Lecture 07
Thread

CS397/497 – Wireless Protocols for IoT

Branden Ghena – Spring 2022

With some advice from Neal Jackson (UC Berkeley)

Materials in collaboration with Pat Pannuto (UCSD)

Northwestern

Administrivia

- Lab 2 due today
  - If you're still having problems with it, let me know
  - Lots of little issues that people have been having
  - It's just not a very robust toolchain

- Lab 3 probably out tomorrow sometime
  - Goal is going to be to actually write some code that sends/receives BLE advertisements
  - Going to be a small group assignment

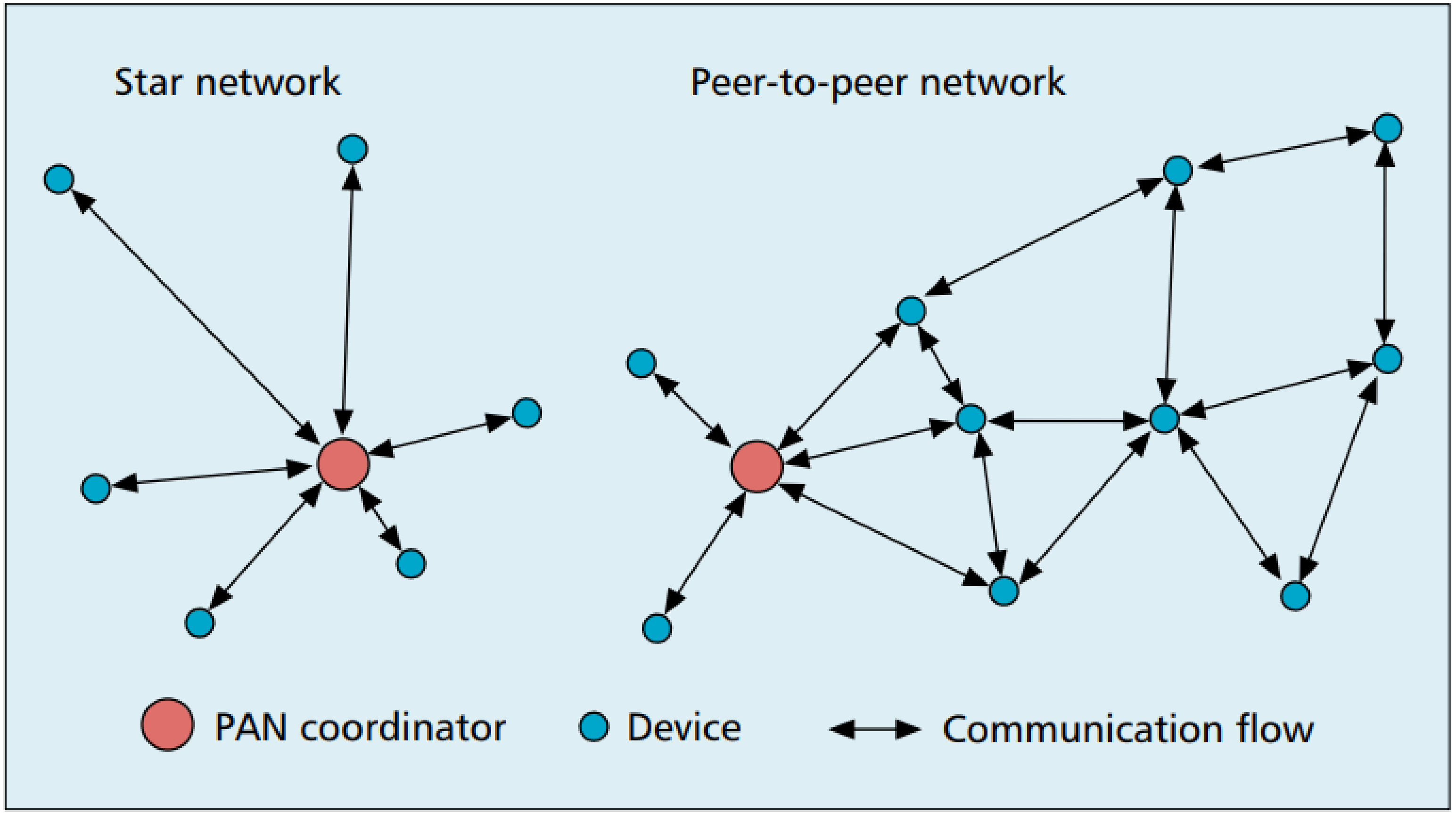
  - I'm going to extend it to be more than a week, probably 2 weeks?
    - I might overlap it with the Thread lab though

Today's Goals

- Explore 802.15.4 packet structure
- Describe goals and capabilities of Thread networks
- Understand addressing in Thread networks
- Describe runtime behaviors like network joining

802.15.4 network topologies

- Only specifies PHY and MAC, but has use cases in mind



Modes of operation

- Beacon-enabled PAN
  - Slotted CSMA/CA
  - Structured communication patterns
  - Optionally with some TDMA scheduled slots

- Non-beacon-enabled PAN
  - Unslotted CSMA/CA
  - No particular structure for communication
    - Could be defined by other specifications, like Thread or Zigbee

Beacon-enabled superframe structure

- Beacons occur periodically [15 ms – 245 seconds]
    - Devices must listen to each beacon
- Contention Access Period
    - Slotted CSMA/CA synchronized by beacon start time
- Inactive Period
    - No communication occurring. Assumes sleepy devices

Non-beacon-enabled PAN

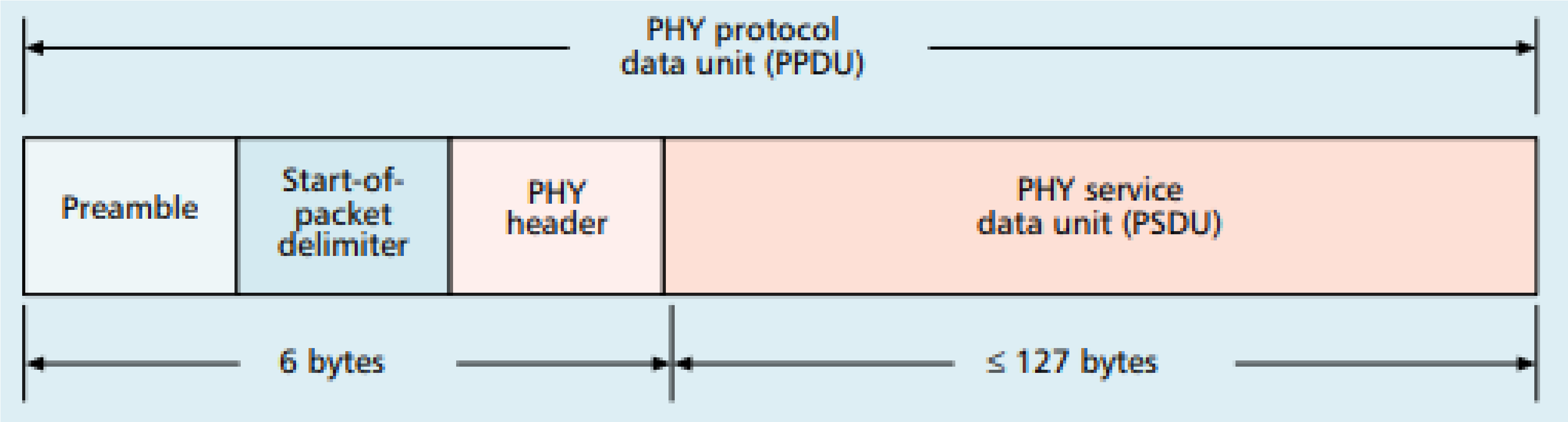
- Same idea, just no beacons
  - Which removes synchronization benefit (and slotted CSMA/CA)
  - Also removes beacon listening cost
    - Devices only need to check for activity before transmitting
  - Still need an algorithm to determine when it should receive data
    - All the time is a huge energy drain
    - Algorithms can get complicated here
    - **Could BLE mechanism of listen-after-send apply?**
      - Only if sending to a high-power device, not among equals

Outline

- **802.15.4 Packet Structure**
- Thread Overview
- Thread Addressing
- Runtime Behavior
- Using IP

