Lecture 05
BLE Connections

CS397/497 – Wireless Protocols for IoT

Branden Ghena – Spring 2024

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

Northwestern

Administrivia

- Lab: BLE
  - Available now! Get started with your group
    - Not every member has to have every individual part working
    - Just one submission per group

- Homework: BLE Packets
  - Should be out later tonight
  - Goal: not busy work, but actually get you interpreting payload data

Today's Goals

- Explore how connections work
  - What does the link layer look like?
  - How do higher layers interact to share data?

- Investigate network questions about connections

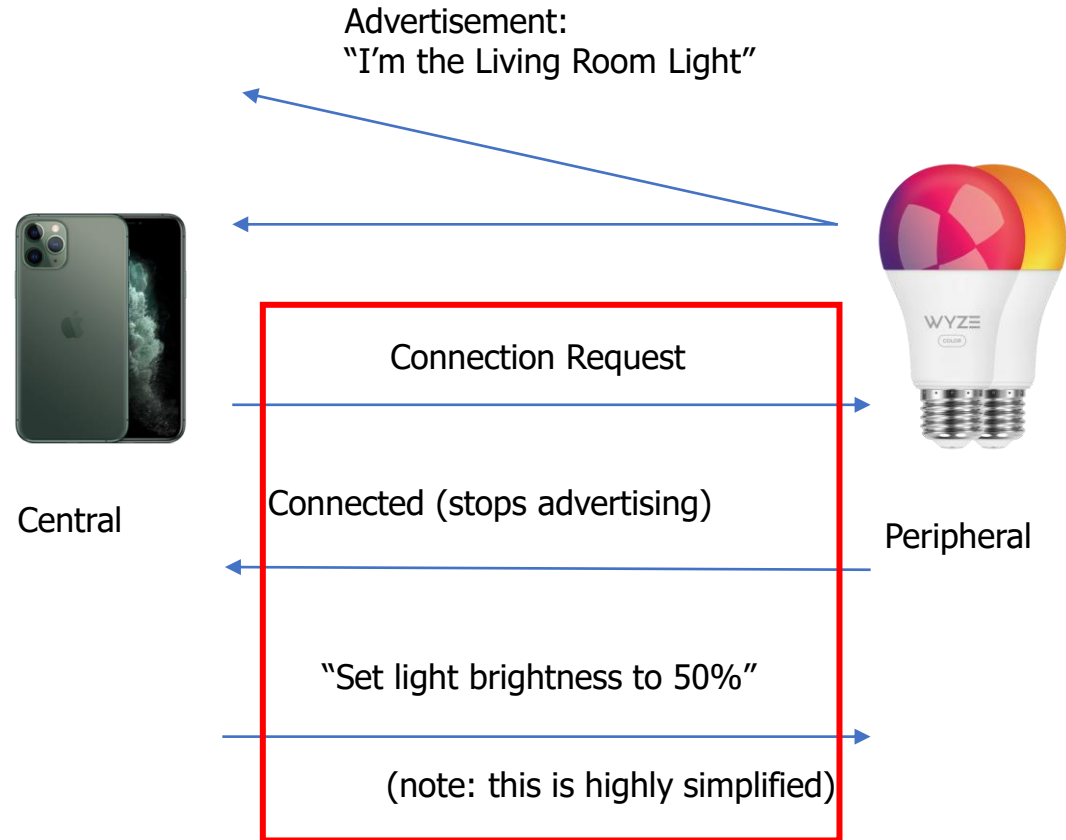- Overview of additions in BLE 5.0

Useful documentation

- [5.2 specification] [4.2 specification] (link to PDF download)

- https://www.novelbits.io/deep-dive-ble-packets-events/

- Thinking about BLE connection data transfer rates
  1. https://punchthrough.com/maximizing-ble-throughput-on-ios-and-android/
  2. https://punchthrough.com/maximizing-ble-throughput-part-2-use-larger-att-mtu-2/
  3. https://punchthrough.com/maximizing-ble-throughput-part-3-data-length-extension-dle-2/

Outline

- **Connection PHY and Link Layer**

- Connection Investigations

- GATT

- BLE 5

Recall: BLE advertisements lead to connections

Advertisement:
"I'm the Living Room Light"

Central

Peripheral

Connection Request

Connected (stops advertising)

"Set light brightness to 50%"

(note: this is highly simplified)

We'll explore what happens here!

Overview of connections

- Connections are for bi-directional communication with higher throughput than advertisements
  - Half-duplex: only one direction at a time

- Simple view
  - A peripheral is either advertising or in a connection
  - A central is scanning and in one or more connections
  - Not quite accurate: devices can have several roles simultaneously

- While in a connection both devices act like servers
  - Either device can read/write fields available on the other device

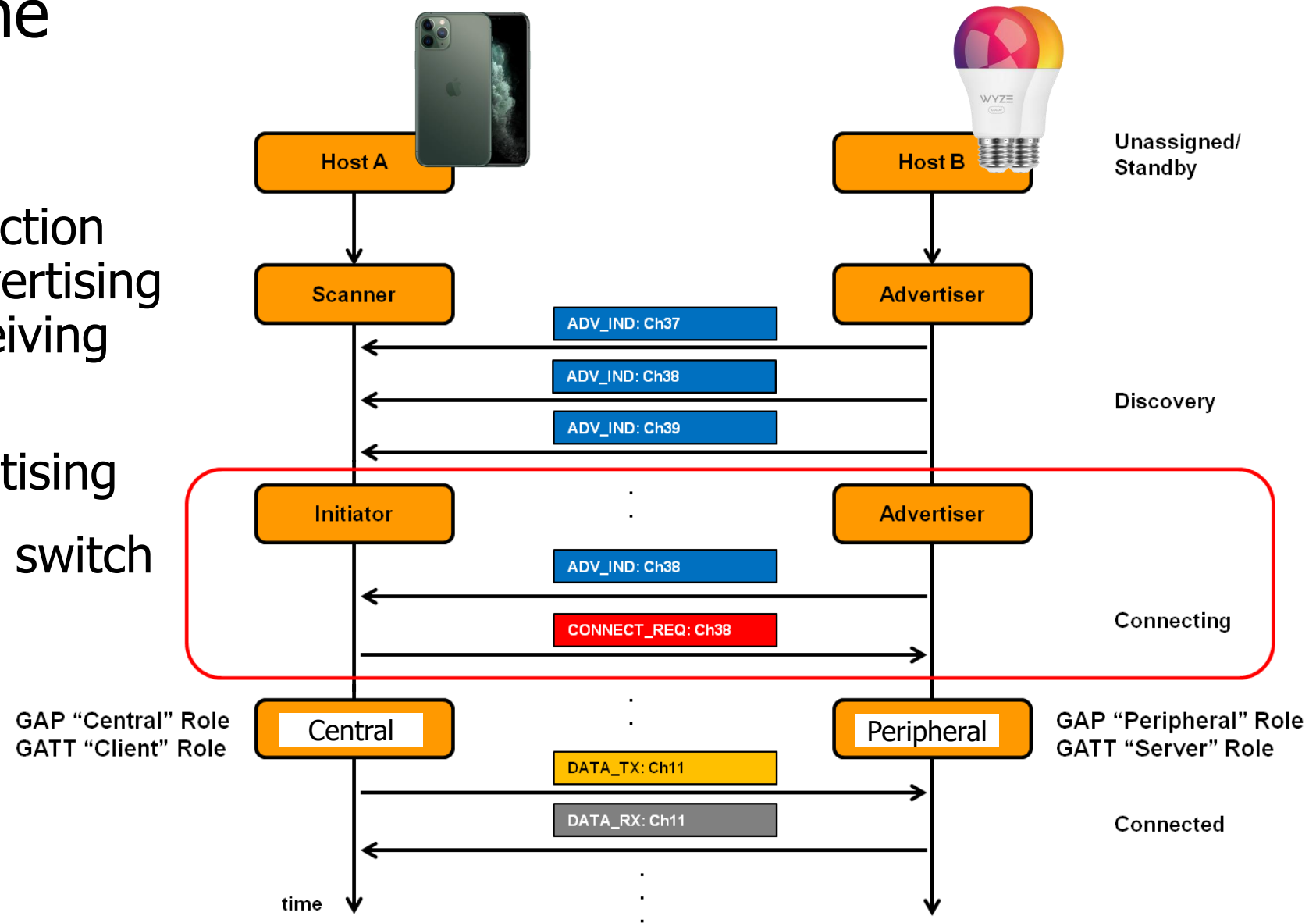# Once a connection is established, BLE has more PHY options

- LE 1M (default)
  - 1 Msym/s
  - BLE encodes 1 bit / symbol, so this is also 1 Mbit/s

- LE 2M (added in BLE 5.0)
  - 2 Msym/s

- LE Coded (added in BLE 5.0)
  - 1 Msym/s + FEC
    - S = 2
      - ~2x range, ½ effective data rate – 500 Kbit/s **goodput**
    - S = 8
      - ~4x range, 125 Kbits/s **goodput**

Data

Each of these are symbols

Modulated Signal
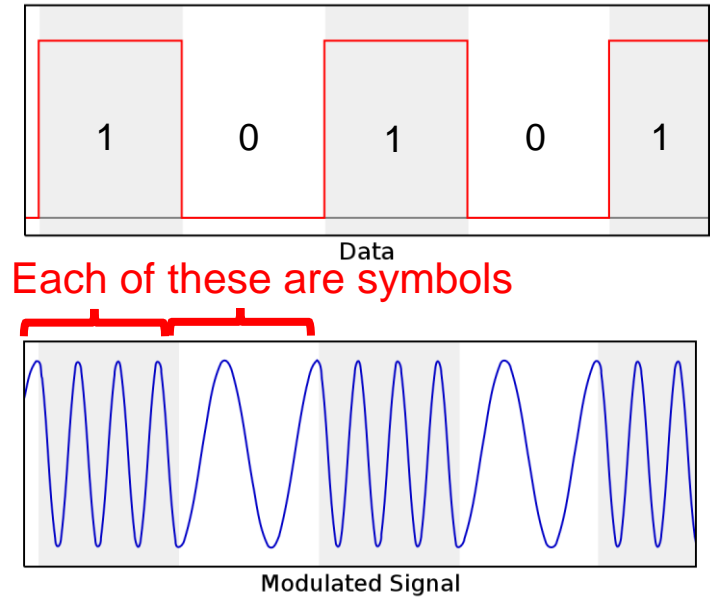
FSK Example

Connection Timeline

- Peripheral advertises
- Central sends a connection request packet on advertising channel just after receiving advertisement
- Peripheral stops advertising
- Central and peripheral switch to a data channel

Host A	Host B — Unassigned/Standby
Scanner	Advertiser
	ADV_IND: Ch37
	ADV_IND: Ch38 — Discovery
	ADV_IND: Ch39
Initiator	Advertiser
	ADV_IND: Ch38
	CONNECT_REQ: Ch38 — Connecting
GAP "Central" Role / GATT "Client" Role — Central	Peripheral — GAP "Peripheral" Role / GATT "Server" Role
	DATA_TX: Ch11
	DATA_RX: Ch11 — Connected
time	

9

Choosing a data channel for communication

- Why do you switch from advertising channel to data channel?

- How to pick a channel out of the 37 data channels?

- What if two pairs pick same channel?

