

Lecture 07

Thread

CS433 – Wireless Protocols for IoT

Branden Ghena – Spring 2026

With some advice from Neal Jackson (UC Berkeley)

Materials in collaboration with
Pat Pannuto (UCSD) and Brad Campbell (UVA)

Administrivia

- Hw: BLE Packets due tonight
 - Do it. It's good for you. Much like eating spinach.
- Lab: BLE due next week Thursday
 - Lab tomorrow to continue work on it
 - Hope is to spend most of the time just working on BLE applications
- You're not strictly required to attend, but strongly encouraged to
 - Make sure you communicate with your team

Today's Goals

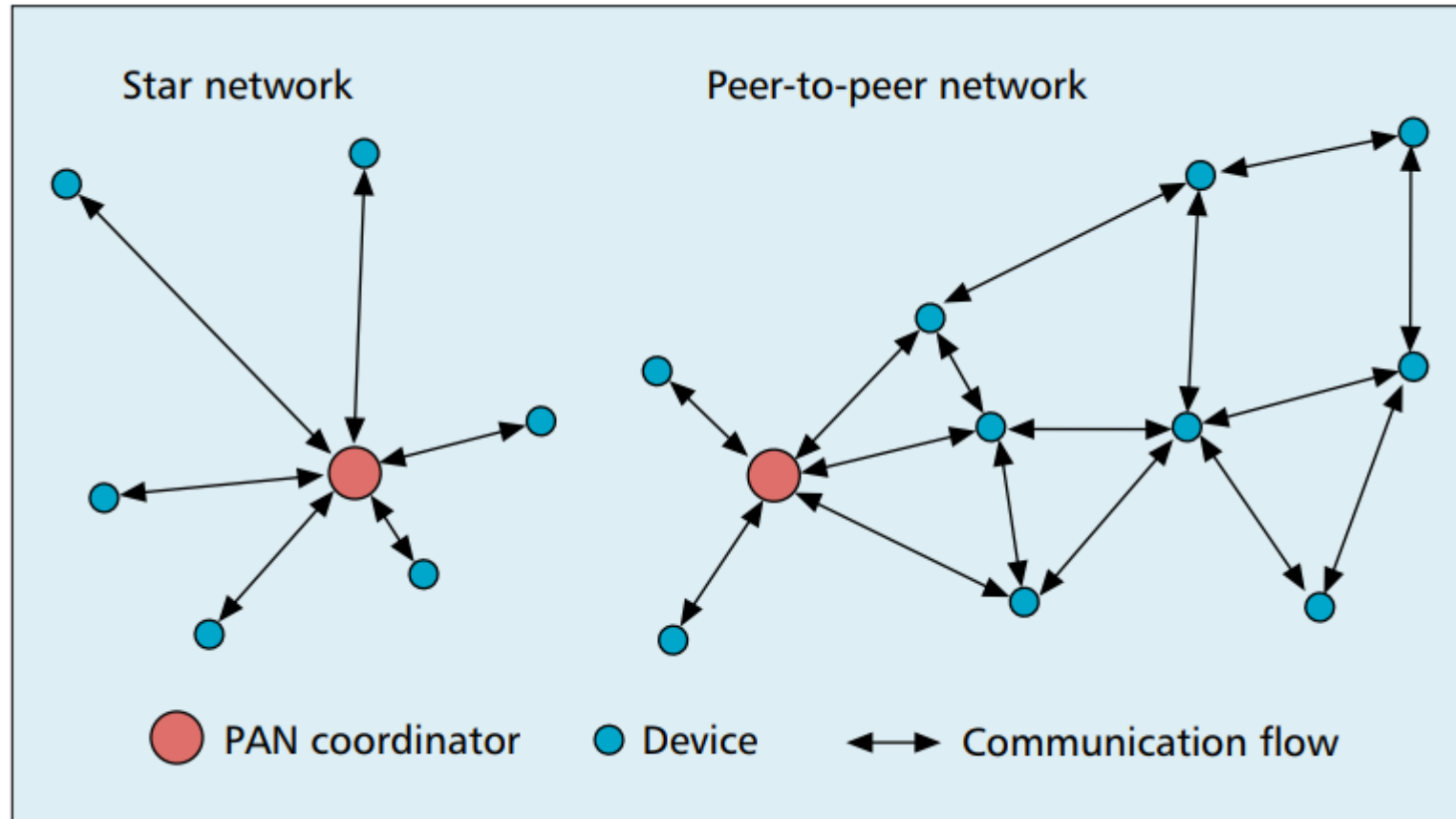
- Explore 802.15.4 packet structure
- Describe goals and capabilities of Thread networks
- Understand addressing in Thread networks
- Describe runtime behaviors like network joining

Outline

- Overview
- Physical Layer
- Link Layer
- **Packet Structure**

802.15.4 network topologies

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



Non-beacon-enabled PAN

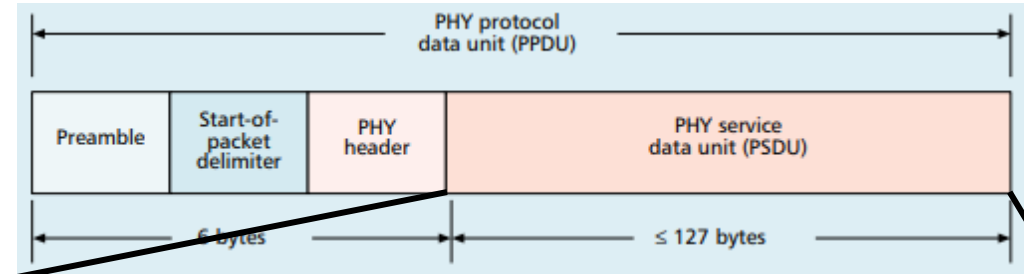


Contention Access Period

- All devices can communicate at any point (no structure)
- So, some *other* protocol needs to come up with rules
 - One simple option: all devices always listen

MAC frame format

- Overhead: 5-25 bytes



- **Frame control**

- Header with flags

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

- **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
- Beacon, Data, Control, Ack

- **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

Frame types - Beacon

- 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 configuration
 - 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

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

Frame types - Data

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

Frame types – MAC Command (i.e., control)

- 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 configuration
 - Source and/or Destination addresses as necessary

1	variable
Command Frame Identifier	Command Payload
MAC Payload	

Frame types - Acknowledgement

- Acknowledgement packet
 - Acknowledges a Data or MAC Command packet
 - Not beacons or other acknowledgements
 - Acknowledging acknowledgements gets problematic...
 - Retransmission if no acknowledgement received
- MAC Header
 - Repeats Sequence Number of acknowledged packet
 - No Source or Destination addresses
- Sent T_{IFS} after the packet it is acknowledging (immediately)
 - Time of Inter-Frame Spacing (standard delay between packets)

Analysis: maximum goodput

- Assume best possible case for data transmission
 - 133 total Bytes per packet (122 payload bytes + 11 bytes of headers)
 - At 250 kbps -> 4.256 ms duration
 - Plus Inter-frame spacing of 40 symbols
 - At 62.5 kBaud -> 0.640 ms duration
- 122 Bytes / 4.896 ms -> **199 kbps goodput**
 - Compare to BLE advertisements: 9.92 kbps goodput
 - Better with BLE 5.0 extended advertisements
 - Compare to BLE connections: 520 kbps goodput
 - Better with BLE 6.0 improvements

Outline

- **Thread Overview**
- Thread Addressing
- Runtime Behavior
- Using IP

Major 802.15.4 uses

- Zigbee & Thread
- Both build upon the 15.4 PHY and Link primitives
 - Using most but discarding some
- Both consider higher-level application considerations
 - Building a network
 - Communicating between devices
 - Application logic

What's the need for Thread?

- Born from Google's Nest team
- Requirements:
 - IP-based: interoperate with rest of the Internet
 - Scalable: hundreds of devices in a network
 - Low power: years of operation on batteries
 - Secure: authenticated and encrypted communication
 - Reliability – mesh networking without single point of failure
- 2014: Thread Group alliance formed
 - Goal: develop, maintain and drive adoption of Thread as an industry networking standard for IoT

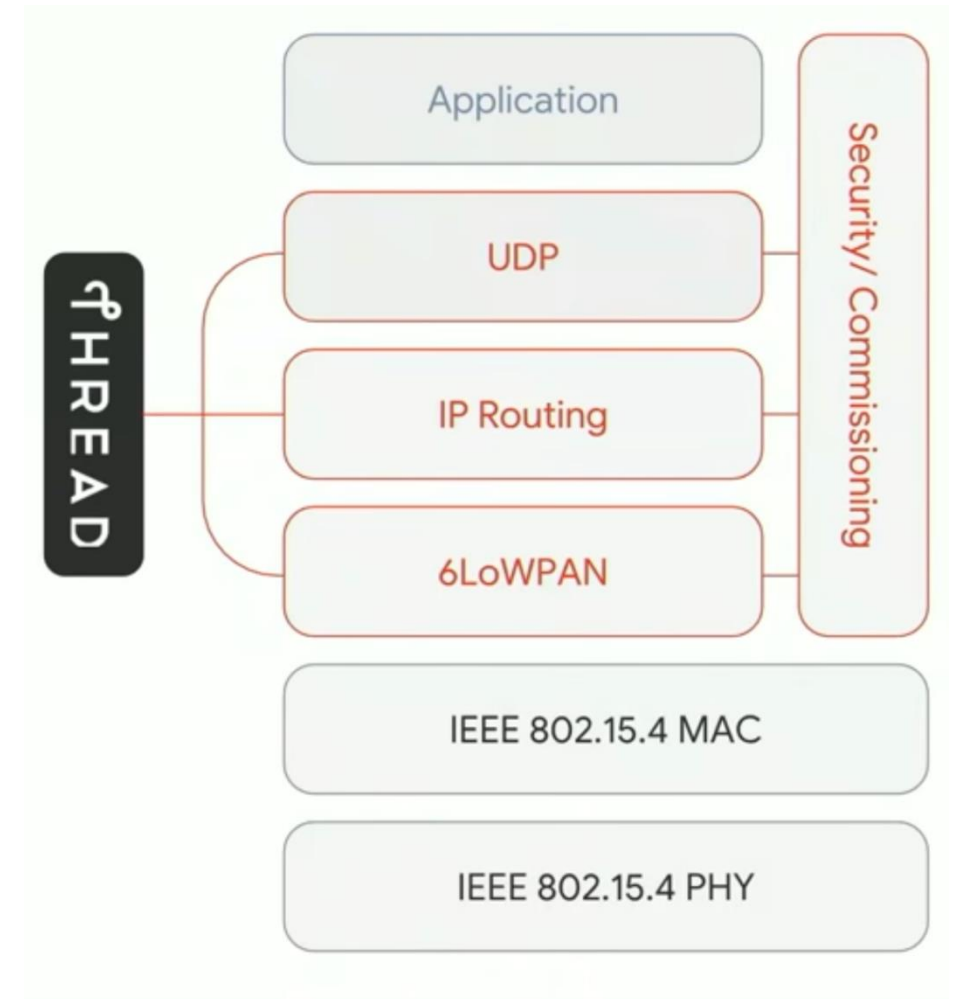


Some consumer devices secretly have Thread radios!