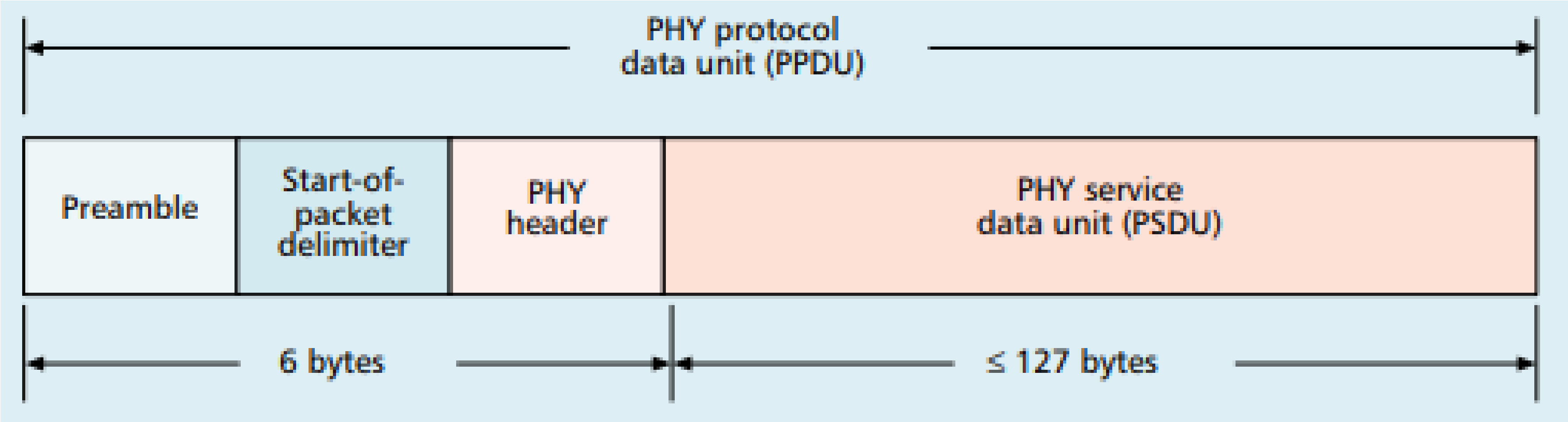
Base packet format

- Synchronization
  - Preamble: four bytes of 0x00
  - Start-of-Packet: 0xA7
- PHY Header
  - One field: length 0-127
  - **Why still 8 bits?**

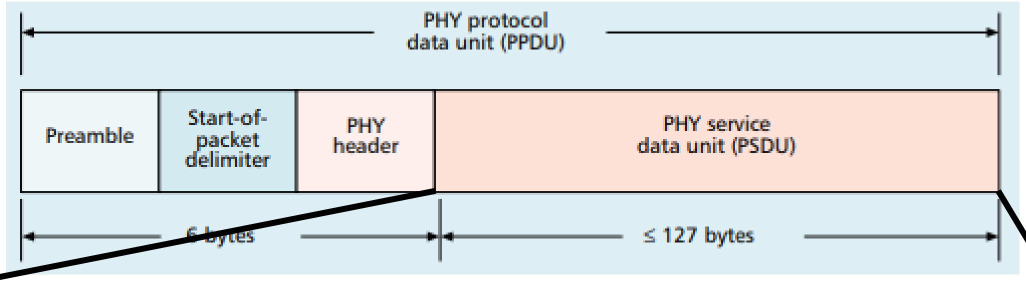


Base packet format

- Synchronization
  - Preamble: four bytes of 0x00
  - Start-of-Packet: 0xA7
- PHY Header
  - One field: length 0-127
  - **Why still 8 bits?** **Because computers depend on bytes**



MAC frame format



Octets:2	1	0/2	0/2/8	0/2	0/2/8	variable	2
Frame control	Sequence number	Destination PAN identifier	Destination address	Source PAN identifier	Source address	Frame payload	Frame check sequence
		Addressing fields					
MAC header						MAC payload	MAC footer

- Frame control
  - Header
- Sequence number
  - 8-bit monotonically increasing
- Addressing fields
  - PAN and addresses
  - Varies based on frame type
- Frame payload
  - Depends on frame type
- Frame check sequence
  - 16-bit CRC

Frame control

Octets:2	1	0/2	0/2/8	0/2	0/2/8	variable	2
Frame control	Sequence number	Destination PAN identifier	Destination address	Source PAN identifier	Source address	Frame payload	Frame check sequence
		Addressing fields					
MAC header						MAC payload	MAC footer

Bits: 0-2	3	4	5	6	7-9	10-11	12-13	14-15
Frame type	Security enabled	Frame pending	Ack. Req.	PAN ID compression	Reserved	Dest. addressing mode	Frame version	Source addressing mode

- Frame type
  - Type of payload included
- Security enabled
  - Packet is encrypted
  - (extra 0-14 byte header)
- Frame pending
  - Fragmented packet
- Acknowledgement required
- PAN ID compression
  - No PAN ID if intra-network
- Addressing modes
  - Which fields to expect

Why no length field?

Frame control

Octets:2	1	0/2	0/2/8	0/2	0/2/8	variable	2
Frame control	Sequence number	Destination PAN identifier	Destination address	Source PAN identifier	Source address	Frame payload	Frame check sequence
		Addressing fields					
MAC header						MAC payload	MAC footer

Bits: 0-2	3	4	5	6	7-9	10-11	12-13	14-15
Frame type	Security enabled	Frame pending	Ack. Req.	PAN ID compression	Reserved	Dest. addressing mode	Frame version	Source addressing mode

- Frame type
  - Type of payload included
- Security enabled
  - Packet is encrypted
  - (extra 0-14 byte header)
- Frame pending
  - Fragmented packet

- Acknowledgement required
- PAN ID compression
  - No PAN ID if intra-network
- Addressing modes
  - Which fields to expect

Why no length field?

Already in prior header

Frame types - Beacon

2	variable	variable	variable
Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload
MAC Payload			

- Beacon
  - Information about the communication structure of this network
  - Sent in response to requests from scanning devices
  - Sent periodically at start of Superframes (if in use)
    - Sent without CSMA/CA
- MAC Header
  - Source address only, broadcast to everyone
- Packet contents
  - Superframe details, including Guaranteed Time Slots (if any)
  - Pending addresses lists devices for which Coordinator has data

Frame types - Data

- Data
  - Data from higher-layer protocols
- MAC Header
  - Source and/or Destination addresses as necessary
- Packet Contents
  - Whatever bytes are desired (122 bytes – address sizes)
  - May be fragmented across packets

Frame types – MAC Command

1	variable
Command Frame Identifier	Command Payload
MAC Payload	

- MAC Command
  - Various commands for supporting link layer
    - Join/leave network
    - Change coordinator within network
    - Request data from coordinator
    - Request Guaranteed Time Slot
- MAC Header
  - Source and/or Destination addresses as necessary

Frame types - Acknowledgement

- Acknowledgement
  - Acknowledges a Data or MAC Command packet
  - Not beacons or other acknowledgements
    - **What happens if acknowledgement isn’t received?**

- MAC Header
  - Repeats Sequence Number of acknowledged packet
  - No Source or Destination addresses

- Sent T_{IFS} after the packet it is acknowledging (immediately)

Frame types - Acknowledgement

- Acknowledgement
  - Acknowledges a Data or MAC Command packet
  - Not beacons or other acknowledgements
    - **What happens if acknowledgement isn't received?**
      - Packet will be transmitted again

- MAC Header
  - Repeats Sequence Number of acknowledged packet
  - No Source or Destination addresses

- Sent T_{IFS} after the packet it is acknowledging (immediately)

Analysis: maximum goodput

- Assume best possible case for data transmission
  - 122 Bytes per packet
    - At 250 kbps -> 3.904 ms
  - Plus Inter-frame spacing of 40 symbols
    - At 62.5 kBaud -> 0.640 ms

  - 122 Bytes / 4.544 ms -> 214 kbps
    - Compare to BLE advertisements: 9.92 kbps
    - Compare to BLE connections: 520 kbps

Break + Open Question

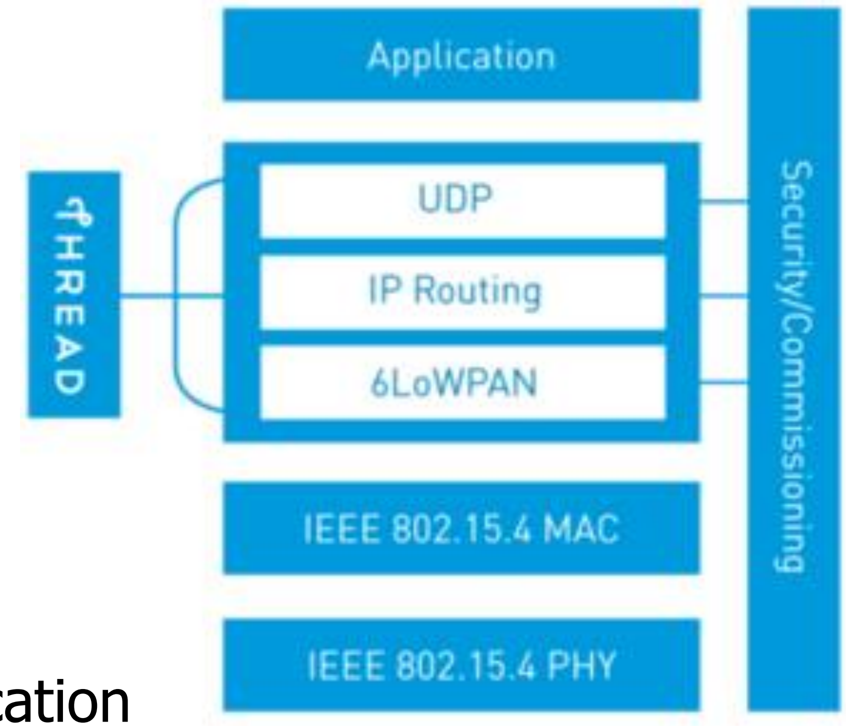
- What is 802.15.4 good for?

