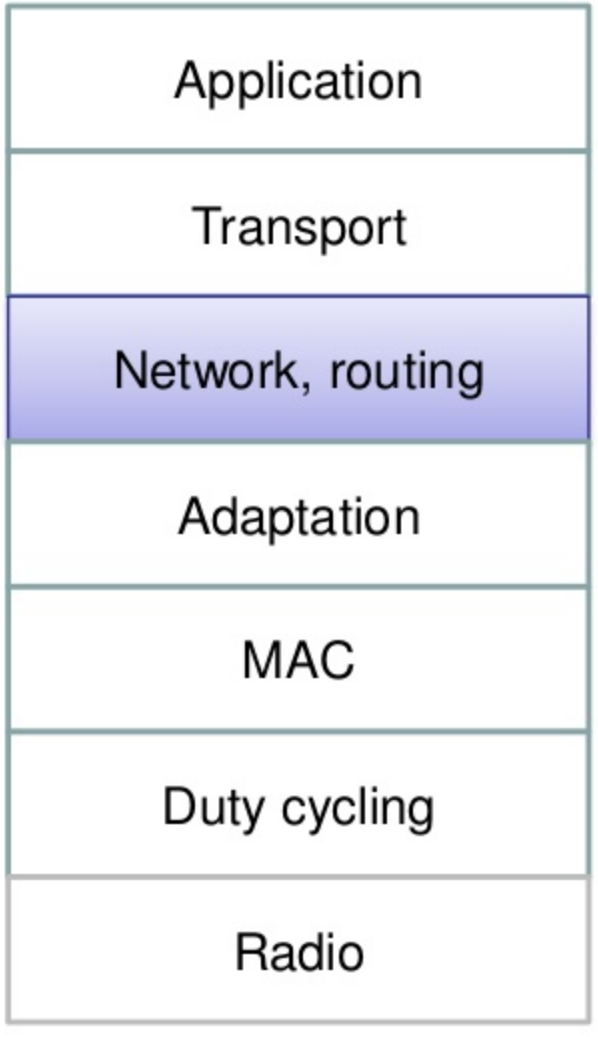
The IoT/IP Protocols

The network layer



The problem

- The general problem
  - Getting data from node A to node C, through node B
- The solutions
  - Addressing, routing
- The wireless problem
  - Intermediate nodes may move
  - Wireless medium is brittle
  - Wireless medium may change over time
- The resource problem
  - Bandwidth-constrained
  - Memory-constrained

Routing

Routing: IETF RPL

- ”Ripple”
- Proactive any-to-any routing protocol
  - But optimized for the many-to-one case
- RPL forms routing graph from root node
- Packet forwarding along graph, routing metric dynamically updated
  - Short-lived routing loops are tolerated

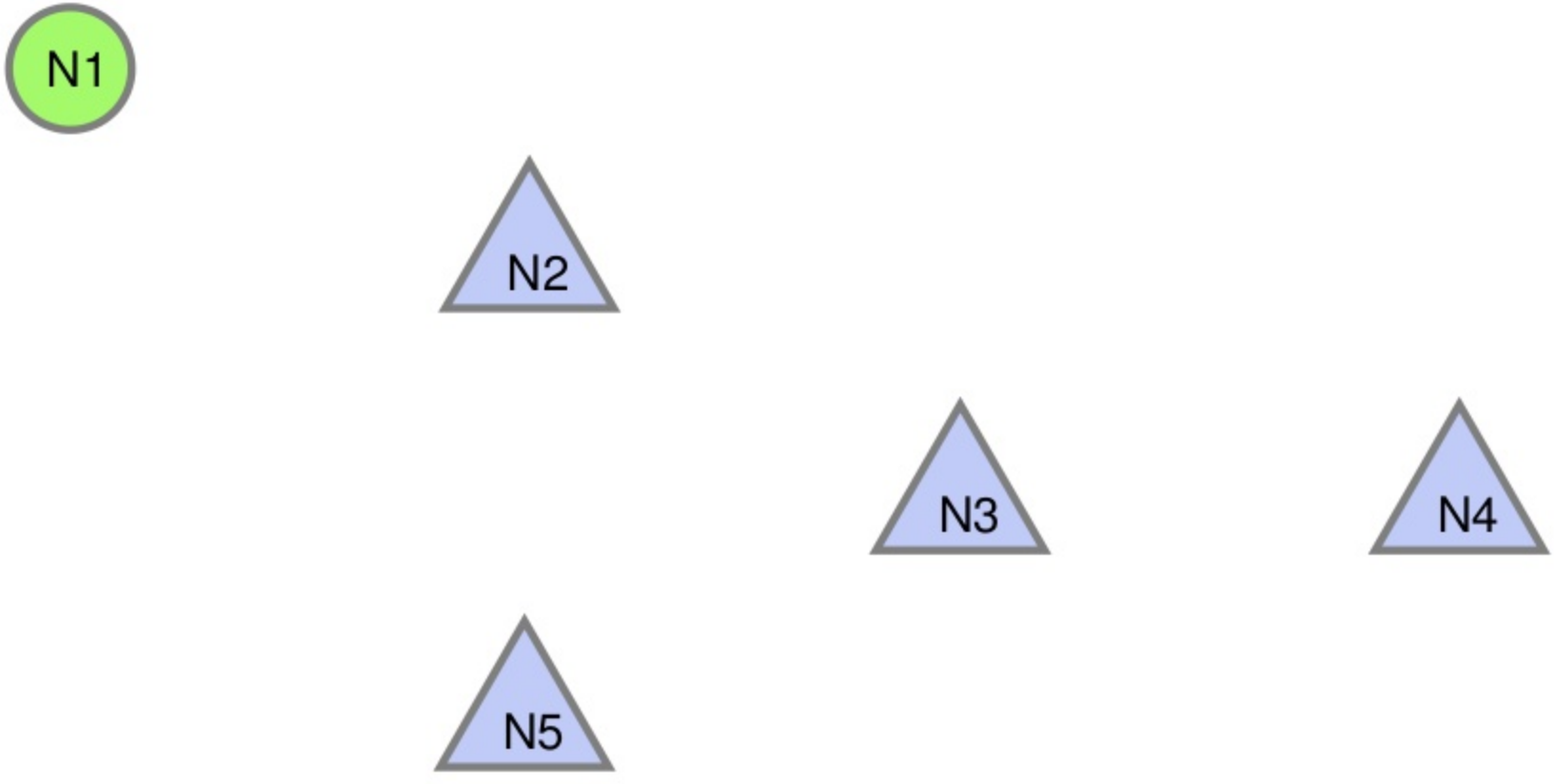
The problems that RPL solves

- Resource constraints
  - Memory constraints
  - Power constraints
  - Bandwidth constraints
- Wireless is unpredictable
  - Pick good routes

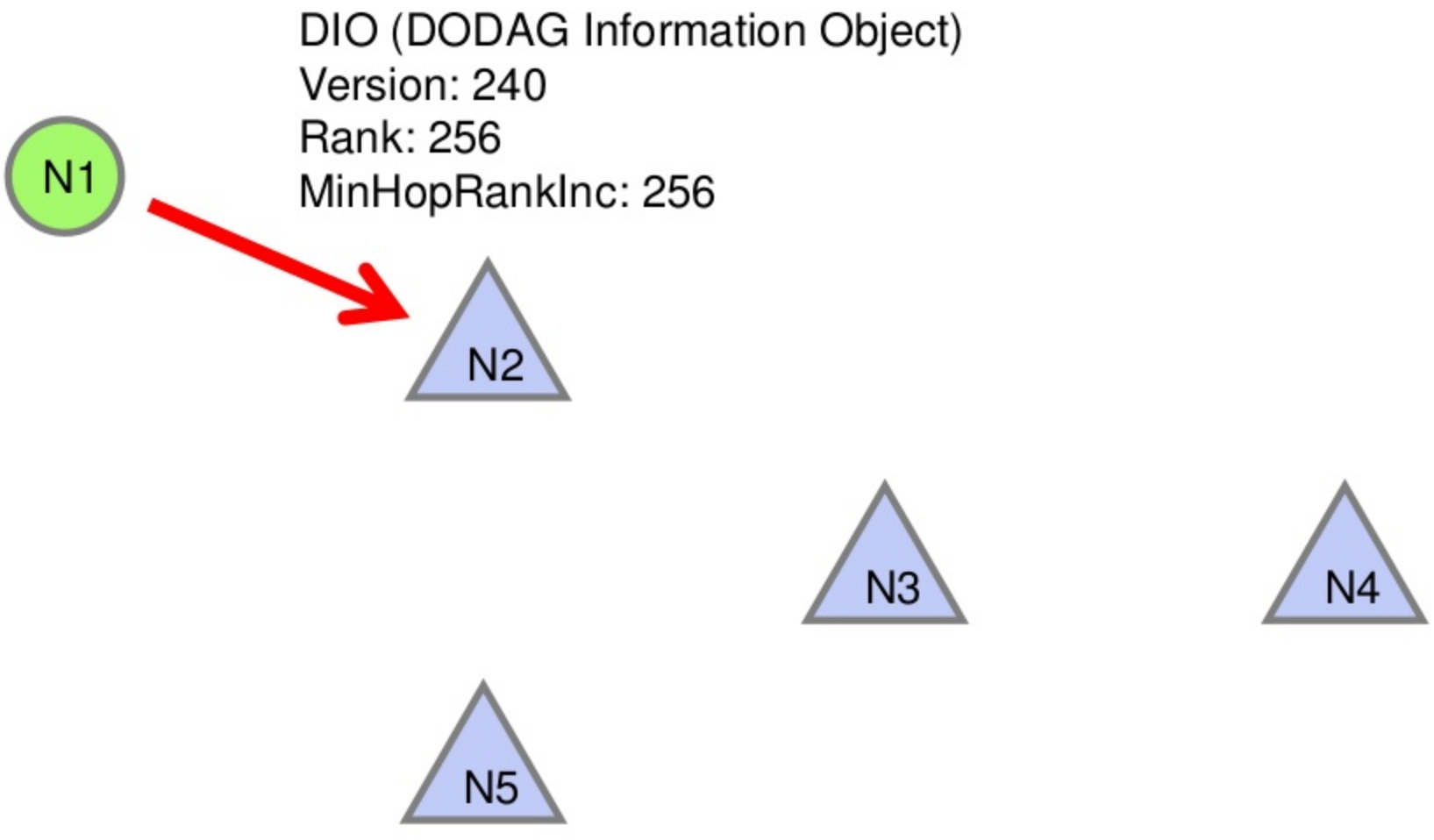
RPL network formation

- Build acyclic graph from root node
  - DODAG – Destination Oriented Directed Acyclic Graph
- DIO messages are broadcast by all nodes, starting from the root node
  - DIO – DODAG Information Object

