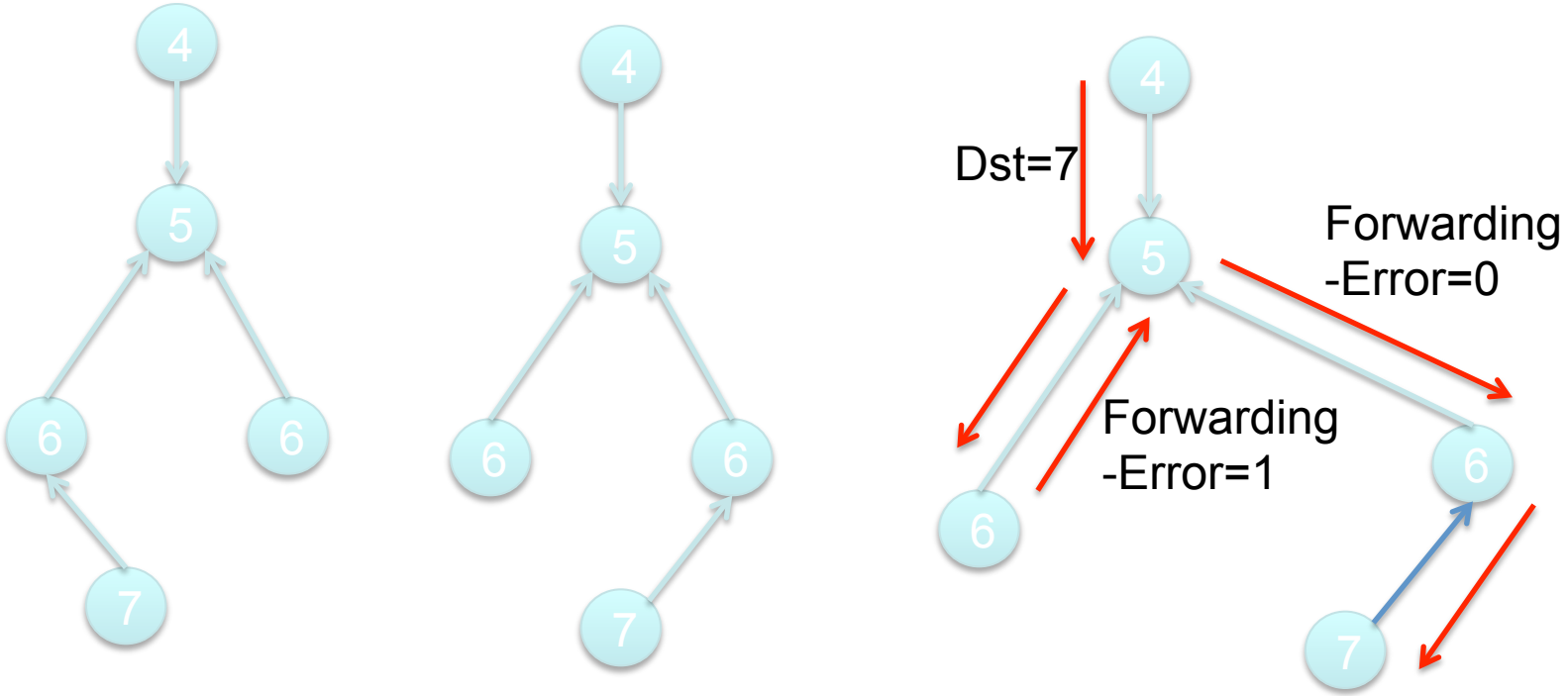
DAG Loop detection

- Packet direction does not match rank relationship



Inconsistency detection and recovery

- DAO inconsistency
  - Downward route is not longer valid in a child
  - Pass back packet to parent if no down route exists
  - Cleanup stale down routes if datagram is passed back