Choosing a data channel for communication

- Why do you switch from advertising channel to data channel?
  - For better reliability and to improve quality of service

- How to pick a channel out of the 37 data channels?
  - Randomize for reduced chance of collision

- What if two pairs pick same channel?
  - Higher chance of collision occurring
  - Some way to not be tied to the same data channel is needed

Frequency Hopping Spread Spectrum (FHSS)

- Recall: Each BLE channel is 2 MHz wide, 40 channels,
  - 3 are used for advertising, remaining 37 are for connections
  - Frequency hopping: $f_{n+1} = (f_n + hop)\ mod\ 37$

- Which exact channels are used and in what order might vary
  - "Adaptive Frequency Hopping" avoids bad channels
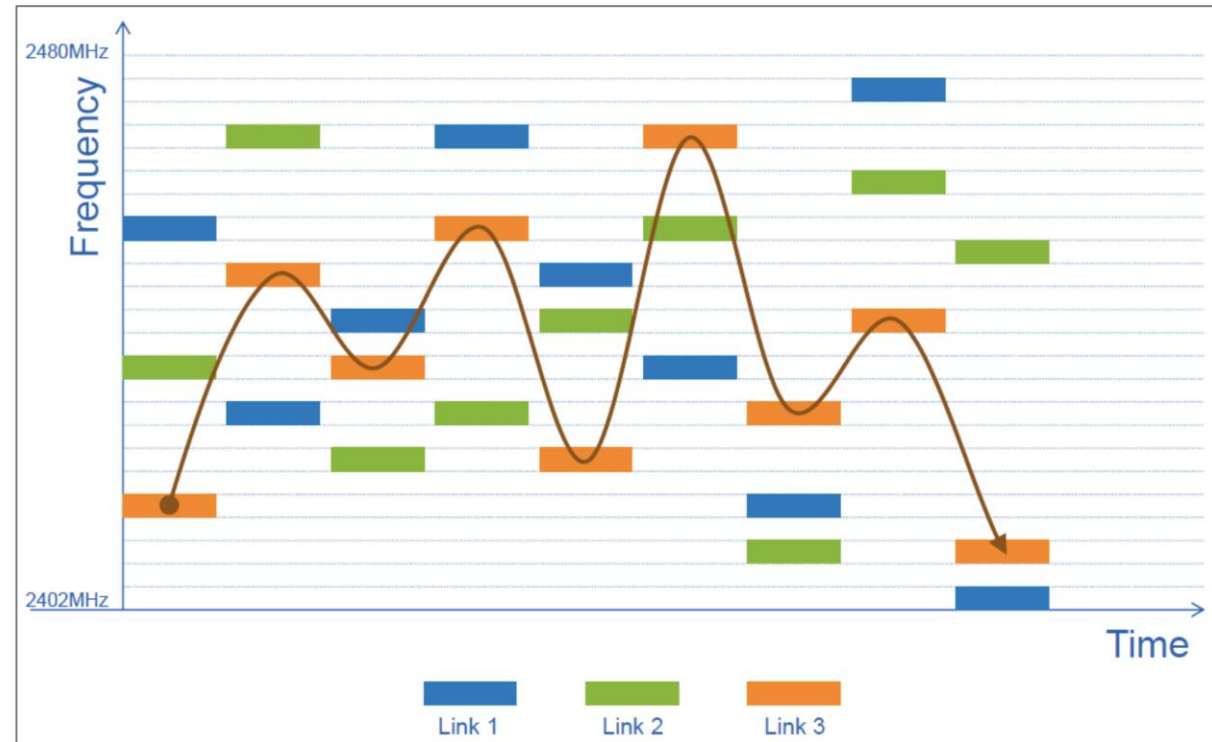
How do the Central and Peripheral agree on things?

- For a packet to be received:
  - Central and peripheral should be listening and sending on same data channel

- For the hopping to be successful, both must:
  - Follow same hop sequence
    - Which <u>channels to use</u> and the <u>order</u> to follow
  - Hop at a given time schedule
  - Agree on when to schedule their first communication

- Some synchronization is needed!

Connection request packet

- Scanner waits until it sees an advertisement from a device it wants to connect to
  - Requesting a connection (in higher layers) just starts this search process

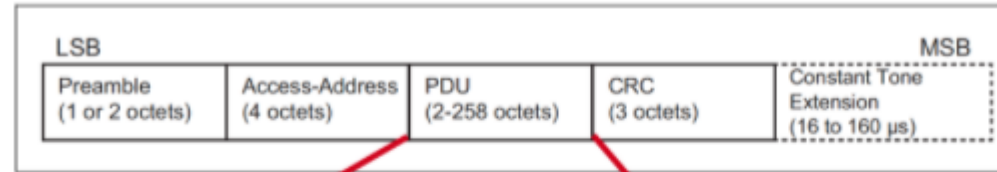- Sends connection request payload in response to advertisement

Figure 2.1: Link Layer packet format for the LE Uncoded PHYs
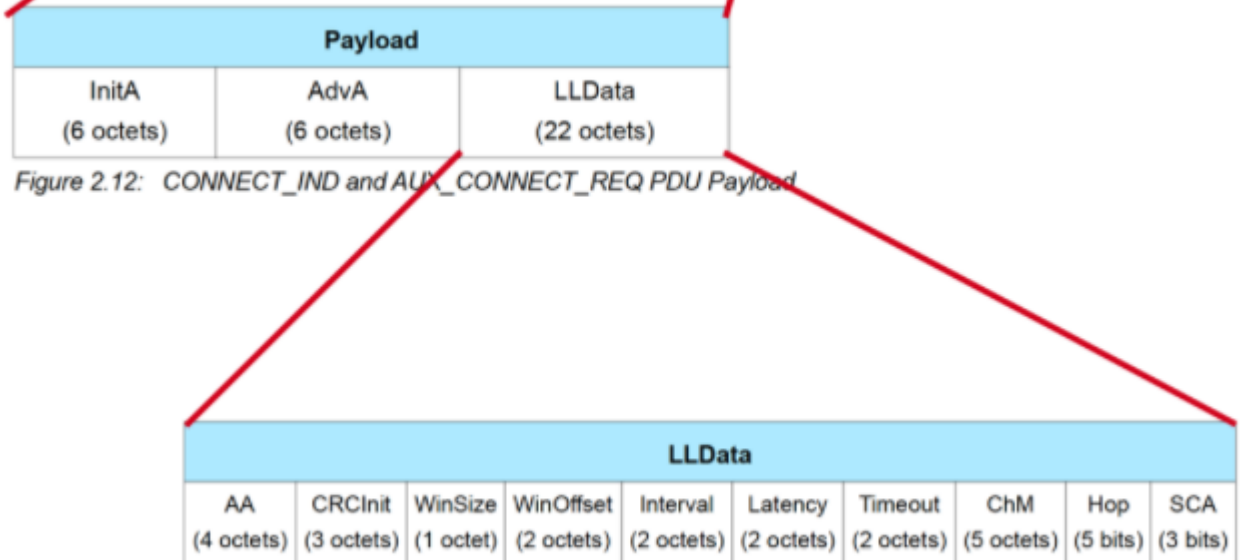
Figure 2.4: Advertising physical channel PDU

Figure 2.12: CONNECT_IND and AUX_CONNECT_REQ PDU Payload

Connection request parameters examined

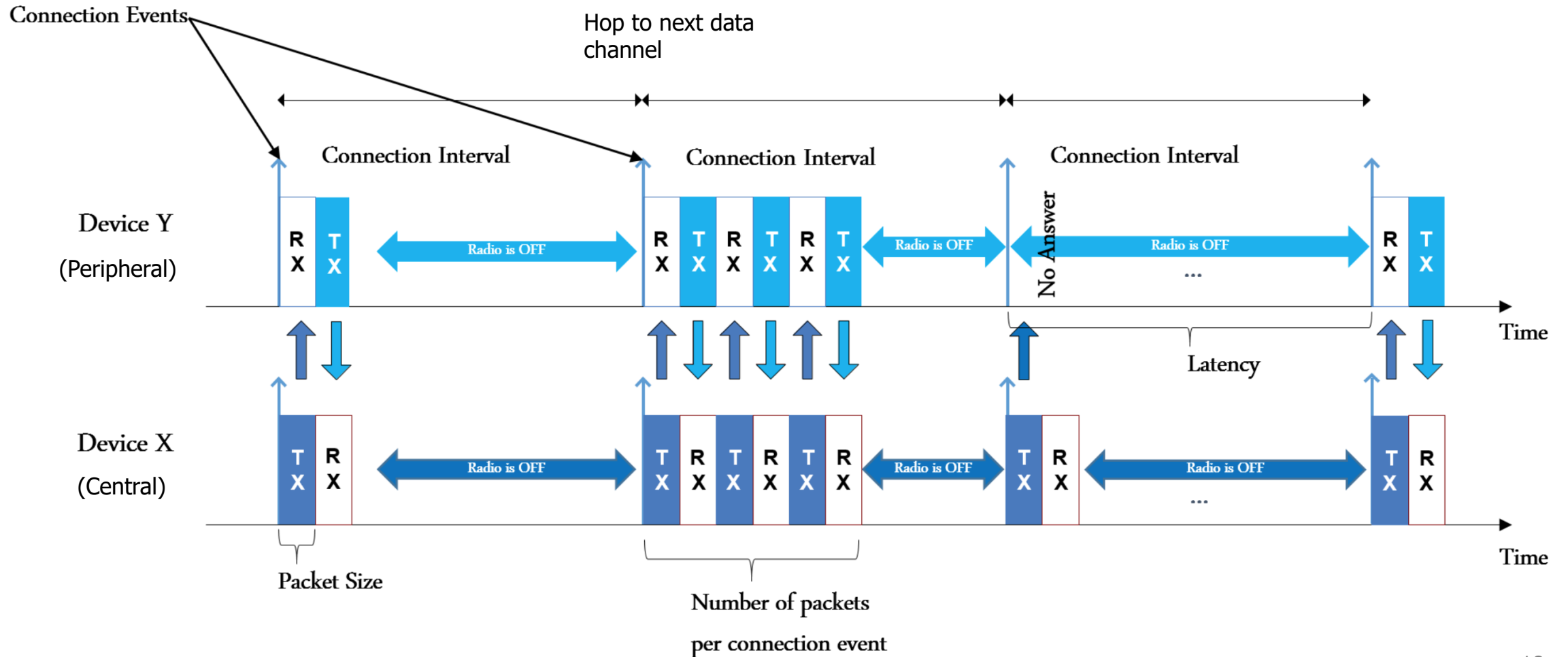| LLData									
AA	CRCInit	WinSize	WinOffset	Interval	Latency	Timeout	ChM	Hop	SCA
(4 octets)	(3 octets)	(1 octet)	(2 octets)	(2 octets)	(2 octets)	(2 octets)	(5 octets)	(5 bits)	(3 bits)

- Specifies parameters of the connection
  - Peripheral must either agree or totally reject the connection
  - Peripheral can later propose a change to the connection parameters

- **Interval** is how frequently connection events occur (7.5ms – 4s)

Steady-state connection timing
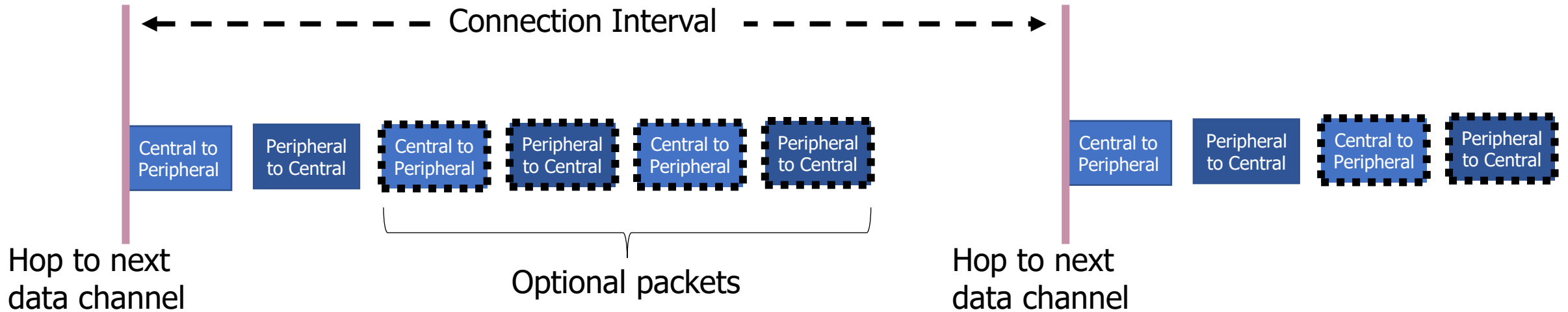
- Some data can be exchanged at each interval
  - Might just be acknowledgements
  - Additional packets can be sent if there is a lot to transmit
  - Each interval is on the next channel in the hopping sequence

- Peripheral can skip a number of intervals to save energy
  - Defined by the Latency connection request parameter

# BLE connection "events" happen at periodic intervals

How do you schedule the very first connection event?