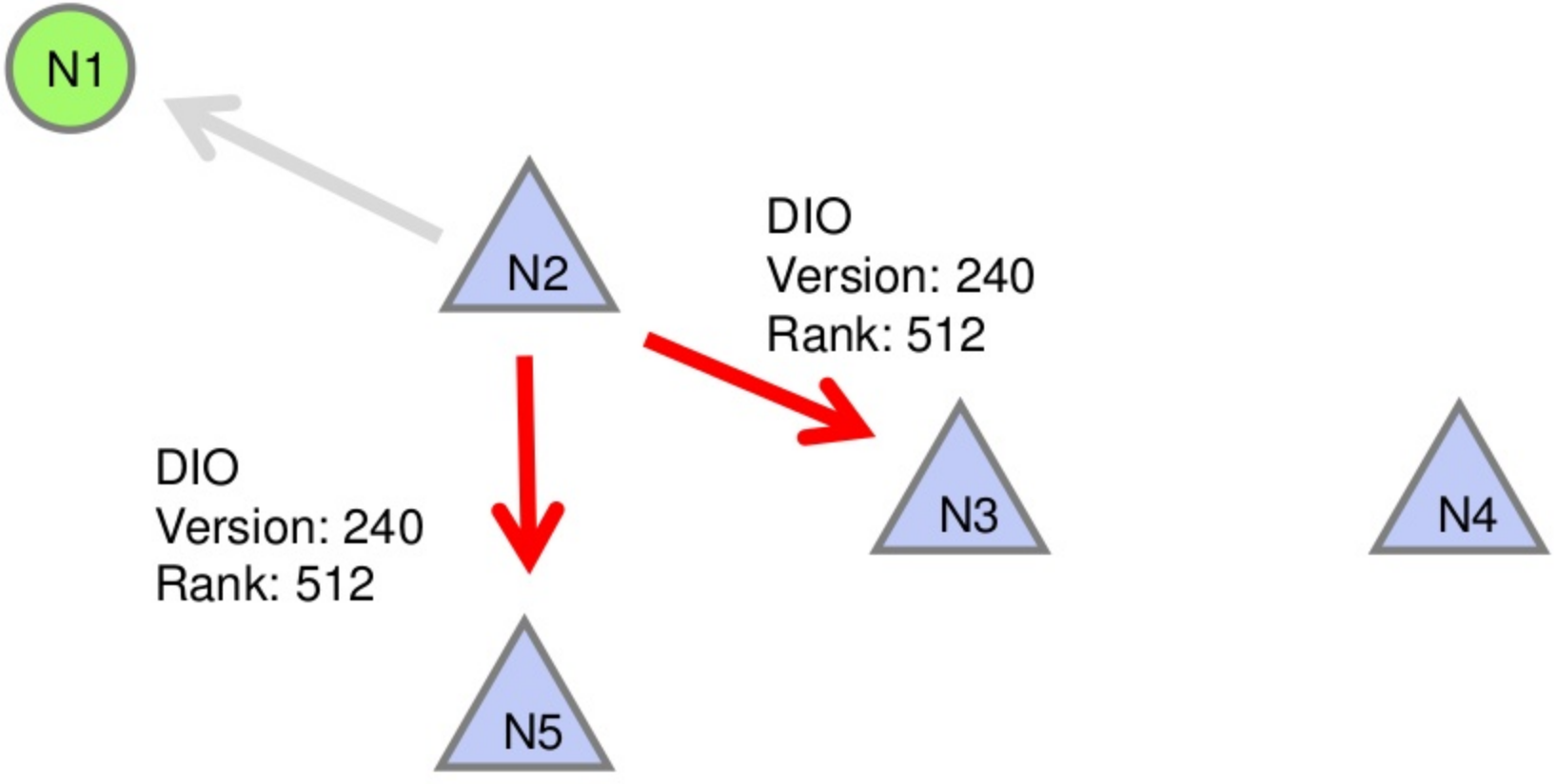
RPL Network Formation



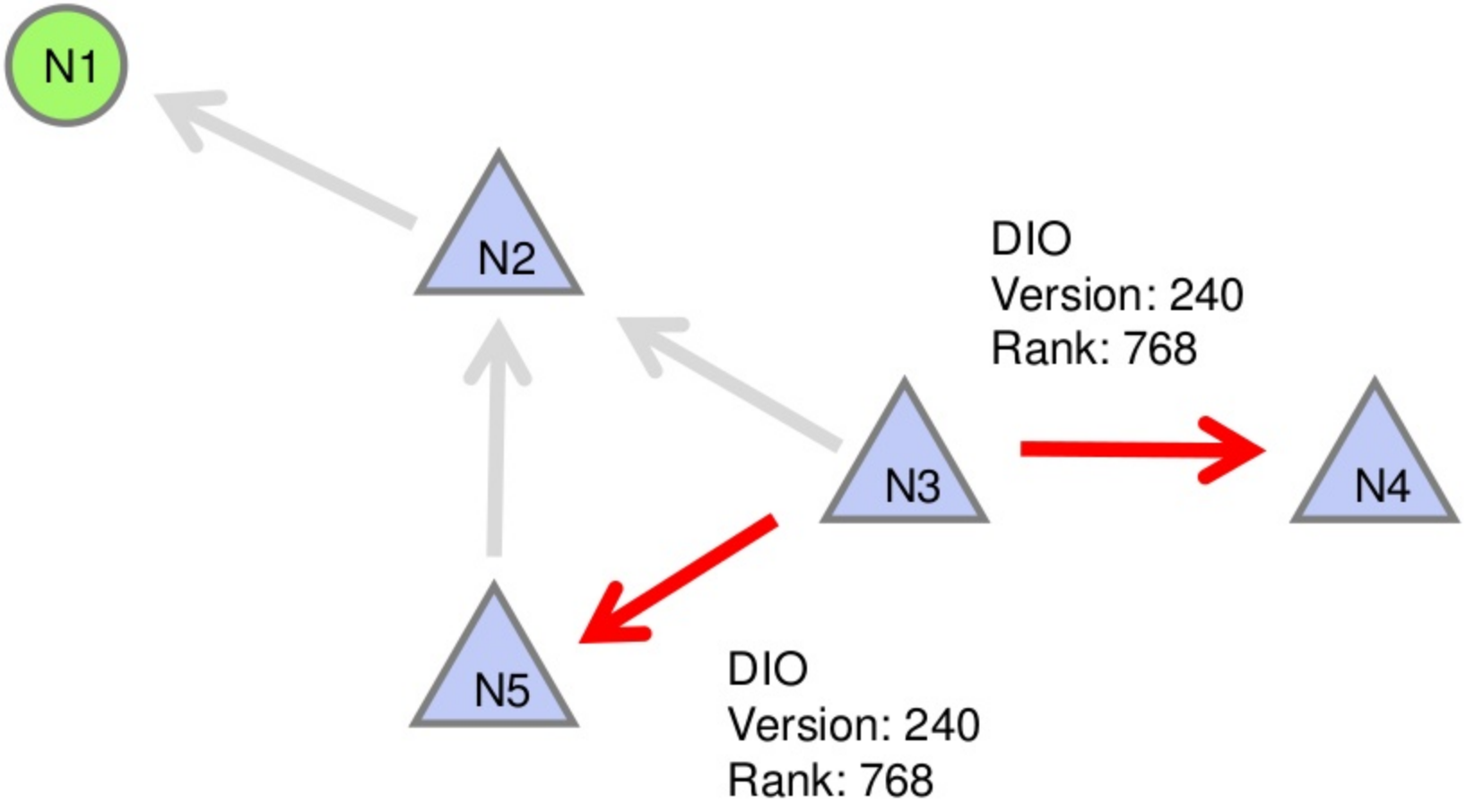
RPL Network Formation



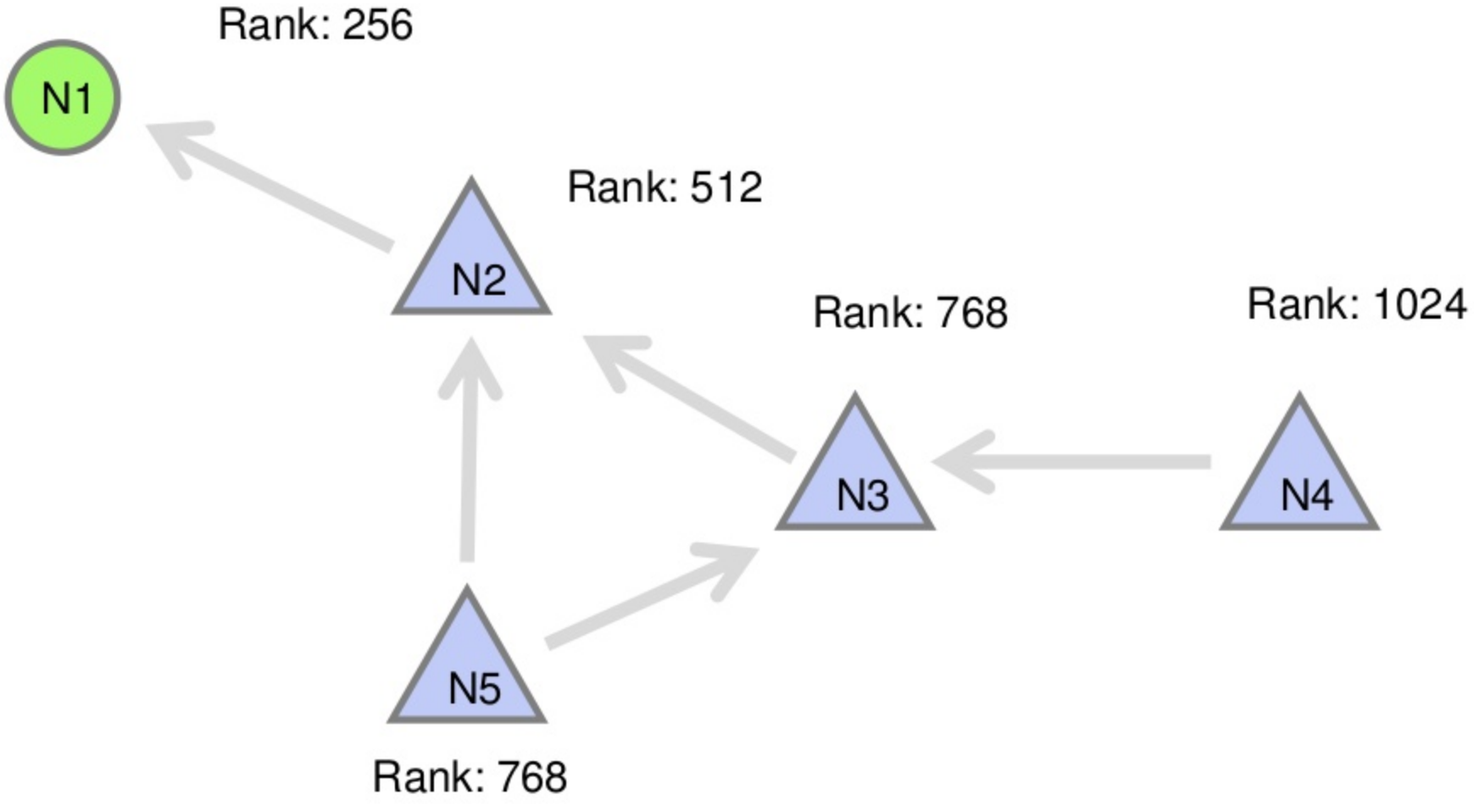
RPL Network Formation



RPL Network Formation



RPL Network Formation



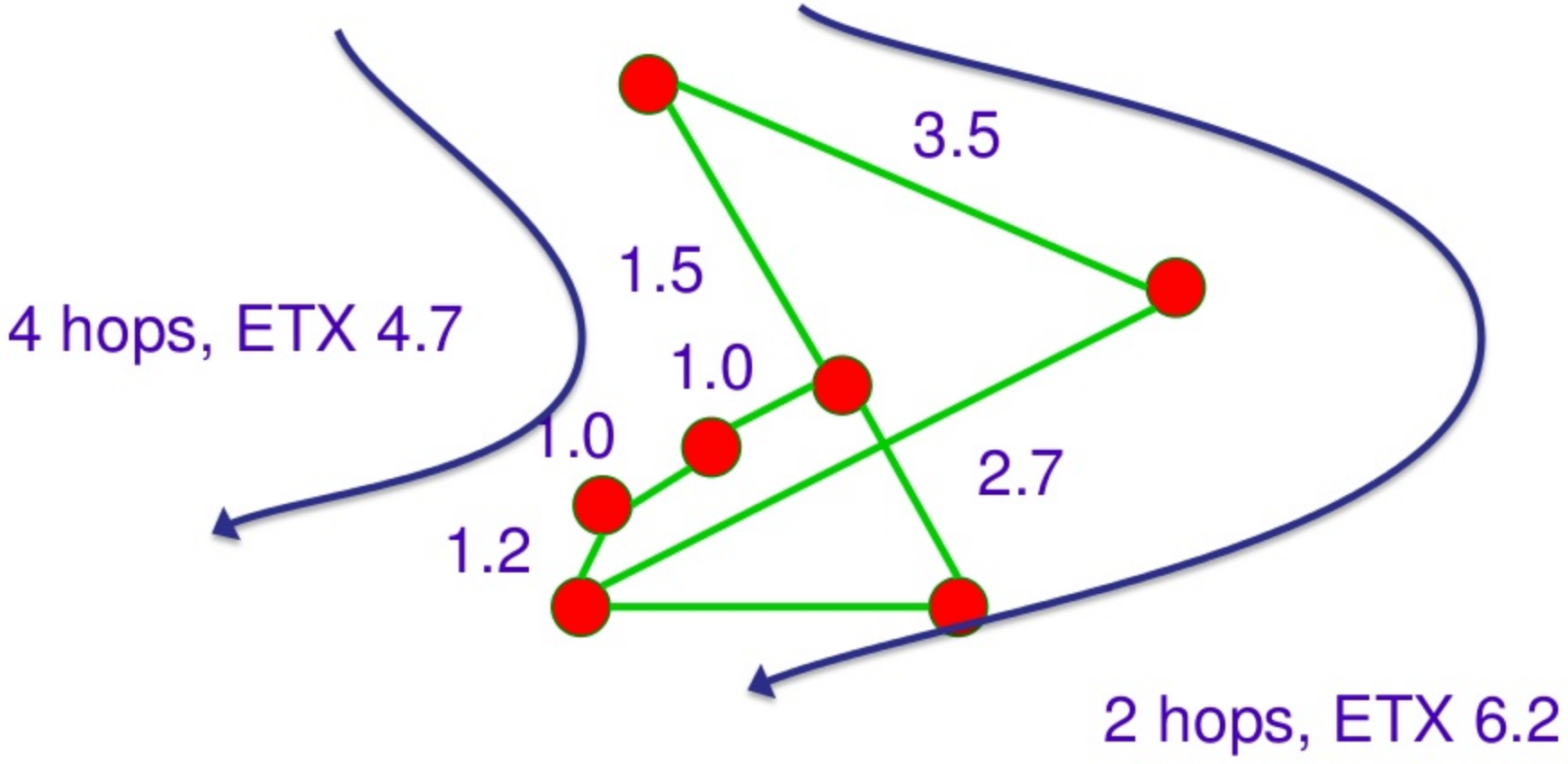
RPL packet forwarding

- Given multiple choices, what route should a packet take?
  - Hop count – shortest amount of hops to the root?
  - Some other dynamic metric?
- RPL leaves this to its Objective Function
- Two Objective Functions specified
  - of0: hop count
  - of1: ETX

ETX – Expected Transmissions

- For every packet transmission, measure how many tries it takes until an ACK is received
  - Use as a measure of how many transmissions to expect
- Keep a moving average, for every neighbor
- To build routes, simply compute the sum of all ETXes