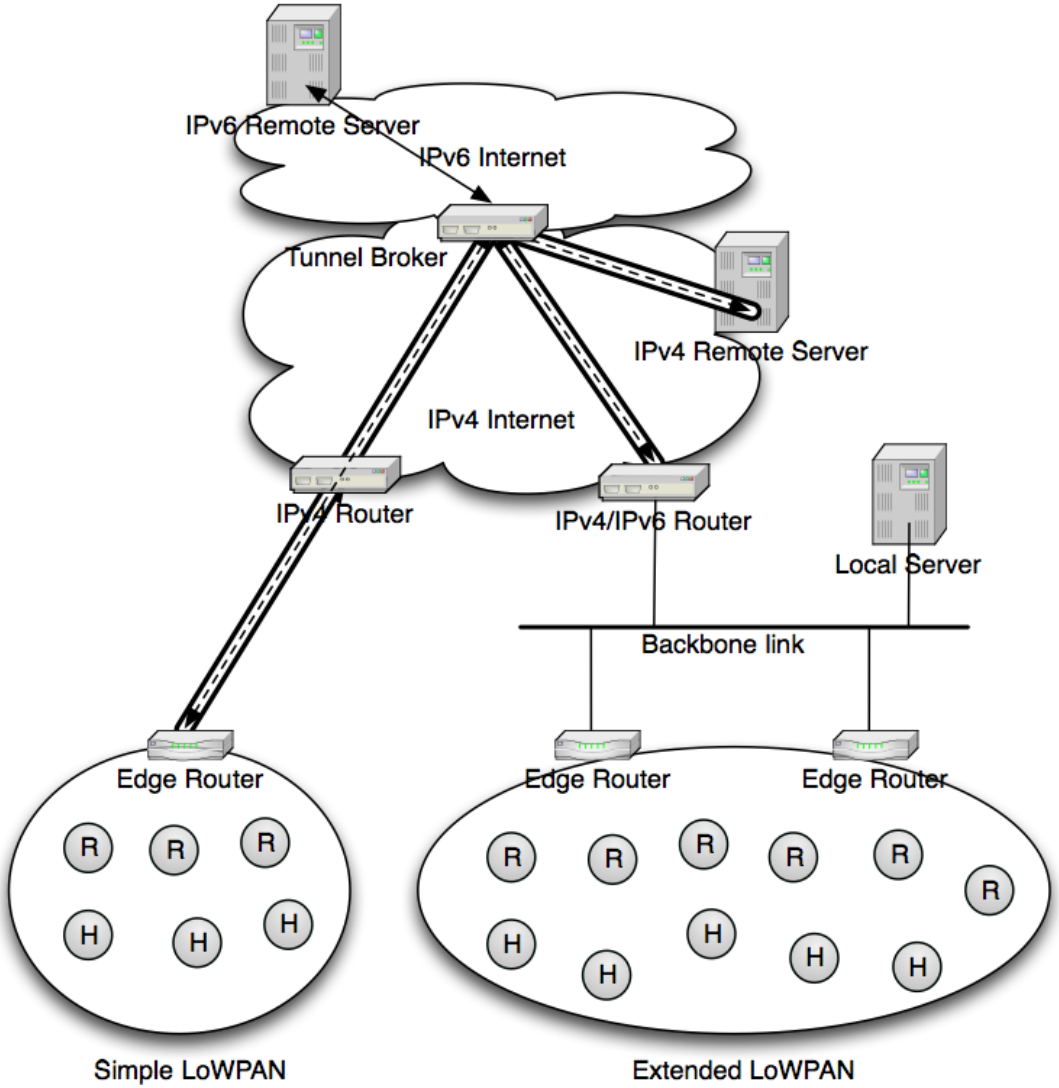
Maintenance of routing adjacency

- Detect if a neighbour is still reachable
- RPL : no keepalive mechanism
  - Expensive: bandwidth and battery
- Need an external mechanism
  - Neighbour Unreachability Detection: RFC4861
  - Layer 2 triggers: RFC5184
    - Association states
    - L2 acknowledgments

RPL: summary

- Distance vector IPv6 routing protocol
- Proactive routing protocol for LLN networks
  - Slow proactive process to construct and maintain a routing topology
  - Steady state: low-rate routing state transmission
- Rank-based data path validation mechanism
  - Include routing information in data datagrams
  - Detect inconsistencies
- Reactive and dynamic to resolve routing inconsistences
  - When inconsistency detected, increase routing state transmission rate
  - Quickly resolve those inconsistency
- Dynamic Trickle timers