- **WinSize** and **WinOffset** specify start of the <u>first</u> connection event
  - First connection event: first exchange after CONNECT_REQ packet)

- WinOffset: Time from now until first event
  - Analogy: "Let's meet 2 hours from now"

- WinSize: Flexible window of time to expect first transmission
  - Analogy: "I might be up to 15 minutes late"

The Central schedules the first connection event

- Place an "anchor" point that defines the TDMA schedule for this device

- Interval specifies duration between connection events starting at "anchor"

- Combination allows Central to determine schedule for this connection and to avoid other connections it manages
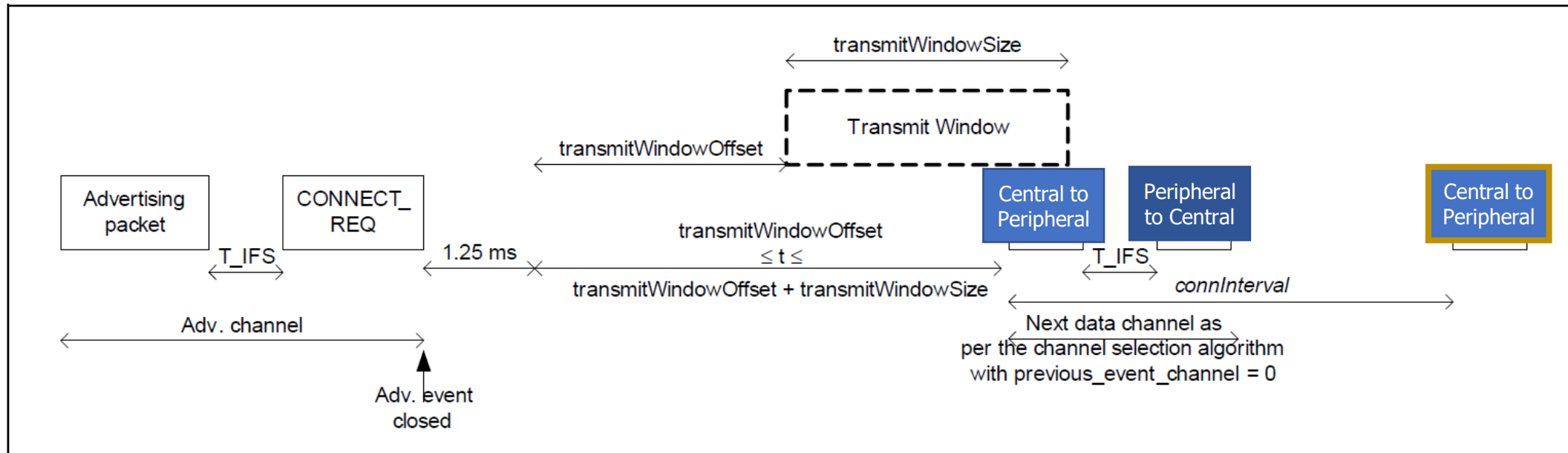
How do you agree on hopping pattern?

LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

- Channel Map (ChM) and Hop define the FHSS pattern

- **Channel Map (ChM)**: Map of which channels to use

- **Hop**: next hop increment
  - Next hop channel $c_{n+1} = (c_n + hop) \mod 37$

Now that a connection exists, packets can be exchanged over it

- Next step: actually communicate between devices
  - Either sending data, or sending "control" messages

- "Control" and "Data" separation: very common networking theme

- LLID – link layer ID
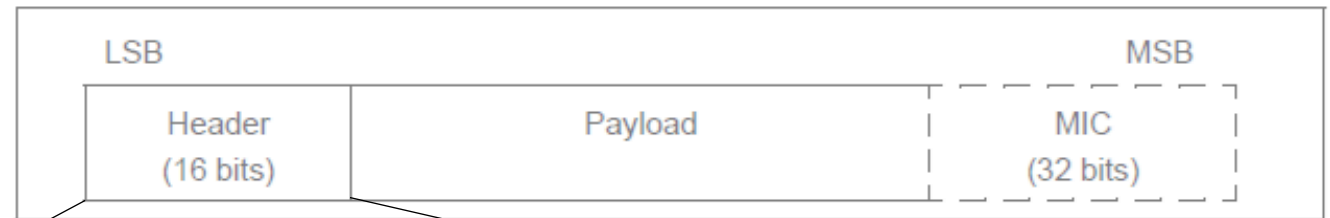 - Empty data / Fragment
  - Data
  - Control

Figure 2.12: Data Channel PDU

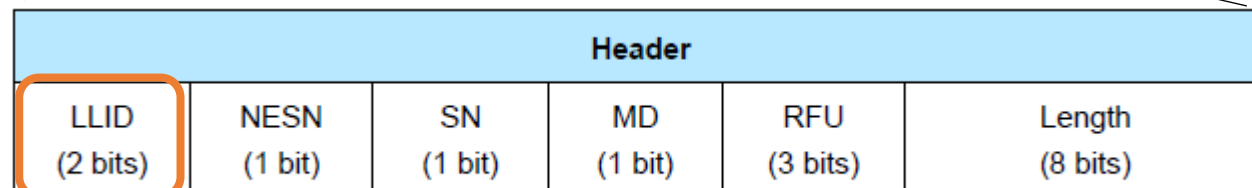| LSB					MSB
Header (16 bits)		Payload			MIC (32 bits)

Header

LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (8 bits)

Figure 2.13: Data channel PDU header

Link Layer header length
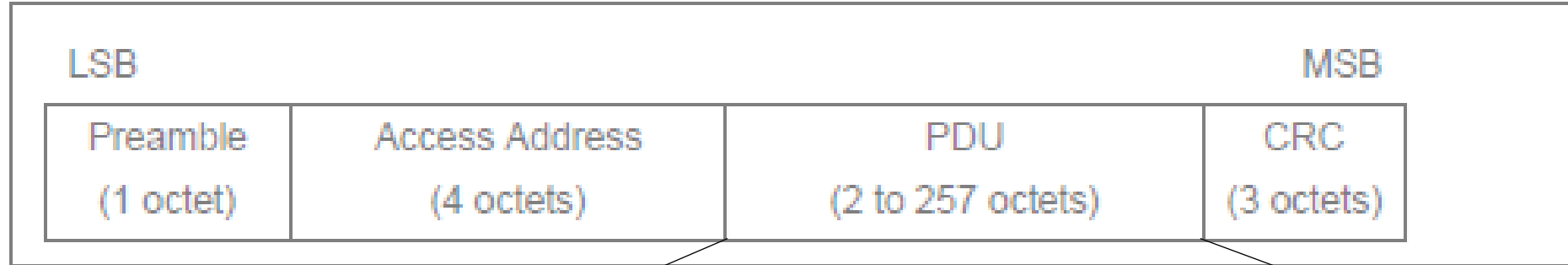
LSB | | | MSB

| Preamble (1 octet) | Access Address (4 octets) | PDU (2 to 257 octets) | CRC (3 octets) |

Figure 2.1: Link Layer packet format

LSB | | MSB

| Header (16 bits) | Payload | MIC (32 bits) |

Figure 2.12: Data Channel PDU

Header

| LLID (2 bits) | NESN (1 bit) | SN (1 bit) | MD (1 bit) | RFU (3 bits) | Length (8 bits) |

Figure 2.13: Data channel PDU header

Question:
What is the maximum possible payload size?

25

Link Layer header length

Figure 2.1: Link Layer packet format
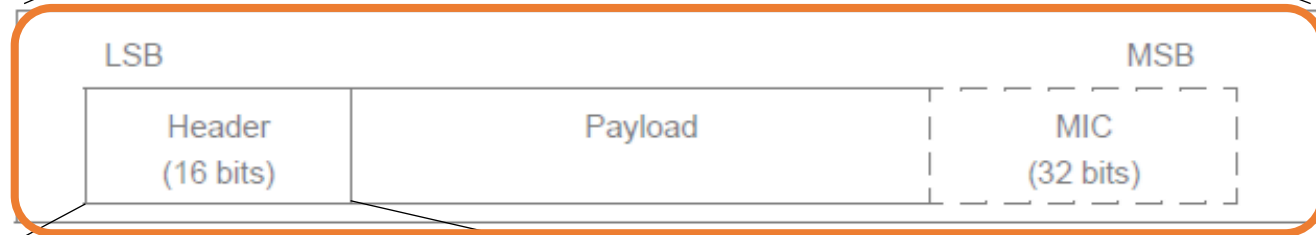
Figure 2.12: Data Channel PDU
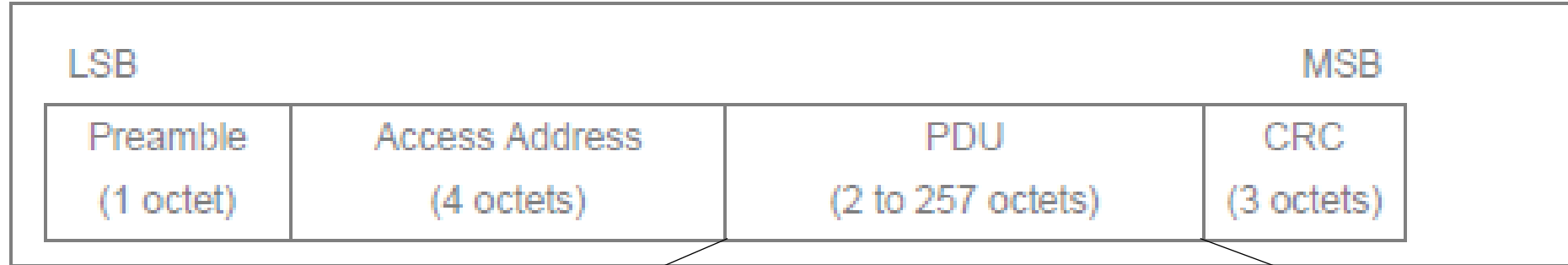
Figure 2.13: Data channel PDU header

Question:
What is the maximum possible payload size?

251 Bytes
= 257 - header - MIC
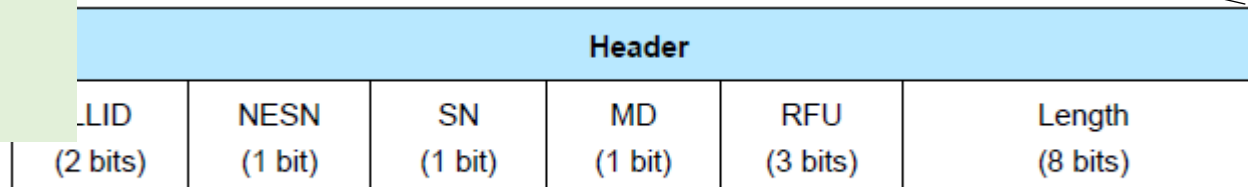
Control payloads

Examples:

- Request update to connection parameters like interval (by peripheral)

- Begin encrypting communication

- Terminate a connection

Note: Control payloads can all be handled by the link layer – not passed up!