- [TechRadar article](#)
 - iPhones, Macbooks, iPads since 2023
 - Pixel 10
- Provides theoretical capability to communicate directly with your smart home devices
 - Not actually implemented on many devices though
- iOS 18 and Android 15 apparently add ability to communicate directly with devices
 - I haven't seen real-world usage of this though

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
- Industry-focused, but based in academic research

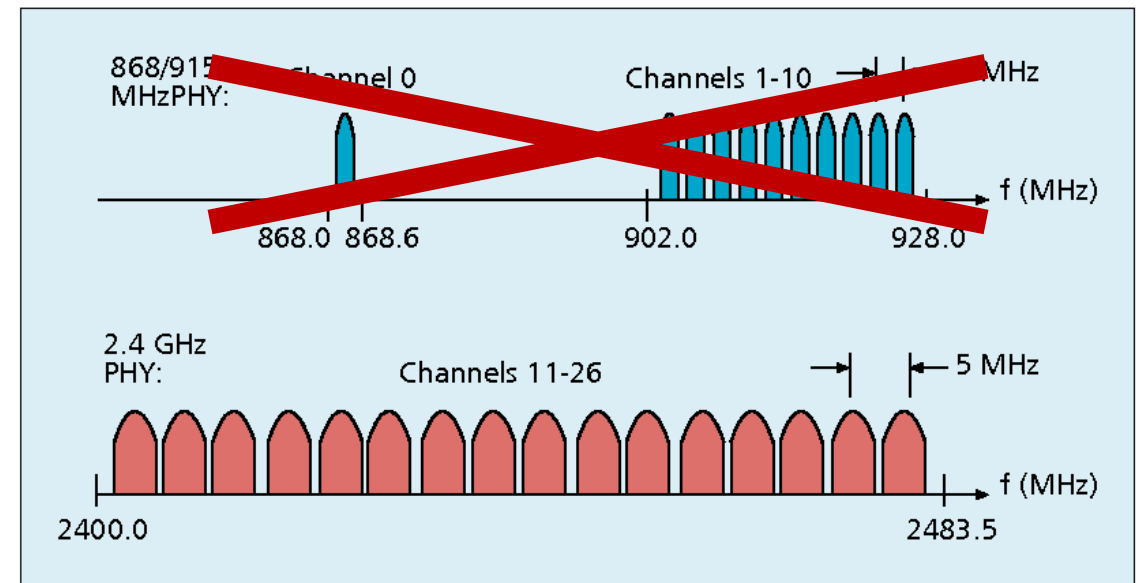


References on Thread

- 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
- Good overview on Thread Networking:
 - <https://www.threadgroup.org/support#Whitepapers>
 - See the “Thread Network Fundamentals” whitepaper

Changes to Physical Layer

- Remove all non-2.4 GHz PHY options
- Otherwise the same
 - O-QPSK
 - 16 channels, 5 MHz spacing
 - Typical TX power 0 dBm
 - Typical RX sensitivity -100 dBm



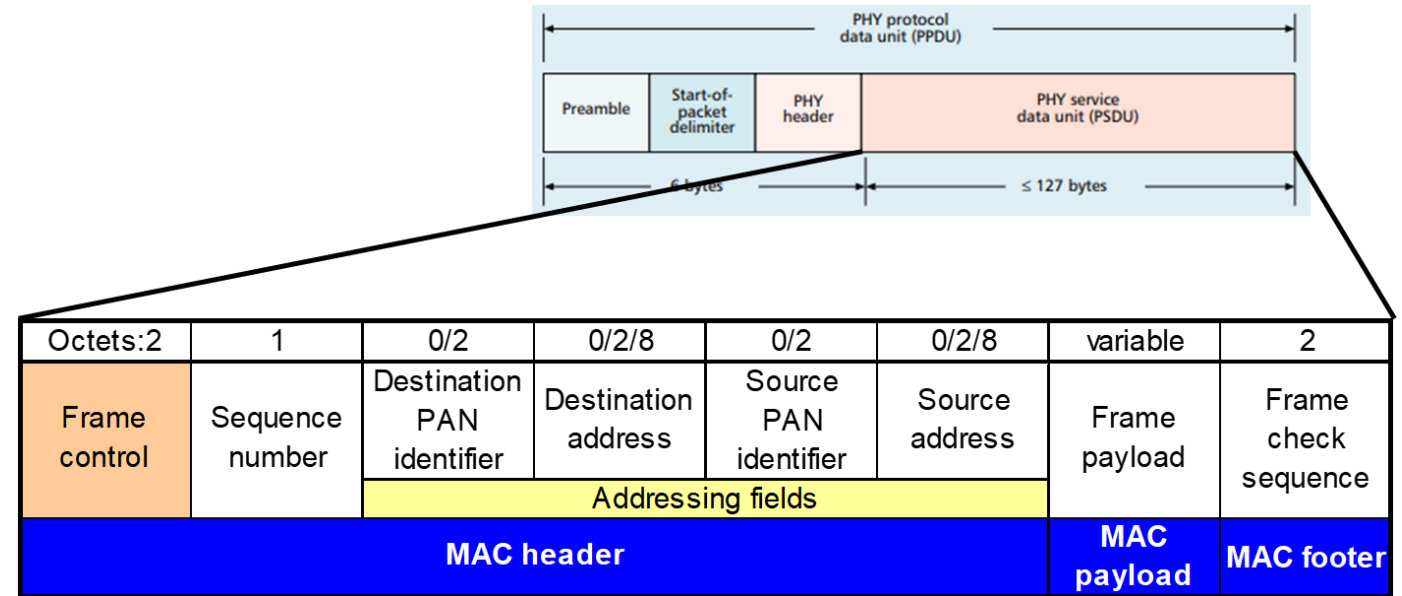
Changes to Link Layer and MAC

- Non-beacon-enabled PAN only
 - No superframe structure
 - No periodic beacons
 - No Guaranteed Time Slots
- Throw out most existing MAC Commands
 - Network joining will be handled at a higher layer
 - Remove network joining/leaving
 - Remove changing coordinators
 - Remove Guaranteed Time Slot request