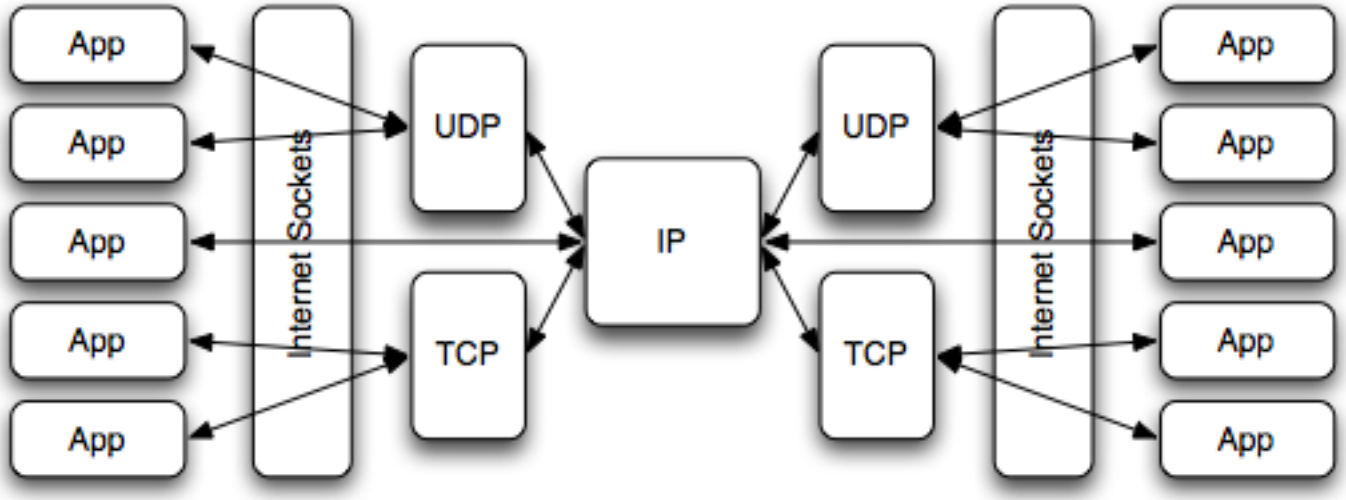
IPv4 Interconnectivity



Application Formats and Protocols

Introduction

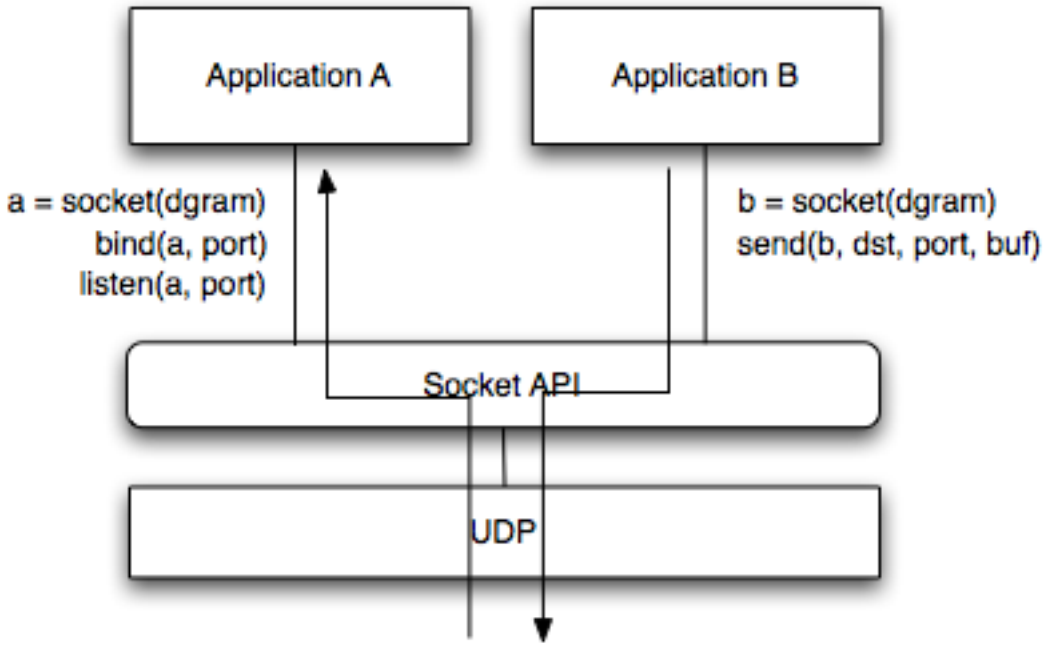
- The processes of applications communicate over IP using an Internet Socket approach
- 6LoWPAN also uses the Internet Socket paradigm
- Application protocols used with 6LoWPAN however have special design and performance requirements



Socket API

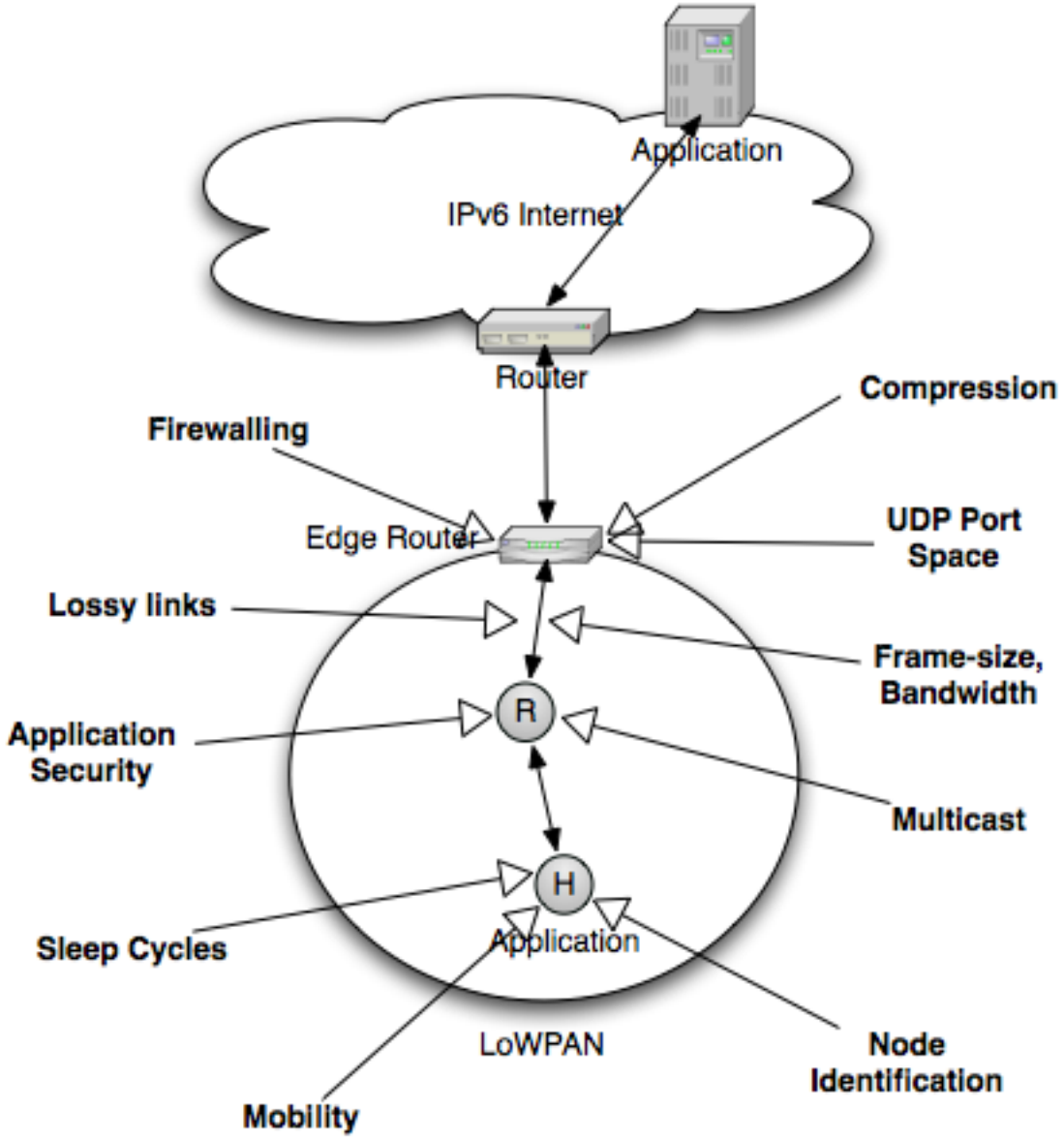
- The Socket API provides access to data communications for applications
- Well-known interface for handling data flow and buffer management via socket
- Supports also control messages to protocols
- Commands include:
  - socket, bind, send, read, close etc.
- Examples of Socket APIs
  - Berkeley sockets in *nix systems
  - Mac OSX (Darwin)
  - Contiki uIP (Pseudo socket approach)