Outline

- 802.15.4 Packet Structure
- **Thread Overview**
- Thread Addressing
- Runtime Behavior
- Using IP

Thread overview

- Build a networking layer on top of 15.4
  - Reuses most of PHY and MAC
  - Adds IP communication
  - Handles addressing and mesh maintenance



- Goals
  - Simplicity – easy to install and operate
  - Efficiency – years of operation on batteries
  - Scalability – hundreds of devices in a network
  - Security – authenticated and encrypted communication
  - Reliability – mesh networking without single point of failure

- Industry-focused, but based in academic research

References on Thread

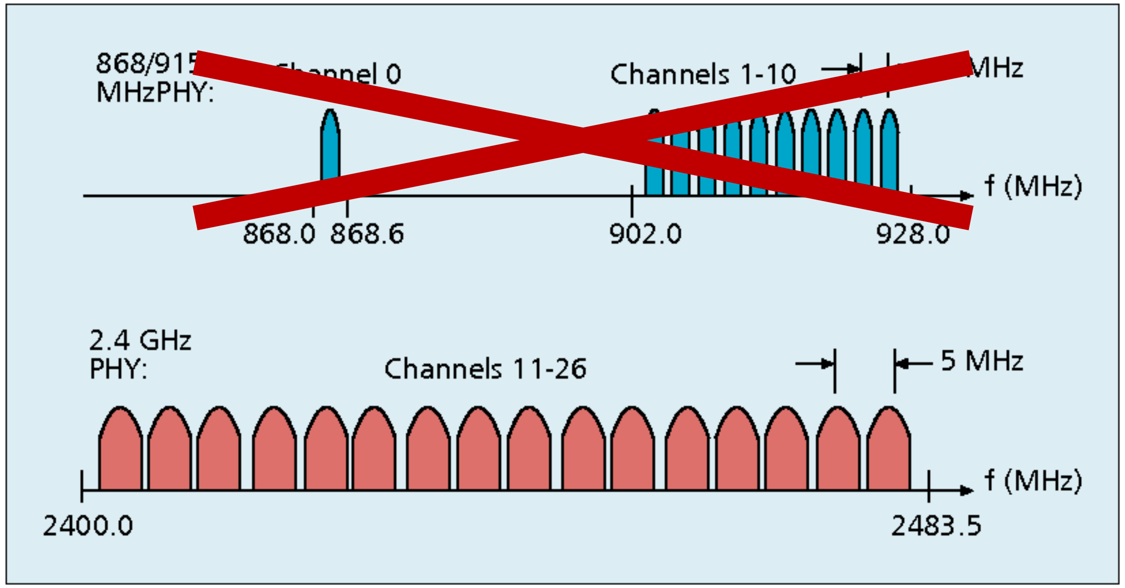
- Request for specification: https://www.threadgroup.org/ThreadSpec
  - Frustratingly locked down 😠

- Overview on capabilities: https://openthread.io/guides/thread-primer
  - Excellent overview
  - Lifting heavily for these slides

Changes to Physical Layer

- Remove all non-2.4 GHz PHY options

- Otherwise the same
  - OQPSK
  - 16 channels, 5 MHz spacing
  - Typical TX power 0 dBm
  - Typical RX sensitivity -100 dBm



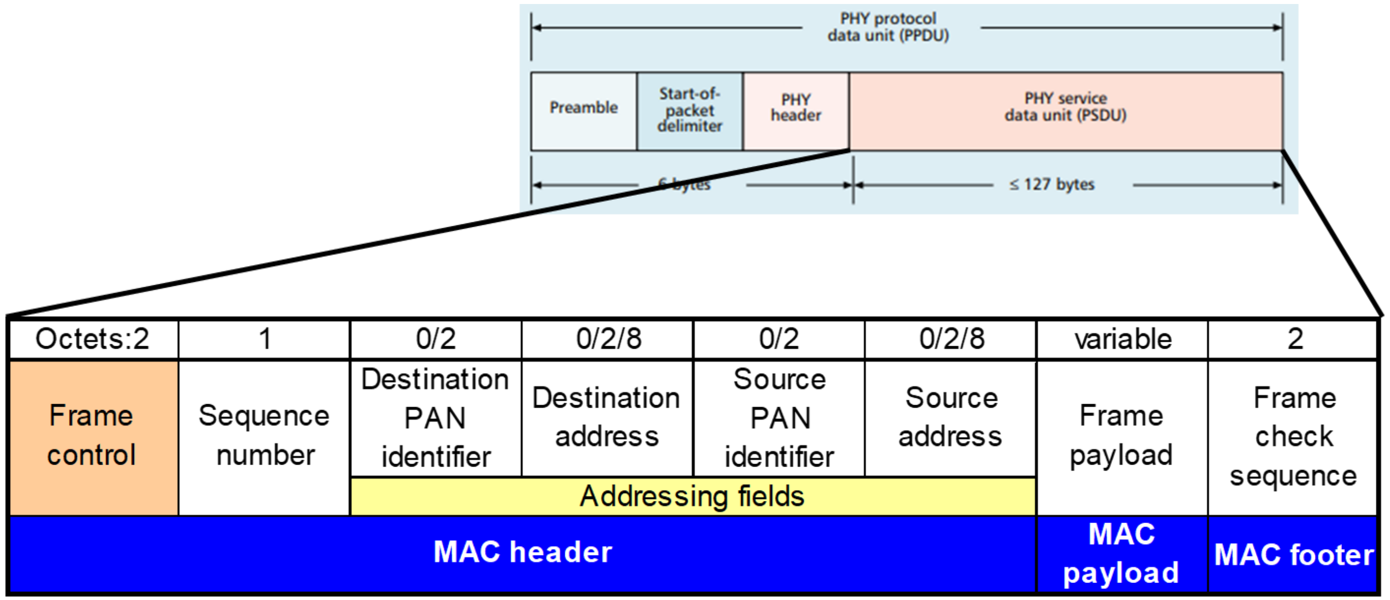
Changes to Link Layer and MAC

- Non-beacon-enabled PAN only
  - No superframe structure
  - No periodic beacons
  - No Guaranteed Time Slots

- Throw out most existing MAC Commands
  - Remove network joining/leaving
  - Remove changing coordinators
  - Remove Guaranteed Time Slot request
  - Network joining will be handled at a higher layer

Changes to Link Layer and MAC

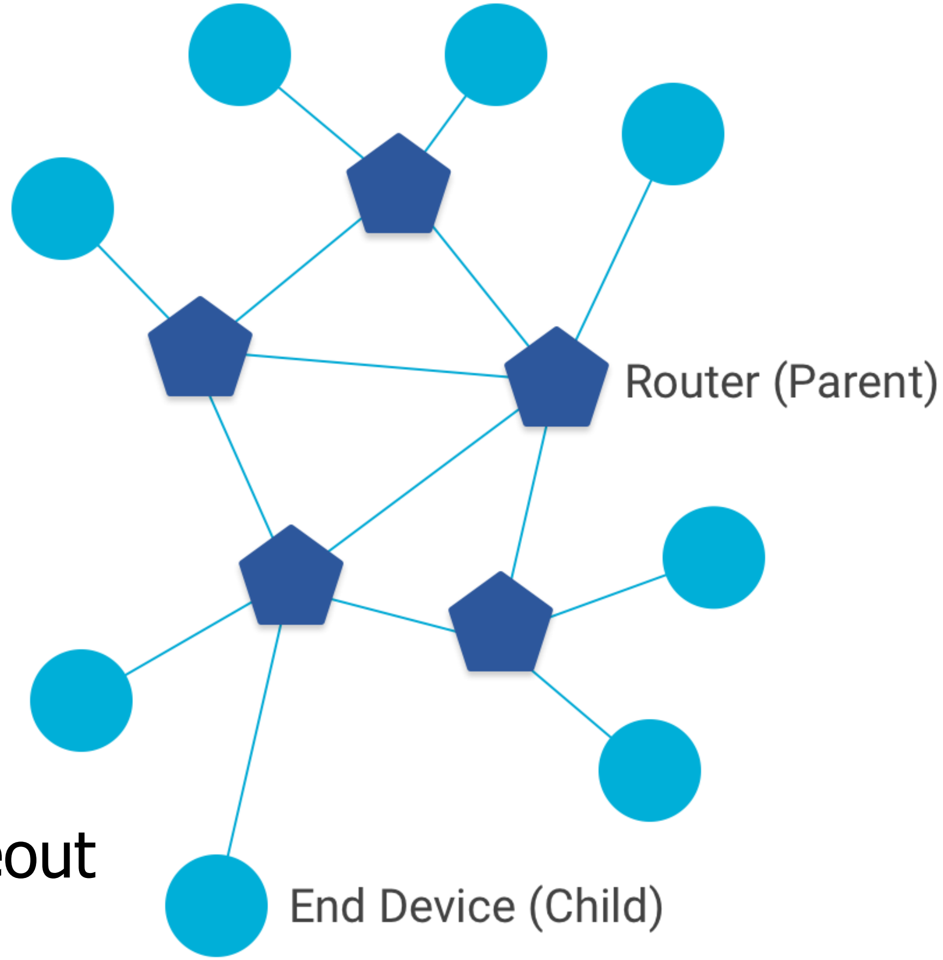
- Keep unslotted CSMA/CA algorithm
- Keep packet structure
- Keep Frame Types
  - Beacon
  - MAC Command
    - Beacon Request
    - Data Request
  - Data
  - Acknowledgement