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



Thread networks use a mix of star and mesh topologies

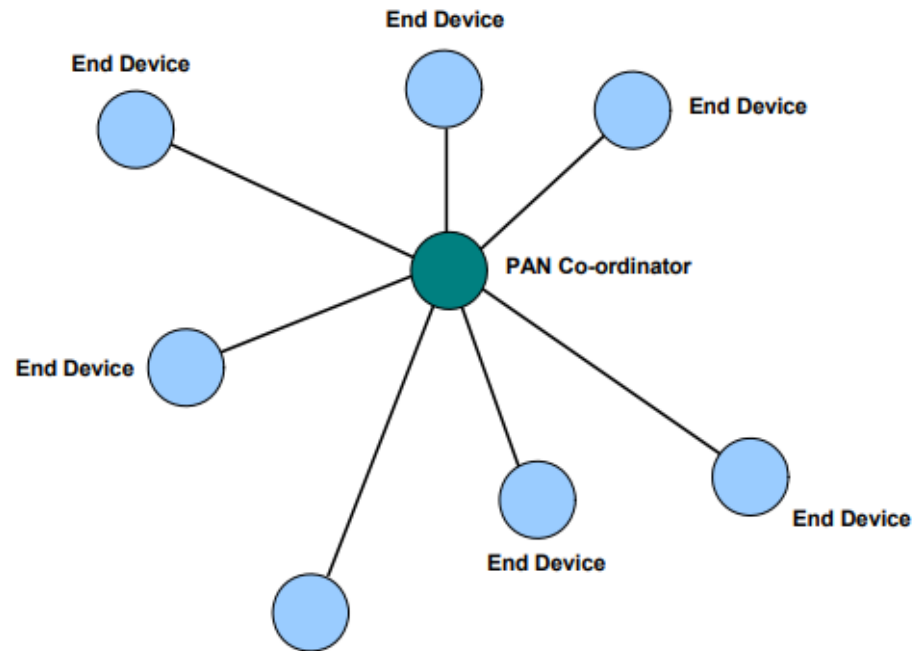


Figure 1: Star Topology

Central PAN coordinator in charge of network

- Single point of failure

End Devices can be less complicated

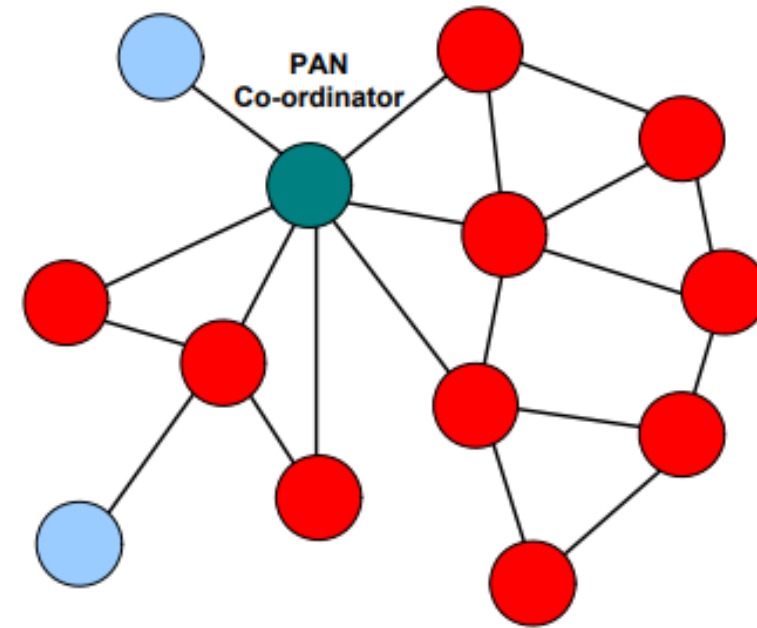


Figure 4: Mesh Topology

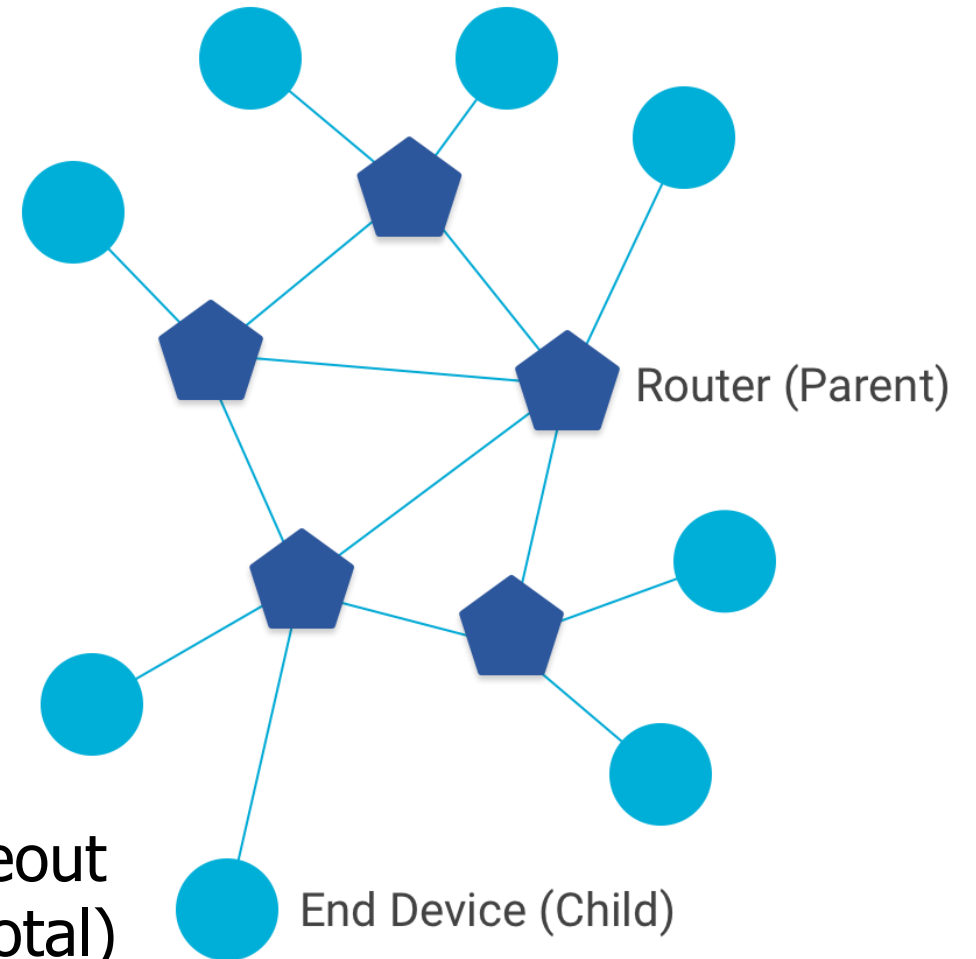
PAN Coordinator still in charge

But mid-level Routers add route redundancy

And End Devices didn't have to change at all

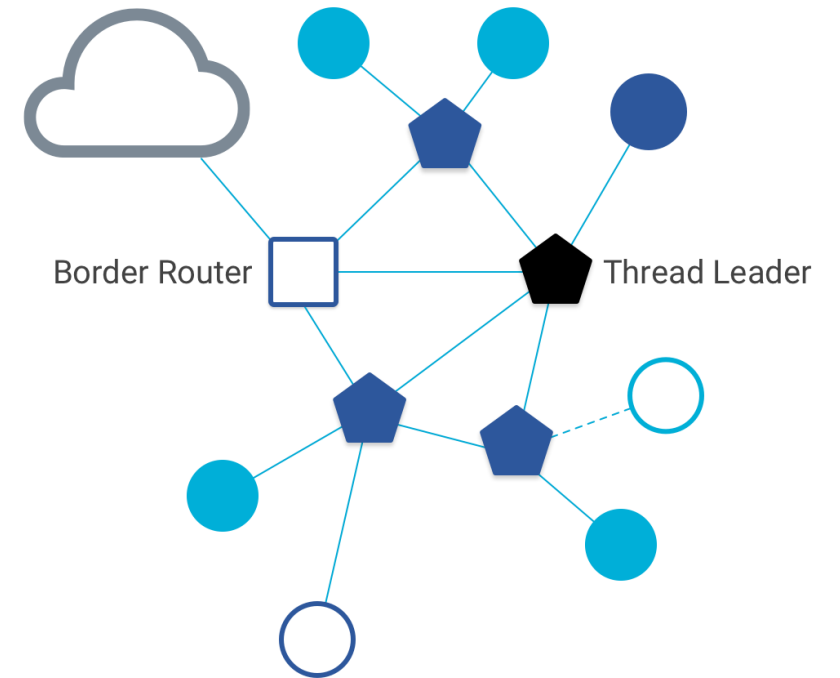
Combination of star and mesh topology

- Routers (parent)
 - Mesh communication with other routers
 - Radio always on
 - Forwards packets for network devices
 - Enables other devices to join network
 - Up to 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
 - Up to 511 end devices per router (~16k total)



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

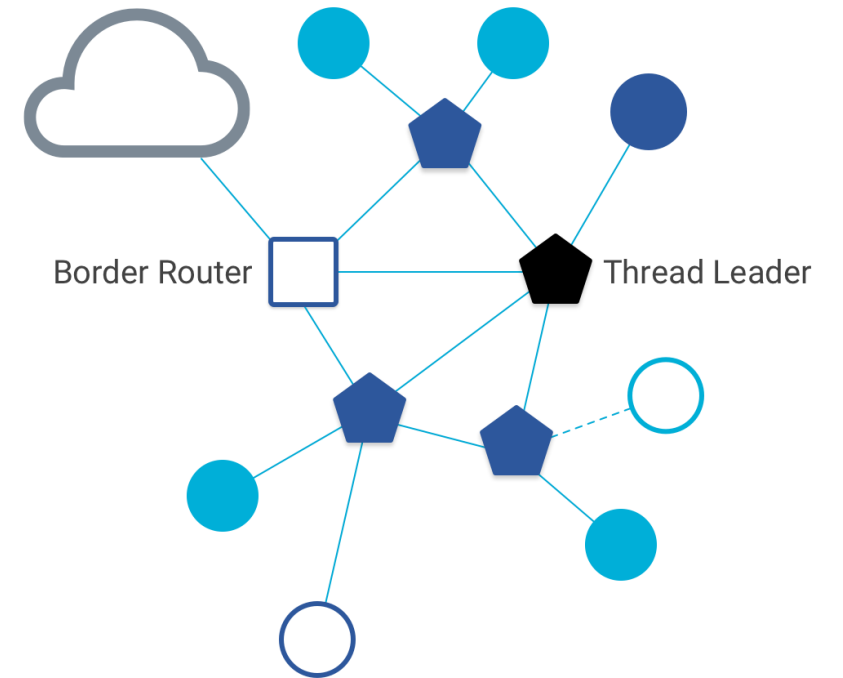


Why use Thread instead of basic 802.15.4?

- Full specification of upper layers
 - Specifies how networks are set up and managed
 - Clarifies how data is transmitted between devices on a network
 - Not just one single hop away
 - 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!

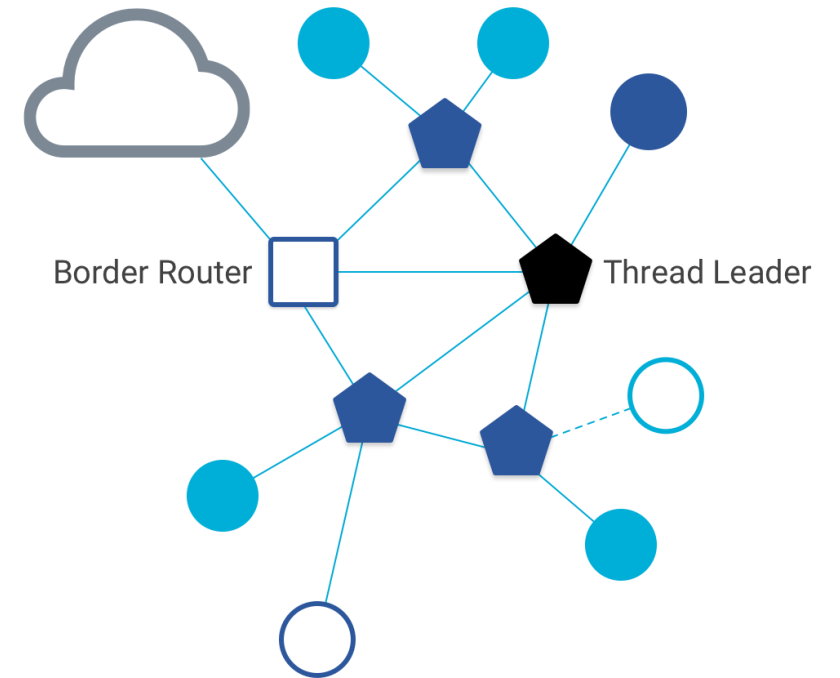
Break + Question

- What kinds of IoT devices are likely to have each thread role?
- End Device
- Router
- Border Router



Break + Question

- What kinds of IoT devices are likely to have each thread role?
- End Device
 - Battery-powered sensor
 - Nest Temperature Sensor
- Router
 - Wall-powered device
 - Nest Thermostat
- Border Router
 - Must have Internet connectivity
 - Custom gateway of some type?



Outline

- 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
 - Shouldn't they run standard protocols wherever possible?
 - Why IPv6?
 - Clean redesign of IPv4 makes it easier to use to support sensor networks
 - Address structure makes device addresses more compressible
- 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

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
 - **340 trillion, trillion, trillion** addresses
 - (> 100 times number of atoms on the surface of the Earth)
 - 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

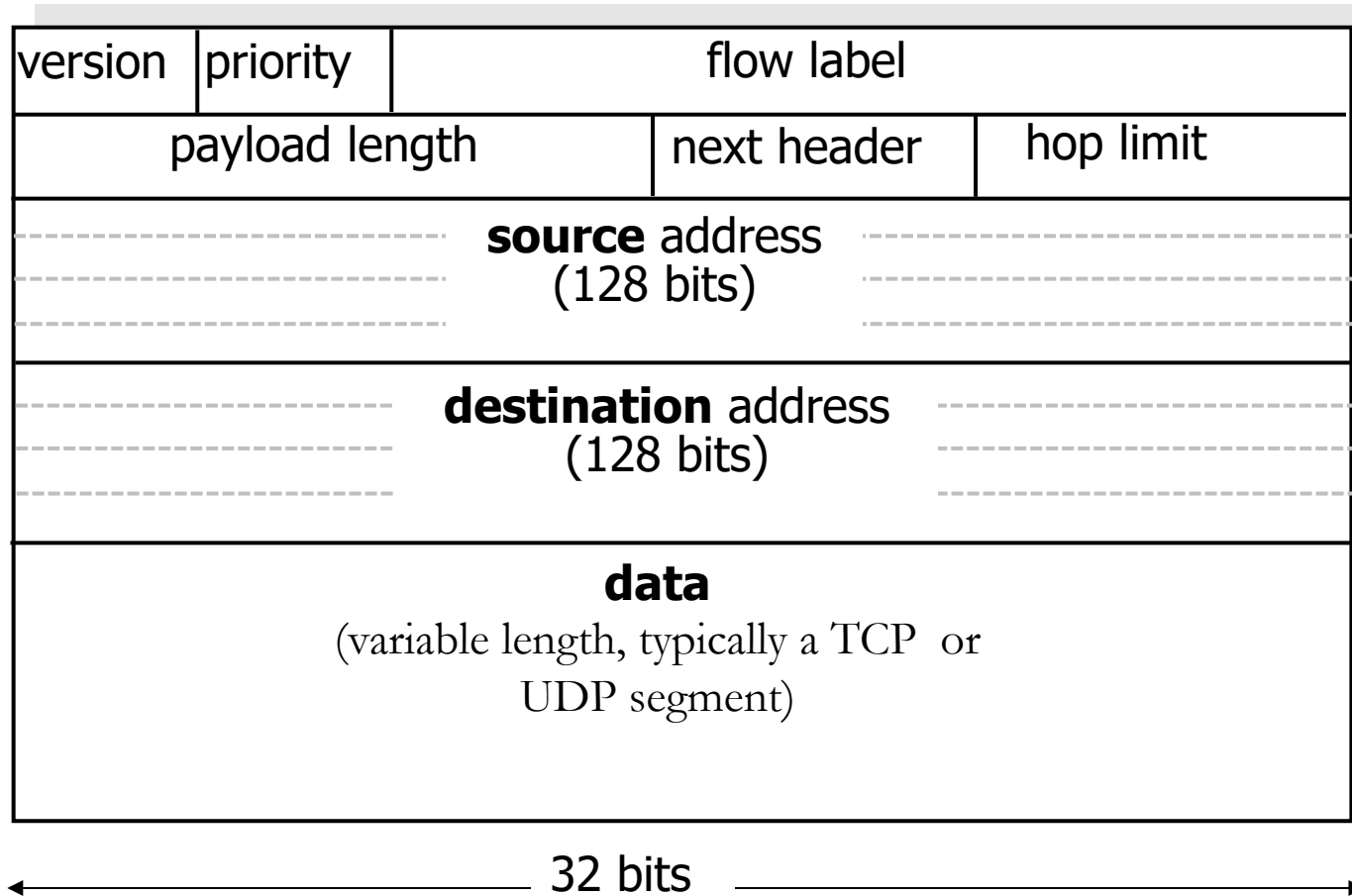
0000:0000:0000:0000:0000:0000:0000:0001 → ::1