Socket API

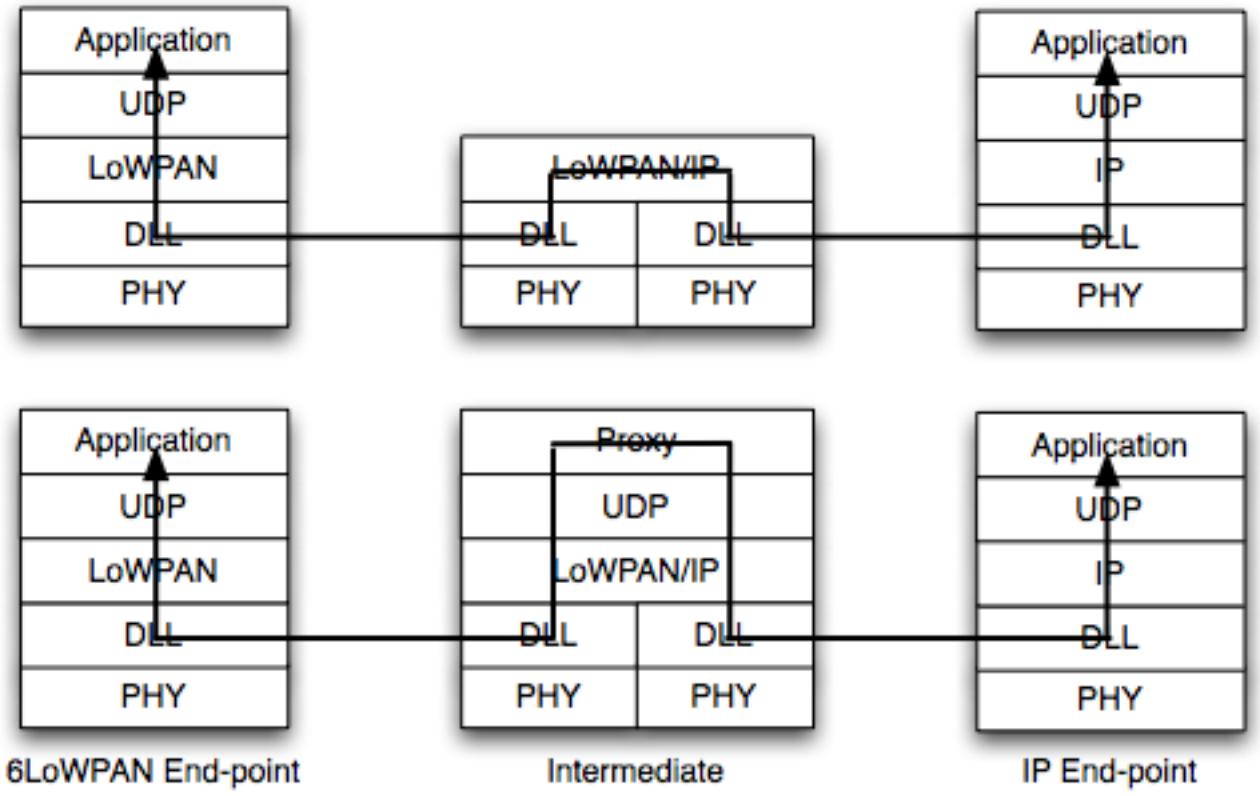

Application A
Application B
a = socket(dgram)
bind(a, port)
listen(a, port)
b = socket(dgram)
send(b, dst, port, buf)
Socket API
UDP