Combination of star and mesh topology

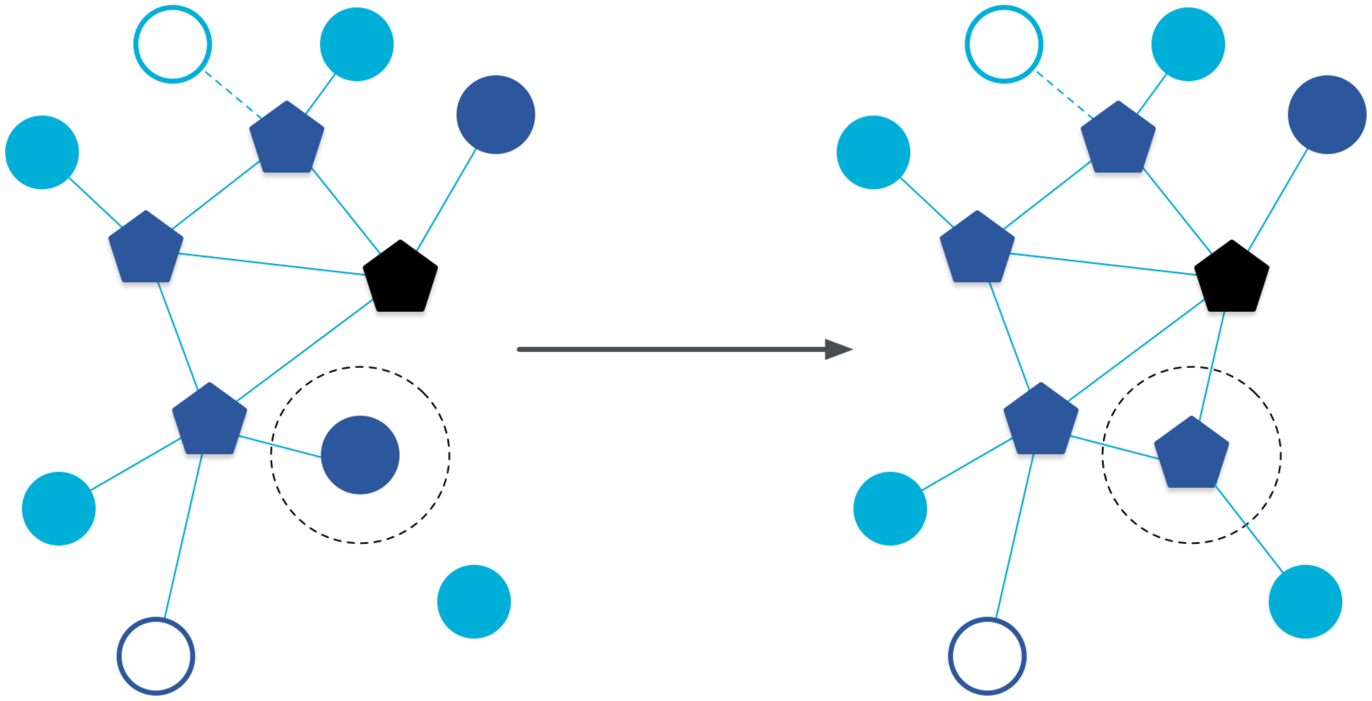
- Routers (parent)
  - Mesh communication with other routers
  - Radio always on
  - Forwards packets for network devices
  - Enables other devices to join network
  - 32 routers per network

- End devices (child)
  - Communicates with one parent (router)
  - Does not forward packets
  - Can disable transceiver to save power
    - Send packets periodically to avoid timeout
  - 511 end devices per router



Router promotion

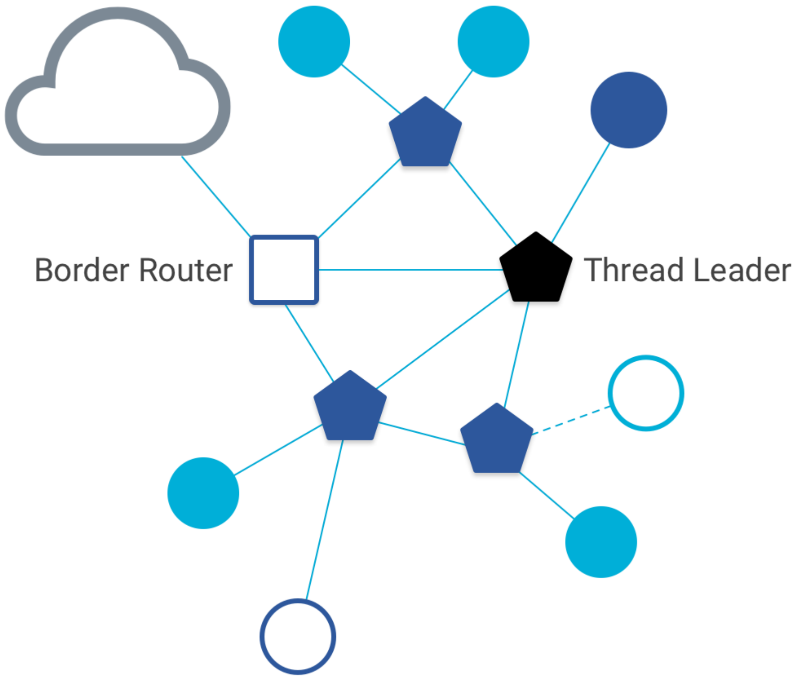
- "Router Eligible End Device"
  - A router without any children
  - Can operate as an end device with one connection (lower power)
  - Promotes to a router when a joining end device relies on it
    - If there is room for an additional router (max 32, typical 16-23)



Other special roles

- Thread leader
  - Device in charge of making decisions
    - Addresses, Joining details
  - Automatically selected from routers
    - One leader at any given time
    - Additional leader is selected if the network partitions

- Border router
  - Router that also has connectivity to another network
    - Commonly WiFi or Ethernet
  - Provides external connectivity
  - Multiple border routers may exist at once



Outline

- 802.15.4 Packet Structure
- Thread Overview
- **Thread Addressing**
- Runtime Behavior
- Using IP

Thread uses IPv6 for communication

- Why IP?
  - If Wireless Sensor Networks represent a future of billions of connected devices distributed throughout the physical world
  - Why shouldn’t they run standard protocols wherever possible?
  - Why IPv6?
    - Generalized, Flexible, Capable

- Benefits
  - Interoperability with normal computers and networks
  - Reuse state of the art developed standards instead of remaking them
    - Security, Naming, Discovery, Services
- Costs
  - Packet overhead can be high (will fix)
  - Complexity for supporting protocols

Hui and Culler, “IP is Dead, Long Live IP for Wireless Sensor Networks”. 2008

Background: IPv6

- Replacement to Internet Protocol v4
  - (Something unrelated used version number 5)
- Extended addressing for devices
  - 32-bits for IPv4 addresses -> 128-bits for IPv6 addresses
  - Example: a39b:239e:ffff:29a2:0021:20f1:aaa2:2112
- Supports multiple transmit models
  - Broadcast: one-to-all
  - Multicast: one-to-many
  - Unicast: one-to-one
- Various other improvements

Background: IPv6 address notation rules

- Groups of zeros can be replaced with "::"
  - Can only use "::" in one place in the address
- Leading zeros in a 16-bit group can be omitted

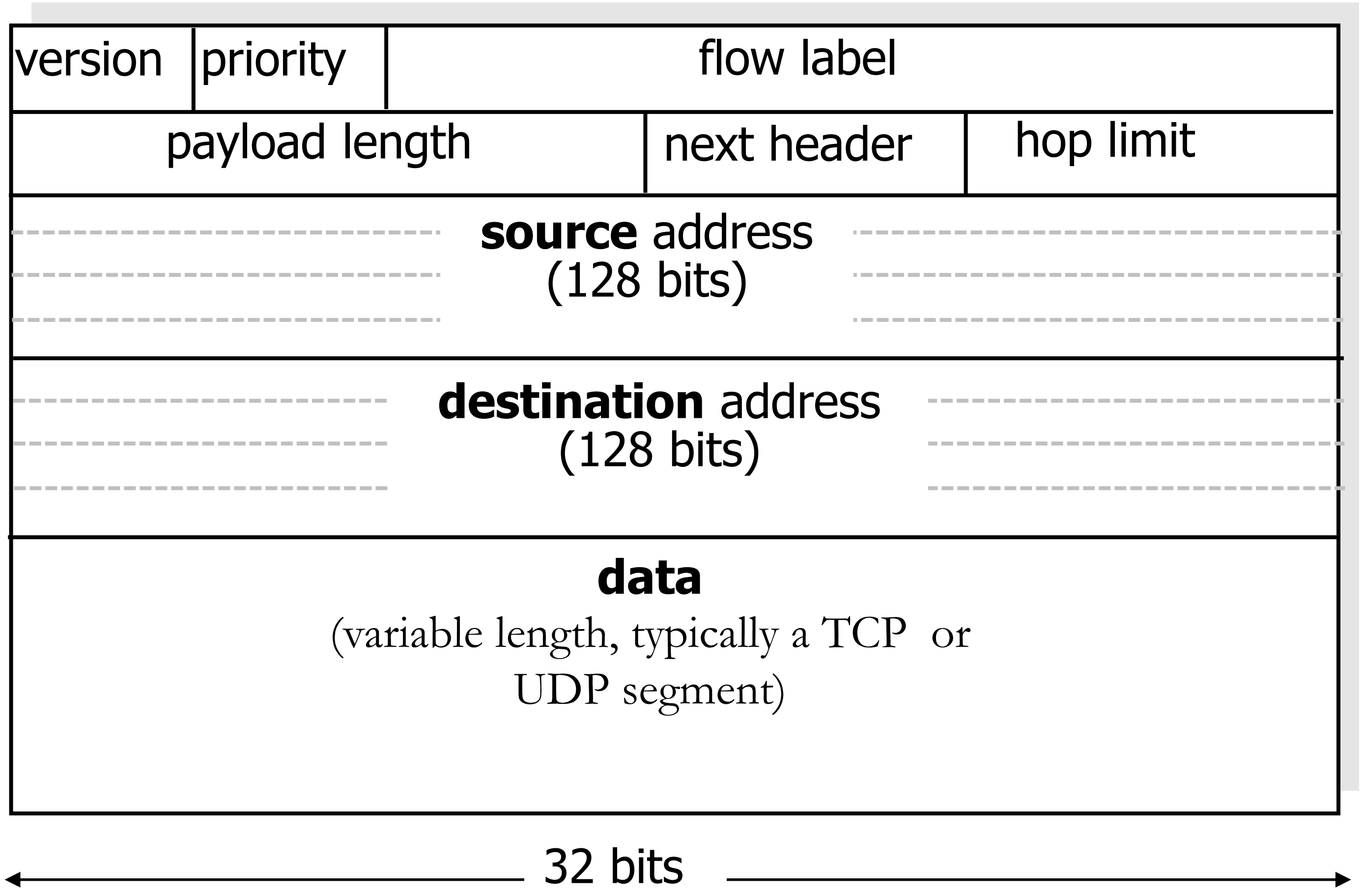
0000:0000:0000:0000:0000:0000:0000:0001 → ::1

