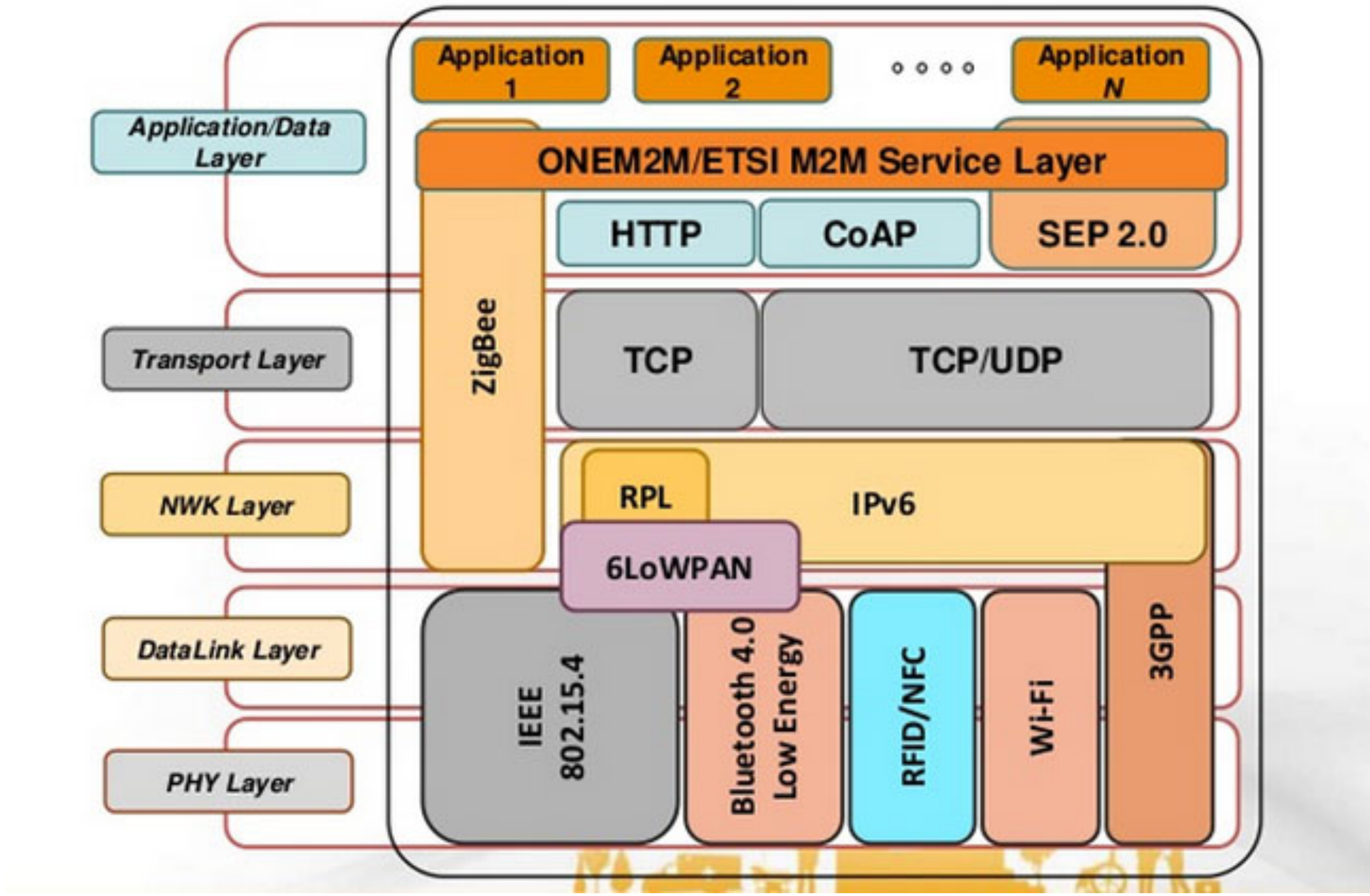
IoT Internet Protocols

黃仁竑 教授
國立中正大學資工系

Outline

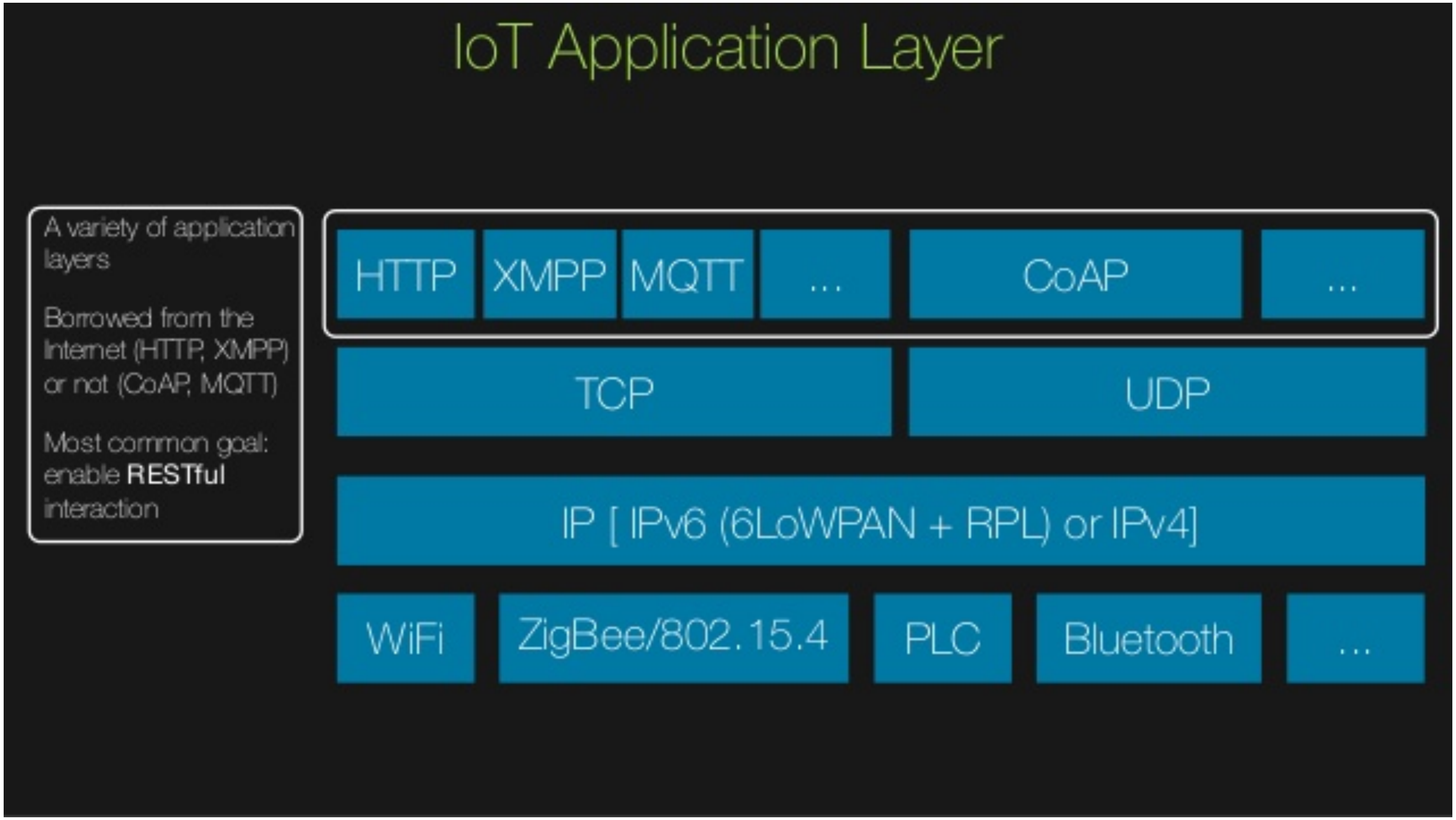
- IPv6
- 6LoWPAN
- RPL
- CoAP
- MQTT
- XMPP

IoT Protocol Stack



Source: https://www.postscapes.com/internet-of-things-protocols/
https://www.slideshare.net/butler-iot/butler-project-overview-13603599

IoT Protocol Stack



Source: https://www.slideshare.net/MarcoPicone/the-android-platform-in-the-era-of-internet-of-things-droidcon-italy-2014