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

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

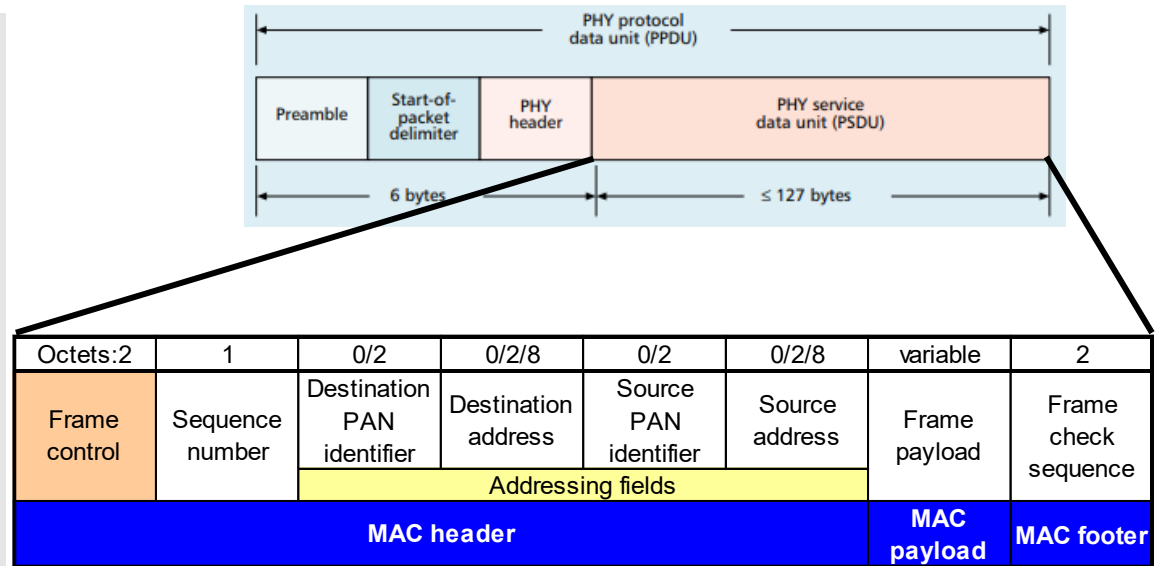
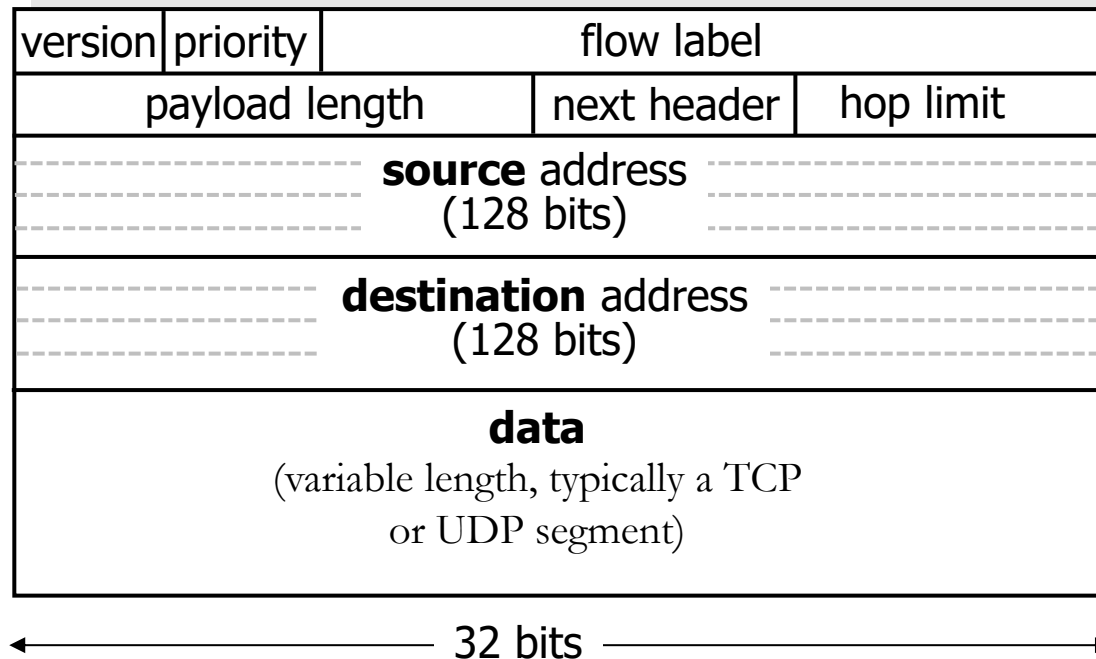
- Special addresses
 - Localhost - ::1 (IPv4 version is 127.0.0.1)
 - Link-Local Network - fe80:: (bottom 64-bits are ~device MAC address)
 - Local Network – fc00:: and fd00::
 - Global Addresses – 2000:: (various methods for allocating bottom bits)

Background: IPv6 datagram format



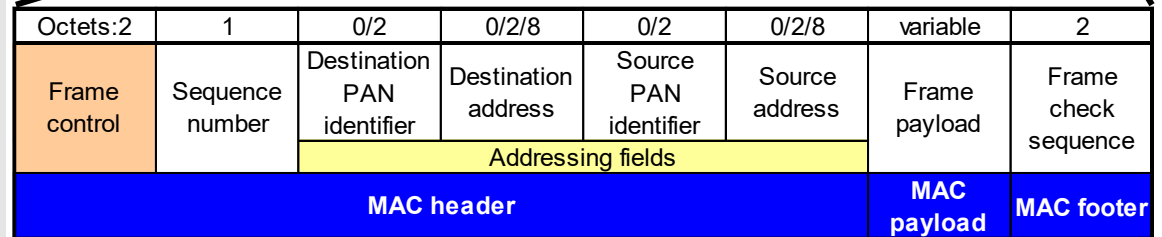
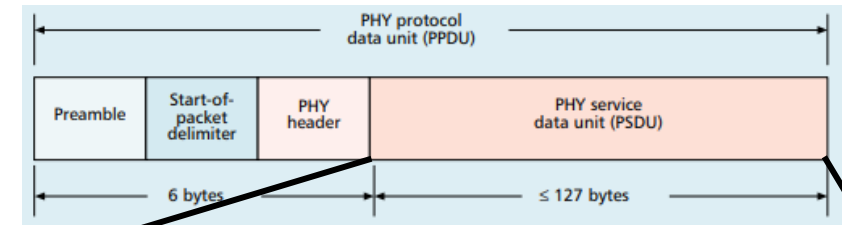
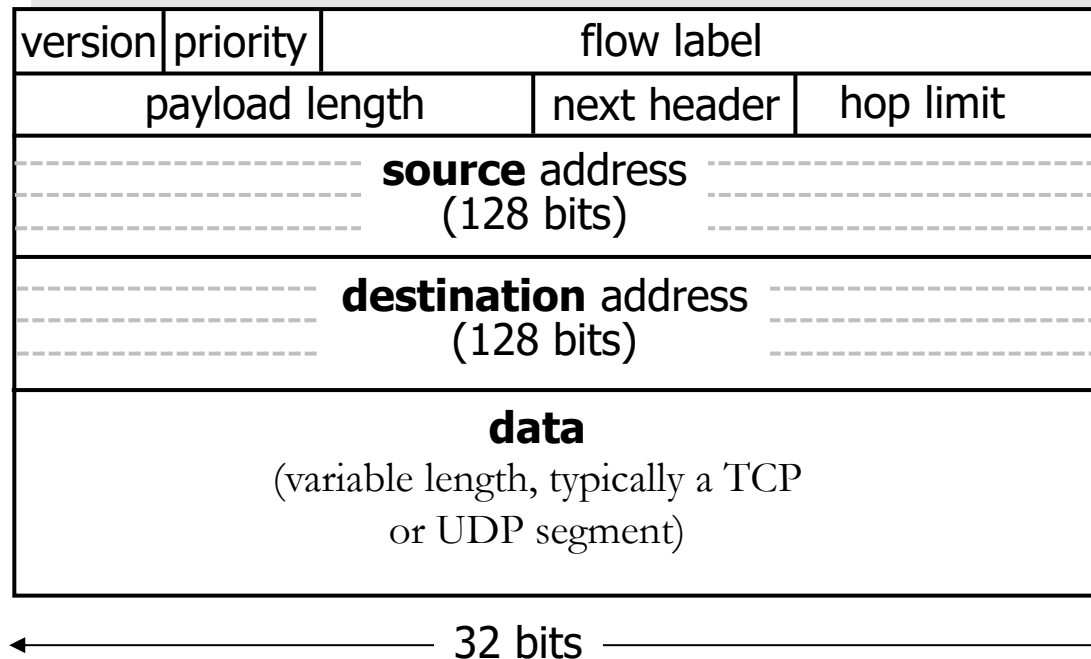
- Header starts with various data for the router
- **Priority**: like “type of service” in IPv4.
- **Flow label**: marks packets that are part of a transaction for routers
- **Next header**: TCP, UDP, ICMP, etc.
- **Hop limit**: hops to live

Fitting an IPv6 datagram in an 802.15.4 frame



- How many bytes for the IPv6 header?
- How much for the MAC header + footer?
- How much room do we get for payload?

Fitting an IPv6 datagram in an 802.15.4 frame



- How many bytes for the IPv6 header? - 8+16+16 = 40 bytes
- How much for the MAC header + footer? - 5 to 25 bytes depending on options
- How much room do we get for payload? - 127-40-N = 82 to 62 bytes

IPv6 tricky to use directly with 802.15.4

- Header overhead very high
 - around half of bytes in PSDU, even more when security enabled
- IP packets may be large, compared to 802.15.4 max frame size
 - IPv6 requires all links support 1280 byte packets [RFC 2460]
- Need some way to compress and fragment!

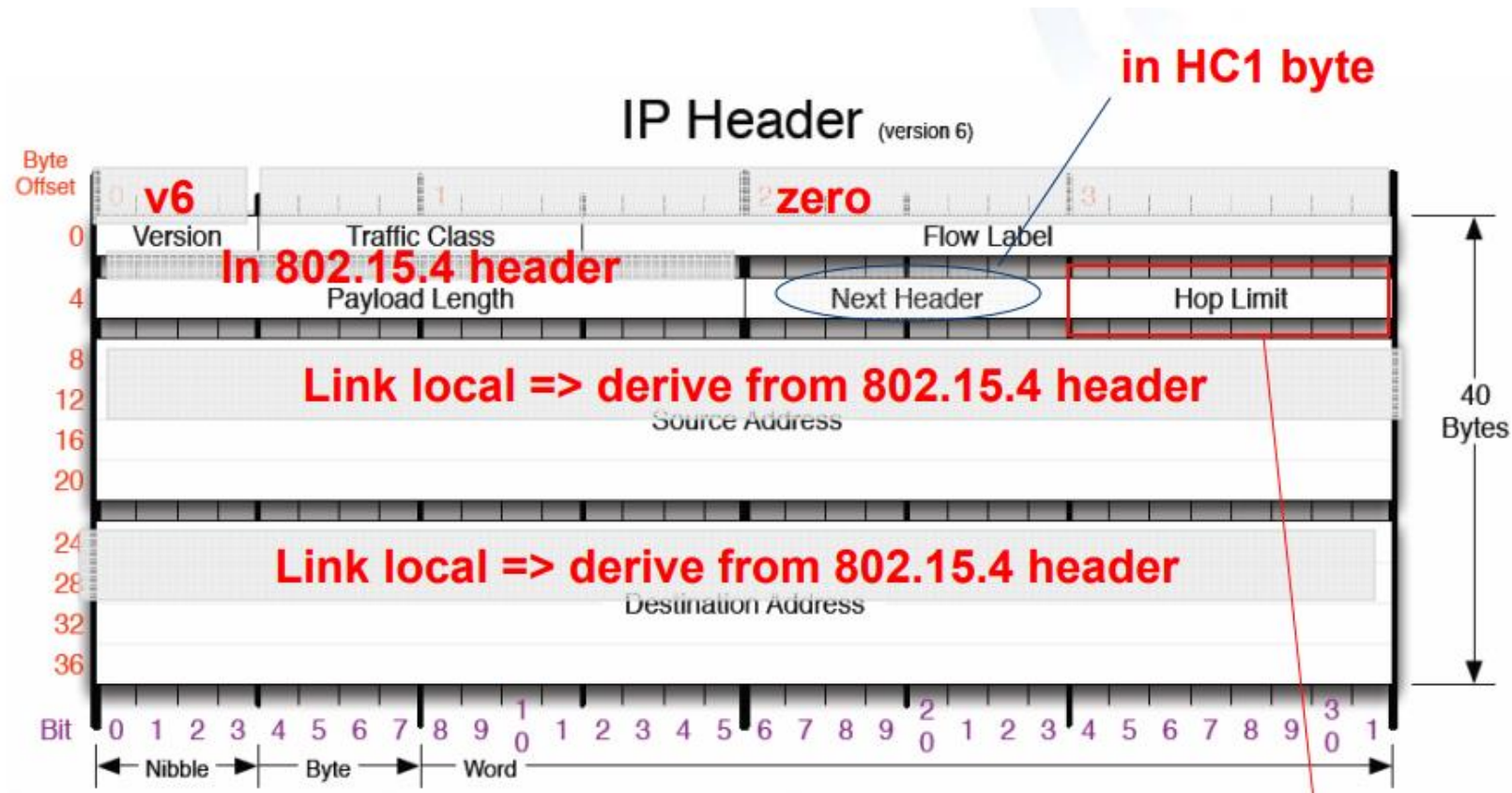
6LoWPAN

- Method for running IPv6 over 802.15.4 links
 - IPv**6** over **L**ow-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 **inside** the 15.4 network should be low-overhead
 - Communication **outside** of the 15.4 network must be possible
 - Still minimize overhead where possible, but might be larger
- 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



version	priority	flow label	
payload length		next header	hop limit
-----		source address	-----
-----		(128 bits)	-----
-----			-----
-----		destination address	-----
-----		(128 bits)	-----
-----			-----
data (variable length, typically a TCP or UDP segment)			