6LoWPAN

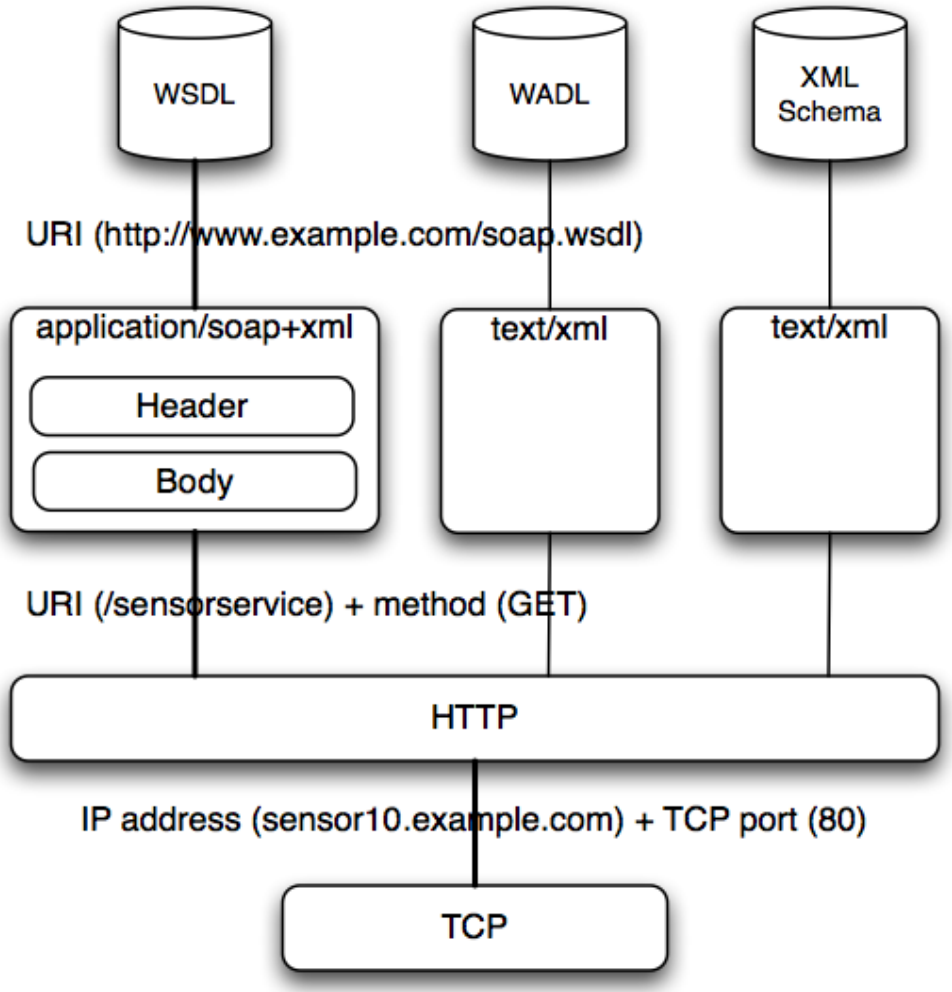
Design Issues



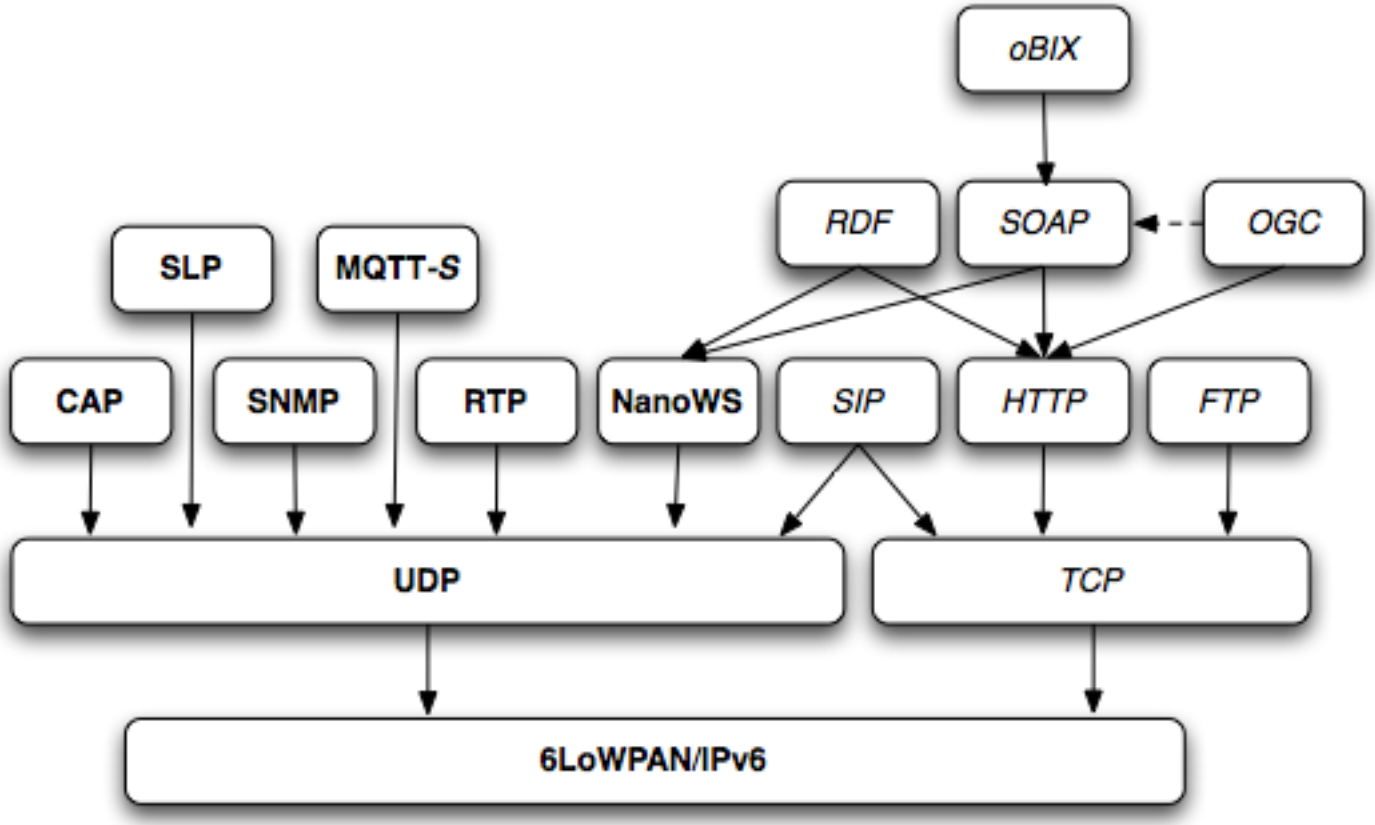
End-to-end Paradigm



Web-service Paradigm

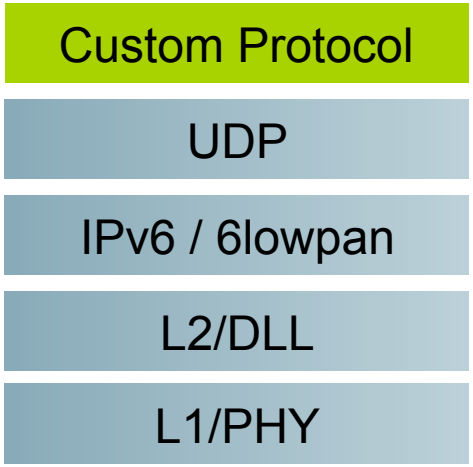


Application Formats and Protocols



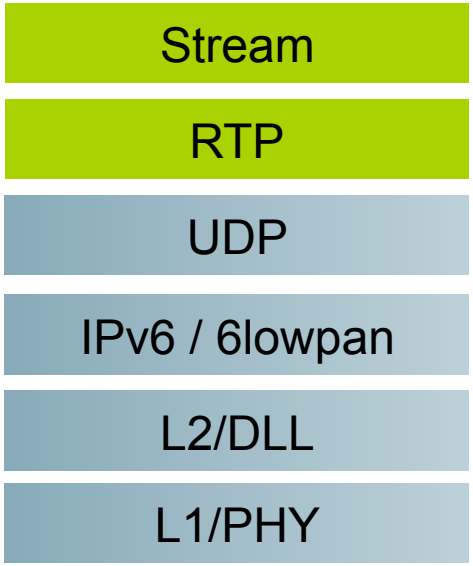
Custom Protocols

- The most common solution today
- Application data typically binary encoded, application specific
- Application protocol uses a specific UDP port, application specific
- As 6LoWPAN is end-to-end IPv6 communications, not a problem
- Advantage:
  - Compact, efficient, security can be integrated, end-to-end
- Disadvantage:
  - Custom server app needed, little re-use, learning curve, interoperability