2345:1001:0023:1003:0000:0000:0000:0000 → 2345:1001:23:1003::

aecb:0222:0000:0000:0000:0000:0000:0010 → aecb:222::10

- Special addresses
  - Localhost - ::1 (IPv4 version is 127.0.0.1)
  - Link-Local Network - fe80:: (bottom 64-bits are ~device MAC address)
  - Local Network – fc00:: and fd00::
  - Global Addresses – 2000:: (various methods for allocating bottom bits)

Background: IPv6 datagram format



- **Priority**: like "type of service" in IPv4.
- **Flow label**: ambiguous
- **Next header**: TCP, UDP
- **Hop limit** = TTL

how much overhead?

- **40 bytes** of IPv6
- 20 more than IPv4

6LoWPAN

- Method for running IPv6 over 802.15.4 links
  - IPv**6** over **Lo**w-Power **W**ireless **P**ersonal **A**rea **N**etworks
  - IETF Standard (RFC4944 + updates in RFC6282)

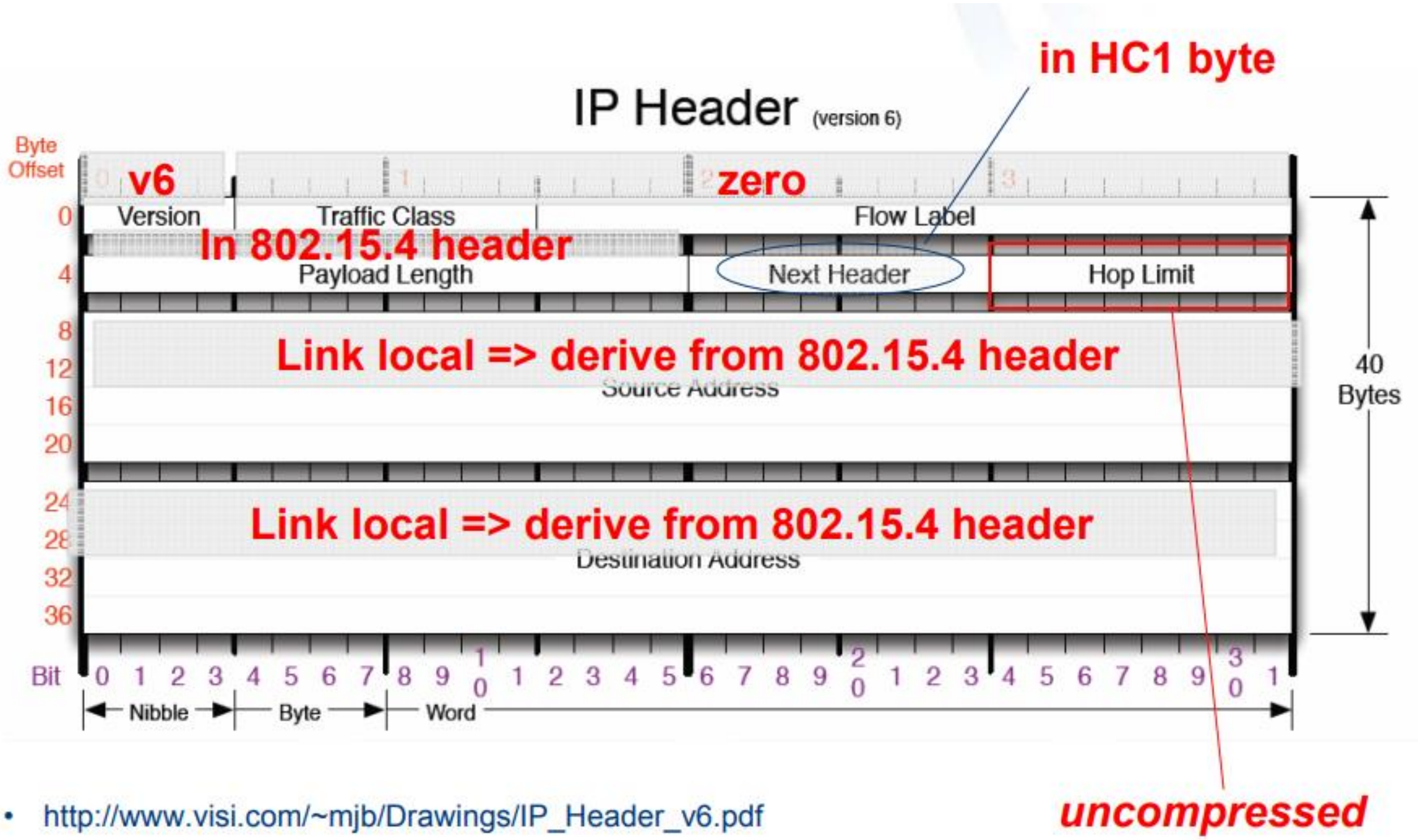
- Directly out of the research world (Jonathan Hui + David Culler)
  - Research Paper: IP is Dead, Long Live IP for Wireless Sensor Networks
  - Thesis of work: sensor networks can and should use IPv6

- Important goals
  - Compress IPv6 headers
  - Handle fragmentation of packets
  - Enable sending packets through mesh

6LoWPAN header compression

- 40 bytes of IPv6 header are a lot for a 127-byte payload
- Most important goals
  - Communication with devices in the 15.4 network should be low-overhead
  - Communication outside of the 15.4 network should still minimize overhead where possible
- Assume a bunch of common parameters to save space
  - A bunch of options are set to default values
  - Payload length can be re-determined from packet length
  - Source/Destination addresses can often be reassembled from link layer data
    - Plus information about network address assignment known by routers
- Border router "inflates" the packet before sending externally

Example of compression



- http://www.visi.com/~mjb/Drawings/IP_Header_v6.pdf

- Note: Thread actually uses IPHC (not HC1) from rfc6282

6LoWPAN fragmentation

- Only the first packet of the fragments will hold the IPv6 header
  - Tag, offset, and size are used to reconstruct