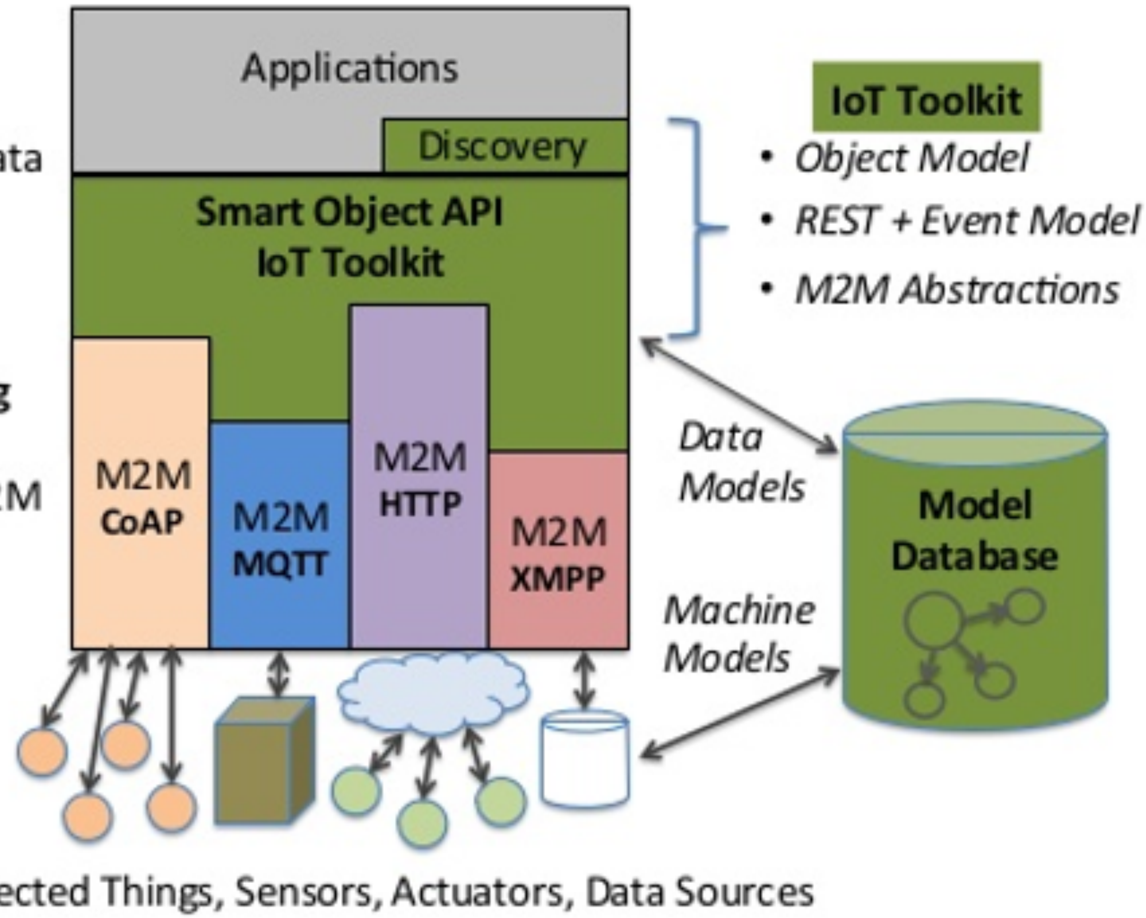
IoT vs. Internet Protocol Stack

TCP/IP Model	**IoT Stack**		**Web Stack**
	IoT Applications	Device Management	Web Applications
Data Format	Binary, JSON, CBOR		HTML, XML, JSON
Application Layer	CoAP, MQTT, XMPP, AMQP		HTTP, DHCP, DNS, TLS/SSL
Transport Layer	UDP, DTLS		TCP, UDP
Internet Layer	IPv6/IP Routing		IPv6, IPv4, IPSec
	6LoWPAN		
Network/Link Layer	IEEE 802.15.4 MAC		Ethernet (IEEE 802.3), DSL, ISDN, Wireless LAN (IEEE 802.11), Wi-Fi
	IEEE 802.15.4 PHY / Physical Radio		

Source https://www.linkedin.com/pulse/emerging-open-standard-protocol-stack-iot-aniruddha-chakrabarti

IoT 2.0 Interoperability

- Easy to deploy new things and applications using data models
- Write once, run anywhere software
- **Any app** to **any thing** via **any M2M**, use-case appropriate M2M
- Network effect enabled



Source: https://www.slideshare.net/michaeljohnkoster/opensourcestackforiot-131017193902phpapp02

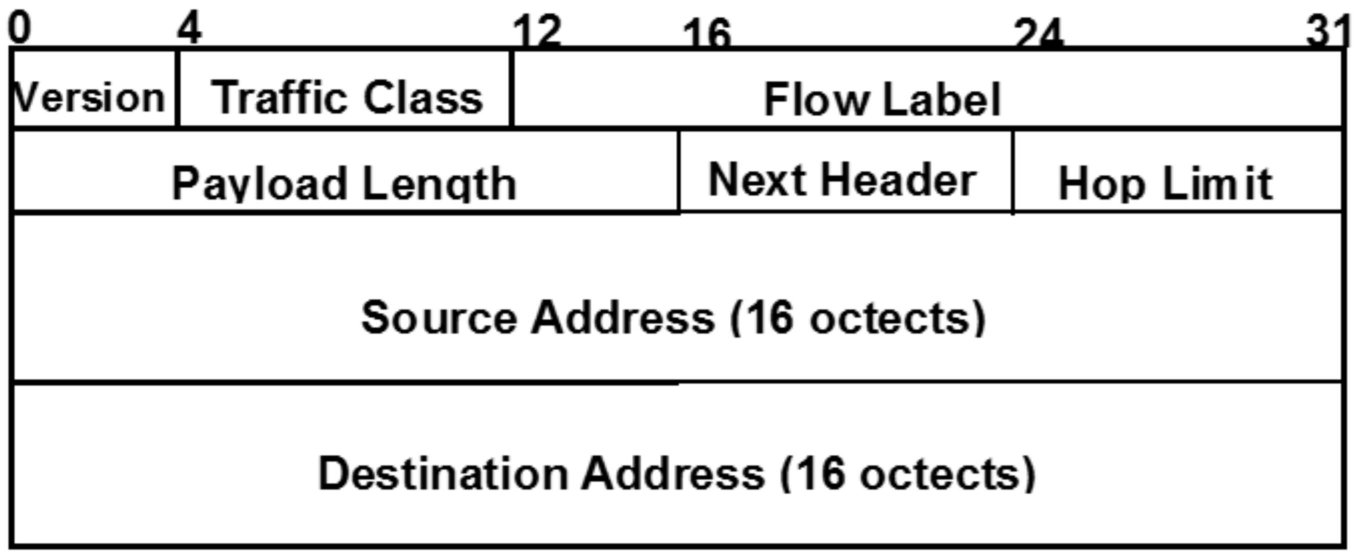
Internet Protocol Version 6 (IPv6)