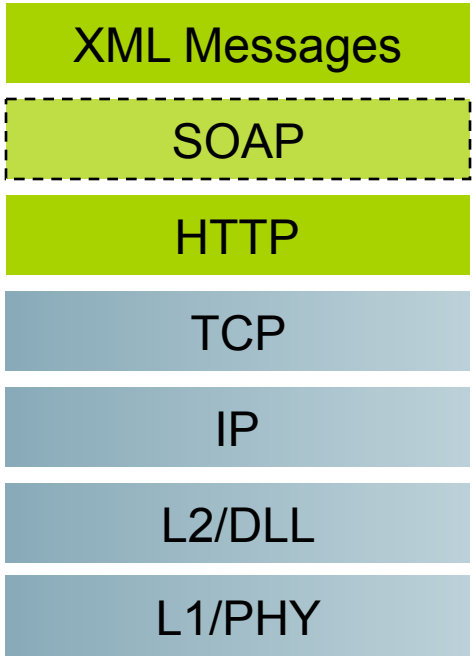
Streaming and RTP

- The correct streaming solution
- For audio or continuous sensor streaming
  - Audio over 802.15.4 needs good codec
- Advantages:
  - RTP can be used over 6LoWPAN
  - Provides end-to-end solution
  - No server modifications needed
  - Jitter control
- Disadvantages:
  - Headers could be more efficient for simple sensor data streaming



XML/HTTP

- De-facto for inter-server communications
- Well-known XML schema important
- All Internet servers speak HTTP/XML
- Useable for RPC, pub/sub and events
- SOAP or REST paradigm
- Advantages:
  - Well known XML schema
  - Formal message sequences
  - Internet-wide support
- Disadvantages:
  - Inefficient, complex
- Solution: Embedded web-services
  - See the IETF 6lowapp effort http://6lowapp.net



Other Application Protocols

- Service Discovery
  - Service Location Protocol (SLP)
  - Device Profile Web Services (DPWS)
- Management
  - Simple Network Mangement Protocol (SNMP)
- M2M Telemetry
  - MQ Telemetry Transport for Sensors (MQTT-S)
- Building Automation
  - BACnet/IP
  - oBIX
- Energy Industry
  - ANSI C12
  - Device Language Message Specification (DLMS)