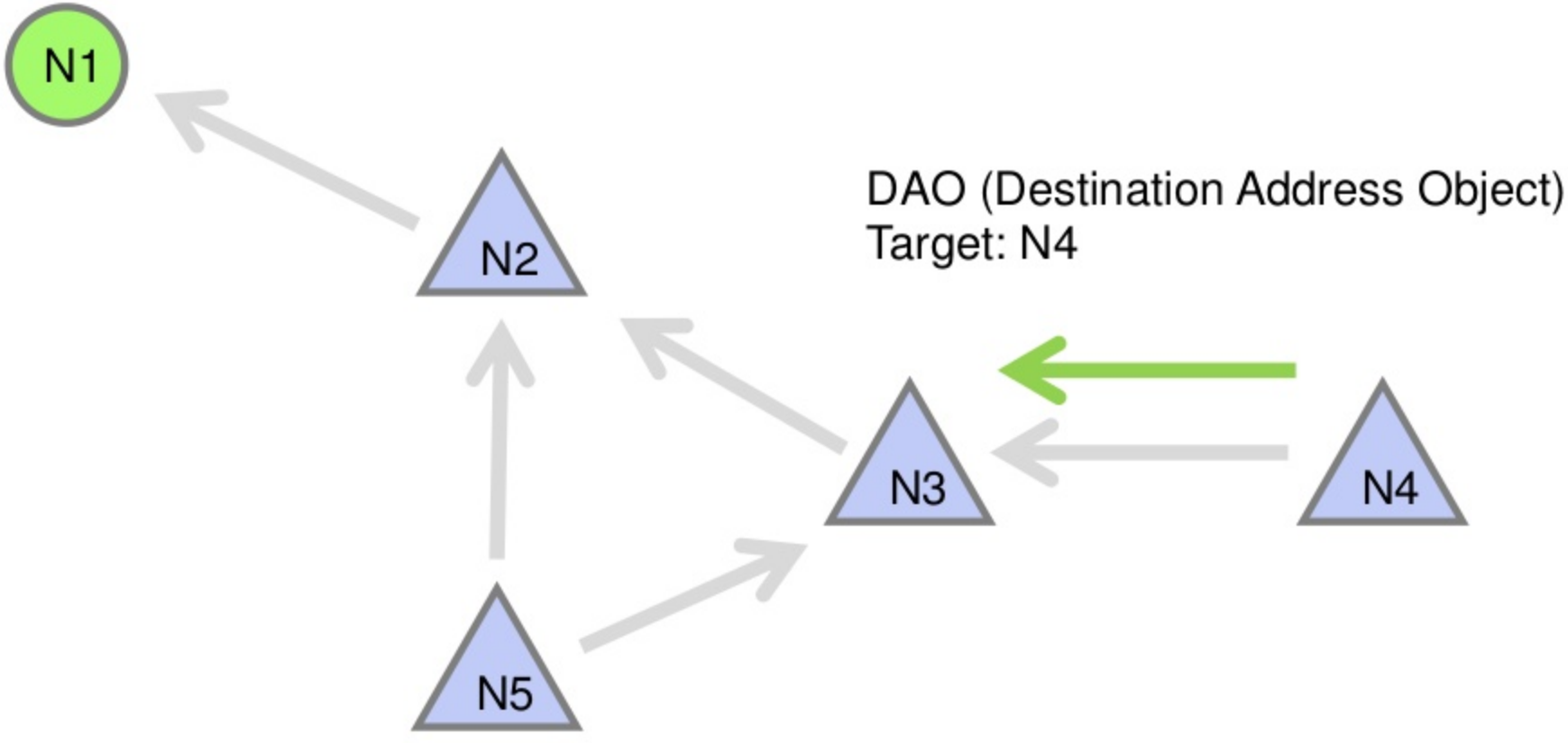
ETX – illustration



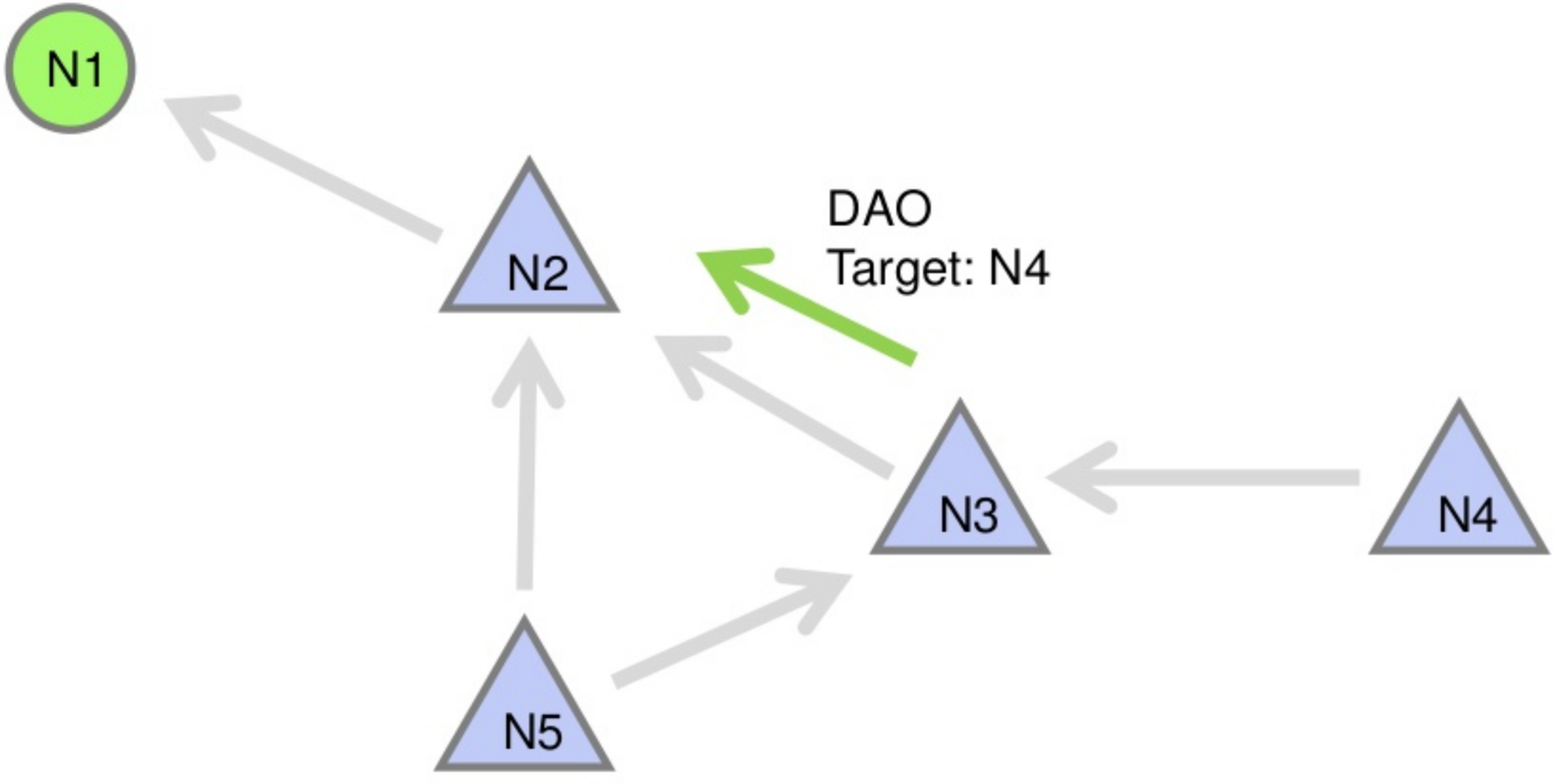
RPL downward routes

- Storing mode vs non-storing mode
  - Storing mode: all nodes store the addresses of their child nodes
    - Drawback: needs routing tables
    - This is how Thingsquare/Contiki does this
  - Non-storing mode: the root node knows the full route to all nodes, sends full route in every packet
    - Drawback: increases header overhead
    - This technique is known as source routing

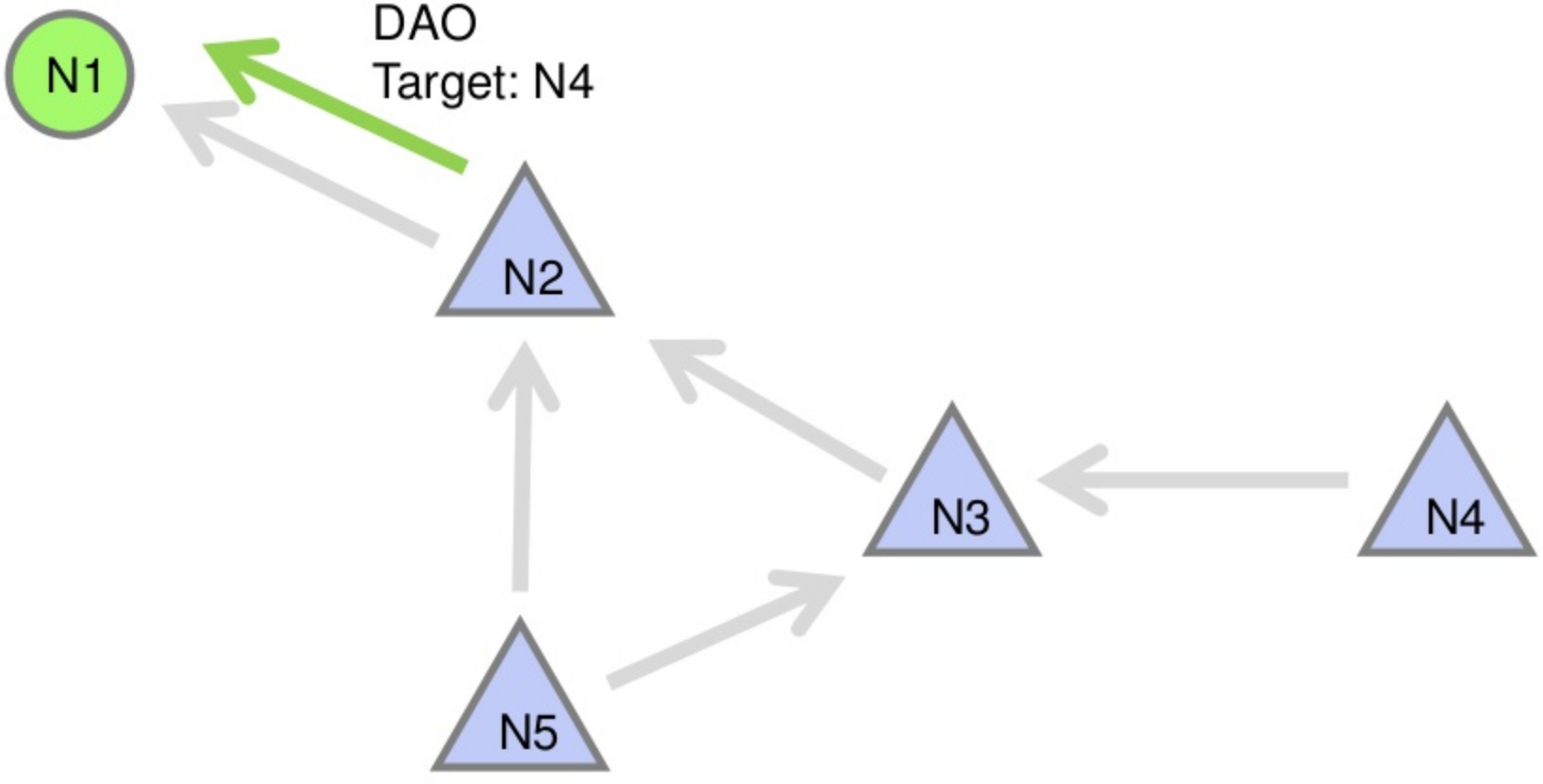
RPL downward routes



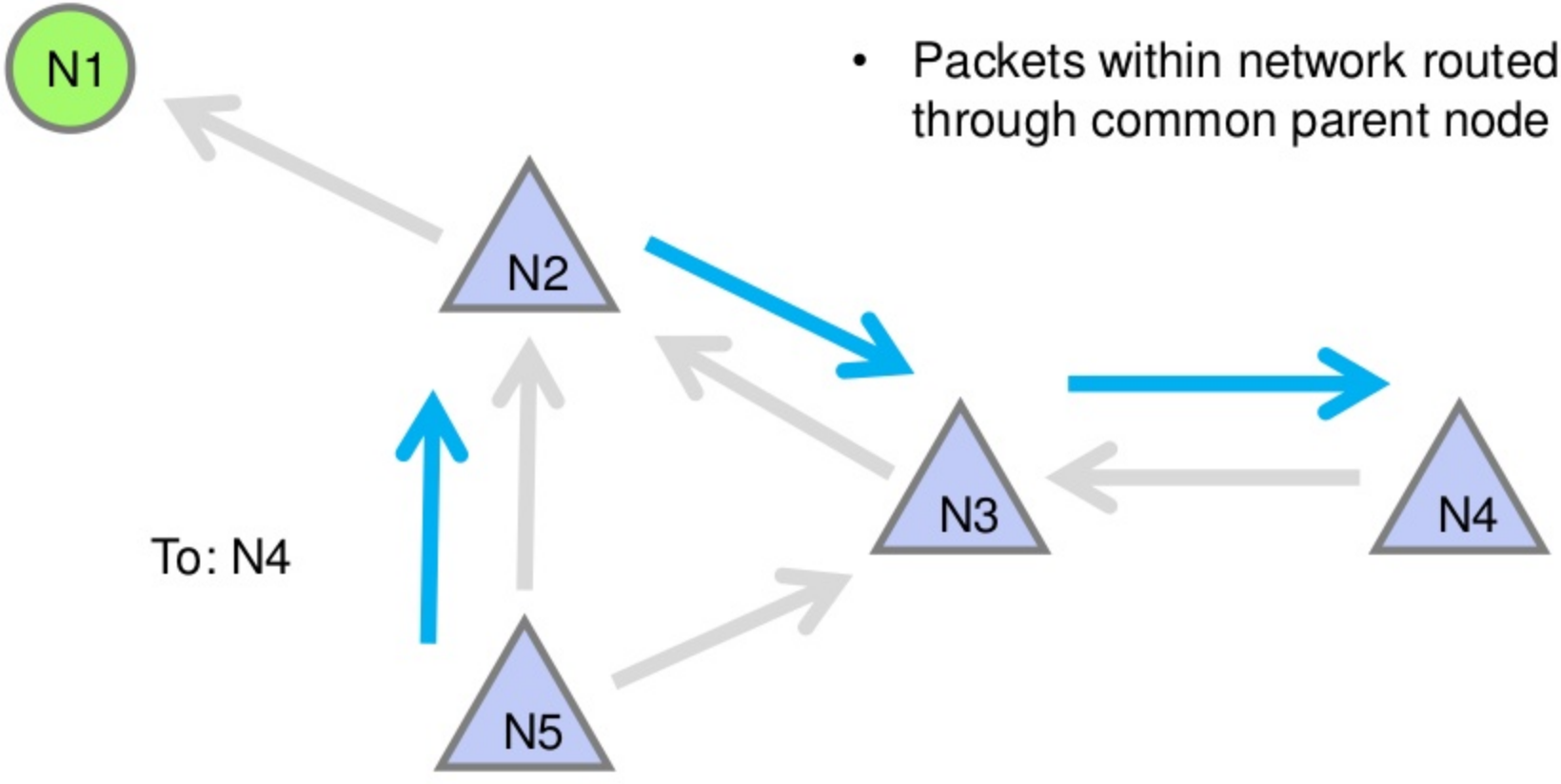
Node IPv6 RPL Network Formation



Node IPv6 RPL Network Formation



RPL Routing: Down



- Packets within network routed through common parent node

RPL network join

- New nodes that appear in the network broadcast Destination Information Solicitation (DIS) packets
- Connected nodes that hear DIS packets respond with a DIO

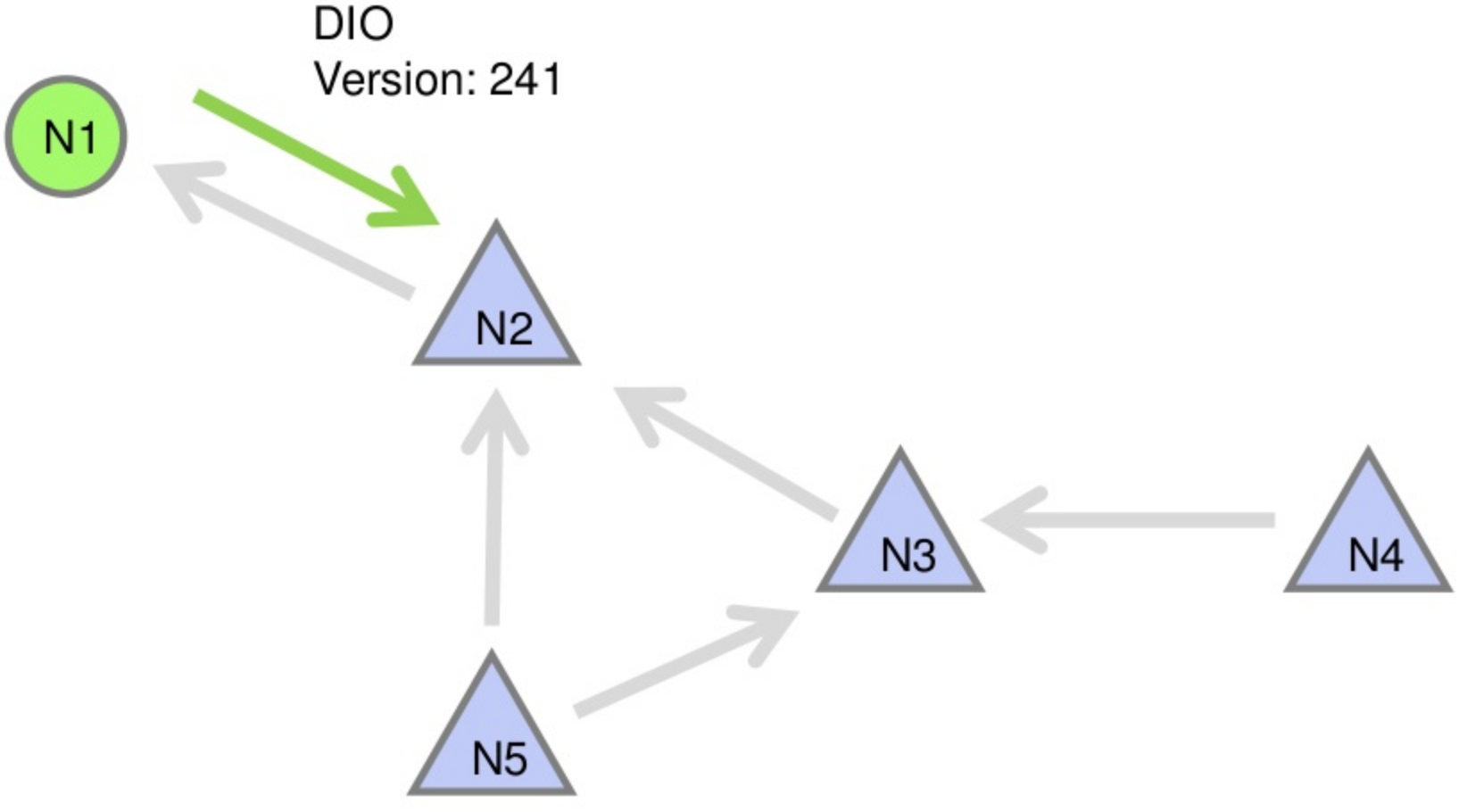
RPL control traffic suppression

- Once the RPL network is established, it reduces the rate of control messages
  - Expontential increase
- To avoid control message explosion, nodes suppress transmissions if it hears too many messages from others
  - Called the Trickle algorithm (Phil Levis et al)