32 bits

- http://www.visi.com/~mjb/Drawings/IP_Header_v6.pdf *uncompressed*
- Example of HC1 header compression
 - Note: Thread actually uses IPHC from rfc6282 (not HC1), but similar idea
 - Good overview: <https://www.youtube.com/watch?v=wRdrCsgJekM>

6LoWPAN fragmentation

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

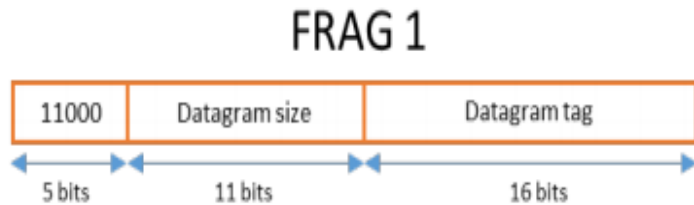


Figure 15. First Fragment Header

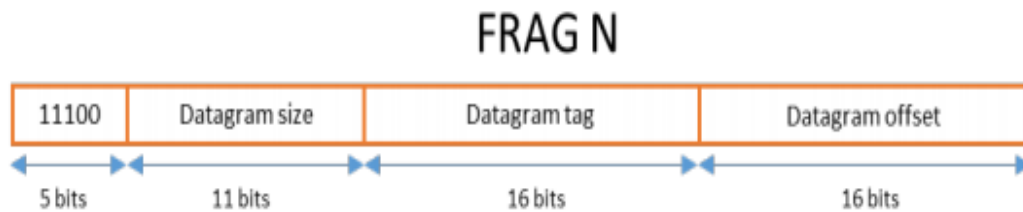


Figure 16. Subsequent Fragment Header

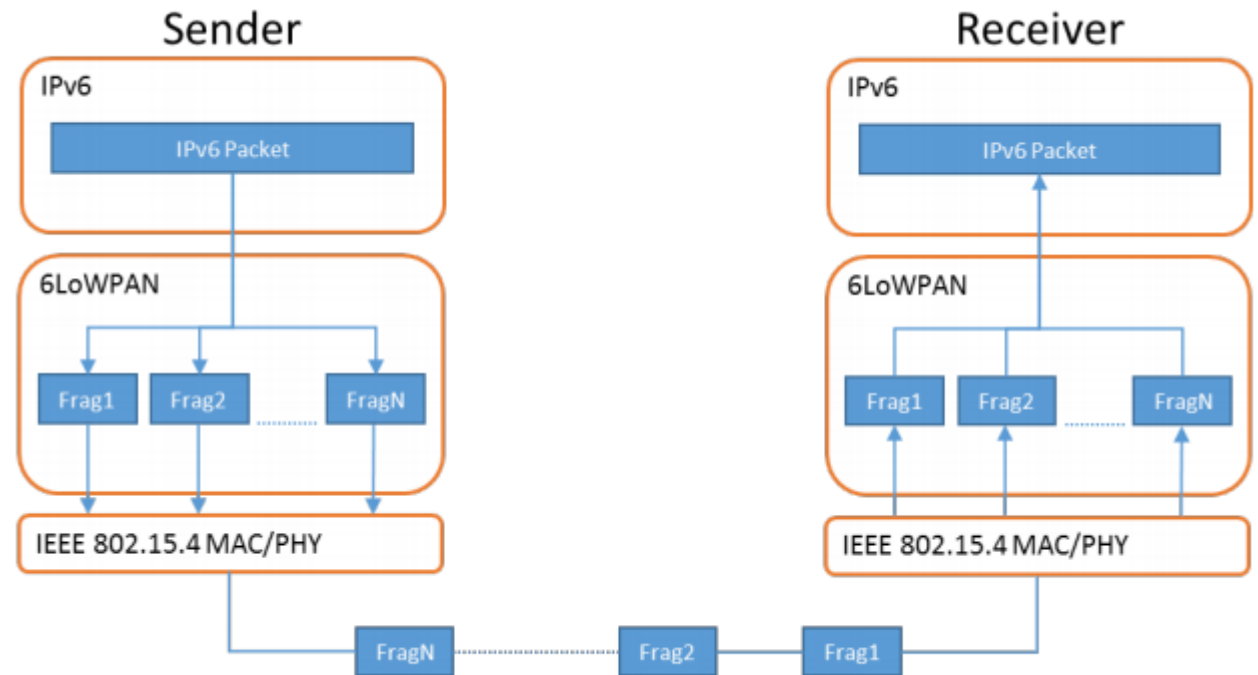


Figure 14. Fragmenting and Reassembling an IPv6 Packet

Sidebar: IPv6 over BLE

- [RFC7668](#) defines 6LoWPAN techniques for BLE connections

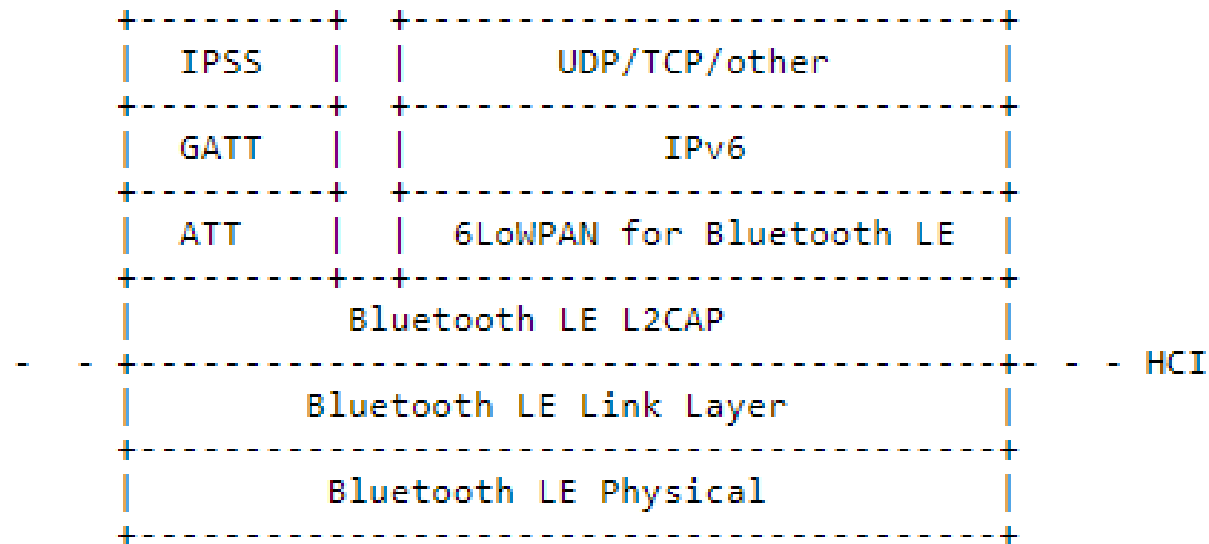
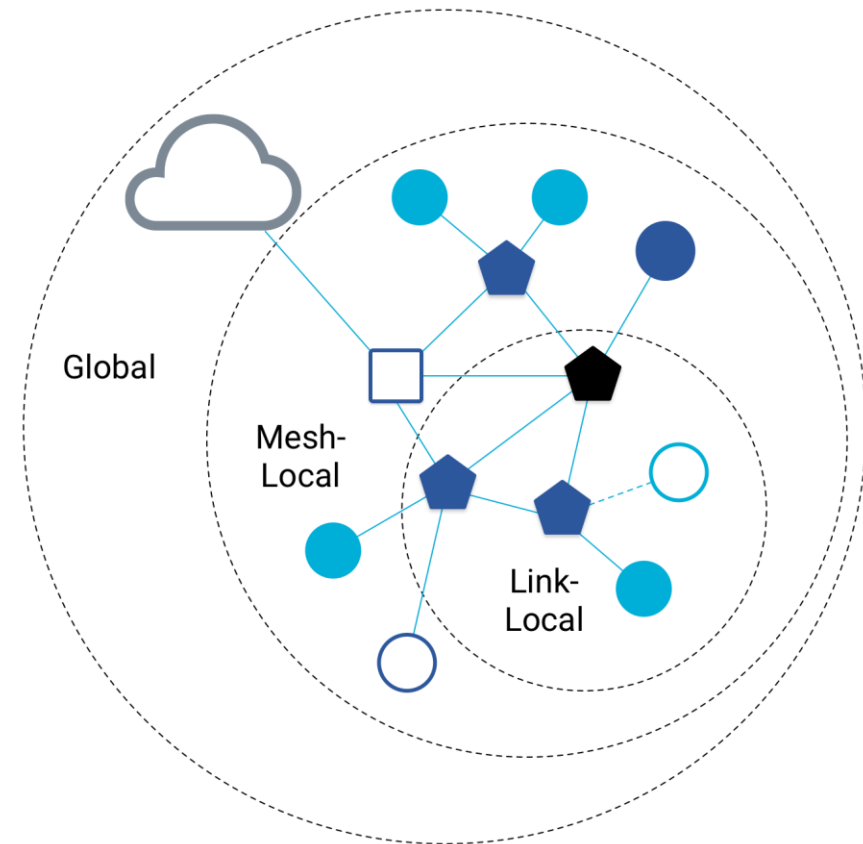


Figure 3: IPv6 and IPSS on the Bluetooth LE Stack

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
 - Send to parent
 - Send to child
- Role-based IP address(es)
 - Send to all Routers
 - Send to Border Router