Ren-Hung Hwang

IPv6

- Problems with IPv4
  - Shortage of address space
  - Lack of Quality of Service guarantee
- New features of IPv6
  - Enlarge address space
  - Fixed header format helps speed processing/forwarding
  - Better support for Quality of Service
  - Neighbor discovery and Auto-configuration
  - Hierarchical address architecture (improved address aggregation)
  - new “anycast” address: route to “best” of several replicated servers

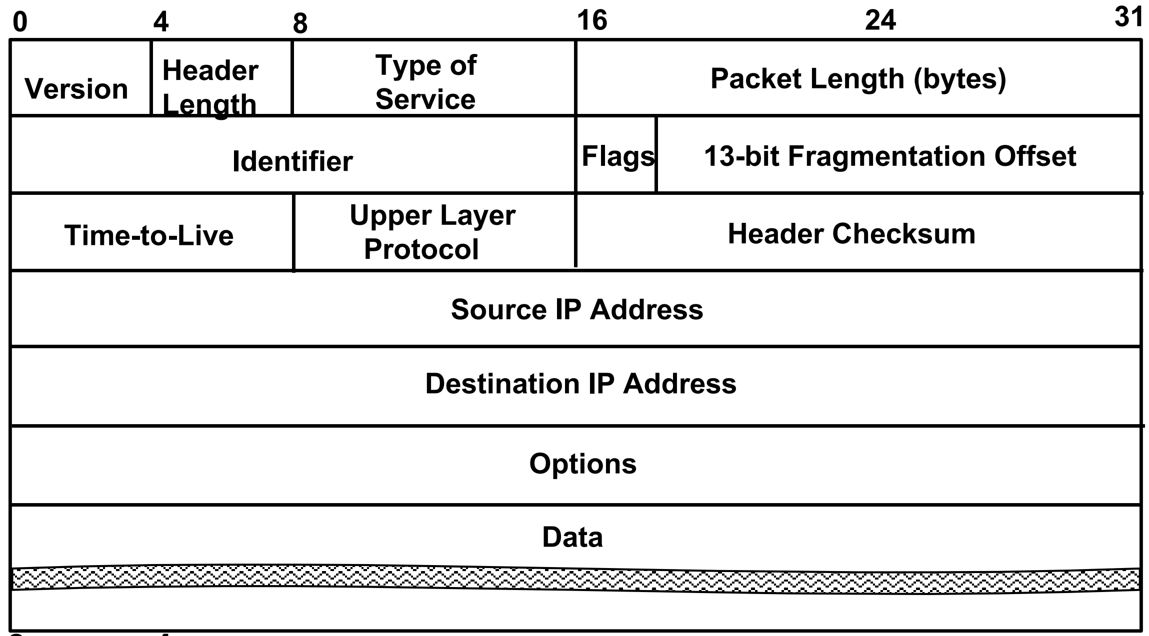
IPv6 Header

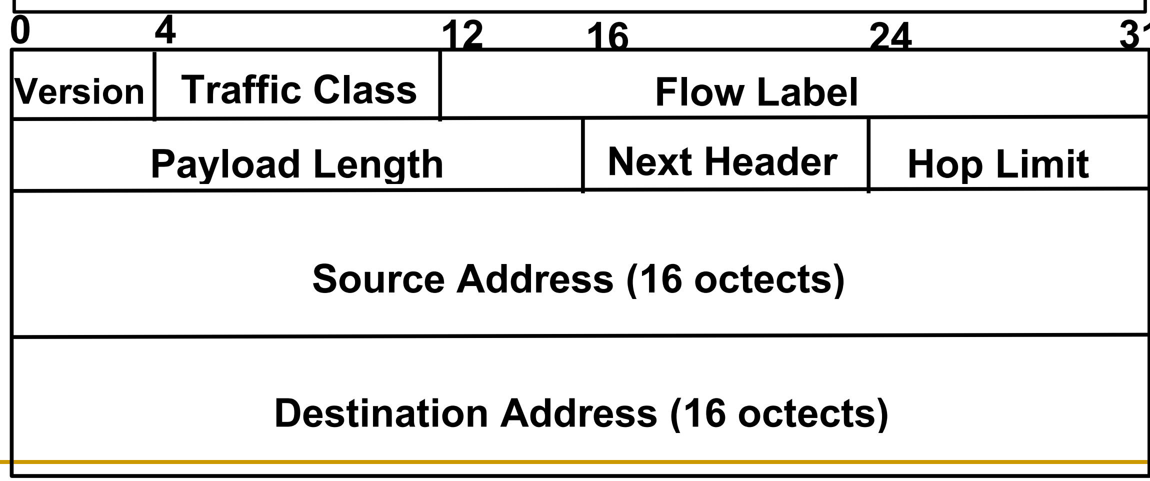

0
4
12
16
24
31
Version
Traffic Class
Flow Label
Payload Length
Next Header
Hop Limit
Source Address (16 octects)
Destination Address (16 octects)

IPv6 Header

- Version: 6
- Traffic class:
  - identify class of service
  - E.g., DiffServ (DS codepoint)
    - The 6 most-significant bits are used for DSCP
- Flow Label:
  - identify datagrams in same “flow”
- Next header:
  - identify upper layer protocol for data