Opcode	Control PDU Name
0x00	LL_CONNECTION_UPDATE_REQ
0x01	LL_CHANNEL_MAP_REQ
0x02	LL_TERMINATE_IND
0x03	LL_ENC_REQ
0x04	LL_ENC_RSP
0x05	LL_START_ENC_REQ
0x06	LL_START_ENC_RSP
0x07	LL_UNKNOWN_RSP
0x08	LL_FEATURE_REQ
0x09	LL_FEATURE_RSP
0x0A	LL_PAUSE_ENC_REQ
0x0B	LL_PAUSE_ENC_RSP
0x0C	LL_VERSION_IND
0x0D	LL_REJECT_IND
0x0E	LL_SLAVE_FEATURE_REQ
0x0F	LL_CONNECTION_PARAM_REQ
0x10	LL_CONNECTION_PARAM_RSP
0x11	LL_REJECT_IND_EXT
0x12	LL_PING_REQ
0x13	LL_PING_RSP
0x14	LL_LENGTH_REQ
0x15	LL_LENGTH_RSP

Data payloads

LSB   MSB

| Preamble (1 octet) | Access Address (4 octets) | PDU (2 to 257 octets) | CRC (3 octets) |
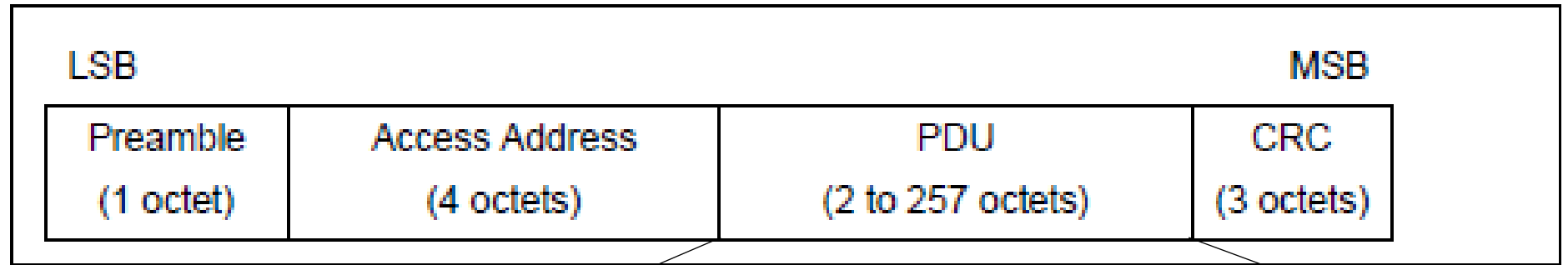
Figure 2.1: Link Layer packet format

SDU – Service Data Unit
- Logical packet that may be split into several physical packets
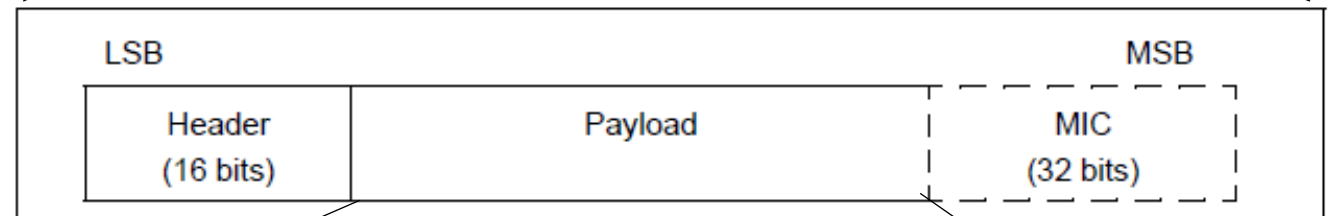
Channel ID
- Destination for data
- ATTributes or Security
- Can also make custom endpoints

LSB   MSB

| Header (16 bits) | Payload | MIC (32 bits) |

Figure 2.12: Data Channel PDU

| Length (16 bits) | Channel ID (16 bits) | L2CAP SDU Length* (16 bits) | Information Payload (0 to 249 octets) |

Very briefly: security in connections

Either side can request secure connection [central -> Pairing; peripheral -> Security Req]

| Pairing Request Opcode (0x01) | IO Capability | OOB Data Flag | AuthReq | Max Encryption Key Size | Initiator Key Distribution | Responder Key Distribution |

Pairing Request Command Format

| Pairing Response Opcode (0x02) | IO Capability | OOB Data Flag | AuthReq | Max Encryption Key Size | Initiator Key Distribution | Responder Key Distribution |

Pairing Response Command Format
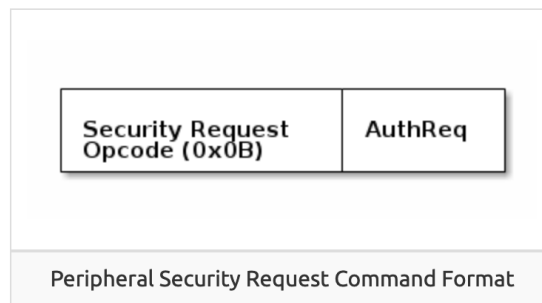
Association Models	Conditions for Use
Just Works	This model is used when there is no Out of Band (OOB) data available and neither device has input capabilities.
Passkey Entry	This model is used when there is no OOB data available, at least one device has the I/O capability to enter a passkey, and the other device has the capability to display a passkey.
Numeric Comparison	This model is used if both devices support Secure Connections, can display a yes or no message and have some input capability.
Out of Band	If OOB data is available on either device, this model will be chosen.

| Security Request Opcode (0x0B) | AuthReq |

Peripheral Security Request Command Format

"Bonding" saves pairing
info between connections

- Excellent TI resource here for more details:
  https://dev.ti.com/tirex/explore/node?node=AOPOY.GDApakIOYjiwoY6A__pTTHBmu__LATEST

Ending connections

- Termination control packet

LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

- Timeout parameter from connection parameters
  - Human-based devices may just wander away from each other
  - Or be shut off, or reprogrammed, etc.

Break + Check your understanding

- What makes connections more reliable than advertisements?

- How does a Peripheral send data to a Central in a connection?

- Why is a "termination" control packet useful?

Break + Check your understanding

- What makes connections more reliable than advertisements?
  - Each packet sent is acknowledged or resent
  - TDMA schedule and FHSS means less likely to collide

- How does a Peripheral send data to a Central in a connection?
  - Wait until next connection interval
  - Respond to packet from Central with a data payload

- Why is a "termination" control packet useful?
  - Timeout could delay for a while
  - Enables re-connection if desired

Outline

- Connection PHY and Link Layer

- **Connection Investigations**

- GATT

- BLE 5

Questions about how a connection "network" works

1. How does TDMA MAC for connections work?
   - Implies a schedule
   - Implies synchronization

2. How many devices can be in a network?

3. How much throughput can a device have?

Questions about how a connection "network" works
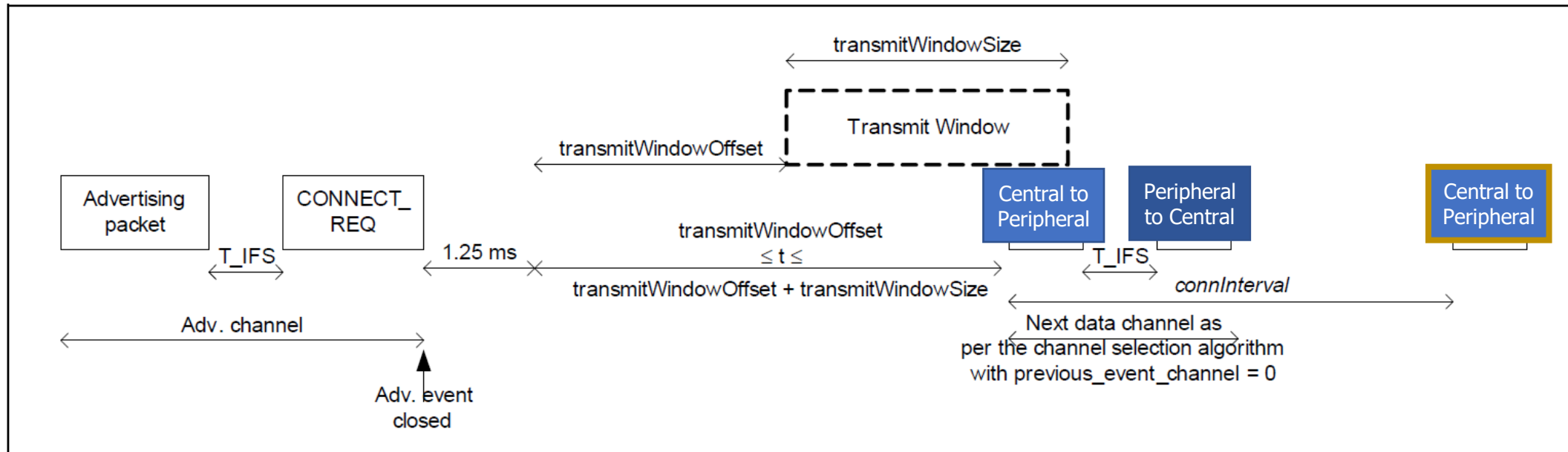
1. **How does TDMA MAC for connections work?**
   - Implies a schedule
   - Implies synchronization

2. How many devices can be in a network?

3. How much throughput can a device have?

How is the TDMA schedule created/managed?

- Only the central needs to know the schedule
  - It controls interactions with the peripheral(s) it is connected to

- Anchor point and connection interval determines the schedule for a specific device

How is synchronization managed?

- Central sends first packet at each connection interval
  - So peripheral must be synchronized to central
  - Resynchronization can occur on each received packet

- Specification describes how a peripheral must widen its listening window based on "Source Clock Accuracy"

$$windowWidening = \left(\frac{centralSCA + peripheralSCA}{1000000} \right) * timeSinceLastAnchor$$

Questions about how a connection "network" works

1. How does TDMA MAC for connections work?
   - Implies a schedule
   - Implies synchronization

2. **How many devices can be in a network?**

3. How much throughput can a device have?

How many devices can be connected?

- We can schedule with granularity of *at least* 1.25 ms (offset steps)
  - That's at least 800 devices per second

- Intervals go up to 4 seconds, so multiply by four
  - That's 3200 devices per max interval

- Plus central can skip intervals on occasion without dropping the connection
  - And really the offset defines a window. More granularity is available

- Answer: thousands of devices
  - Although each is sending minimum-sized packets each interval

How many devices can be connected in the real world?

- The limit is much much lower on real devices
  - Example: Android sets a limit at around 4-15
  - nRF52 s140 softdevice allows up to 20

- Connection management is often done in firmware
  - Softdevice for nRF, firmware on the radio chip in smartphones

- Limited by memory and complexity

Questions about how a connection "network" works

1. How does TDMA MAC for connections work?
   - Implies a schedule
   - Implies synchronization

2. How many devices can be in a network?

3. **How much throughput can a device have?**

How much throughput can a device have?

- **1 Mbps? That's the PHY data rate**

How much throughput can a device have?

- **1 Mbps? That's the PHY data rate**
  - Not even close. Packet overhead plus timing delays

- Step 1: decrease connection interval as much as possible
  - More connection events per second mean more data
  - Range: 7.5 ms to 4.0 s
  - Somewhat device-specific configuration
    - Android allows 7.5 ms
    - iOS allows 15 ms

How much throughput can a device have?
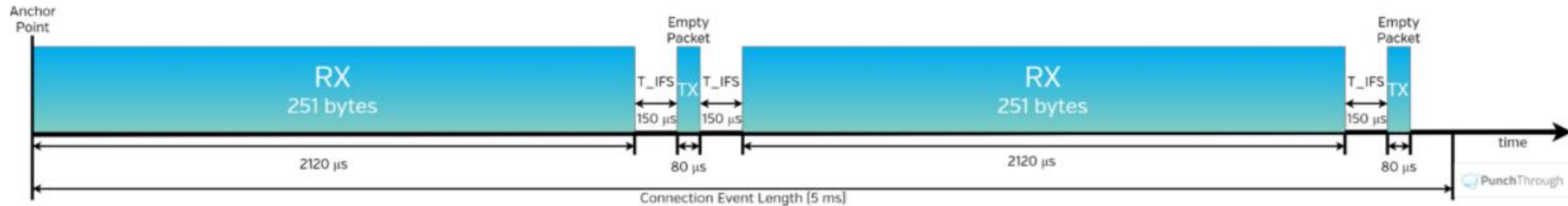
- Step 2, increase packet size

Figure 5 – A single connection event (from the slave's perspective) in a DLE-enabled connection in which the master is transmitting as much data as possible within the effective Connection Event Length
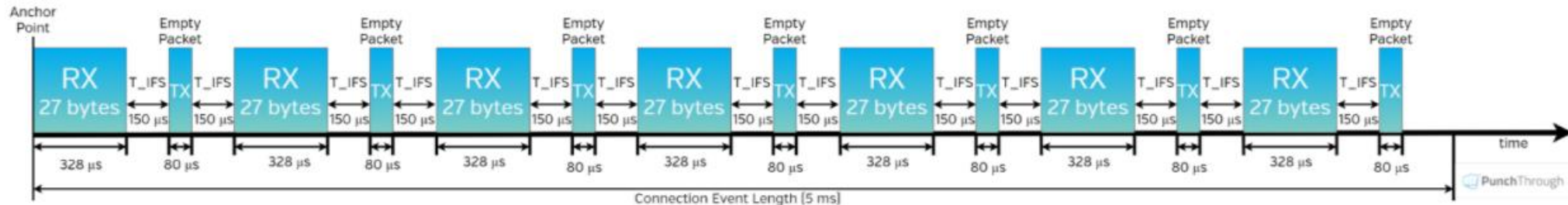

Figure 6 – A single connection event (from the slave's perspective) in a connection without DLE, in which the master is transmitting as much data as possible within the effective Connection Event Length

How much throughput can a device have?