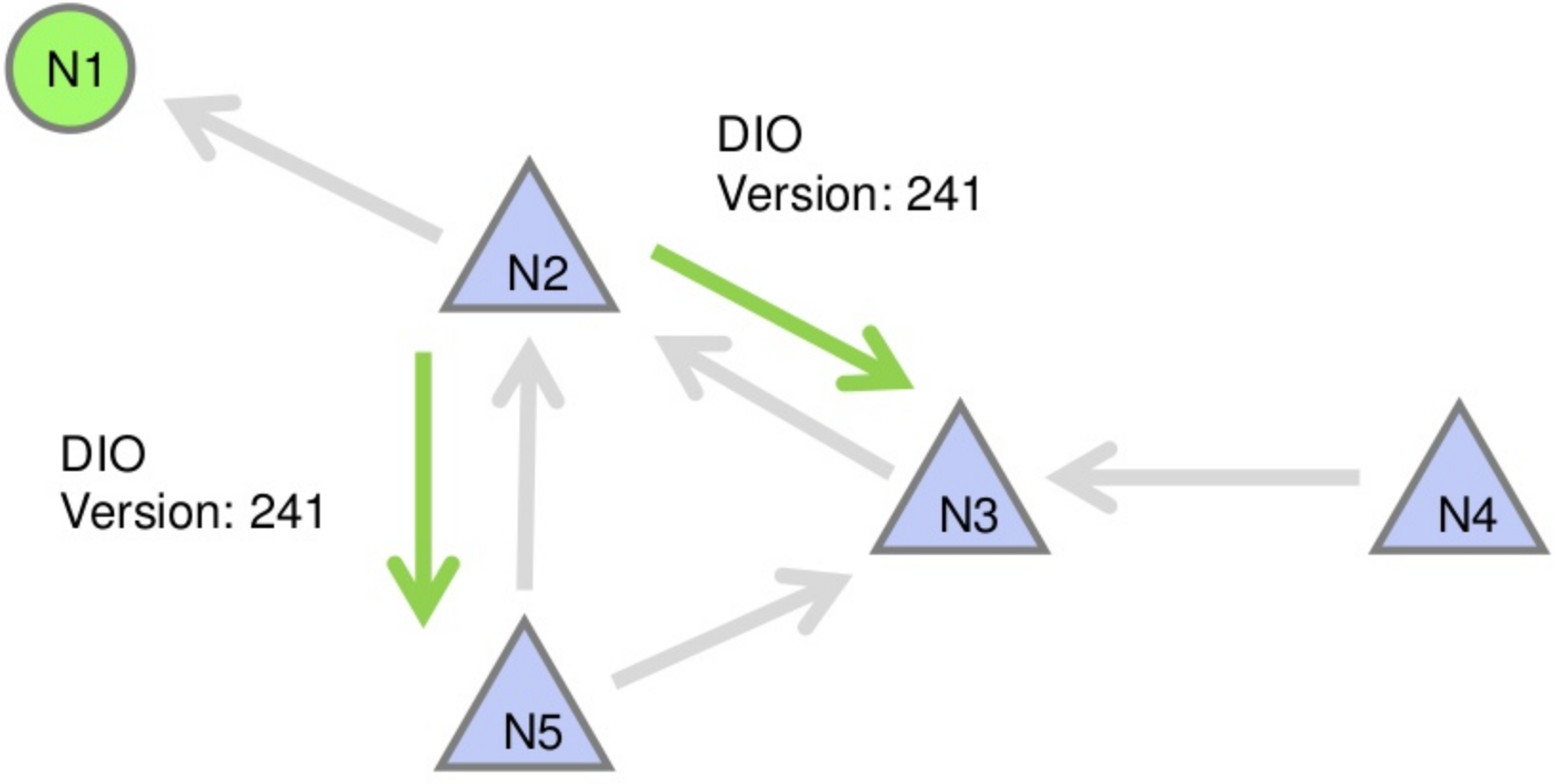
RPL repair

- Loop detection: packets carry backward path information in header
- When a loop is detected, the first few packets are allowed through the loop
- If the same packet is seen once again, a local repair is initiated
  - Node sets rank to infinity, sends DIS
- RPL global repair
  - Increase version number, rebuild DODAG

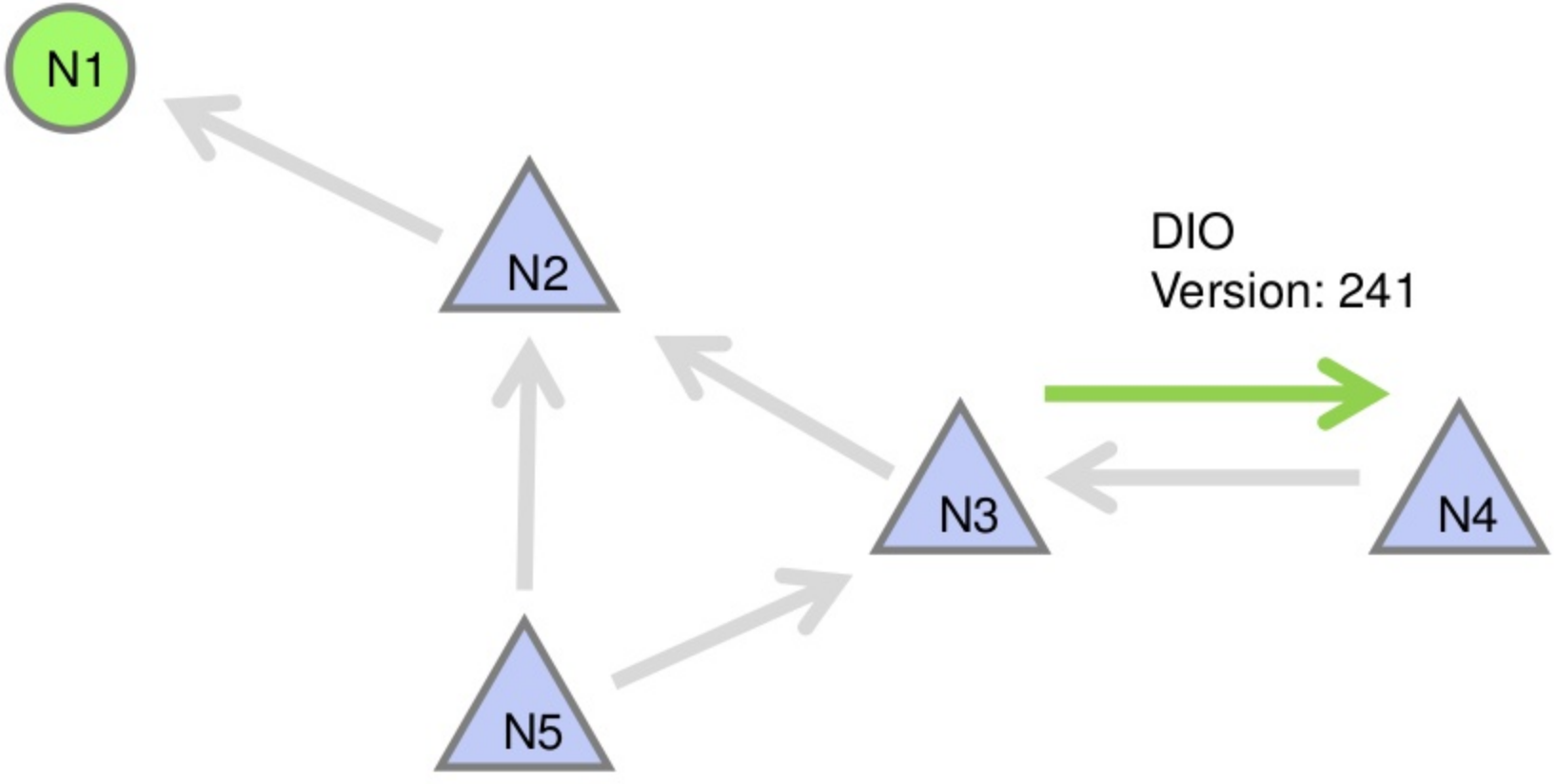
RPL Global Repair



RPL Global Repair



RPL Global Repair



RPL any-to-any routing

- RPL is optimized for many-to-one routing
  - All report packets to the sink
- Any-to-any routing must go through closest common ancestor
  - The root, in the worst case
- RPL P2P mode a suggested solution
  - AODV-like reactive route discovery with source routing
  - Contiki implementation exists: http://contiki-p2p-rpl.gforge.inria.fr/

RPL pitfalls

- Rebooting the root node
  - Must listen for a while to obtain network version
- Global repair may take a while

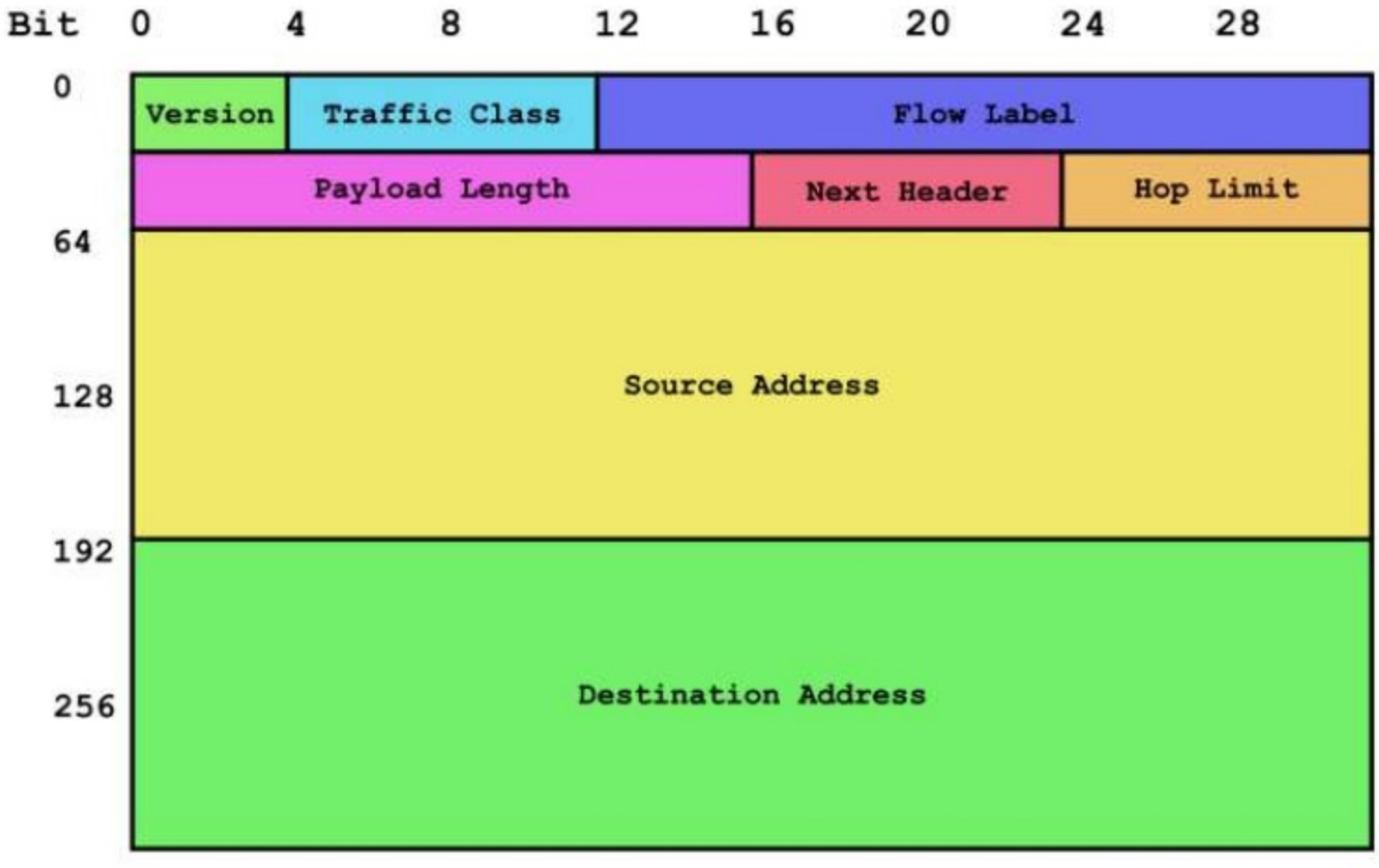
In Contiki

- ContikiRPL code is difficult to use
  - Integrated with the uIPv6 stack
  - Complex API
- Thingsquare provides a simple-rpl module
  - Gracefully handles root node reboots
  - Provides global repair / local repair API

In Contiki

- RPL node types
  - Mesh nodes: RPL routers, can be routed to
  - Leaf nodes: does not route RPL traffic, can be routed to
  - Feather nodes: routes RPL traffic, cannot be routed to

Addressing: IPv6



IPv6 addresses

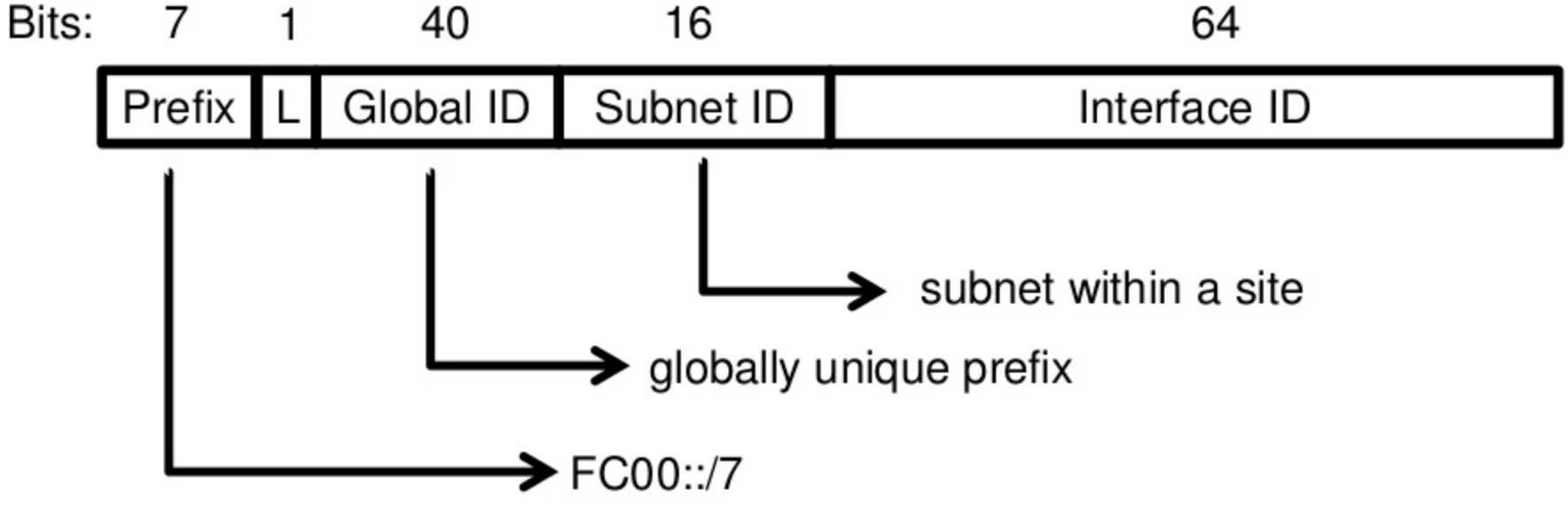
- 128 bits
  - Much more than the 32 bits of IPv4
- Examples:
  - 2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - fe80::12:7400:1:100
- An IPv6 host has multiple addresses
- An IPv6 network interface has multiple addresses
- IPv6 addresses are typically automatically generated
  - From a given prefix