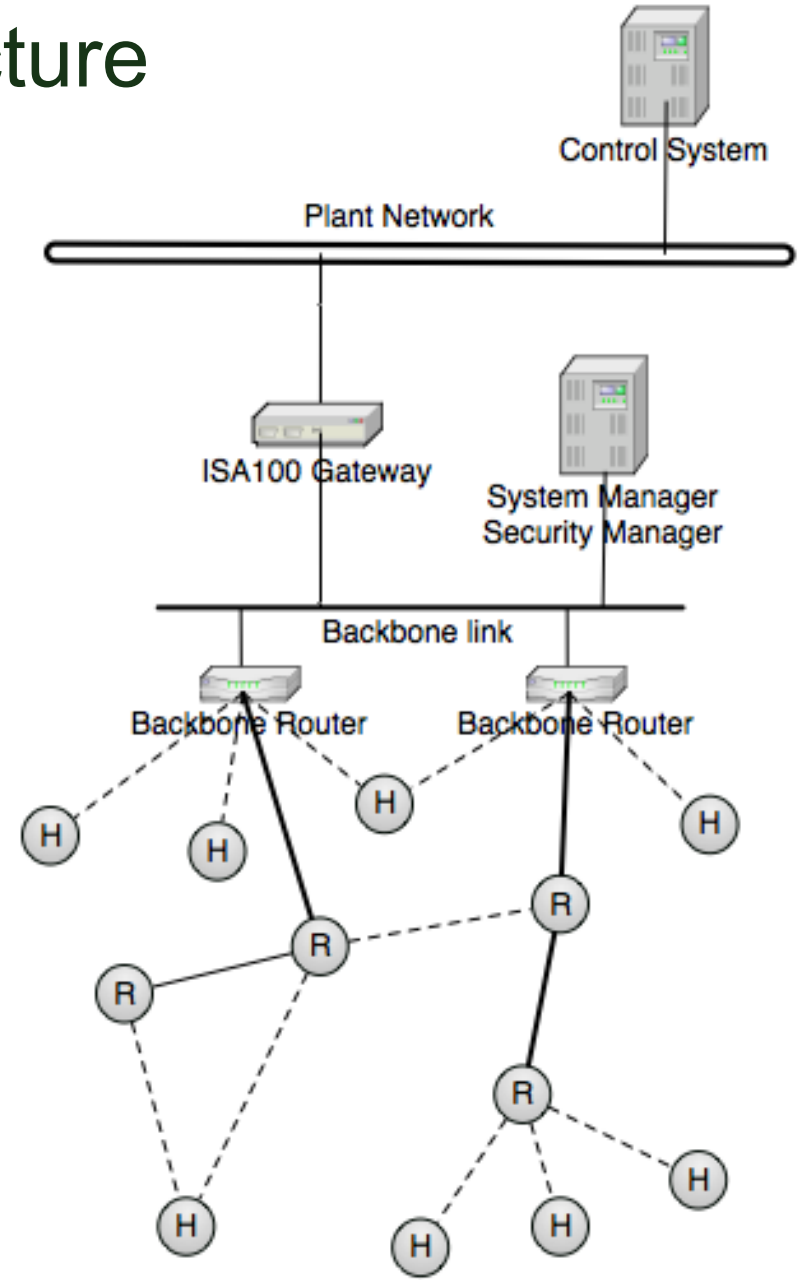
System Examples

ISA100 Industrial Automation

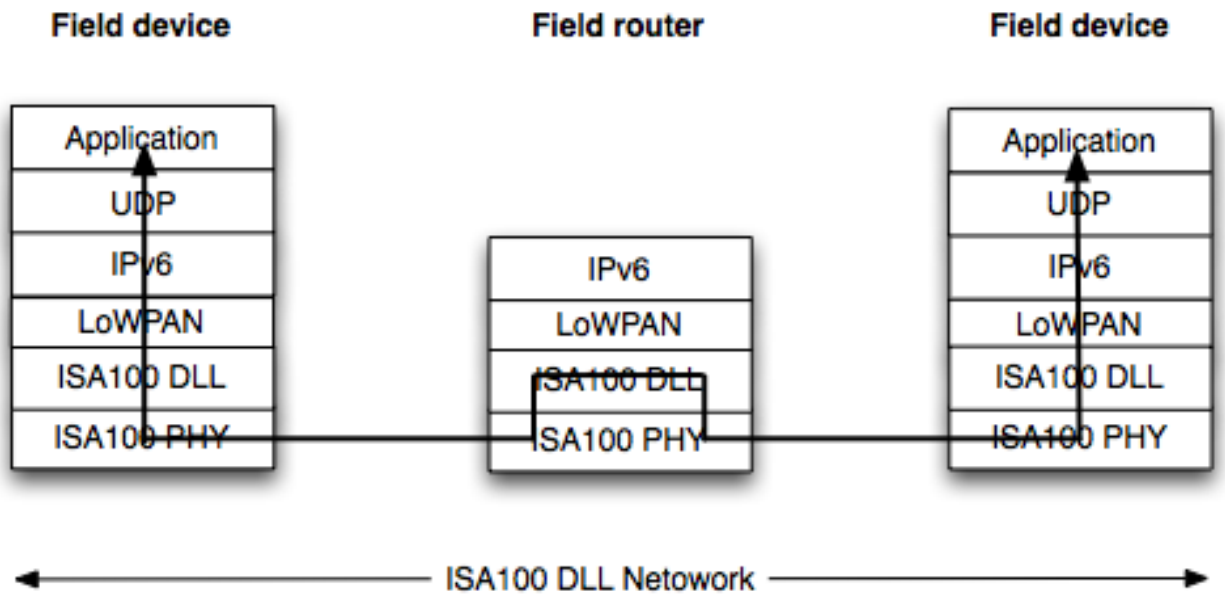
ISA100

- Standard effort from the Instrumentation, Systems, and Automation Society (ISA)
  - Standardization activities accredited by ANSI
- Has been estimated that 55% of industry will support ISA100 in the next few years
- ISA100 group standardizes wireless systems for automation
- ISA100.11a standardization is in progress
  - IEEE 802.15.4-2006 Radio Standard
    - With frequency hopping improvements
  - 6LoWPAN Networking (6LoWPAN, IPv6, UDP)
  - Network gateways, monitoring, deployment, interoperability
  - Defining reliability classes 0 to 5
  - First version of approved standard released in 2009

ISA100 Architecture



ISA100 Forwarding



Usage Classes

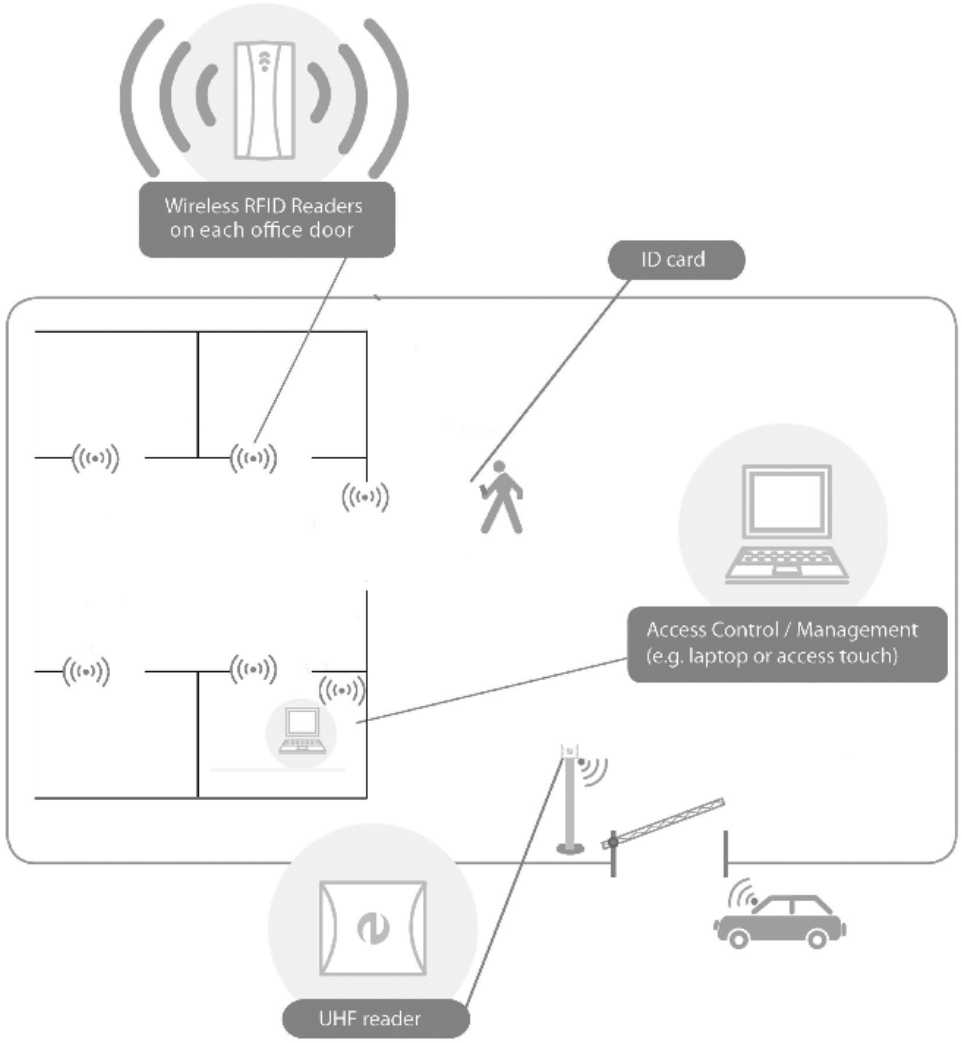
Safety	Class 0: Emergency action	Always critical	Importance of message timeliness increases →
Control	Class 1: Closed loop regulatory control	Often critical	
	Class 2: Closed loop supervisory control	Usually non-critical	
	Class 3: Open loop control	Human in the loop	
		NOTE Batch levels* 3 & 4 could be class 2, class 1 or even class 0, depending on function *Batch levels as defined by ISA S88; where L3 = unit and L4 = process cell	
Monitoring	Class 4: Alerting	Short-term operational consequence (e.g., event-based maintenance)	
	Class 5: Logging and downloading / uploading	No immediate operational consequence (e.g., history collection, sequence-of-events, preventive maintenance)	