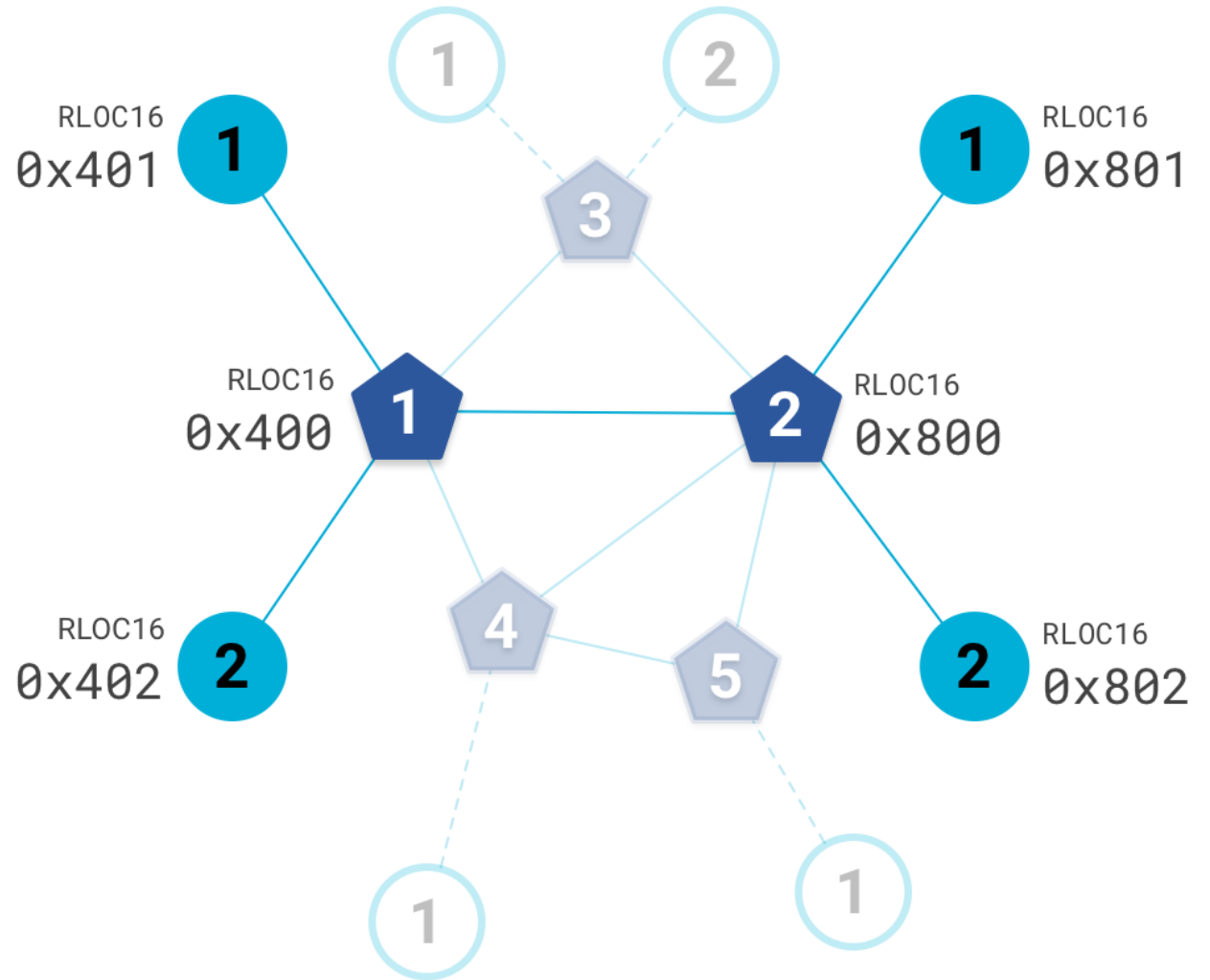
Traditional addresses in Thread

- 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 in IPv6)
 - Remaining bits are randomly chosen as part of joining the network
 - Permanent while connection is maintained to a network
 - Used for application-layer interactions within the network
- Global Addresses
 - 2000::/3 (2000:: are for global, unicast IP addresses in IPv6)
 - 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 concatenated with Child ID
- | Router ID | R | Child ID |
|-------------|---|-----------------------|
| 0 0 0 0 0 1 | 0 | 0 0 0 0 0 0 0 0 0 0 1 |

→ 0x401
- Changes with network topology
 - Used for link-layer interactions and for routing packets



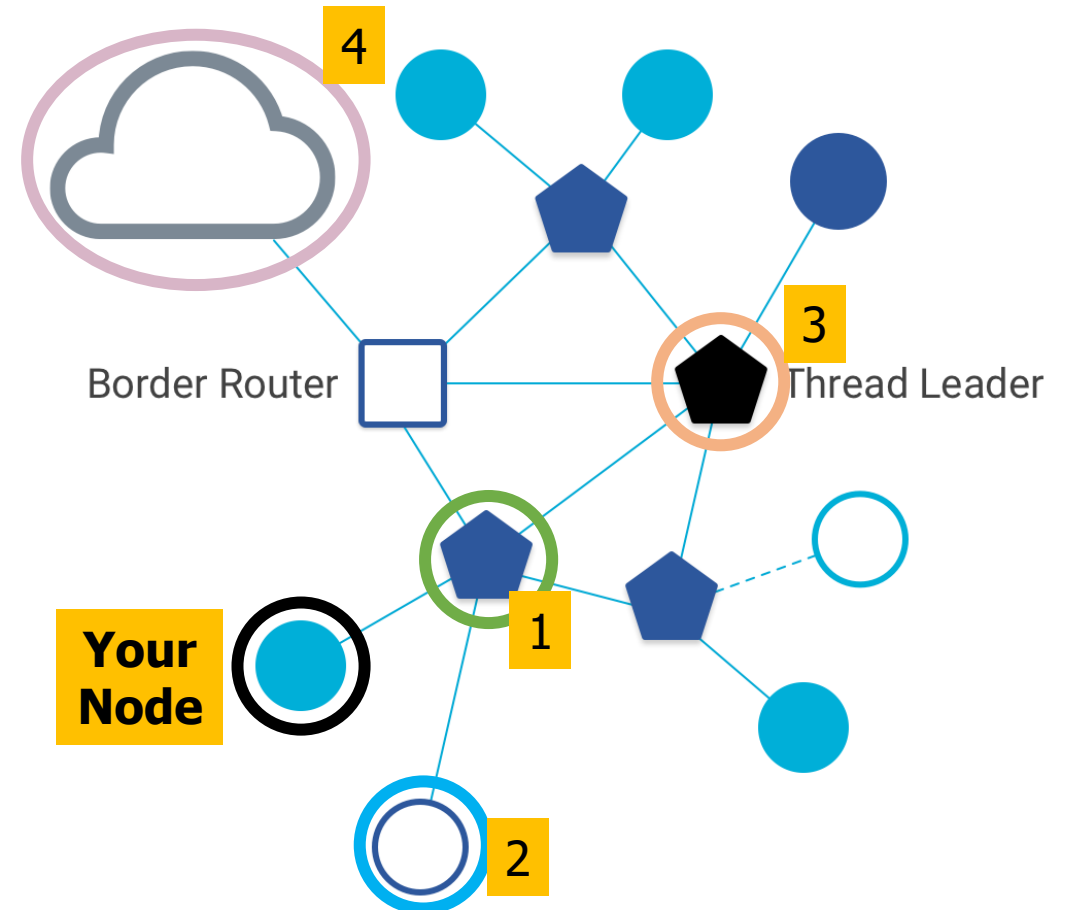
Role-based addresses in Thread

- Multicast (to all devices with role)
 - 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 (to any device with role)
 - FD00::00FF:FE00:FCxx
 - 00 – Thread Leader
 - 01-0F – DHCPv6 Agent
 - 30-37 – Commissioner
 - etc.
- Purpose:
- Link-layer communication
 - Configuration after joining network
 - Especially *discovery* of devices

Break + Question

Which type of address(es) could you use for communication?