IPv6 duplicate address detection

1. Pick a tentative address
   - Usually from your EUI-64 address
     - Or timestamp
2. Send a neighbor discovery message for the tentative address
   - Send a Neighbor Solicitation (NS), wait for a Neighbor Advertisement (NA)
3. If there is a reply, go back to step 1
4. If nobody replies, claim the tentative address
   - The tentative address is now preferred address

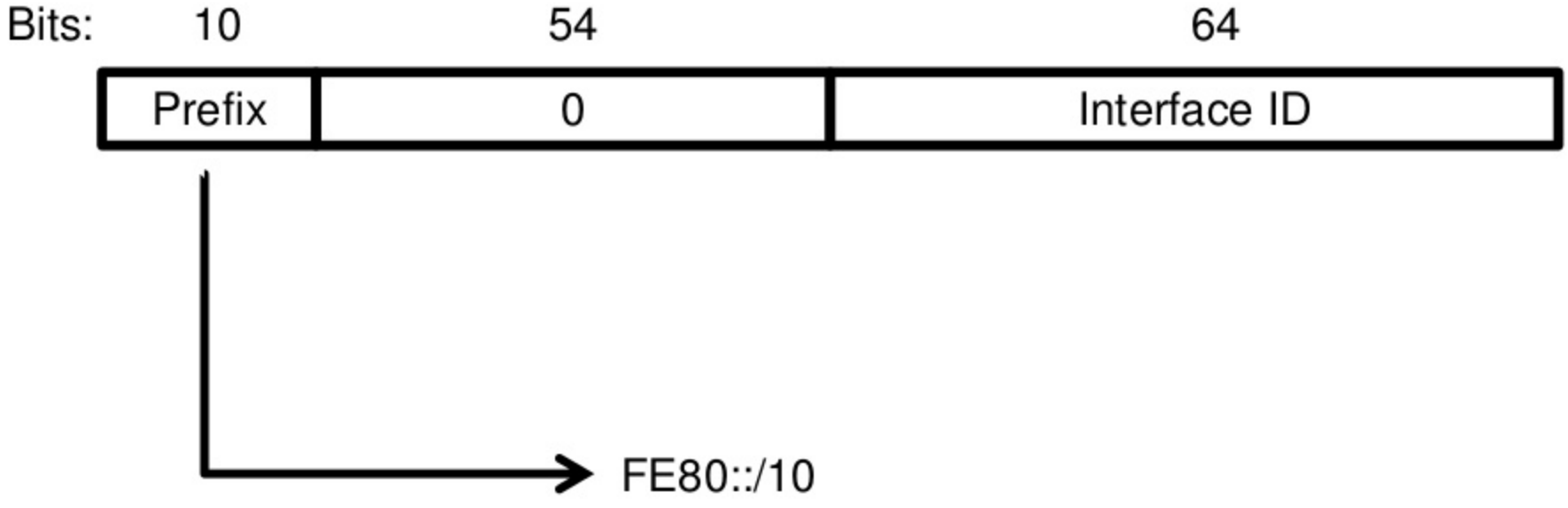
IPv6 address generation

Unique local unicast IPv6 address (RFC 4193)



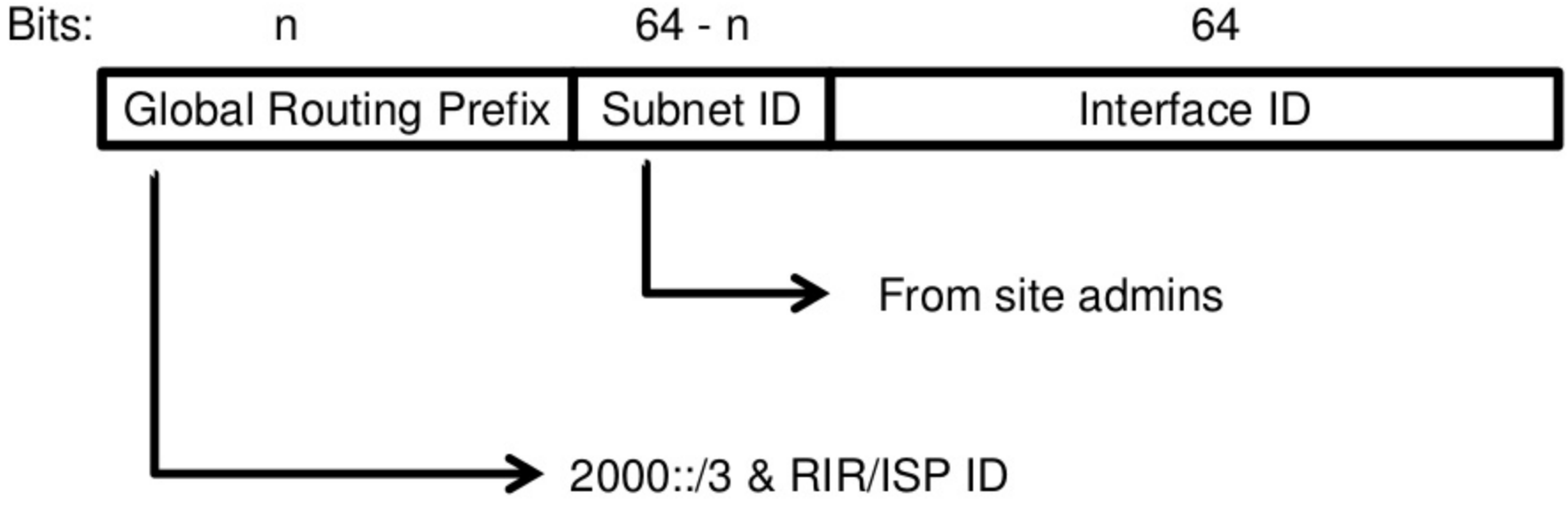
IPv6 address generation

Link-local unicast IPv6 address (RFC 4291)



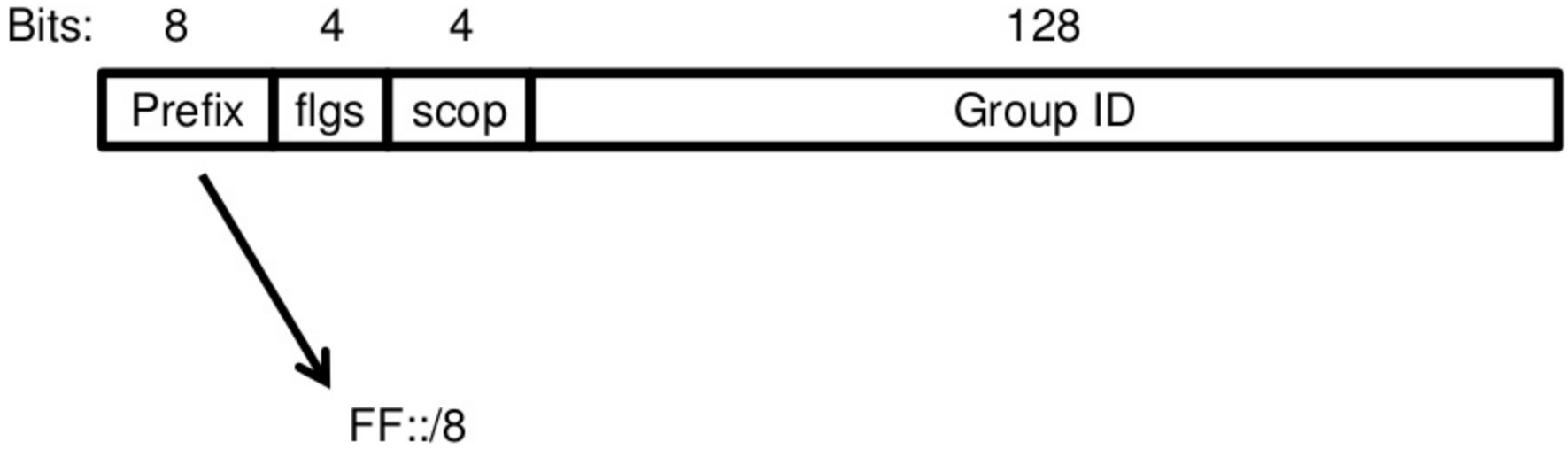
IPv6 address generation

Global unicast IPv6 address (RFC 3587)



IPv6 address generation

Multicast IPv6 addresses (RFC 3513)



IPv6 shortening

- Replace any number of zeroes with ::
  - But only once
- Replace any leading zeroes in 16-bit field omitted
  - 01ff -> 1ff
- Hex as lower-case letters
- Eg
  - fe80::1db:9

IPv6 Good-to-know

- /n is CIDR notation: first n bits are fixed
- Multicast ff::/8
  - but ff0x:: reserved (where x = 0..f)
- all nodes multicast
  - ff01::1 (interface-local)
  - ff02::1 (link-local)
- all routers multicast
  - ff01::2 (interface-local)
  - ff02::2 (link-local)
  - ff05::2 (site-local)
- fe80::/10 – link local addresses
- fc00::/7 – unique local addresses
- Global addresses 2000:: – 3ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

In Contiki

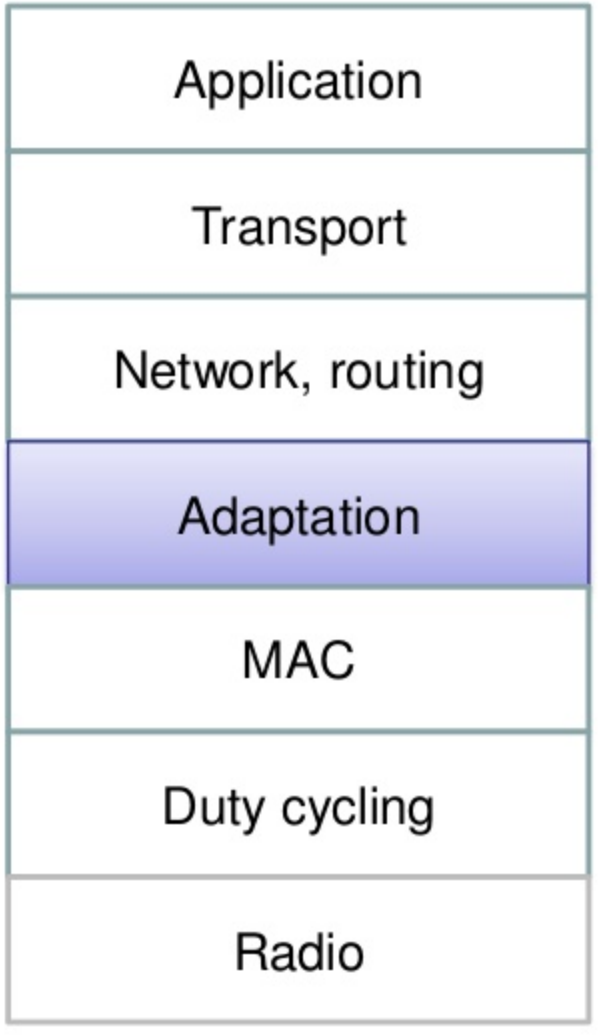
- Contiki provides a full IPv6 stack
  - With support for IPv6 address management, multiple addresses, duplicate address detection, neighbor tables, routing tables

Takeaway network layer

- The network layer addresses devices and routes packets
- The IoT/IP stack uses IPv6 and RPL

The IoT/IP Protocols

The adaptation layer



The problem

- How do we transmit IP packets over a medium like 802.15.4?
  - 802.15.4 frames are tiny
  - 802.15.4 bit rates are slow
  - IP headers are large – IPv6 headers are even larger
  - IPv6 packets may be huge
- The solutions
  - Header compression
  - Fragmentation

IPv6 over 802.15.4

- IP headers are large
  - IPv6 headers are 40 bytes – minimum
  - UDP headers are 8 bytes, TCP headers 20 bytes – minimum
- IP packets may be huge
  - 1280 bytes must be supported
- 802.15.4 frames are 127 bytes – maximum

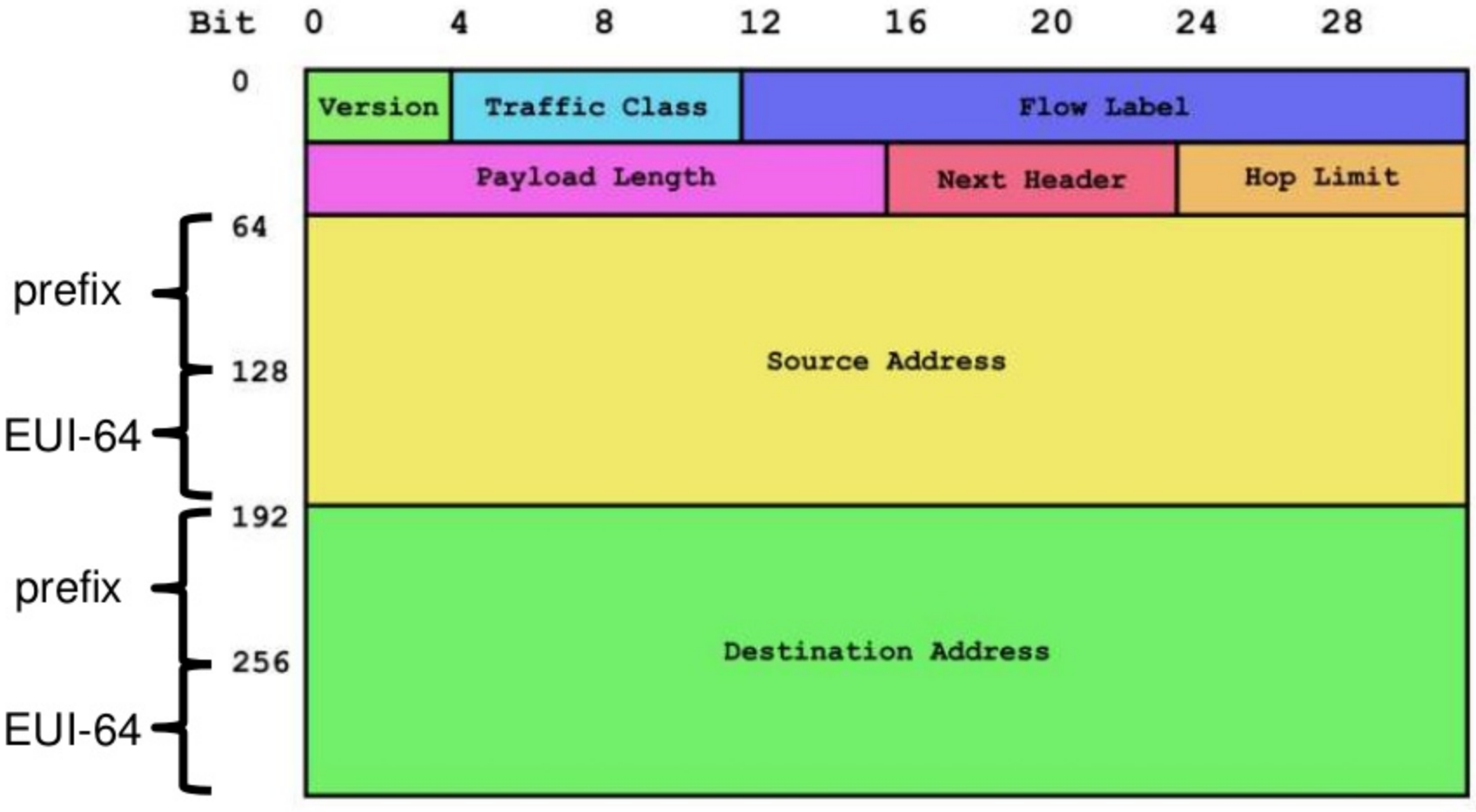
6lowpan

- standard defined by the 6lowpan working group
  - 6lowpan = IPv6 for Low-power Wireless Personal Area Networks
  - Because of the name of the standard, many people refer to IPv6 over 802.15.4 as ”6lowpan networks” or ”6lowpan stacks”
    - We don’t