- 488 bytes per connection event
  - Maximum sized packets, discounting headers and timing delays

- Connection event every 7.5 ms

- Result: 520 kbps (65 kilobytes per second)
  - iOS result 260 kbps
  - Original BLE 4.1 result on Android: 128 kbps
  - A little lower in practice due to lost packets
  - Modern results will be better, but a similar ballpark

Outline

- Connection PHY and Link Layer

- Connection Investigations

- **GATT**

- BLE 5

How is data actually exchanged between central and peripheral?

Central
ATT Client

Attribute Protocol

Peripheral
ATT Server
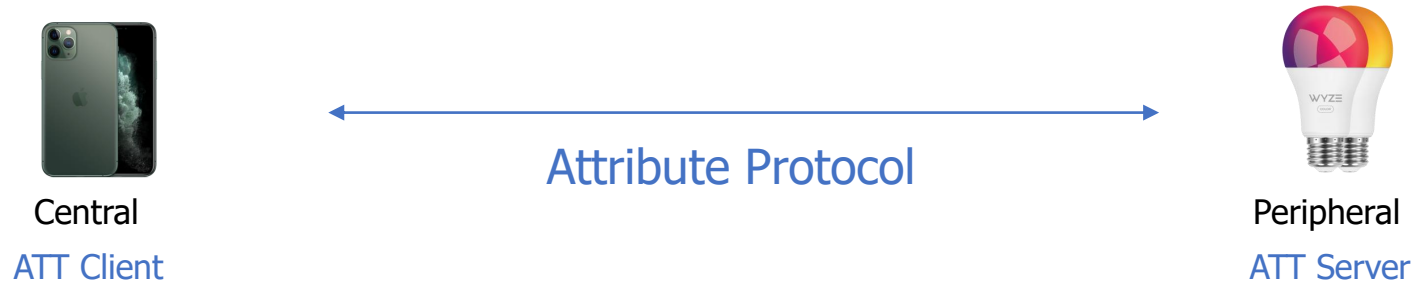
- Attribute Protocol (ATT) and Generic Attribute Profile (GATT) used for this!

- ATT is the underlying framework for GATT
  - Two roles: Client and Server

- Data that the server exposes is structured as **attributes**

Attribute server keywords

- Characteristic
  - A field with properties and a value
  - Descriptor: metadata about the characteristic (readable, writeable, notifications when it updates)

- Service
  - Collection of characteristics

- Profile
  - Collection of services
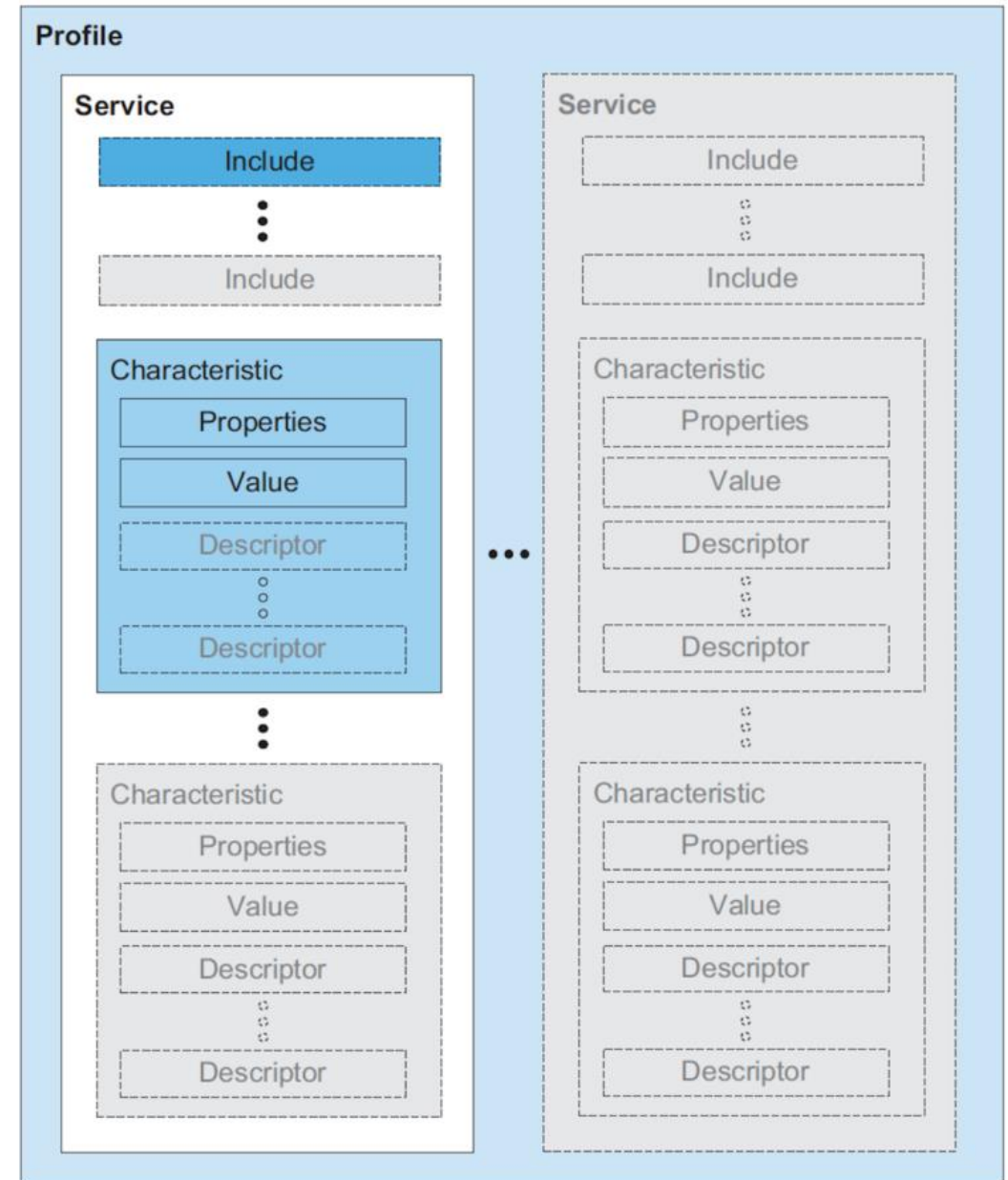
Overview of Generic Attribute Profile (GATT)
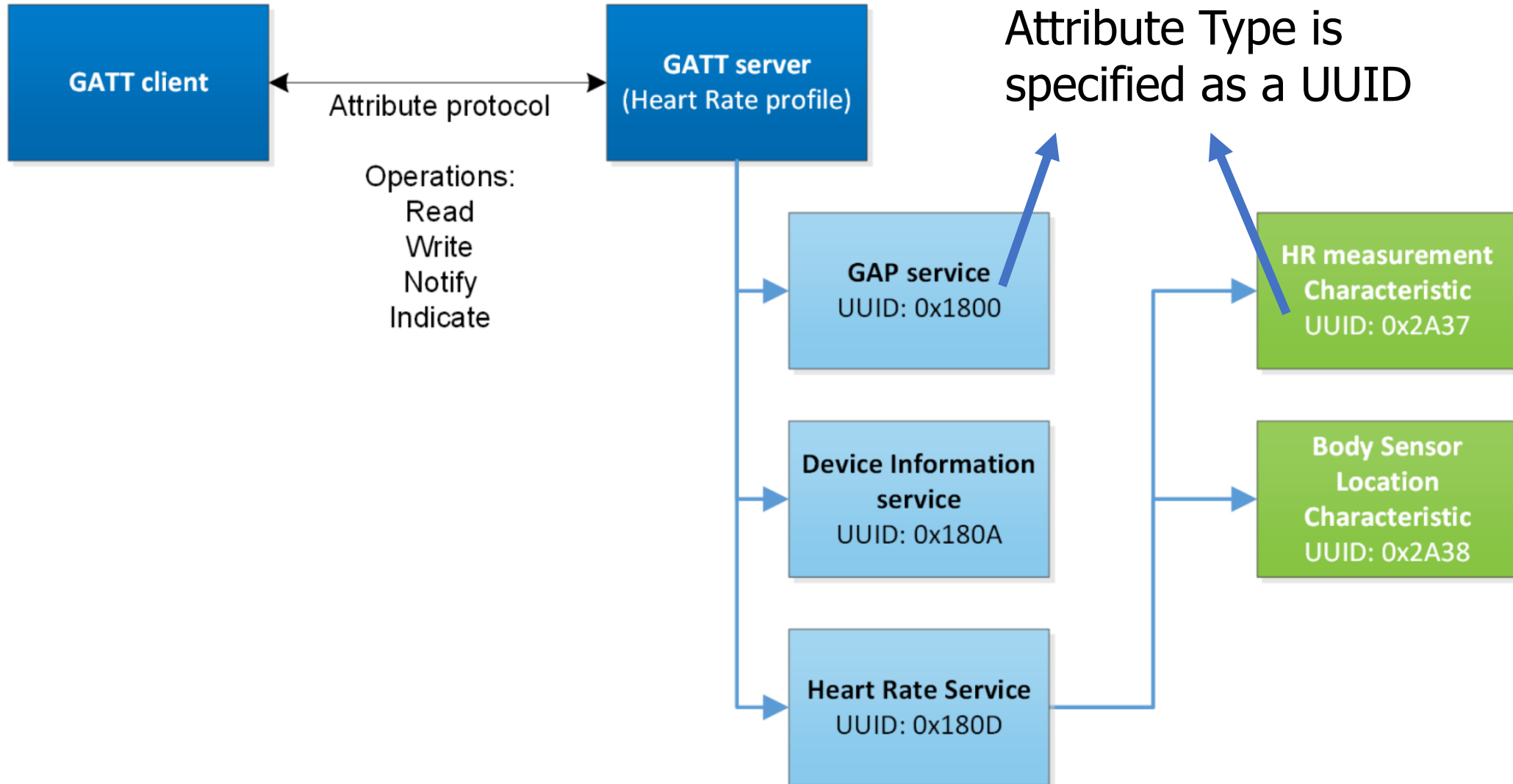
UUIDs and handles

- Universally Unique Identifiers
  - 128-bit, mostly random with a few bits for versioning
  - Example: 00000000-0000-1000-8000-00805F9B34FB
    - This is the default BLE UUID for *known* services
    - You can generate your own UUID for custom services

- Handles
  - Too long to pass around all the time, so pick 16 bits that mean that UUID
    - Must be unique among services/characteristics on that device
  - Taken from UUID: 0000**xxxx**-0000-1000-8000-00805F9B34FB
  - Handle often sequentially incremented for each new characteristic within a service