Changes from IPv4 (1/3)





Changes from IPv4 (2/3)

- Expanded Addressing Capabilities
  - from 32 bits to 128 bits (more level and nodes)
  - improve multicast routing ("scope" field)
  - "anycast address": send a packet to any one of a group of nodes
- Header Format Simplification
  - reduce bandwidth cost
- Extensions
  - more flexibility

Changes from IPv4 (3/3)

- Checksum
  - removed to reduce processing at routers
- Fragmentation
  - Not allowed at intermediate routers

6LoWPAN(RFC 6282): IP on IEEE 802.15.4 Low-Power Wireless Networks

黃仁竑 教授
國立中正大學資工系

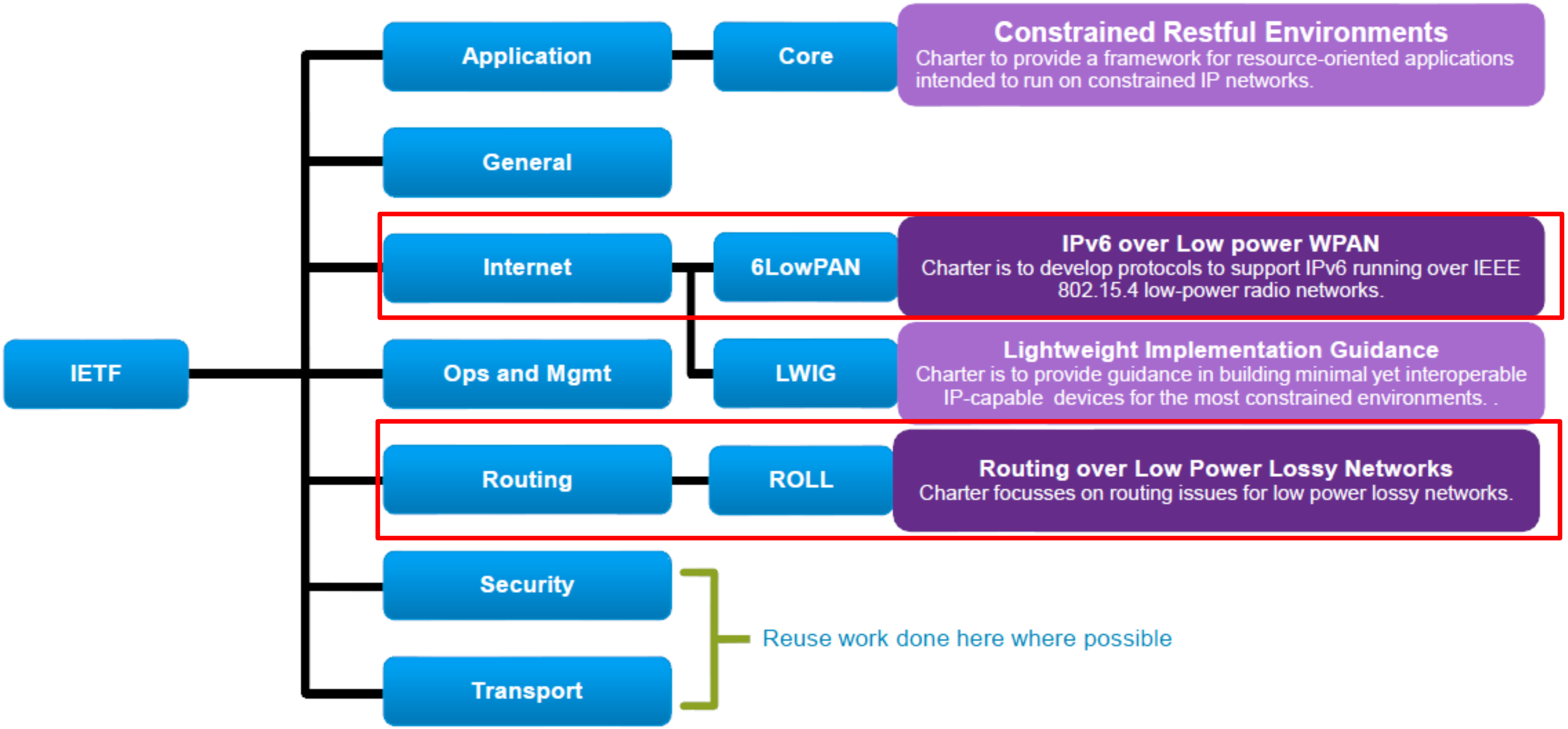
Outline

- What is 6LoWPAN?
- Motivation and Goal
- Key Elements
- Topology
- 6LoWPAN Adaptation Layer