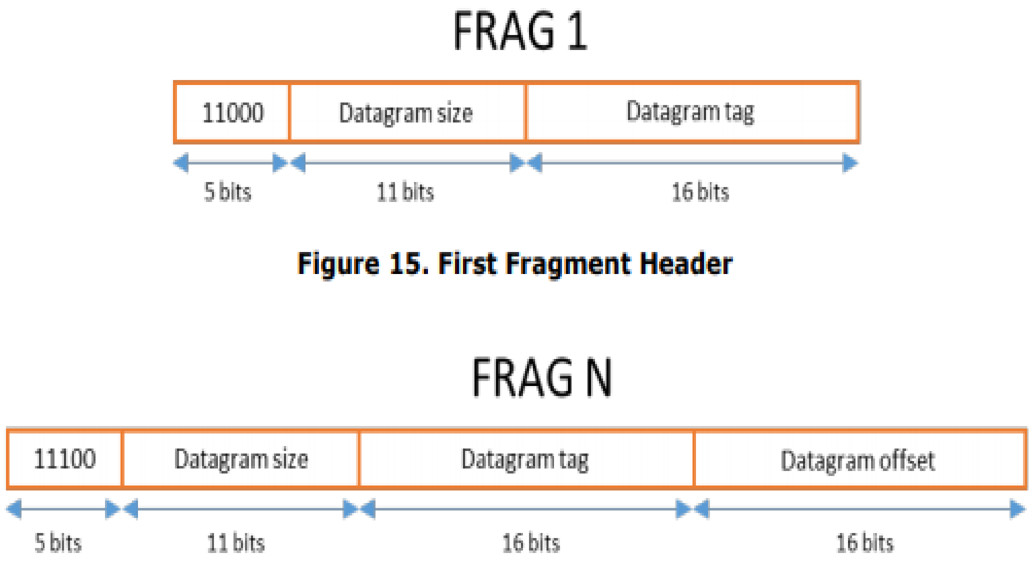


Figure 15. First Fragment Header

Figure 16. Subsequent Fragment Header

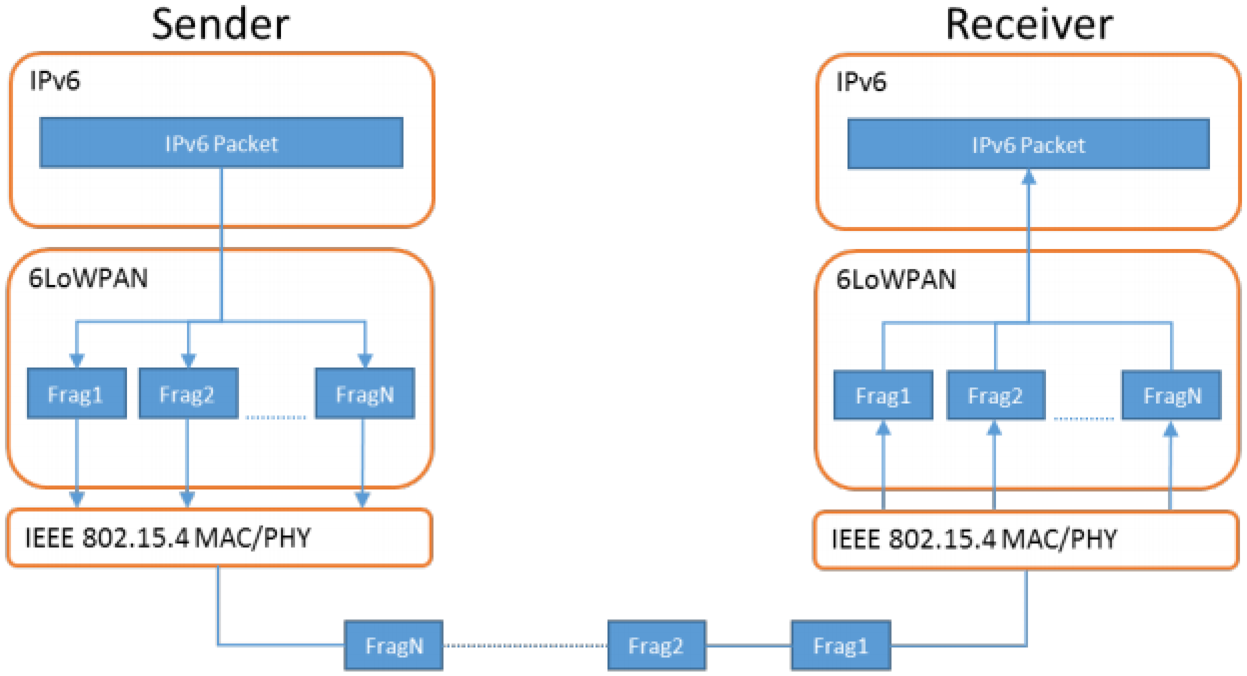


Figure 14. Fragmenting and Reassembling an IPv6 Packet

6LoWPAN mesh forwarding

- Additional header with originator and final addresses

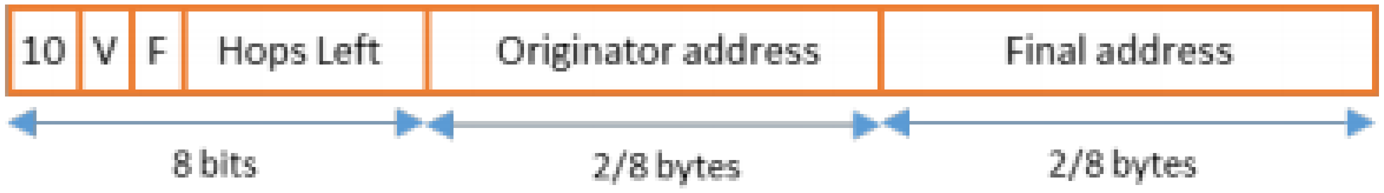


Figure 17. Mesh Header Format

- Which of these headers are used depends on the packet

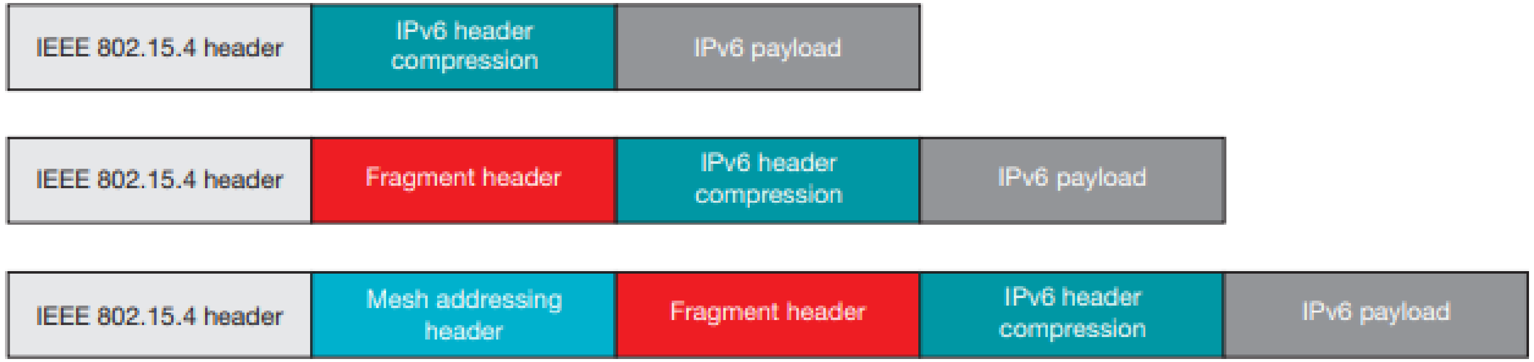


Figure 4. 6LoWPAN stacked headers

Sidebar: IPv6 over BLE

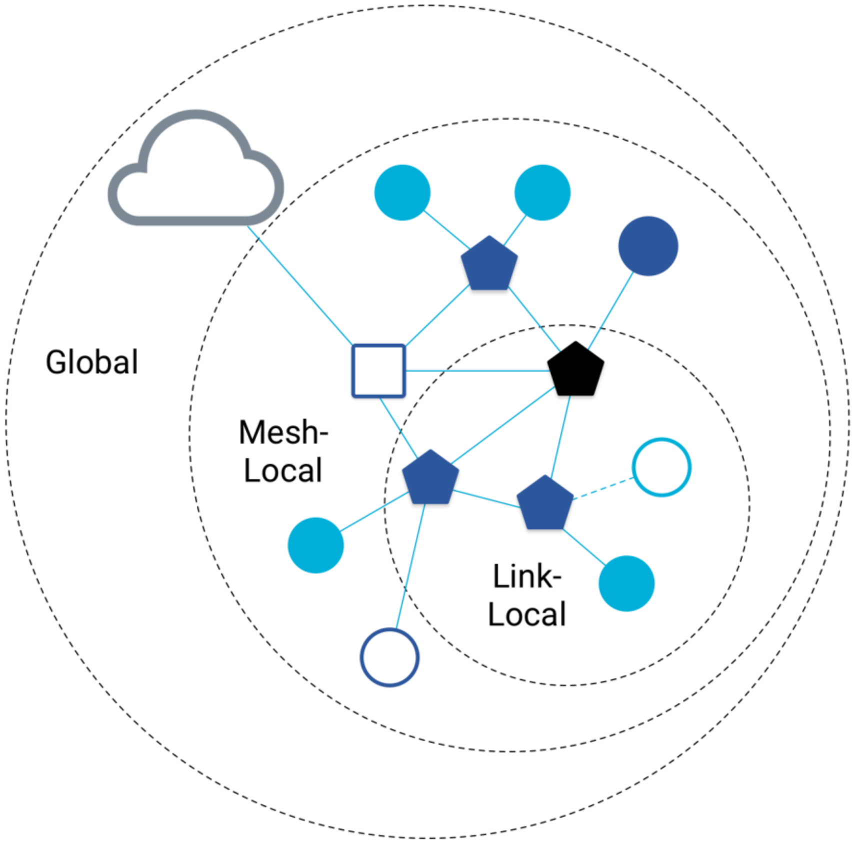
- RFC7668 defines 6LoWPAN techniques for BLE connections

```
          +----------+  +----------------------------+
          |   IPSS   |  |        UDP/TCP/other       |
          +----------+  +----------------------------+
          |   GATT   |  |            IPv6            |
          +----------+  +----------------------------+
          |   ATT    |  |  6LoWPAN for Bluetooth LE  |
          +----------+--+----------------------------+
          |          Bluetooth LE L2CAP              |
     -  - +------------------------------------------+- - - HCI
          |        Bluetooth LE Link Layer           |
          +------------------------------------------+
          |         Bluetooth LE Physical            |
          +------------------------------------------+
```

Figure 3: IPv6 and IPSS on the Bluetooth LE Stack

Benefit to IPv6: multiple address spaces per Thread device

- Each device gets an IPv6 address for each way to contact it
  - Global IP address
  - Mesh-local IP address
  - Link-local IP address
  - Topology-based IP address
  - Role-based IP address(es)