6lowpan header compression

- Don’t send what the receiver(s) already know/can compute
- IPv6 addresses often automatically constructed from the MAC address – if so, just set a bit
- Transport layer may contain a length field
  - Can be computed from link-layer header – set a bit
- Some port numbers are more common than others
  - Set a bit
- And so on

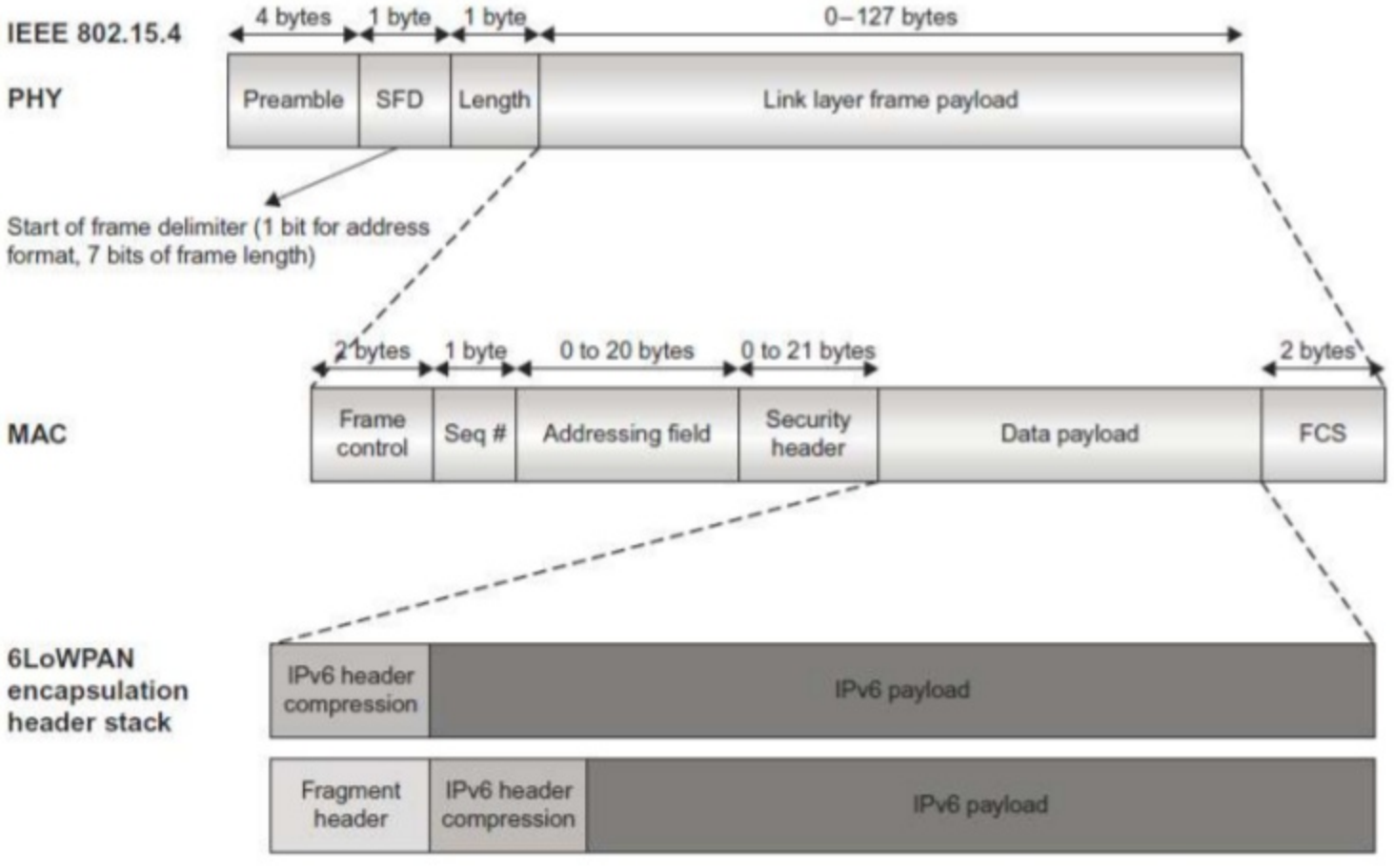
IPv6 headers



6lowpan fragmentation

- Split large IPv6 packets across multiple link-layer frames
  - Reassemble at every hop
- Set fragmentation bit in first packet
  - Fragmentation header in subsequent packets
  - When all packets have been received, send packet upwards to network layer

IPv6-in-802.15.4 frame



Further issues

- IPv6 neighbor management defined for transitive links
  - Transitive: if A hears B, B will also hear A
  - Wireless multi-hop links are non-transitive
- This impacts duplicate address detection
  - If node A and B happen to pick the same address, they may not know about it
- Multicast is expensive for wireless
- Optimizations for 802.15.4 links: RFC 6775

In Contiki

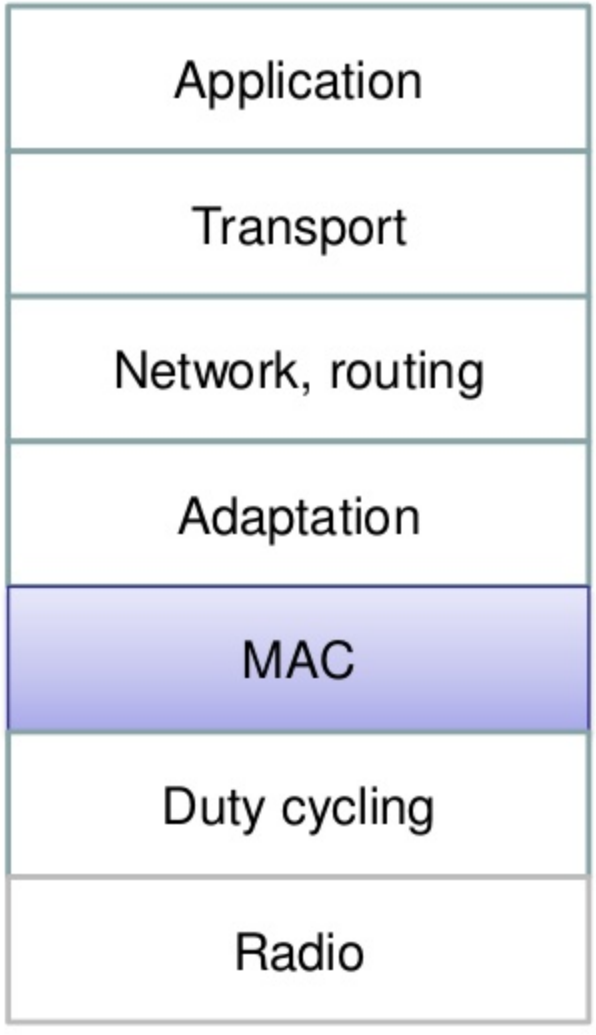
- The 6lowpan layer does header compression, fragmentation
- Informs the upper layers about the number of retransmissions needed by the MAC layer
- Shields the IPv6 stack from radio-level details
  - Such as signal strength

Takeway from adaptation layer

- Specifies how to transmit IP packets over the radio link layer
  - Header compression
  - Fragmentation
  - Called 6lowpan for IEEE 802.15.4

The IoT/IP Protocols

The MAC layer



The MAC layer

- The simplest layer in the IoT/IP stack
- CSMA/CA: Carrier Sense Multiple Access with Collision Avoidance
  - Sense the medium before sending
  - Back off if someone else is sending
- May result in hidden terminal problem
  - But the capture effect mitigates this to some degree [c.f. Österlind et al, IPSN 2012]

In Contiki

- Two MAC layers
  - CSMA/CA
    - Regular CSMA/CA layer
    - Timing depends on RDC layer
    - Network layer decides number of transmissions
  - nullmac
    - A MAC layer that doesn't do anything

Takeaway from MAC layer

- Avoid collisions, back-off if there is too much other traffic

The IoT/IP Protocols

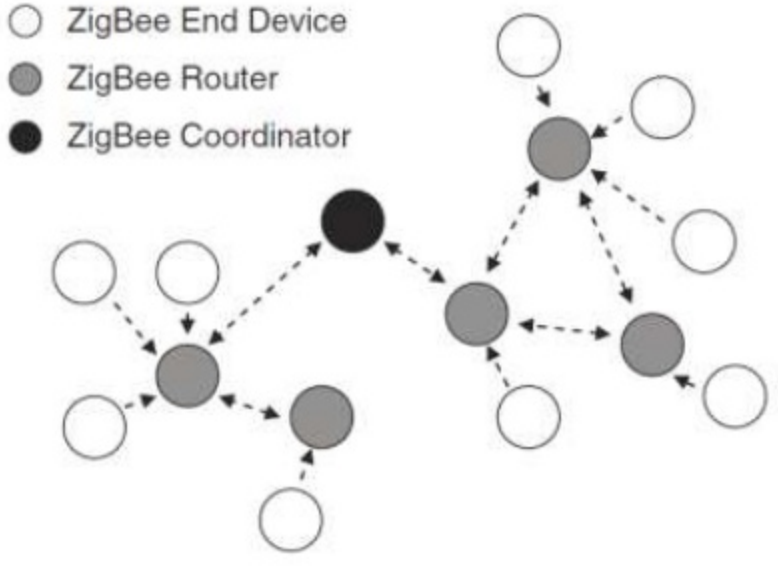
The radio duty cycling layer

The duty cycling problem

- Radio transceivers consume (significant) power when just listening
  - Often as much as or more as when transmitting
- Need to duty cycle the radio transceiver
  - Keep it off as much as possible
  - Coordination needed

Simple duty cycling methods

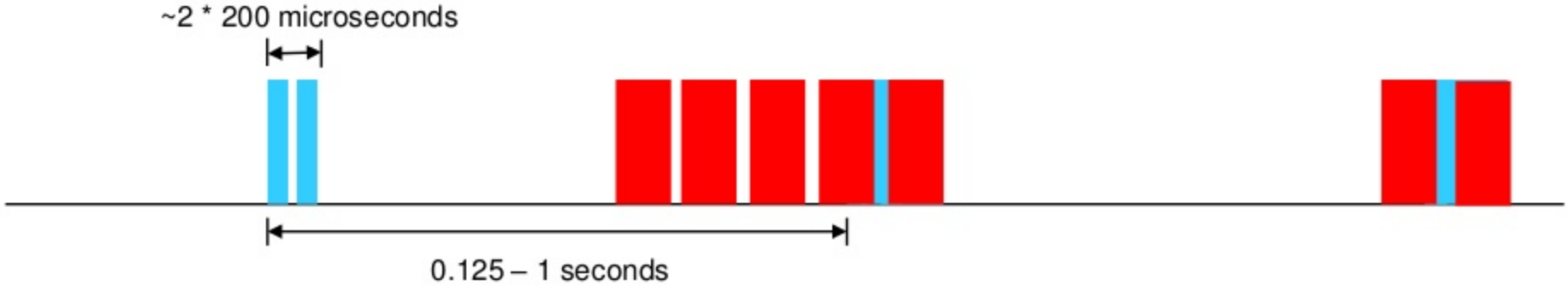
- Star network duty cycling (ZigBee)
  - Routers and coordinators are always on
    - Must be plugged into power source
  - Leaf nodes may be off



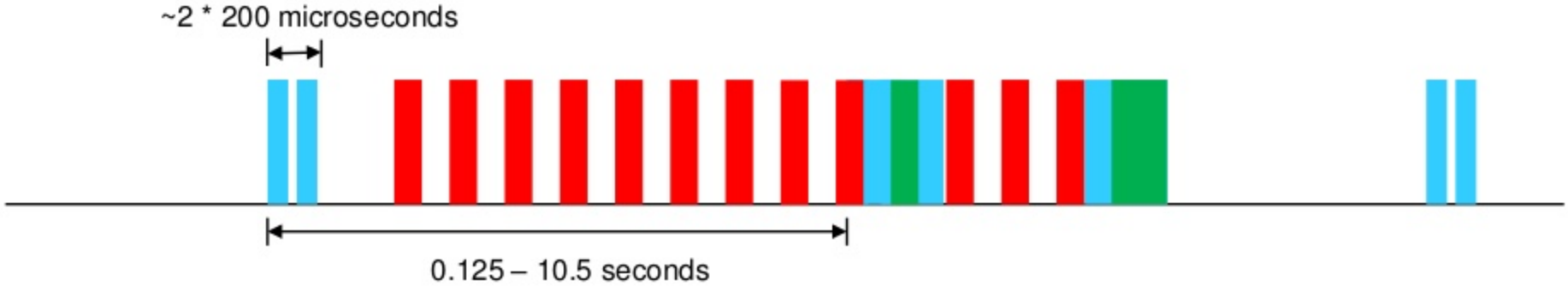
Sleepy mesh duty cycling

- Allow arbitrary mesh communication between nodes
  - ContikiMAC
    - Highly optimized for 802.15.4
  - Drowsie (Thingsquare firmware)
    - Portable across radios
  - 802.15.4 CSL
    - Standardized radio duty cycling