Resource: Useful post on identifying handles in the wild

Example: Time Profile
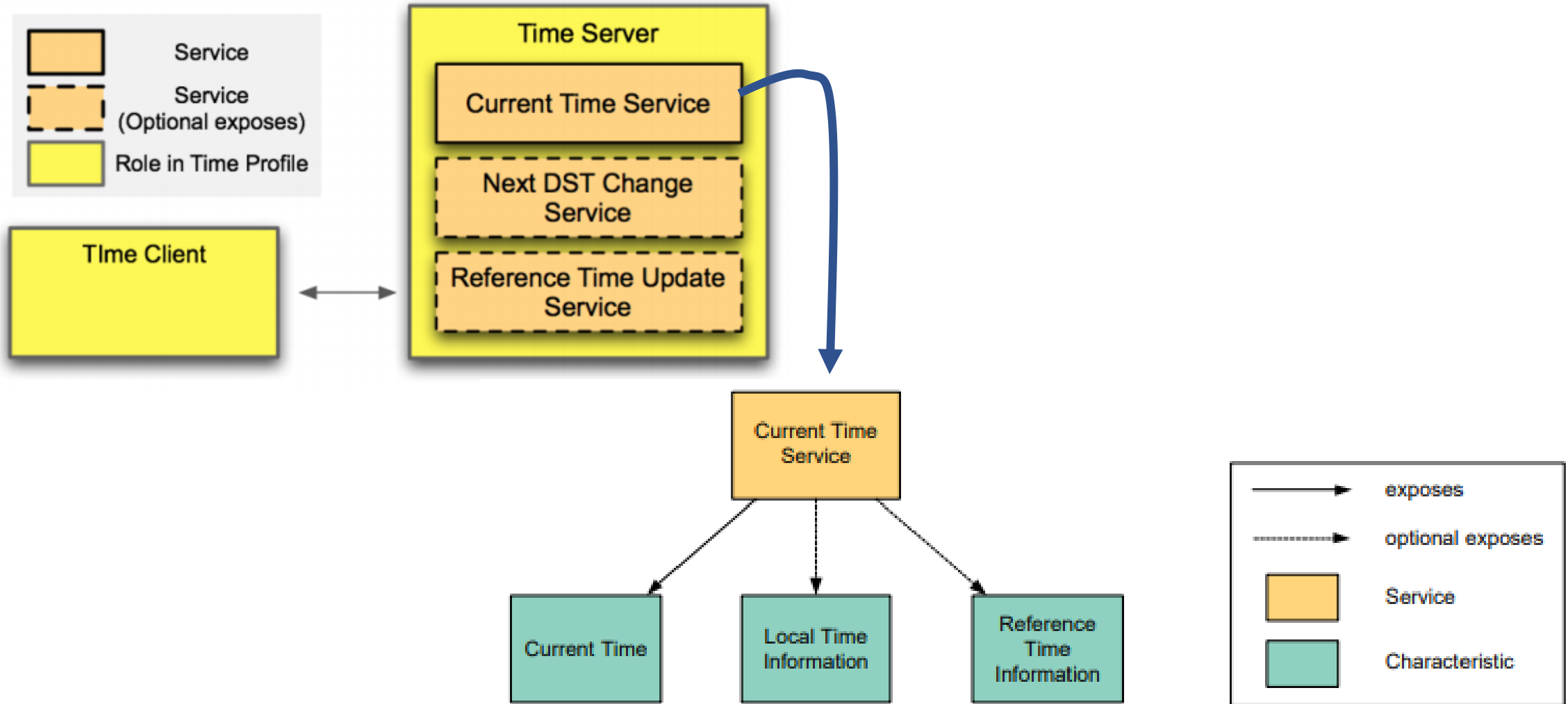
Current time characteristic

Field	Data Type	Size (in octets)	Description
Exact Time 256	struct	9	Refer to the Exact Time 256 characteristic in Section 3.64
Adjust Reason	uint8	1	See Section 3.58.2.1

Bit	Bit Name
0	Manual Time Update
1	External Reference Time Update
2	Change of Time Zone
3	Change of DST
4–7	Reserved for Future Use

Field	Data Type	Size (in octets)	Description
Day Date Time	struct	8	Refer to the Day Date Time characteristic in Section 3.54.
Fractions256	uint8	1	The number of 1/256 fractions of a second. Valid range 0–255.

Field	Data Type	Size (in octets)	Description
Date Time	struct	7	Refer to the Date Time characteristic in Section 3.53
Day of Week	struct	1	Refer to the Day of Week characteristic in Section 3.55

Field	Data Type	Size (in octets)	Description
Year	uint16	2	Year as defined by the Gregorian calendar. Valid range 1582 to 9999. A value of 0 means that the year is not known. All other values are reserved for future use (RFU).
Month	uint8	1	Month of the year as defined by the Gregorian calendar. Valid range 1 (January)

The Bluetooth SIG are pedantic people

Interacting with characteristics

- Depends on their permissions
  - Readable, Writable, Notify-able, etc.

- Notify
  - Automatically get a message sent whenever the characteristic value updates
  - Note: have to enable this on both sides, it's not the default behavior

- Long characteristics are automatically fragmented across multiple packets and/or connection events
  - Lower layers are in charge of this without the application having to consider it

Discovery

- When a connection first occurs, each device can query the other for a list of services
  - And can further query for a list of characteristics in that service
  - Gets a list of handles/UUIDs

- Standardized UUIDs can be interpreted immediately
  - Custom services/characteristics need documentation from the manufacturer

Documentation of GATT standards

- https://www.bluetooth.com/specifications/gatt/
  - Various profiles and services that have been standardized

- GATT Specification Supplement
  - Various characteristic definitions that have been standardized

- Both incredibly specific and woefully inexhaustive

Break + Example Services

Allocated UUID	Allocated for
0x1803	Link Loss
0x1804	Tx Power
0x1805	Current Time
0x1806	Reference Time Update
0x1807	Next DST Change
0x1808	Glucose
0x1809	Health Thermometer
0x180A	Device Information
0x180D	Heart Rate
0x180E	Phone Alert Status
0x180F	Battery
0x1810	Blood Pressure
0x1811	Alert Notification
0x1812	Human Interface Device
0x1813	Scan Parameters

Allocated UUID	Allocated for
0x1814	Running Speed and Cadence
0x1815	Automation IO
0x1816	Cycling Speed and Cadence
0x1818	Cycling Power
0x1819	Location and Navigation
0x181A	Environmental Sensing
0x181B	Body Composition
0x181C	User Data
0x181D	Weight Scale
0x181E	Bond Management
0x181F	Continuous Glucose Monitoring
0x1820	Internet Protocol Support
0x1821	Indoor Positioning
0x1822	Pulse Oximeter
0x1823	HTTP Proxy
0x1824	Transport Discovery
0x1825	Object Transfer
0x1826	Fitness Machine

https://btprodspecificationrefs.blob.core.windows.net/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf

Outline

- Connection PHY and Link Layer

- Connection Investigations

- GATT

- **BLE 5**

Changes in BLE 5

- Major changes
  - Multiple physical layers
  - Advertising extensions
  - Localization extensions (will discuss later with localization)

- Minor changes
  - Various quality of life improvements
  - Examples:
    - Advertise on channels in any order
   - Better data channel hopping algorithm

Revised physical layers

- 2 Mbps PHY
  - Transmit data faster
  - Transmit more data in the same time

- Coded PHY
  - Forward Error Correction in the data stream
    - 1 bit -> 2 symbols or 8 symbols
  - Makes bits more reliable -> longer distance
  - 500 kbps and 125 kbps modes

- Connections can switch to these PHYs after creation

- Advertisements can use these, but only in a special "extended" mode
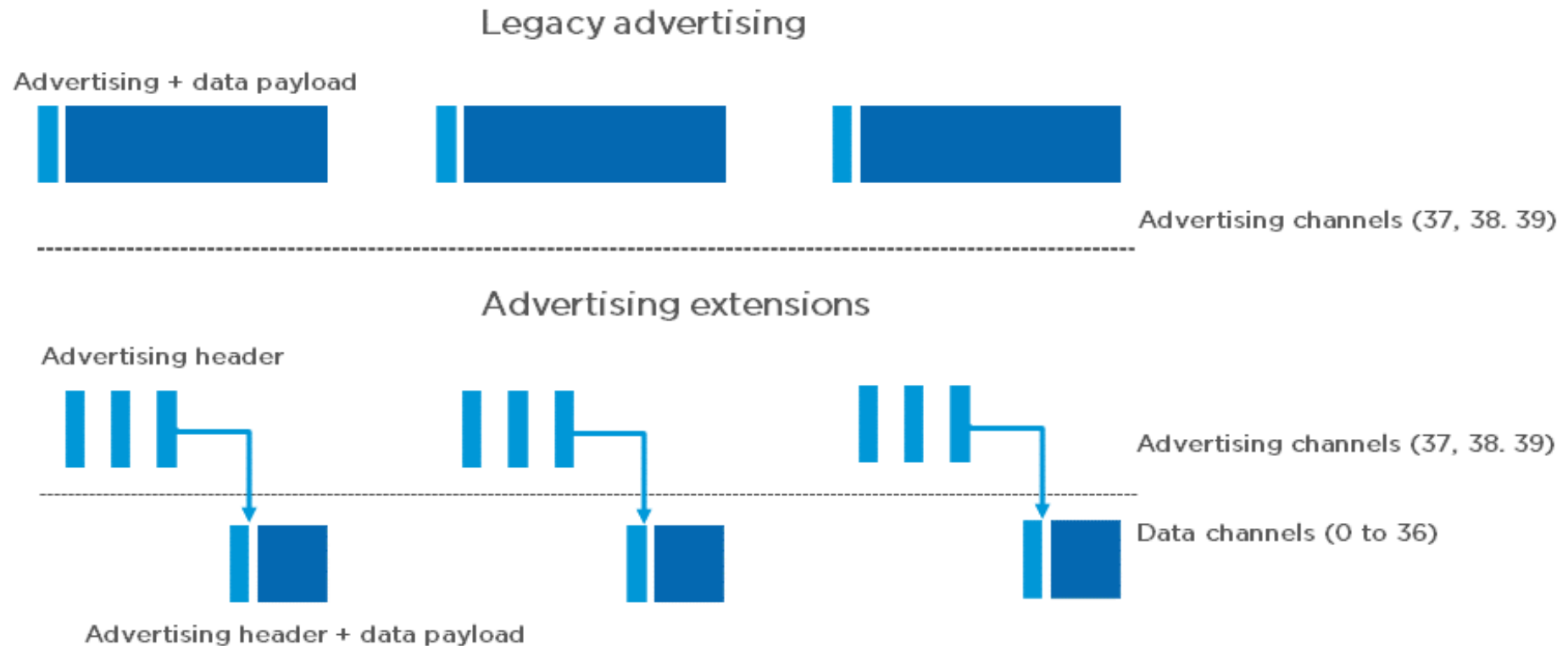
Coded PHY mixes physical and link layers