Wireless RFID Infrastructure

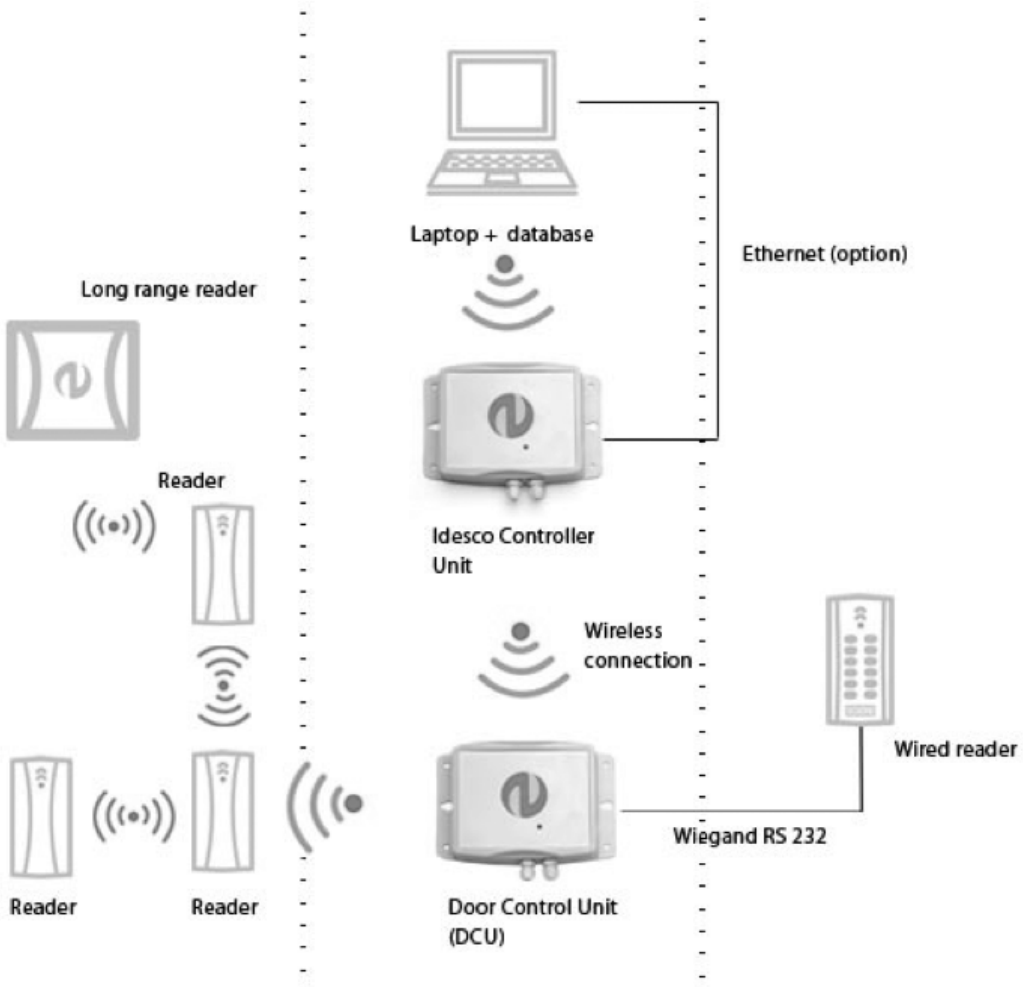
Wireless RFID Infrastructure

- Access control is an important part of building automation
- Idesco is a Finnish RFID system provider
  - http://www.idesco.fi
- Idesco Cardea System
  - World's first wireless infrastructure RFID access control system
  - 6LoWPAN networking between RFID components
- System components
  - Idesco Cardea readers
  - Idesco Cardea door control unit
  - Idesco Cardea control unit and Access Touch
- Benefits of using 6LoWPAN
  - Significant reduction in installation time and cost
  - Flexibility and use in temporary installations
  - Makes RFID access control practical for small installations

Wireless RFID Infrastructure



Wireless RFID Infrastructure



Building Energy Savings

Building Energy Savings

- Energy savings is important in commercial buildings
  - 52% of electricity consumption in the UK
  - UK businesses waste up to 30% of energy purchased
- LessTricity projects aims at this problem
  - Consortium of companies in property management, building design, and management software
  - Based on 6LoWPAN technology from Jennic Ltd.
- Centralized mangement solution
  - Eliminate the wasteful use of electricity in buildings
- System architecture
  - LessTricity power controllers - *measure consumption*
  - LessTricity network interface - *Ethernet router*
  - Link layer mesh, 6LoWPAN and Jennic SNAP protocol

Building Energy Savings