1. Global
 2. Mesh-local
 3. Link-local
 4. Topology
 5. Role-based
- Communicate with each circled target



Break + Question

1. Green node

- Mesh-local
- Topology (parent)
- Link-local

2. Blue node

- Mesh-local
- Topology (other child of parent)
- Maybe link-local

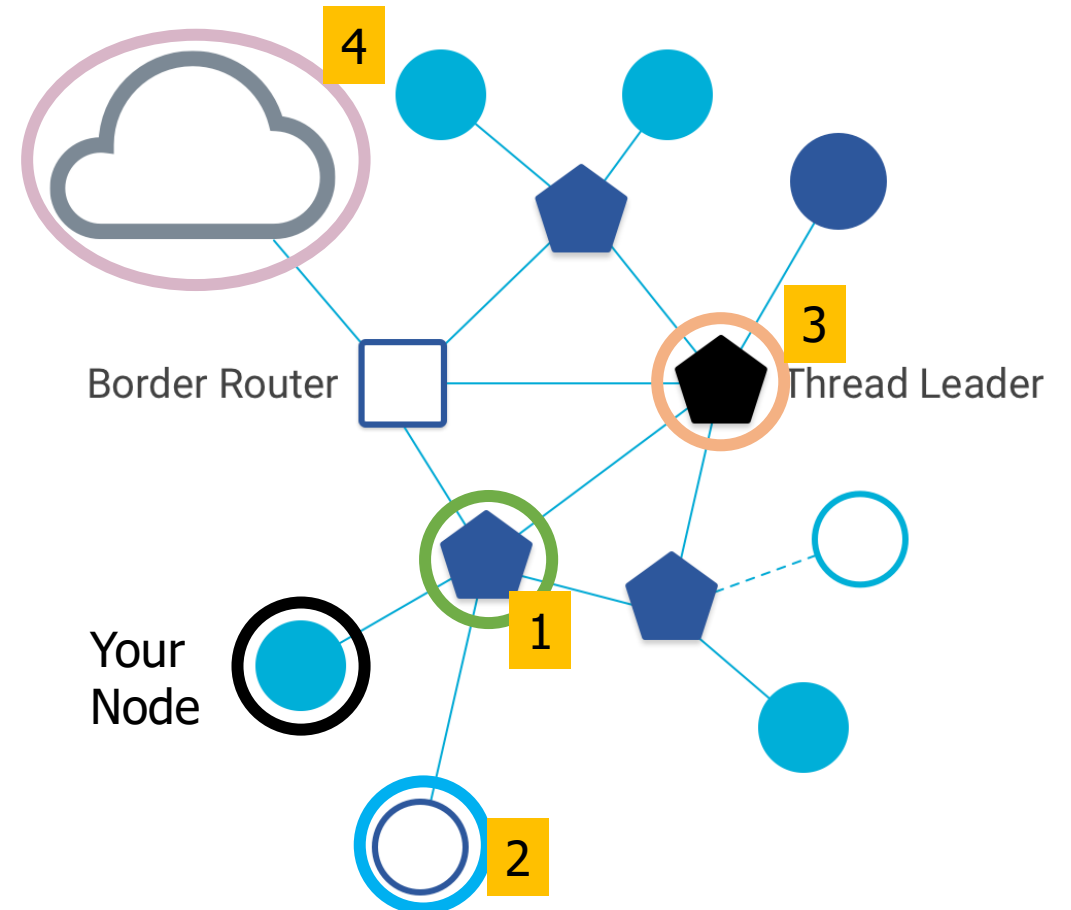
3. Orange node

- Mesh-local
- Role-based

4. Purple “cloud”

- Global

- Options
1. Global
 2. Mesh-local
 3. Link-local
 4. Topology
 5. Role-based

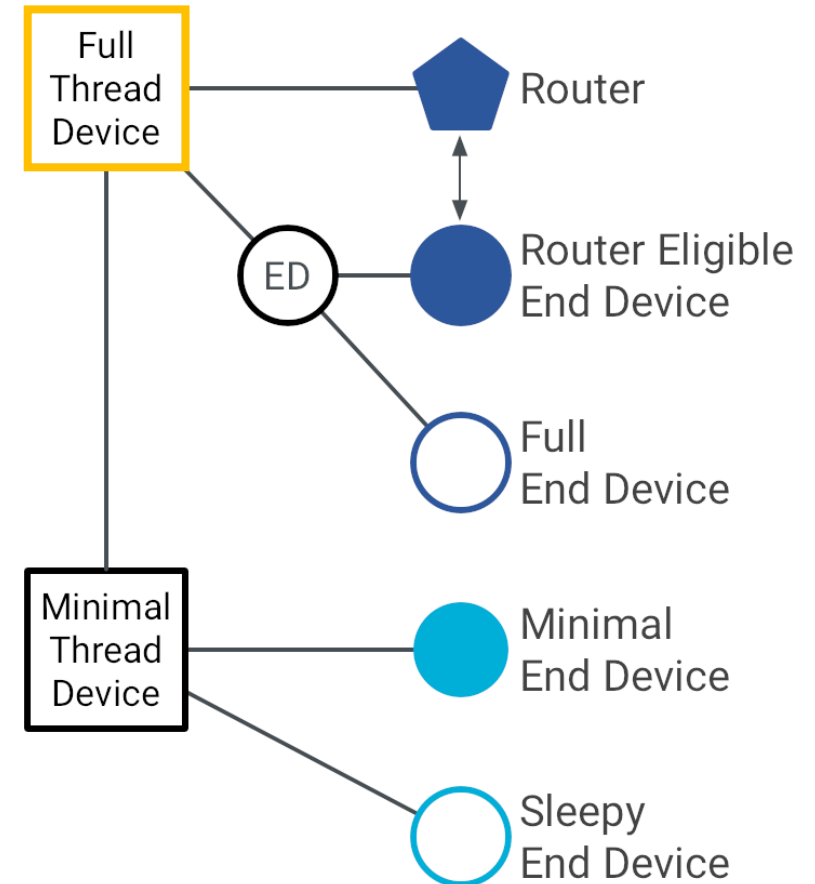


Outline

- Thread Overview
- Thread Addressing
- **Runtime Behavior**
- Using IP

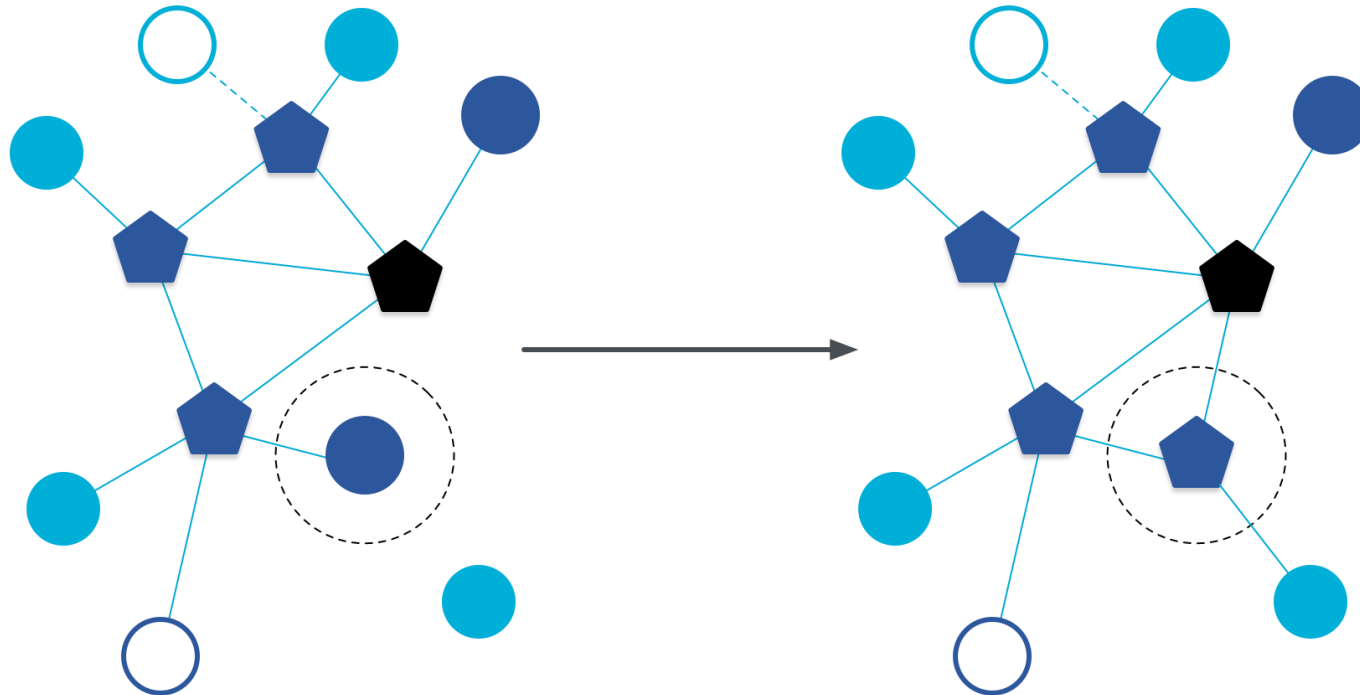
Thread device types

- Full Thread Devices
 - MAC: Always-on
 - Participate in routing
 - Examples:
 - Routers
 - Thread Leader
 - Border Router
 - Router-Eligible End Device (REED)



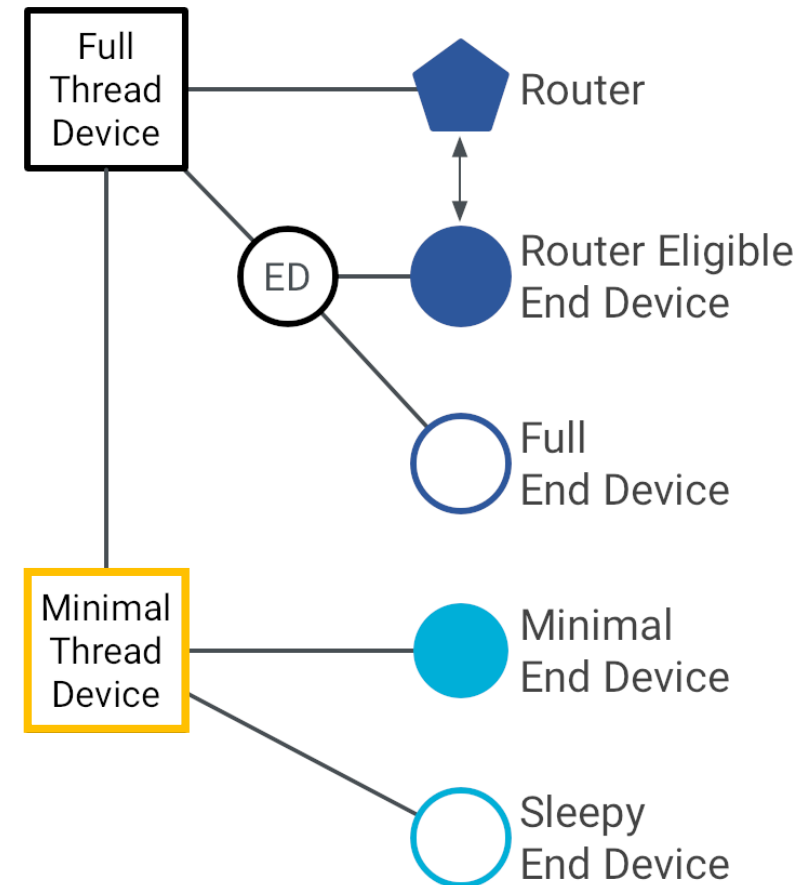
Router promotion

- Router Eligible End Device (REED)
 - 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)



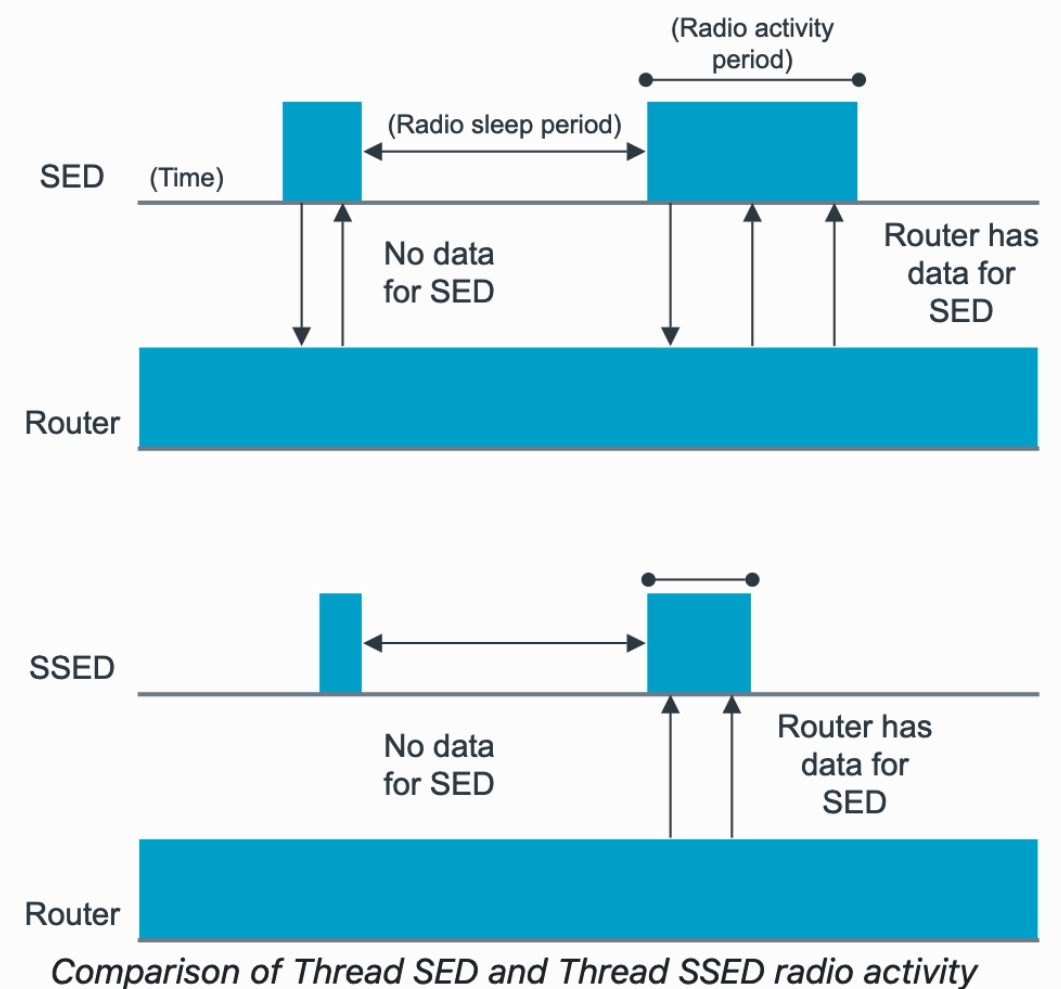
Thread device types

- Full Thread Devices
 - MAC: Always-on
 - Participate in routing
 - Examples:
 - Routers
 - Thread Leader
 - Border Router
 - Router-Eligible End Device (REED)
- Minimal Thread Devices
 - MAC: Can be low power
 - Only send/receive to/from parent
 - Examples:
 - Minimal End Device (MED)
 - Sleepy End Device (SED)
 - Synchronized Sleepy End Device (SSSED)



Sleepy End Devices

- These are the low power devices
- Periodically must exchange messages with Router
- Sleepy End Device (SED)
 - MAC:
 - TX: Router always-on, so send as needed (with CSMA)
 - RX: Periodically check in with router if new data pending
- Synchronized Sleep End Device (SSED)
 - MAC:
 - TX: Send as needed (with CSMA)
 - RX: Listen during pre-configured windows (still CSMA from Parent)