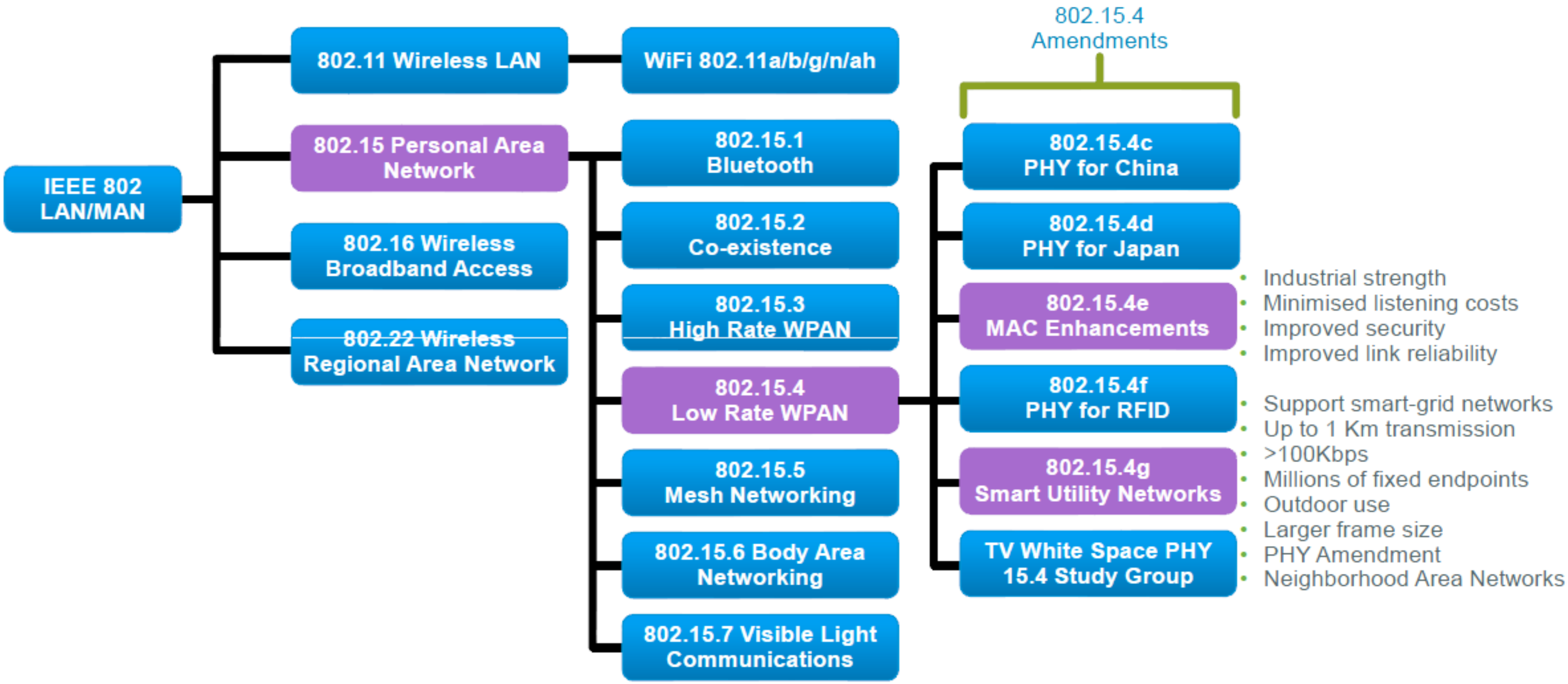
What is 6LoWPAN?

- 6LoWPAN is an acronym of IPv6 over Low power Wireless Personal Area Networks.
- It is designed by the 6LoWPAN working group in IETF (Internet Engineering Task Force).
- RFC 4919 (6LoWPAN Overview, Assumptions, Problem Statement, and Goals) included a detailed review of requirements, which were released in 2007.

IETF Low Power Lossy Network Related Working Groups



IEEE Wireless Standards



6LoWPAN WG Documents

draft-ietf-6lowpan-btle-11	Transmission of IPv6 Packets over BLUETOOTH Low Energy	2012-10-12
RFC 4919	IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals	2007-08
RFC 4944*	Transmission of IPv6 Packets over IEEE 802.15.4 Networks	2007-09
RFC 6282	Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks	2011-09
RFC 6568	Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)	2012-04
RFC 6606	Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing	2012-05
RFC 6775	Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)	2012-11 new

*RFC 4944 (Proposed Standard) Updated by RFC 6282, RFC 6775

6LoWPAN WG Documents

RFC 7388	Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)	2014-10
RFC 7400	6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)	2014-11
RFC 8025	IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch	2016-11
RFC 8138	IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header	2017-4
RFC 8180	Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration	2017-5

Motivation

- Traditionally, battery-powered networks or low-bitrate networks, such as most fieldbus networks or 802.15.4 were considered **incapable of running IP**.
- In the home and industrial automation networks world, the situation compares to the situation of corporate LANs in the **1980**s:
  “should I run Token-Ring, ATM or IPX/SPX?” translates to “should I run ZigBee, LON or KNX?”