Traditional addresses in Thread

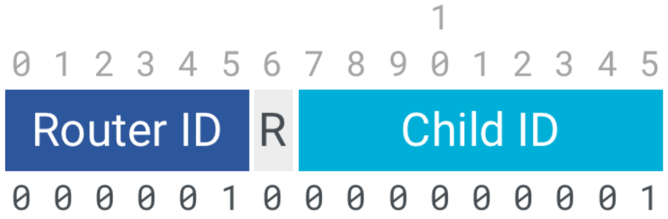
- Link-Local Addresses
  - FE80::/16
  - Bottommost 64-bits are EUI-64 (MAC address with 0xFFFE in the middle)
  - Permanent for a given device (no matter the network)
  - Used for low-layer interactions with neighbors (discovery, routing info)

- Mesh-Local Addresses
  - FD00::/8 (FD00:: and FC00:: are for local networks)
  - Remaining bits are randomly chosen as part of joining the network
  - Permanent while connection is maintained to a network
  - Used for application-layer interactions

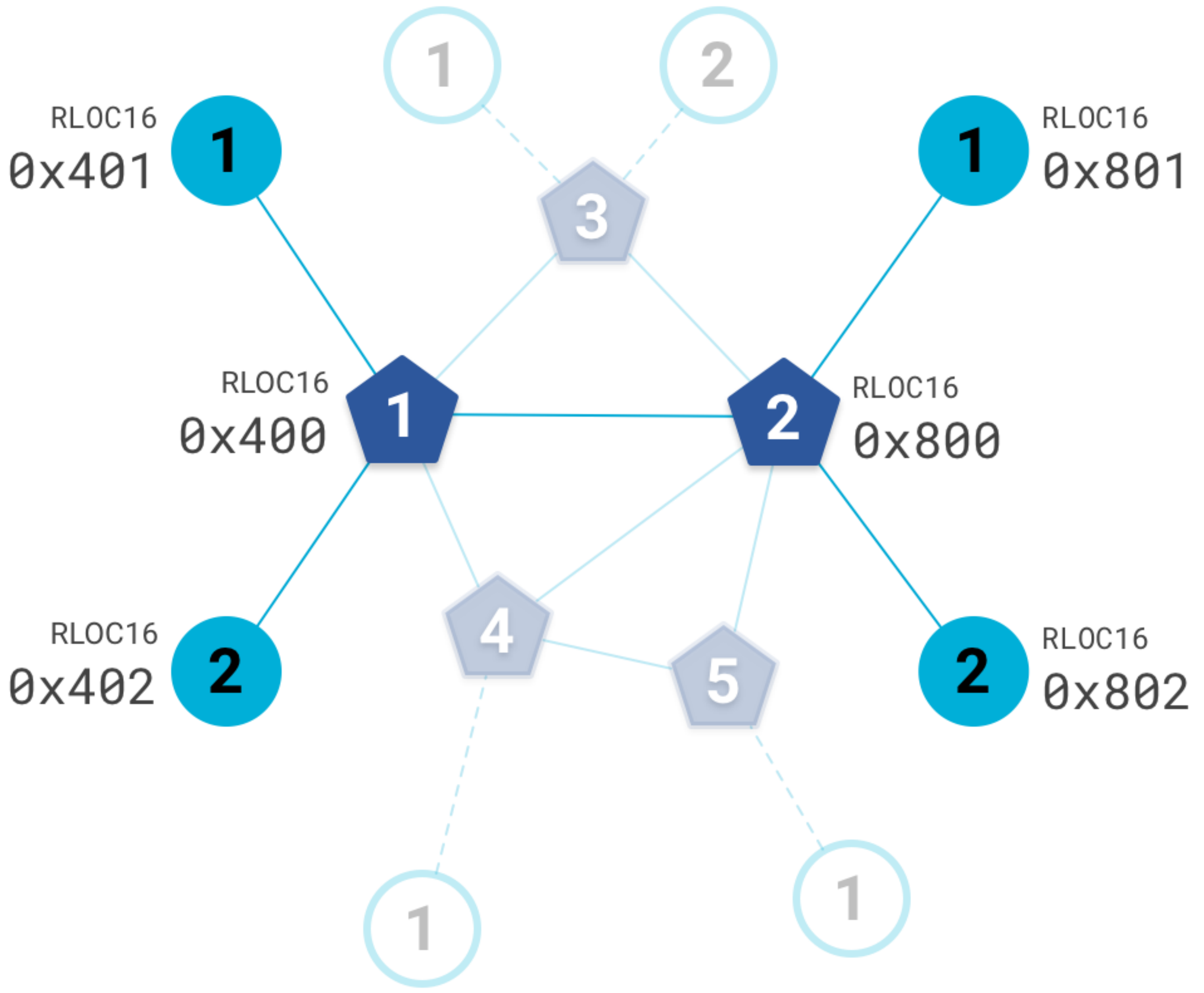
- Global Addresses
  - 2000::/3
  - Public address for communicating with broader internet through Border Router
  - Various methods for allocation (SLAAC, DHCP, Manual)

Topology-based addresses in Thread

- FD00::00ff:fe00:RLOC16
  - Same top bits as mesh-local
- Routing Locator (RLOC)
  - Router ID plus Child ID



- Changes with network topology
  - Used for routing packets



Role-based addresses in Thread

- Multicast
  - FF02::1 – link-local, all listening devices
  - FF02::2 – link-local, all routers/router-eligible
  - FF03::1 – mesh-local, all listening devices
  - FF03::2 – mesh-local, all routers/router-eligible

- Anycast
  - FD00::00FF:FE00:FC**xx**
    - 00 – Thread Leader
    - 01-0F – DHCPv6 Agent
    - 30-37 – Commissioner
    - etc.

Break + Open Question

- Why use Thread instead of basic 802.15.4?

Break + Open Question

- Why use Thread instead of basic 802.15.4?

  - Full specification of upper layers
    - Clarifies how data is transmitted between devices on a network
    - Cleans up a lot of things otherwise left implementation-dependent

  - Interaction with the world *outside* of the sensor network!
    - Gateway can be a dumb forwarder of packets
    - Devices can directly talk to NTP servers or POST data to a website!

Outline

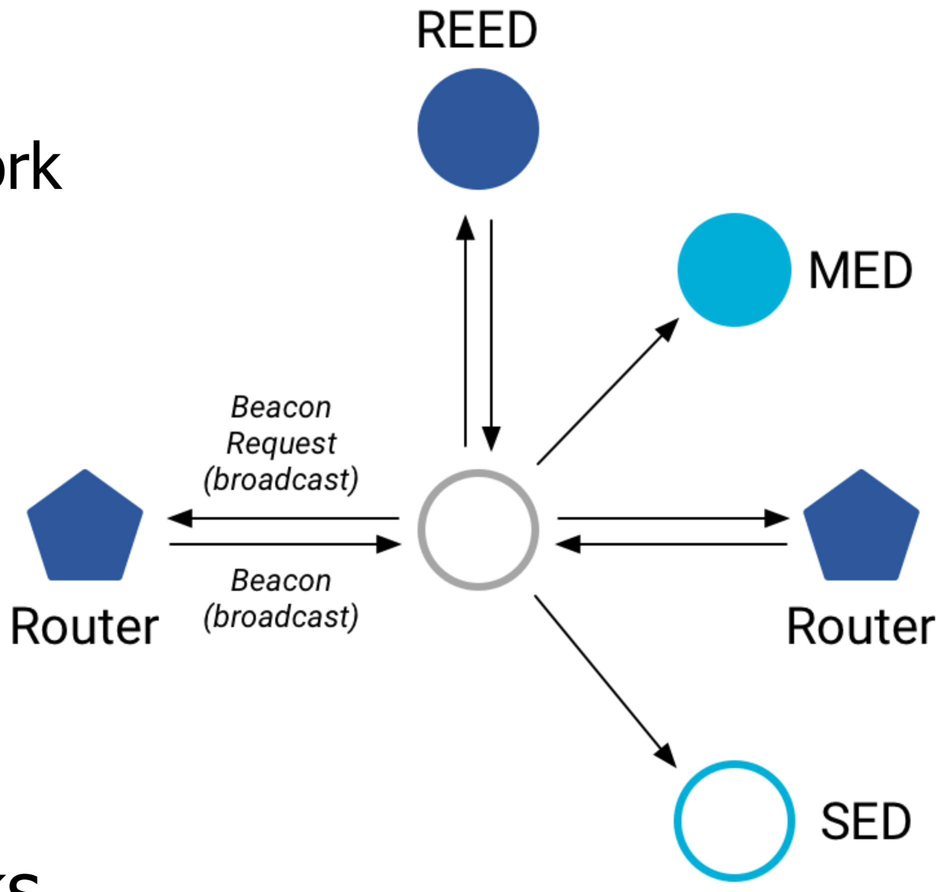
- 802.15.4 Packet Structure
- Thread Overview
- Thread Addressing
- **Runtime Behavior**
- Using IP

Discovering Thread networks

- Beacon request MAC command
  - Routers/Router-eligible devices respond
  - Payload contains information about network

- Thread network specification
  - PAN ID – 16-bit ID
  - XPAN ID – extended 64-bit ID
  - Network Name – human-readable

- Active scanning across channels can quickly find all existing nearby networks



Creating a new network

- Select a channel (possibly by scanning for availability)

- Become a router
  - Elect yourself as Thread Leader
  - Respond to Beacon Requests from other devices