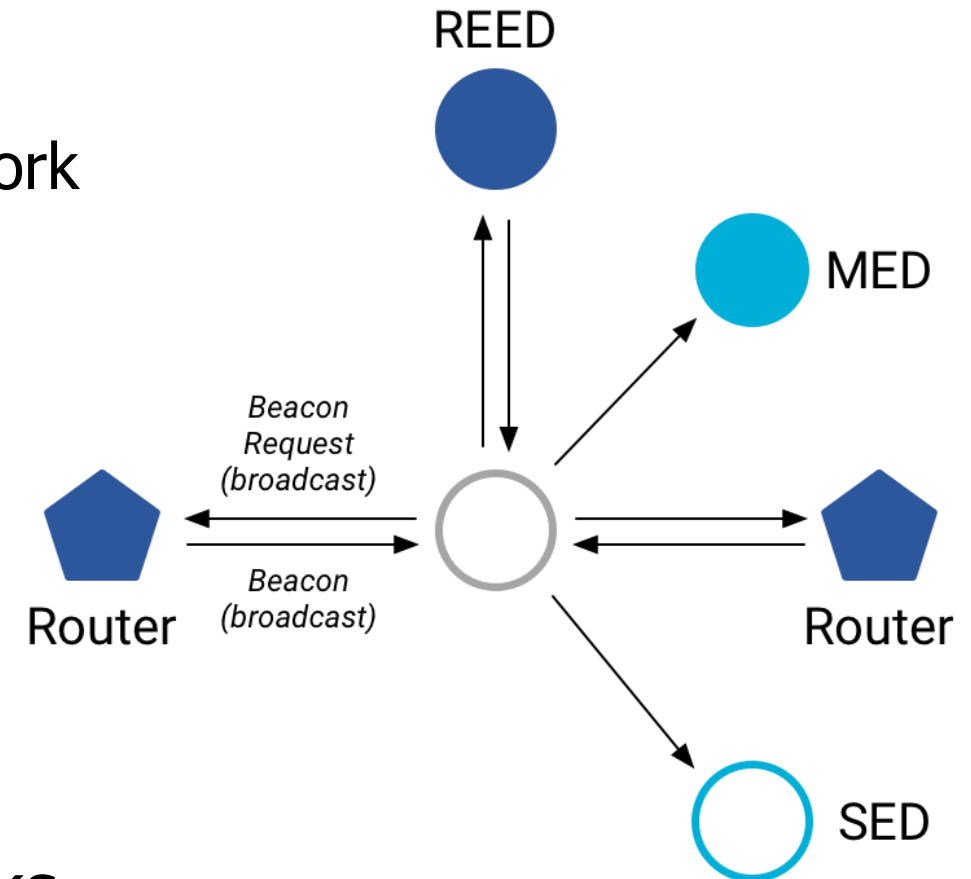


Creating a new network

- 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

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



Mesh-Level Establishment (MLE)

- 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
-----	---------------	-----------------	-----	-----	-----	-----

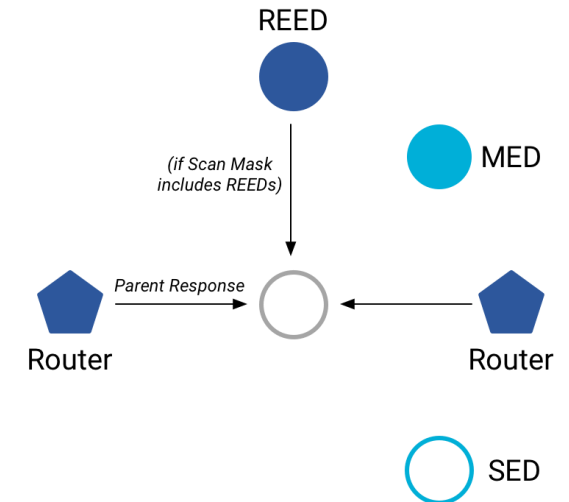
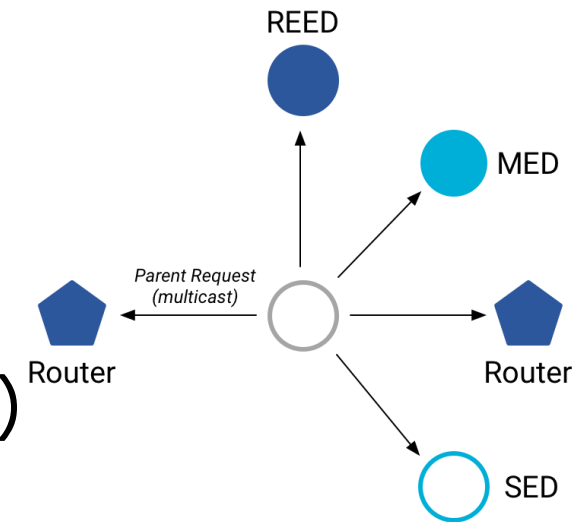
- TLVs (Type-Length-Value)

- Various data types that may be helpful within those packets
 - Addresses, Link Quality, Routing Data, Timestamps

Joining an existing network

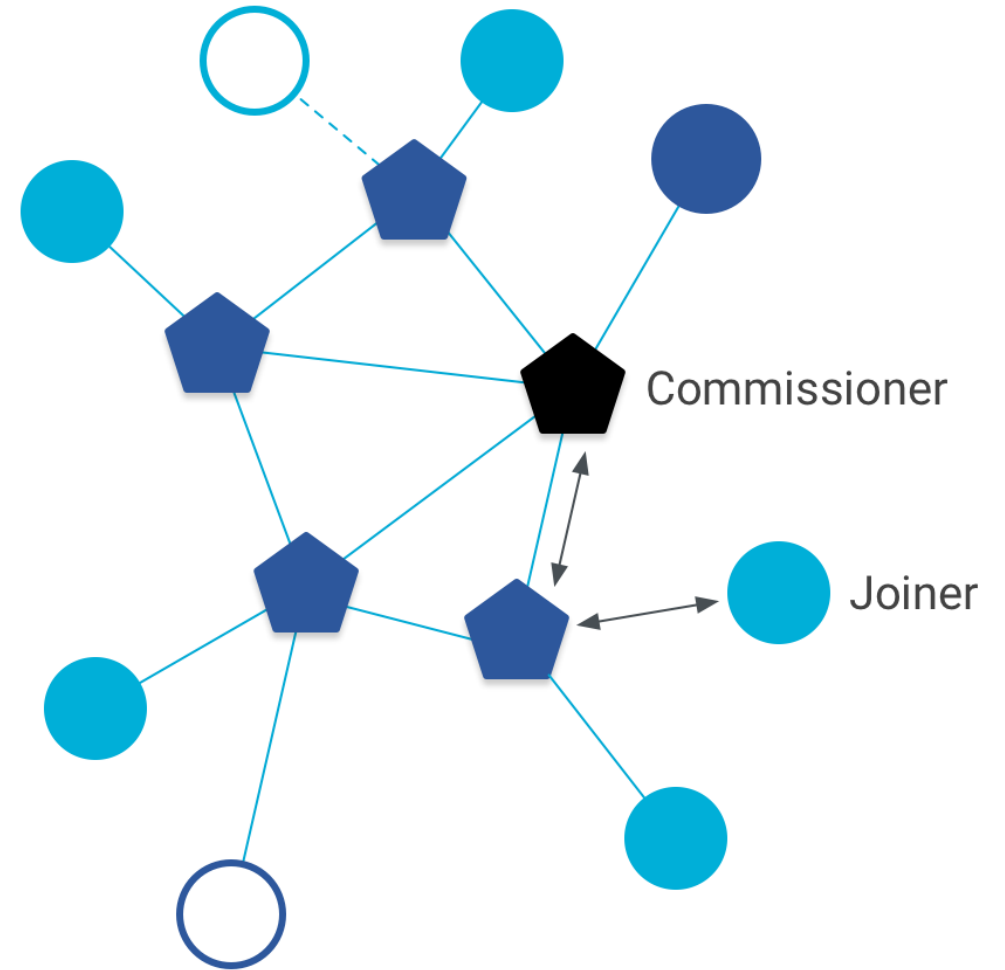
- All devices join as a child of some existing router

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



Thread Commissioning

- As part of joining a network, a thread device is also “commissioned”
- Bonus step in the joining process for real-world networks
- Authenticates the joining device allowing it to join and encrypt communications



Updates to Thread specification

- Thread version 1.2
 - Simpler, lower-power end devices
 - Any packet sent is enough to keep the device attached to network
 - All acknowledgements identify if packets are available for it
 - Power control
 - Reduce transmission power to minimum needed for “good connection”
 - Reduces potential for “hidden terminal problem”
- Thread version 1.3
 - Better IPv6 support
 - Coordinate address selection with IP network border router connects to
 - Shuttle DNS Service Discovery messages between networks
 - Support for TCP
- Thread version 1.4
 - Credential-sharing mechanism to make joining a thread network simpler

Outline

- 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



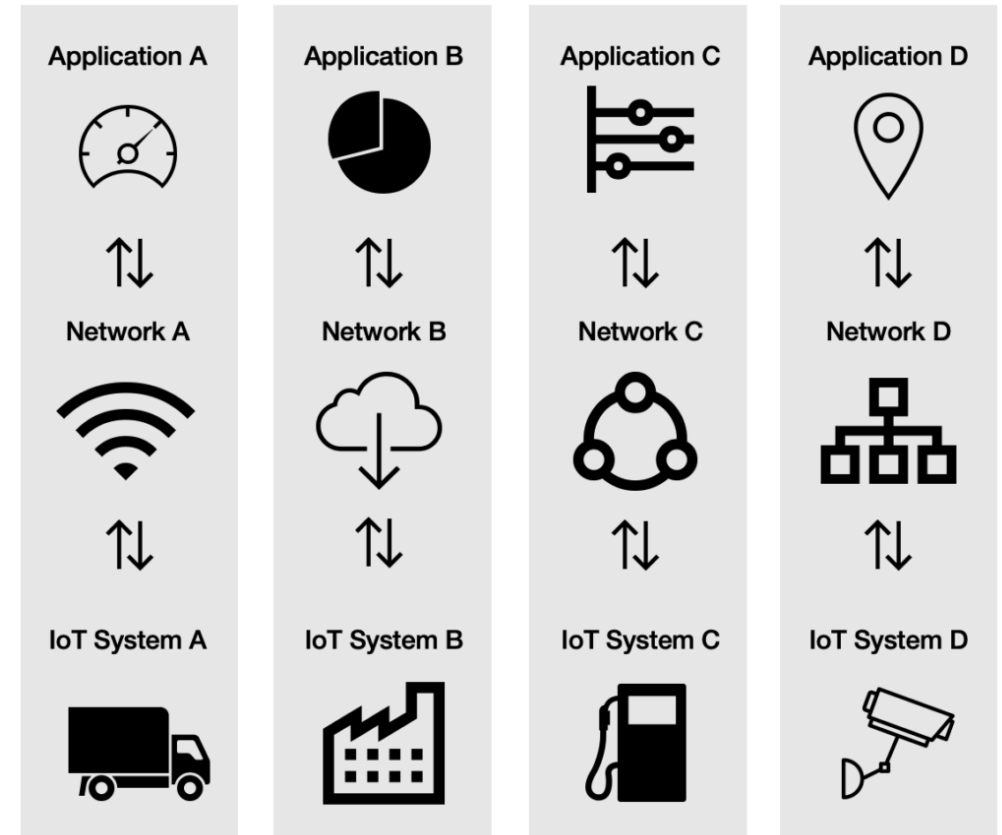
- CoAP Requests
 - Have a type: GET, POST, PUT, DELETE
 - Have a URL: /file/etc
 - Have data up to 65 KB

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
- Active research:
 - Sam Kumar, Michael Anderson, Hyung-Sin Kim, David Culler. ["Performant TCP for Low-Power Wireless Networks"](#). 2020.
 - The debate is still very much open
- 2023 update: Thread now supports TCP!!
 - Primarily for large data payloads, like firmware updates

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
- Concerns
 - What IP address do you send data to?
 - Manufacturer's server is an obvious choice
- We'll talk about the Matter standard as an approach to this issue



Outline

- Thread Overview
- Thread Addressing
- Runtime Behavior
- Using IP