Motivation

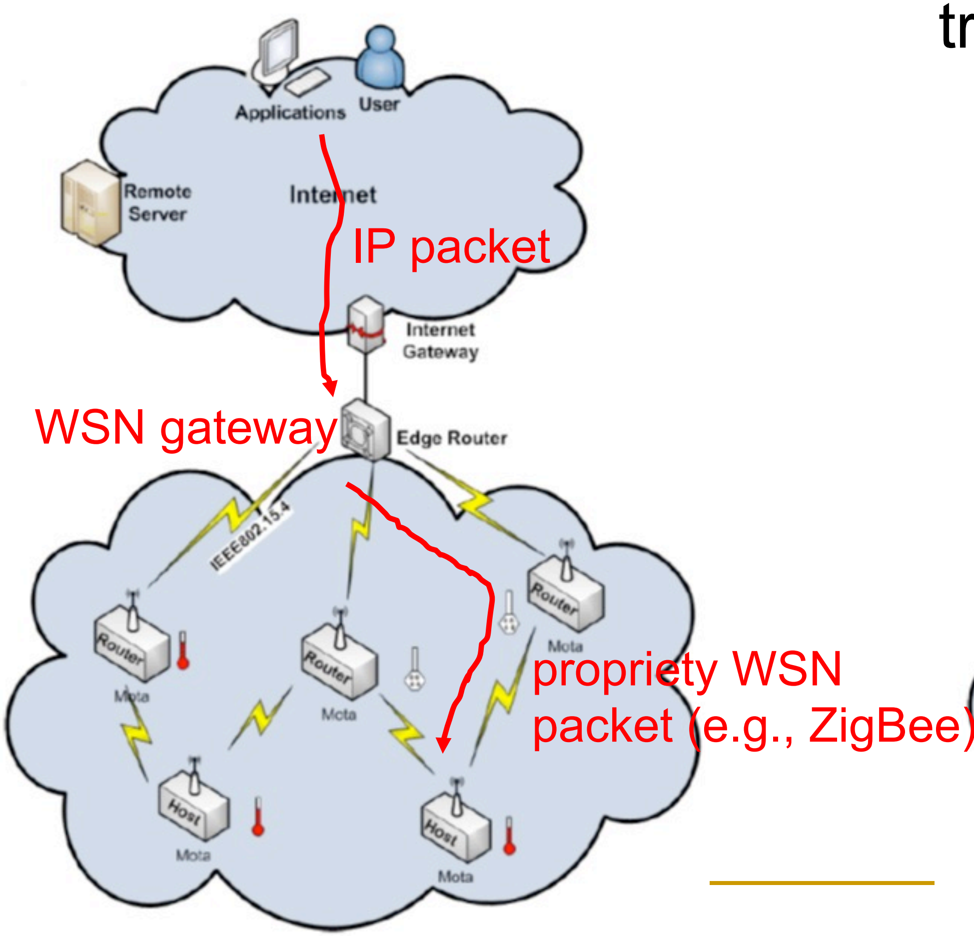
- IP, with its concept of layer 3 routing and internetwork technology, has made those debates about incompatible networks obsolete:
  - the vast majority of LANs and WANs today run IP, and many people can hardly remember which layer 2 technology their IP networks are running on.

Motivation

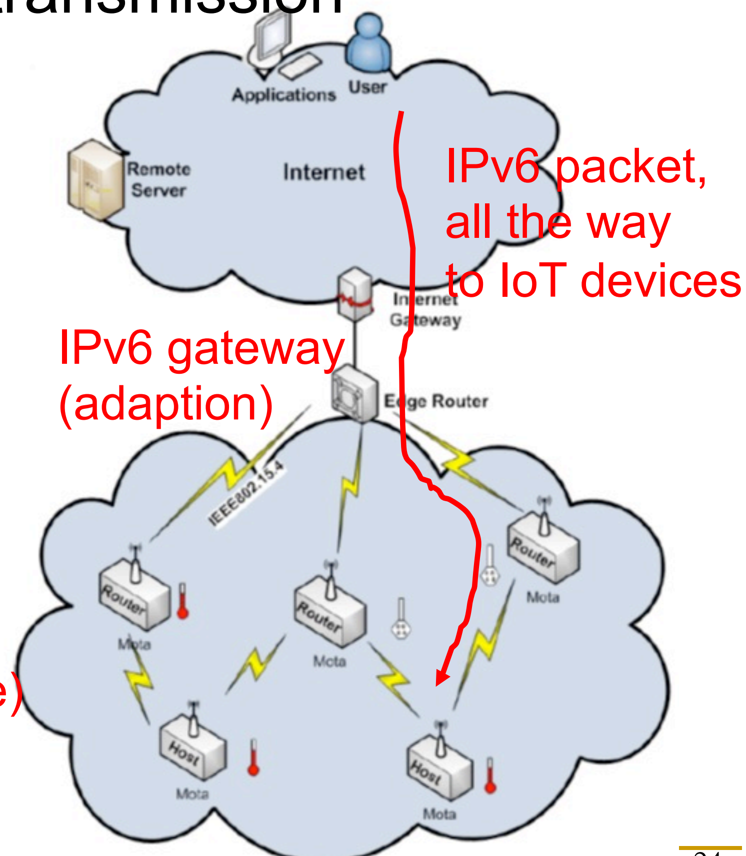
- Almost any layer 2 technology can be used and will simply extend the IP internetwork.
- The same transition to IP is now happening in the home and industrial automation worlds. 6LoWPAN and RPL have made this possible.