- Further organization occurs through Mesh-Level Establishment protocol

Mesh-Level Establishment

- Creating and configuring mesh links
  - Payloads placed in UDP packets within IPv6 payloads

- Commands for mesh
  - Establish link
  - Advertise link quality
  - Connect to parent

0	Command Type	TLV	…	TLV

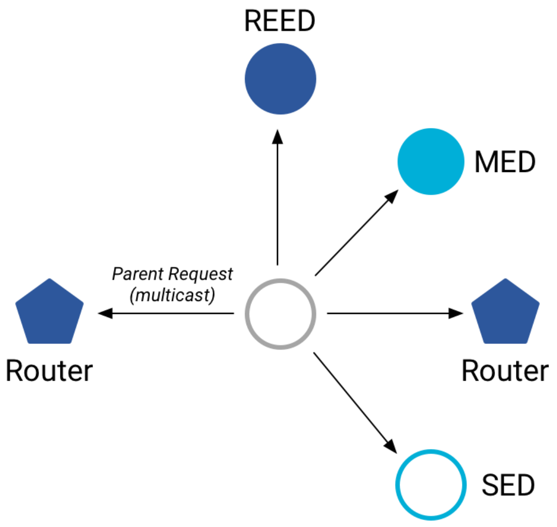
OR (secure version)

255	Aux Header	Command Type	TLV	…	TLV	MIC

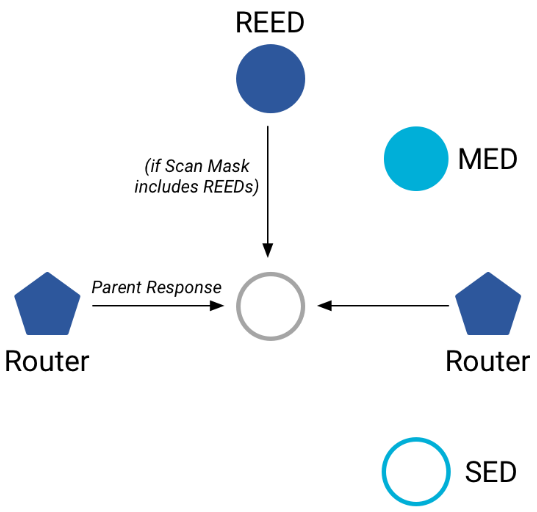
- TLVs (Type-Length-Value)
  - Various data types that may be helpful within those packets
  - Addresses, Link Quality, Routing Data, Timestamps

Joining an existing network

- All devices join as a child of some existing router



1. Send a Parent Request (to all routers/router-eligible)
   - Using the multicast, link-local address
2. Receive a Parent Response (from all routers/router-eligible separately)
   - Contains information on link quality



3. Send a Child ID Request (to router with best link)
   - Contains parameters about the new child device
4. Receive a Child ID Response (from that router)
   - Contains address configurations

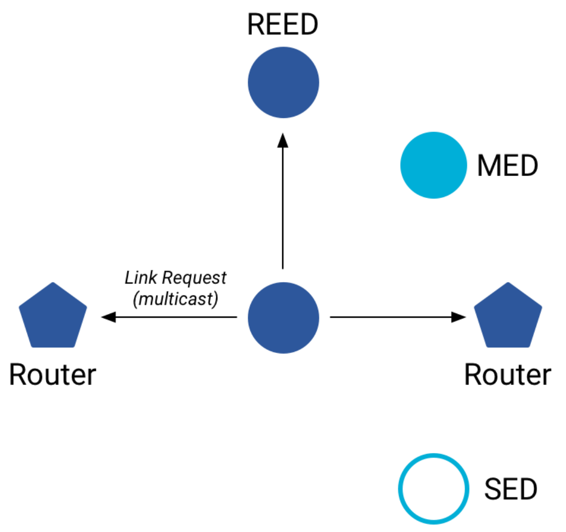
Becoming a router

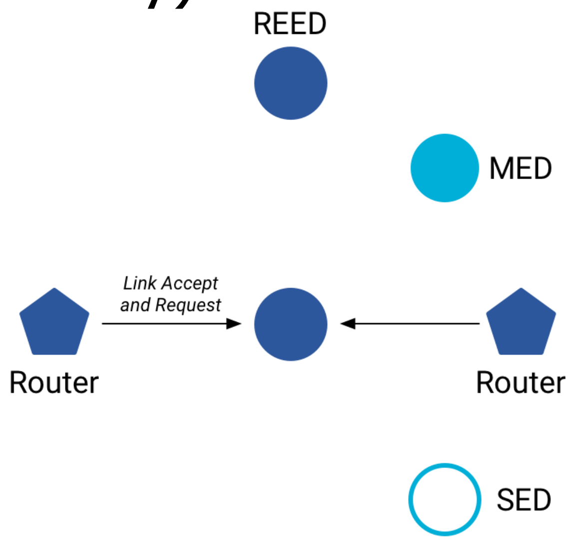
- Thread tries to maintain 16-23 routers (max 32)
  - Goals: path diversity, extend connectivity

1. Send a Link Request (to all routers/router-eligible)
   - Using the multicast, link-local address

2. Receive Link Accept and Request (from each router separately)
   - Forms bi-directional link

3. Send a Link Accept (to each router individually)





Outline

- 802.15.4 Packet Structure
- Thread Overview
- Thread Addressing
- Runtime Behavior
- **Using IP**

Communicating with IP

- Any communication that layers on top of IP is now possible
  - If there is a library to support it

- Common choices
  - UDP
    - DNS – translate hostnames into IP addresses
    - SNTP – get real-world time, accuracy better than 1 second
    - CoAP – send and receive data

Constrained Application Protocol - CoAP

- HTTP, but over UDP targeting less-capable devices
  - Same REST architecture
  - Adds capability for automatic retransmissions

- CoAP Requests
  - Have a type: GET, POST, PUT, DELETE
  - Have a URL: /file/etc
  - Have data up to 65 KB

Sensor networks don't use TCP (yet?)

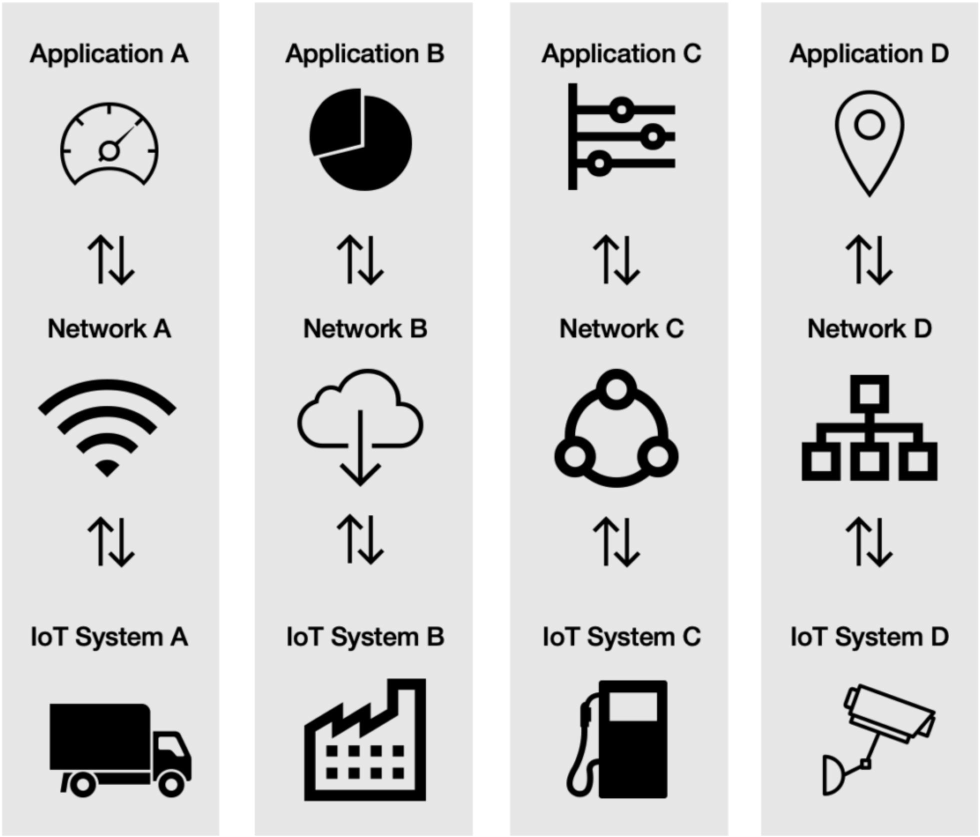
- Uncommon choice: TCP
  - Concerns: Too large, too slow, poorly suited to lossy networks
  - Also concerning: We're just replicating TCP poorly

  - Work in progress:
    - Sam Kumar, Michael Anderson, Hyung-Sin Kim, David Culler. "Performant TCP for Low-Power Wireless Networks". 2020.

    - The debate is still very much open

A problem: the siloed internet of things

- Problem: companies are more interested in selling you the whole stack
  - Which then makes it harder for devices to be interoperable
- This is not Thread or IP-specific, but a problem all IoT devices are facing
- Branden’s concern:
  - What IP address do you send data to?
  - Manufacturer’s server is an obvious choice



Outline

- 802.15.4 Packet Structure
- Thread Overview
- Thread Addressing
- Runtime Behavior
- Using IP