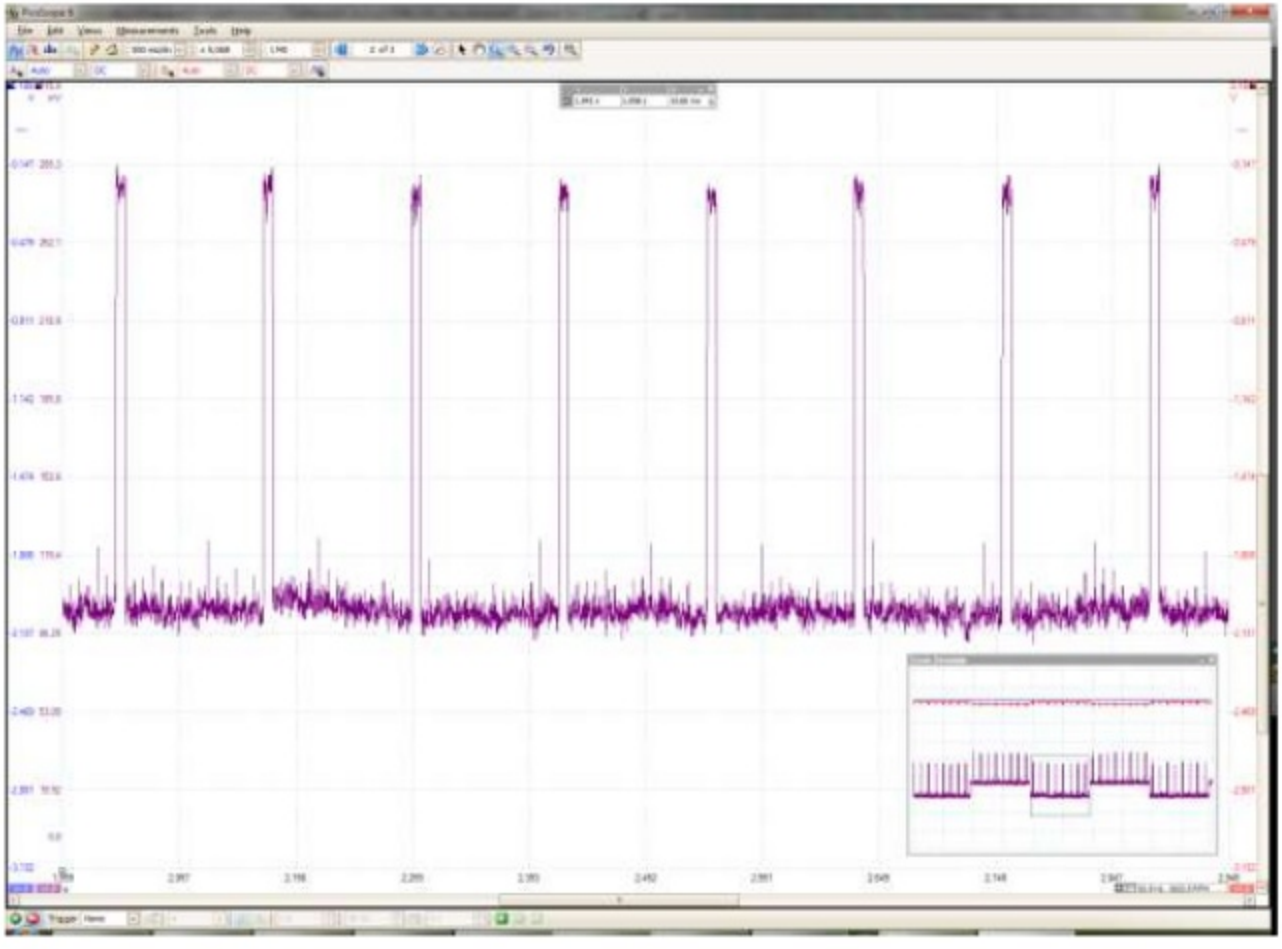
ContikiMAC



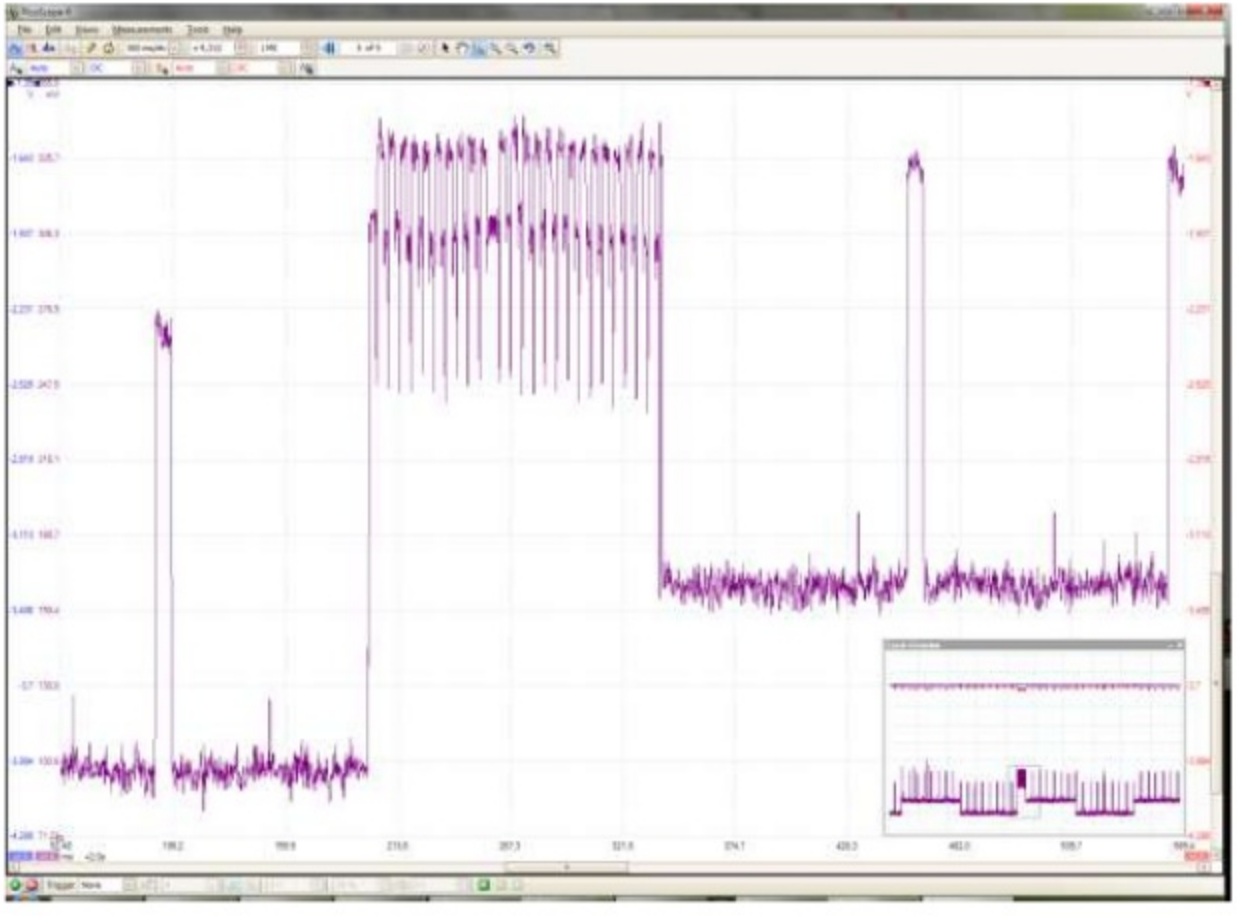
Thingsquare 802.15.4e CSL



Drowsie



Drowsie transmission



Implications of duty cycling

- Slight increase in response time
  - Must wait until receiver is awake
- Throughput reduced
- But we save power!
  - Thingsquare CSL idle ca 0.5% DC

In Contiki

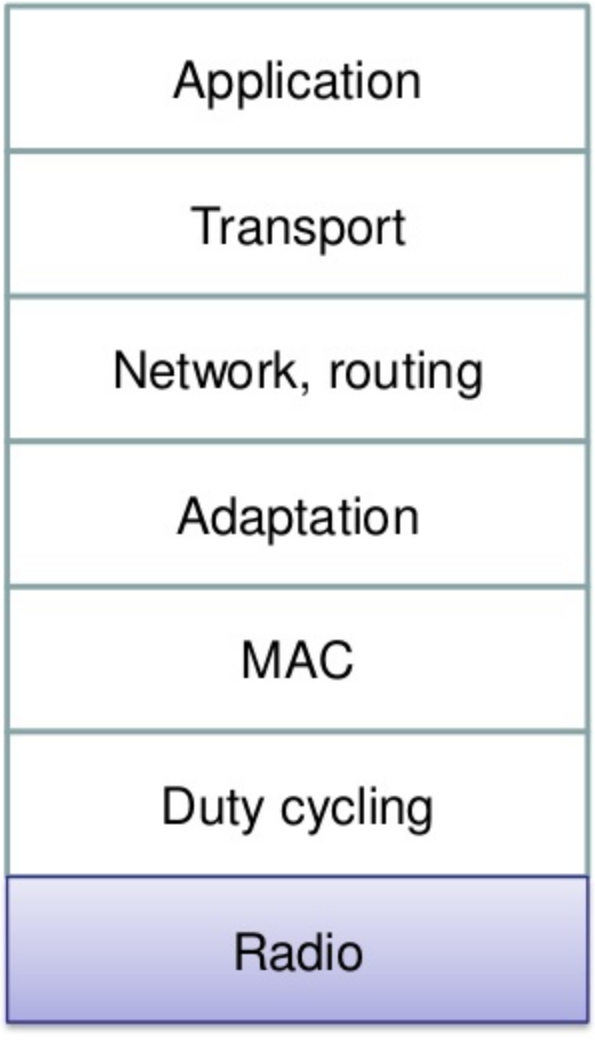
- The RDC layer also does encryption, duplicate frame filtering, address filtering
  - AES decryption / encryption
  - For sub-GHz radios, support for 802.15.4-like behavior
- ContikiMAC
  - Highly optimized, for 802.15.4
- Drowsie
  - More relaxed, works on more platforms
    - Less power-efficient
- 802.15.4e CSL
  - Standardized power-saving!
  - Similar to ContikiMAC
- nullrdc
  - An RDC layer that always keeps the radio on

Takeways from radio duty cycling layer

- Duty cycling allows sleepy meshing
  - Unlike many other technologies (e.g. ZigBee, Bluetooth Low-Energy)

The IoT/IP Protocols

The radio layer



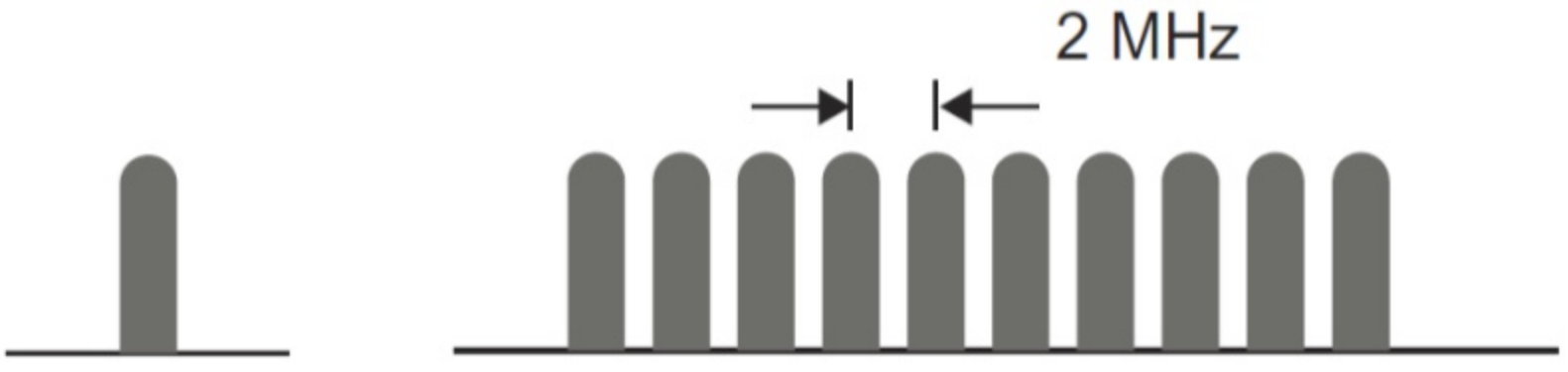
The radio layer (PHY)

- Getting 1s and 0s across the air
- Modulation and encoding
  - How the 1s and 0s are represented
- Modulation examples
  - On off keying (OOK), amplitude shift keying (ASK), frequency shift keying (FSK), phase shift keying (PSK)
  - Direct sequence spread spectrum (DSSS), offset quadrature phase-shift keying (O-QPSK)
- Resilience and EMC
  - Forward Error Correction (FEC)
  - Whitening

IEEE 802.15.4 PHY

- 27 logical channels
  - 0-10: sub GHz bands
  - 11-26: 2.4 GHz band
- Other various modulations, freq bands etc
- Low-power mechanisms 2012