- Different PHY settings at different times
  - Make beginning headers extra-reliable
  - Data might be slightly less reliable as a trade for faster speed
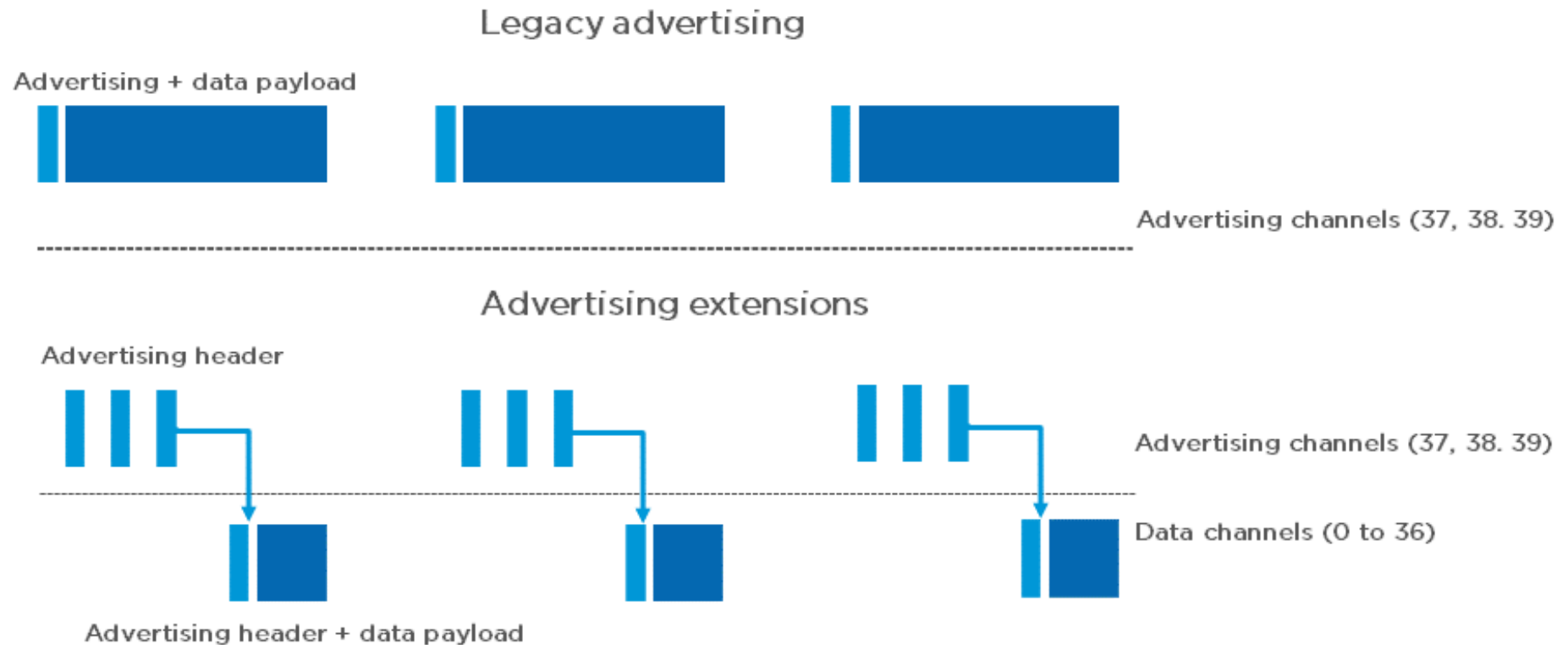
Packet sent with coded PHY
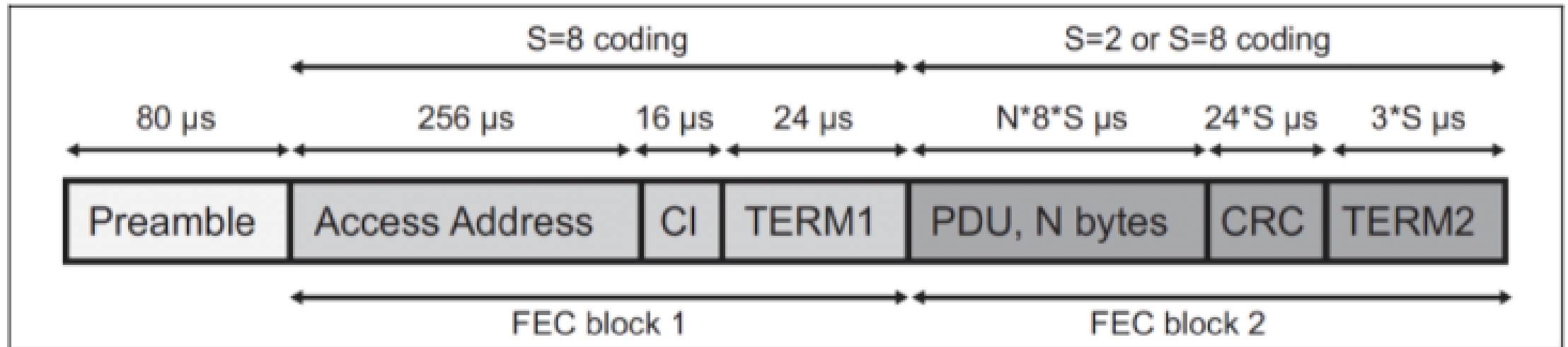
Extended advertising

- Allow bigger payloads and/or different PHYs
  - Uses Data Channels to do so. **Why?**
  - Regular advertisements point to extended advertisements

Legacy advertising

Advertising + data payload

Advertising channels (37, 38. 39)

Advertising extensions

Advertising header

Advertising channels (37, 38. 39)

Data channels (0 to 36)
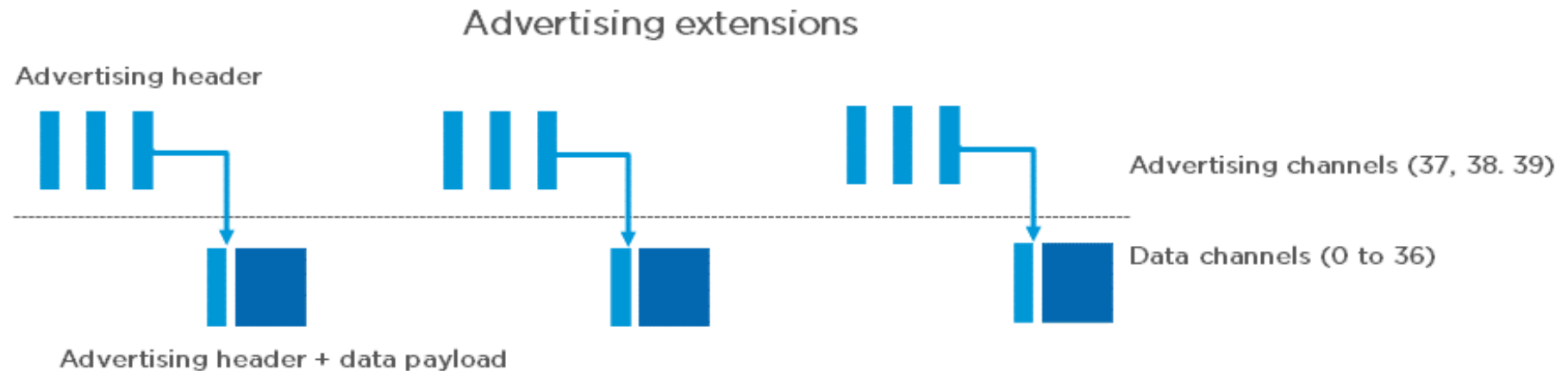
Advertising header + data payload

Extended advertising

- Allow bigger payloads and/or different PHYs
    - Uses Data Channels to do so. **Why?   Packet collisions!**
    - Regular advertisements point to extended advertisements

Legacy advertising

Advertising + data payload

Advertising channels (37, 38. 39)

Advertising extensions

Advertising header

Advertising channels (37, 38. 39)

Data channels (0 to 36)

Advertising header + data payload

Procedure for scanning extended advertisements

1. Scan on 3 primary channels for advertising packets.

2. If ADV_EXT_IND is scanned, record the secondary channel information (which channel and when etc.)

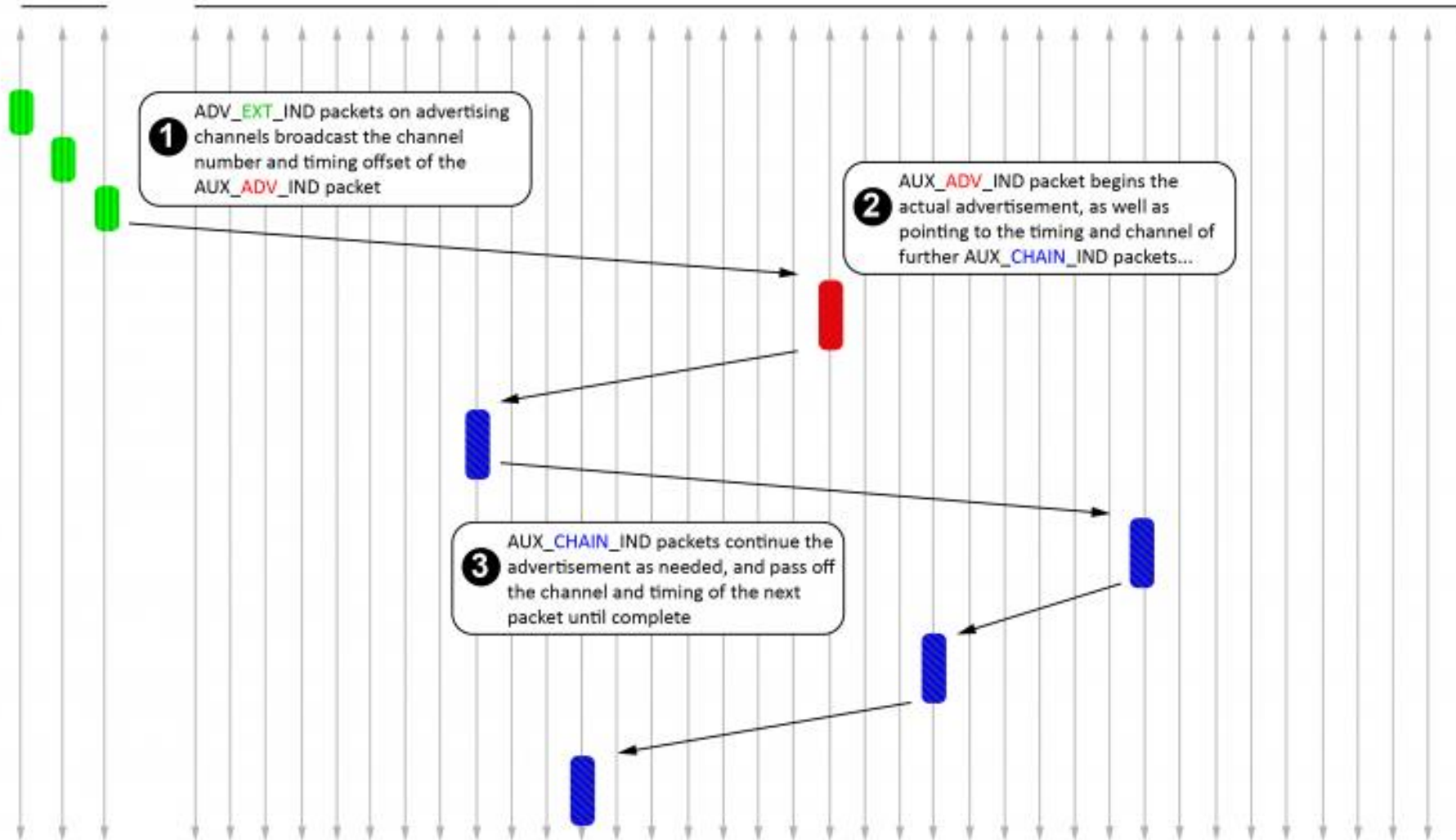3. Scan the specific secondary channel at the given time.

Advertising extensions

Advertising header

Advertising channels (37, 38. 39)

Data channels (0 to 36)

Advertising header + data payload

Extended advertisement train on data channels

Bluetooth 5 Advert Extensions - Overview

Laird

3 Advertisement Channels

37 Data Channels

1. ADV_EXT_IND packets on advertising channels broadcast the channel number and timing offset of the AUX_ADV_IND packet

2. AUX_ADV_IND packet begins the actual advertisement, as well as pointing to the timing and channel of further AUX_CHAIN_IND packets...

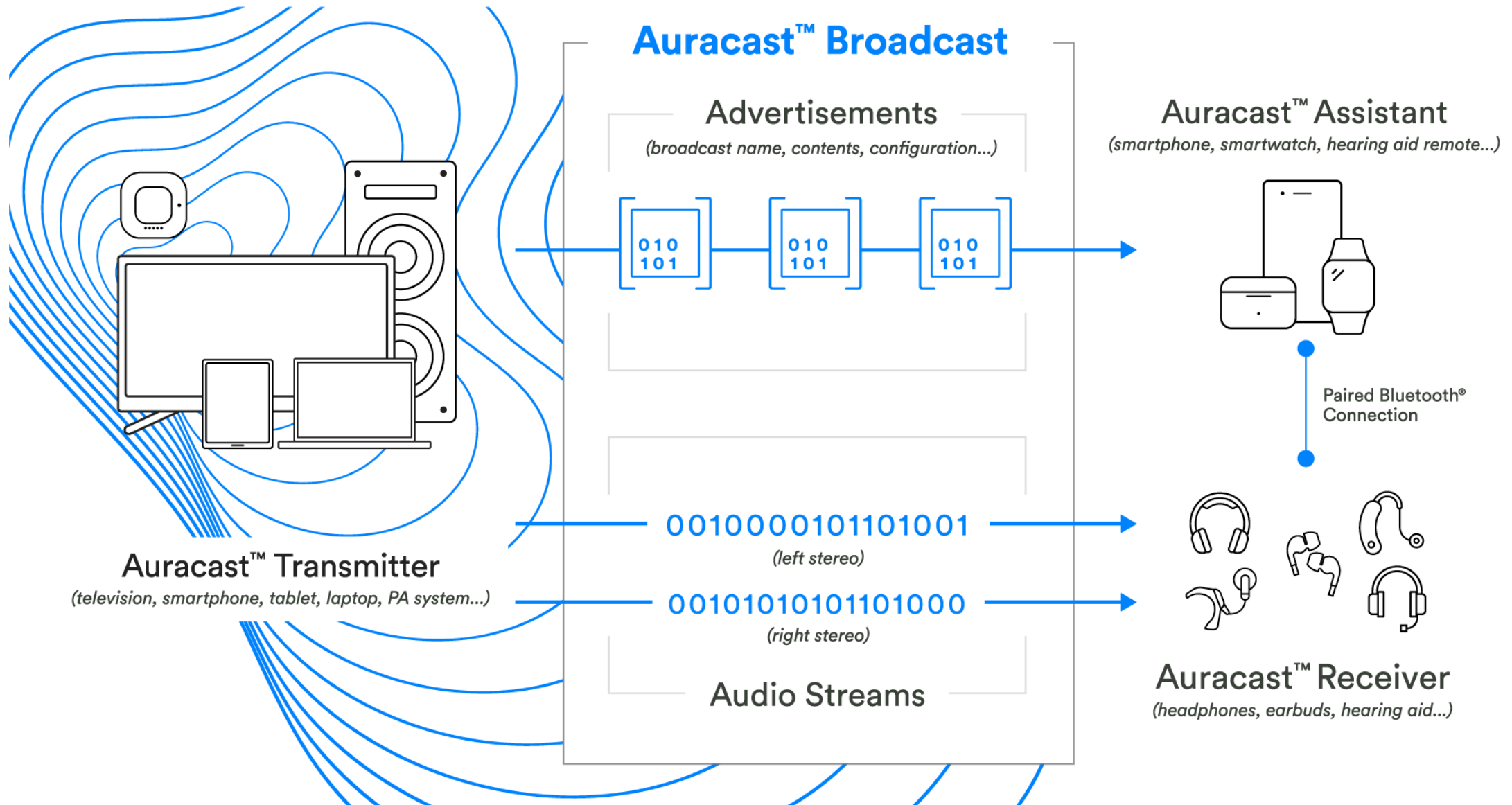3. AUX_CHAIN_IND packets continue the advertisement as needed, and pass off the channel and timing of the next packet until complete

Use case:
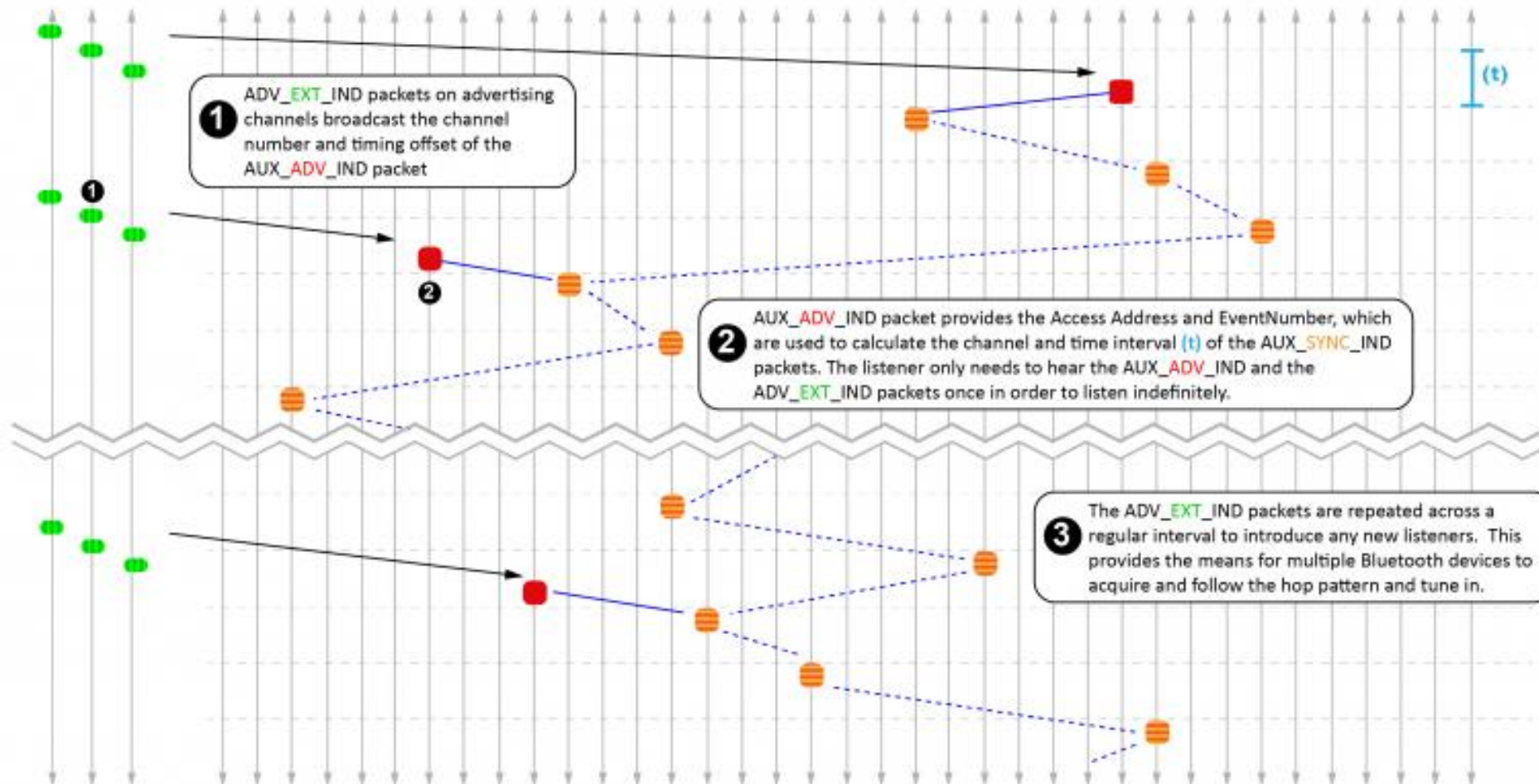long advertisements that need to be fragmented

Up to 1650 bytes total

BLE "Auracast"

Periodic advertising on data channels

Bluetooth 5 Advert Extensions - Sync Packets (e.g. Compressed Audio)
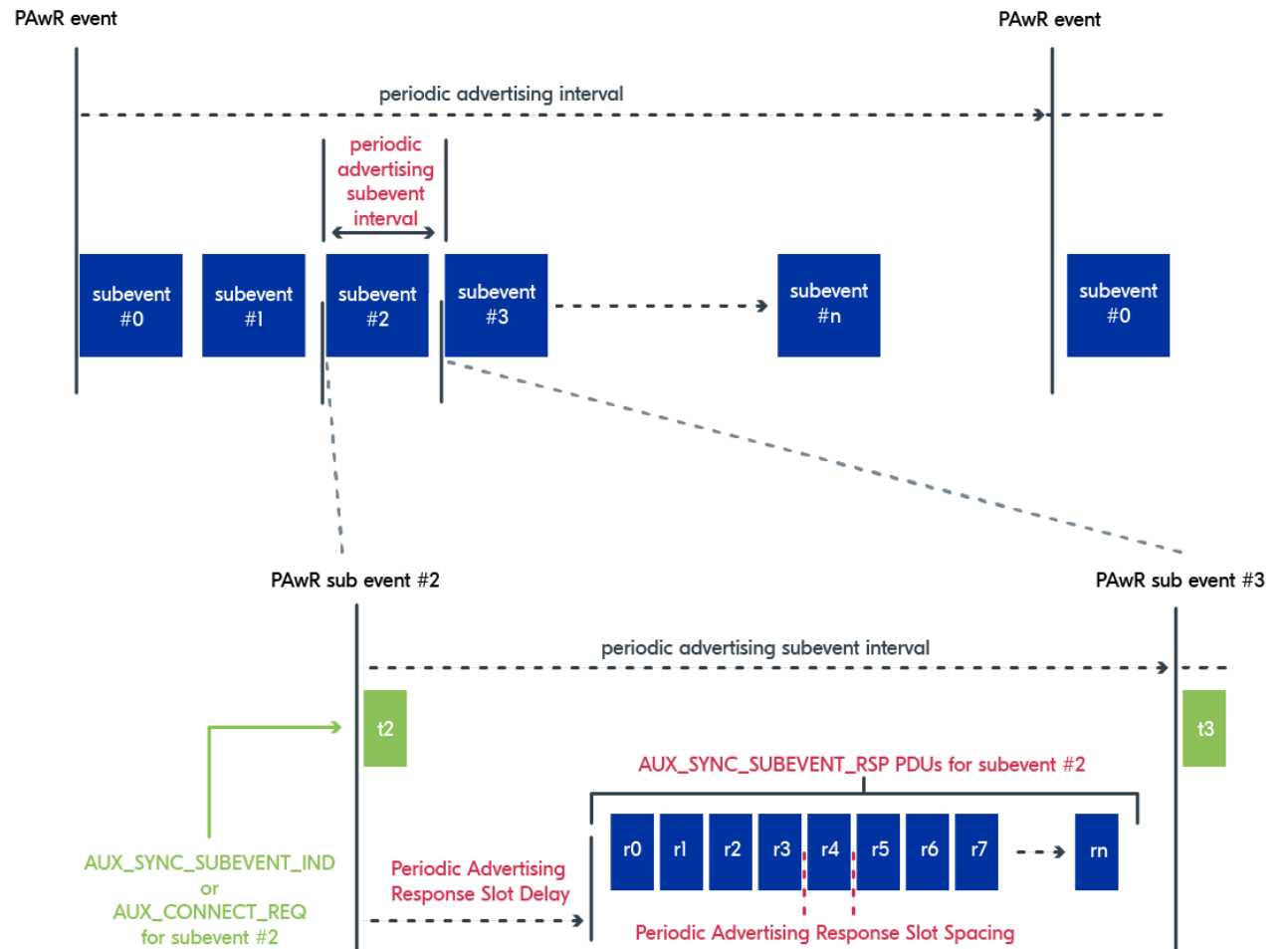
Use case: publicly-available localized audio sources

- TV in a bar
- Commentary in a museum

Broadcast analogy of a connection

Probably only able to follow one of these at a time

Bi-directional "Periodic Advertisement with Responses"

- BLE 5.4 addition

- Between advertisements send a "subevent indication" packet, followed by a number of reception windows
  - A scanner can transmit back to advertiser during one of those windows

  - Up to the application to determine which window each scanner should use

Outline

- Connection PHY and Link Layer

- Connection Investigations

- GATT

- BLE 5