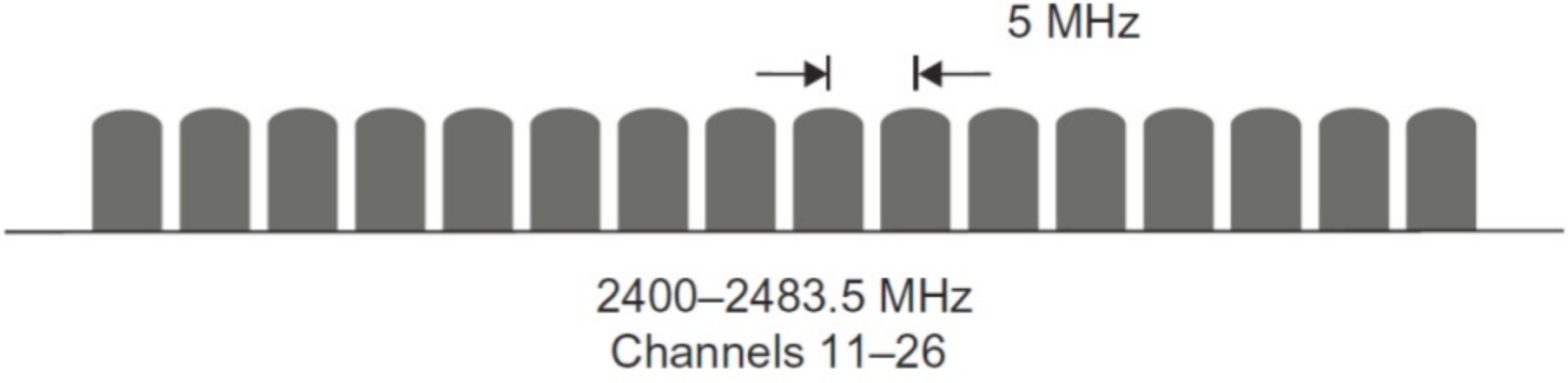
IEEE 802.15.4 channels



868-868.8 MHz
Channel 0

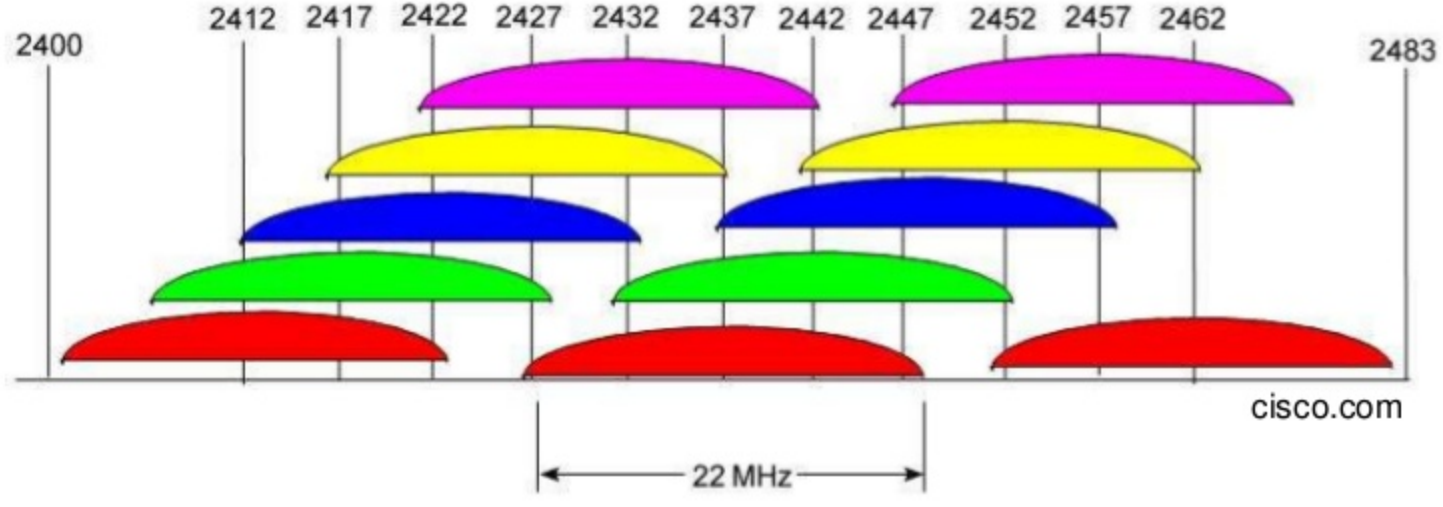
902 - 928 MHz
Channel 1-10

IEEE 802.15.4 channels

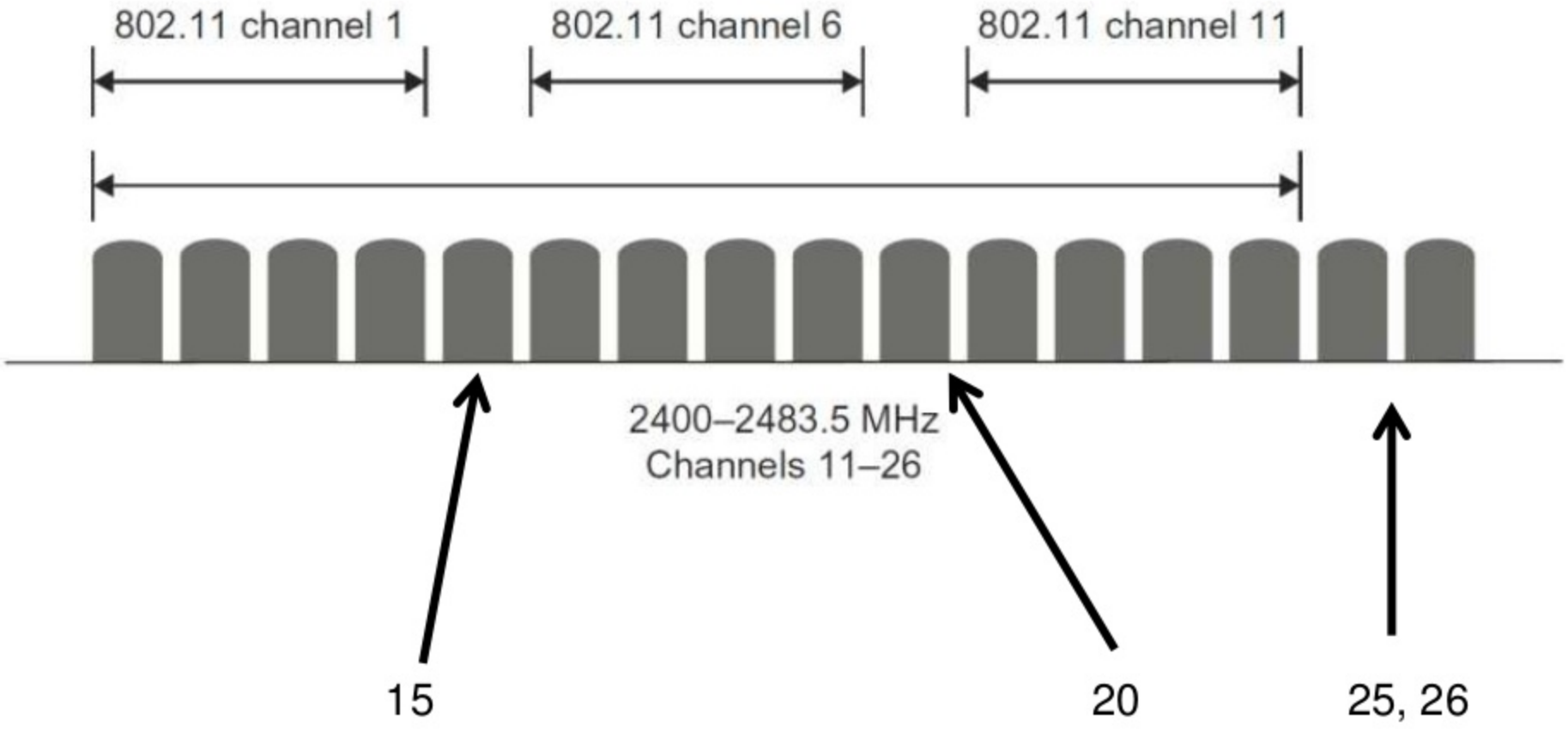


IEEE 802.11 (Wi-Fi)

- Channels 1, 6, 11 are not overlapping



802.15.4 and Wi-Fi overlap



IEEE 802.15.4 is more

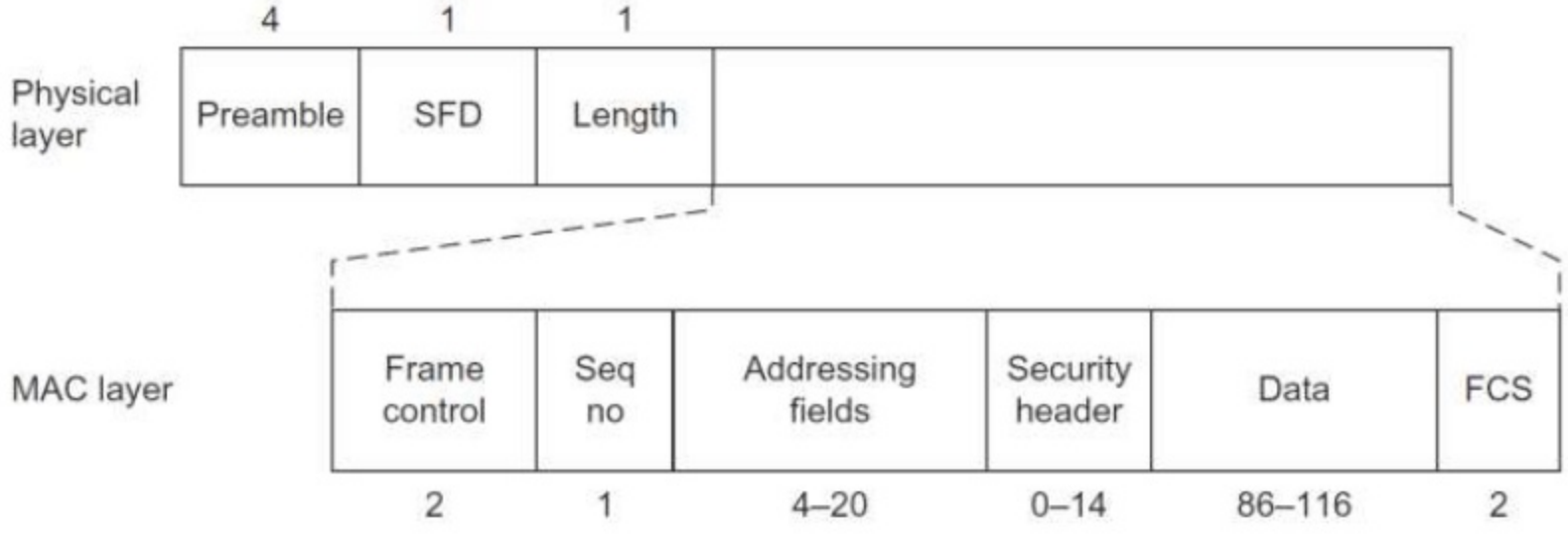
- IEEE 802.15.4 does not only specify a PHY layer
  - Node addressing
  - Network addressing
  - MAC layer framing
  - MAC layer mechanisms
  - Encryption
  - Node types: Full Function Devices (FFDs) and Reduced Function Devices (RFDs)
- IEEE 802.15.4e, IEEE 802.15.4g: even more

IEEE 802.15.4 addressing

- Node addresses
  - Three address types: simple, short and long
  - Short addresses
    - 2 bytes
    - Dynamically assigned at run-time
  - Long addresses
    - 8 bytes – EUI 64™
    - Statically assigned by manufacturer
- Network addresses – PAN ID
  - Dynamically assigned

802.15.4 MAC layer framing

- Data frames



- ACK frames
  - Like data frames, but no addressing, data, security

802.15.4e MAC layer frame

Octets: 1/2	**0/1**	**0/2**	**0/1/2/8**	**0/2**	**0/1/2/8**	**0/1/5/6/10/14**	**variable**		**variable**	**2**
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Information Elements		Frame Payload	FCS
		Addressing fields					Header IEs	Payload IEs		
MHR							MAC Payload			MFR

802.15.4 framing

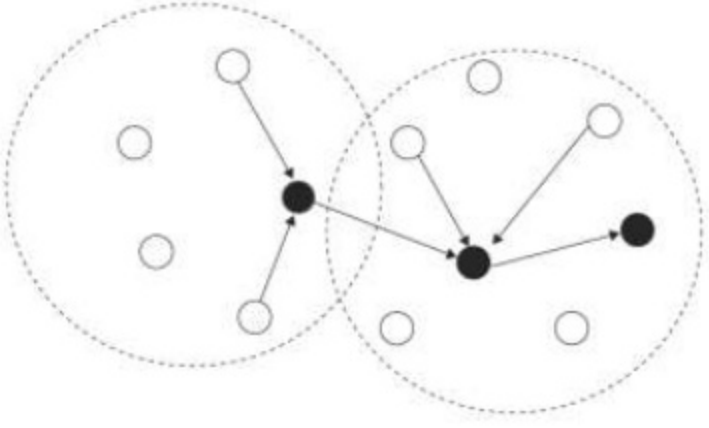
- Frames are protected by CRC-16
- 127 bytes maximum data frame size
- ACK frames must be sent within a specific time
  - Usually handled automatically by hardware

802.15.4 security

- AES 128 encryption
- Message Integrity Check (MIC)
  - Protects against the message being altered
  - Sequence number protects against replay attacks
  - Timestamp may be used to protect against delay attacks
- Uses AES CBC-MAC mode
  - Authentication and confidentiality

IEEE 802.15.4 node types

- Not used (much) with low-power IP, but good to know about
- FFDs
  - Always on
  - PAN coordinators
  - Assigns short addresses
  - Use so-called Beacon-mode
- RFDs
  - May be off a lot
  - Cannot be mesh nodes



IEEE 802.15.4e

- Standardized duty cycling
  - Coordinated Sampled Listening (CSL)
    - Very similar to ContikiMAC
  - Time Synchronized Channel Hopping (TSCH)
    - Duty cycling and channel hopping
    - Based on TSMP / WirelessHART
  - Receiver Initiated Transmission (RIT)
    - Similar to Contiki's LPP

IEEE 802.15.4g

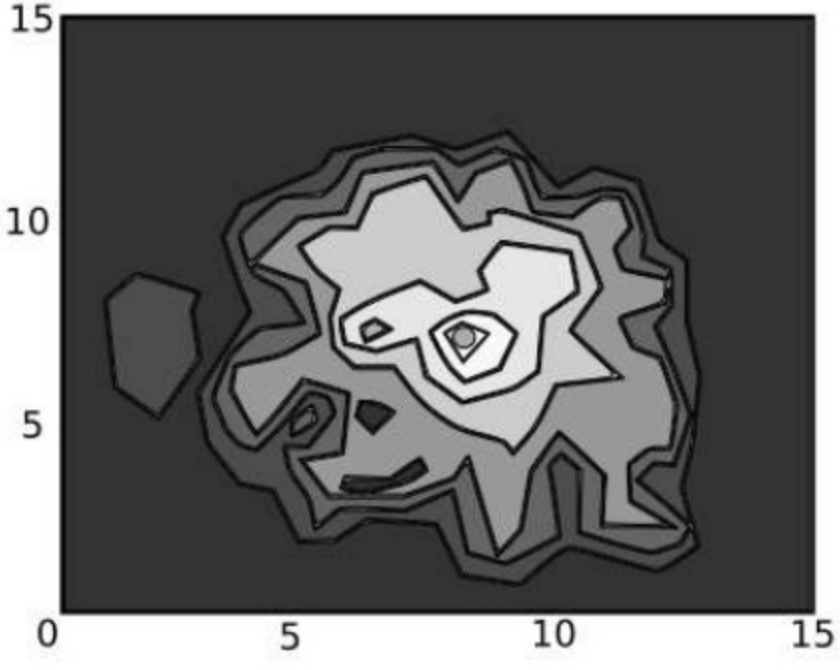
- Smart metering
- Better support for sub-GHz modes
- Possible to run with some existing sub-GHz chips
  - TI CC1101, TI CC1120

In Contiki

- The radio layer is about low-level drivers
- Complexity such as encryption is put into the RDC layer

Wireless connectivity

- Counter-intuitive
- Not a simple replacement to cables

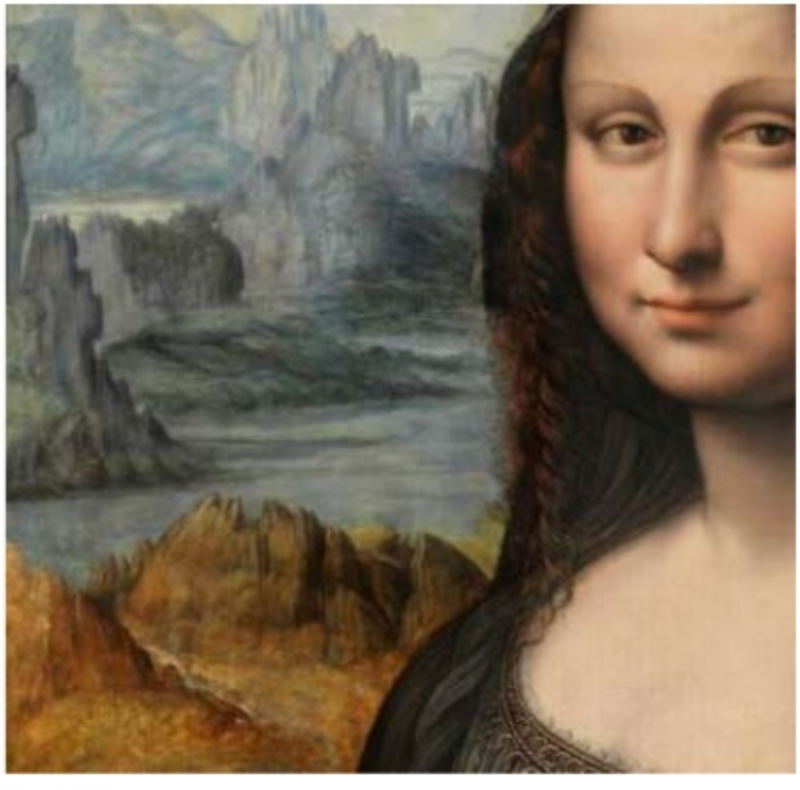


Takeways from the radio layer

- Different radios have different properties, benefits, drawbacks
- IEEE 802.15.4 is currently a popular standard
  - But don't worry too much about the details

Hands-on: UDP sockets, light sensors, LEDs, timers

Challenge!



© Leonardo

- Scenario: Old art in museum.
- Color ages/deteriorates from sunlight / UV exposure
- Hypothetical application:

Light alarm

Challenge:
Light alarm

- Too bright? Notify us!
- Logged on the backend
- Warn by blinking LEDs on the device

Challenge: Light alarm

- Bonus points for taking into consideration:
  - Periodic logging of ambient light
  - Reducing alarm latency (read often)
  - Be a responsible network citizen, don't spam/flood
  - Manually be able to remotely enable/disable the application

Light alarm

- Use udp-broadcast.c, light-sensor.c, cookbook.c

Conclusions

- What we have done
  - Built an IoT cloud service
  - Built an IoT cloud-connected device
- What we have seen
  - The protocols underneath it all
- What we can do next
  - Develop our own connected product

More



http://thingsquare.com