Goal of 6LowPAN

- Traditional way: 2-stage



- End-to-end IP transmission



Constraints of LoWPAN

- Low-cost nodes communicating over multiple hops to cover a large geographical area
- Operate unattended for years on modest batteries.
- Capabilities are more limited
  - small frame sizes, low bandwidth, and low transmit power, limited memory and compute power.
- Proprietary protocols and link-only solutions, presuming that IP was too memory and bandwidth-intensive

Key Factors for IP over 802.15.4

- Header
  - Standard IPv6 header is 40 bytes [RFC 2460]
  - Entire 802.15.4 MTU is 127 bytes [IEEE 802.15.4]
  - Often data payload is small
- Fragmentation
  - Interoperability means that applications need not know the constraints of physical links
  - IP packets may be large, compared to 802.15.4 max frame size
  - IPv6 requires all links support 1280 byte packets [RFC 2460]

Key Factors for IP over 802.15.4

- Allow link-layer mesh routing under IP topology
  - 802.15.4 subnets may utilize multiple radio hops per IP hop
  - Similar to LAN switching within IP routing domain in Ethernet
- Allow IP routing over a mesh of 802.15.4 nodes
  - Options and capabilities already well-defined
  - Various protocols to establish routing tables

Topology

- 6LoWPAN network can be organized around three topologies:
  - Star topology
  - Meshed
  - Routed

Star topology

- All sensor nodes can reach and are reachable from the LBR(LoWPAN Border Router)

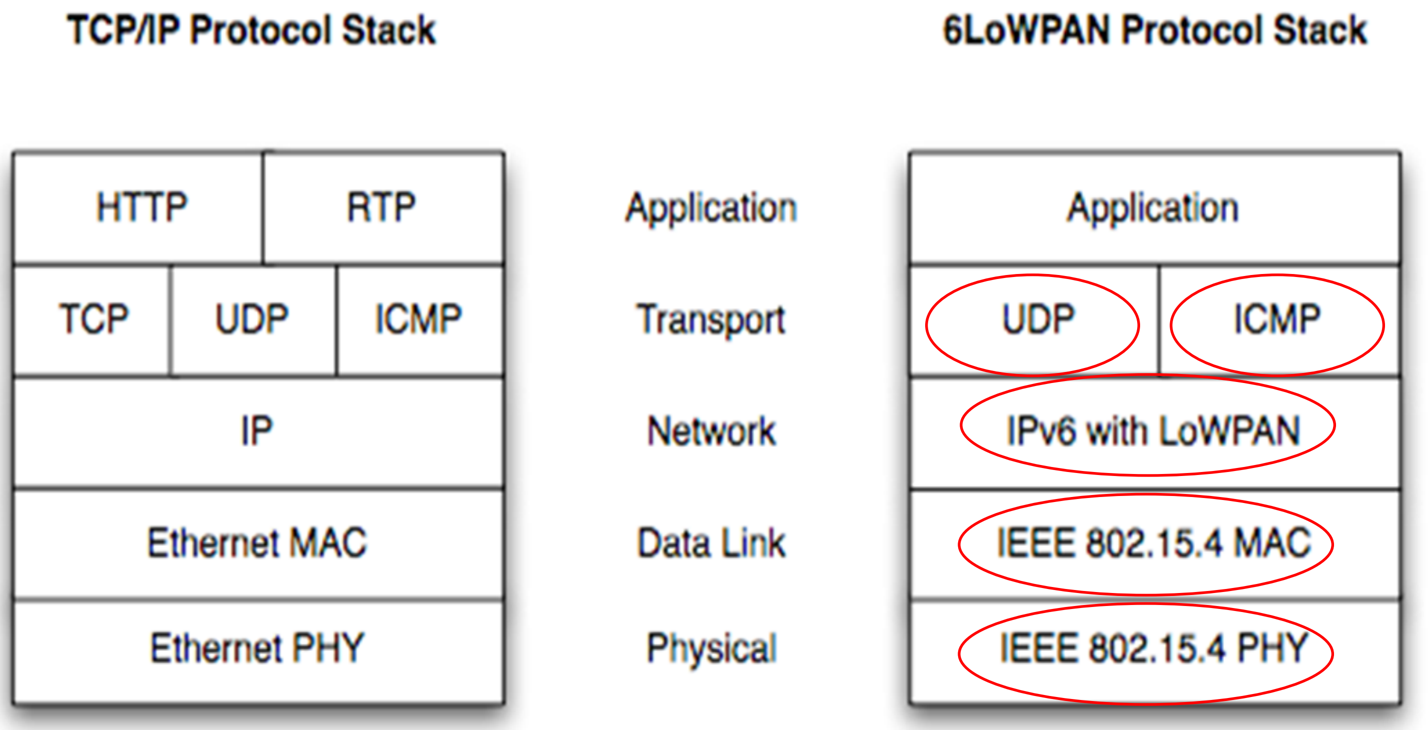
Meshed

- Nodes are organized at Layer 2 in order to relay frames toward the destination.
- From point of view of the Internet, a meshed network is similar to an Ethernet network where every node shares the same prefix.
- 6LoWPAN refers to that technology as **mesh-under (MU)**.

Routed

- Nodes act as routers and forward packets toward the destination.
- Nodes acting as a router inside the LoWPAN network and not directly connected to the Internet are called LoWPAN routers(LRs).
- 6LoWPAN refers to that technology as **route-over(RO)**. The best example is RPL protocol.

6LoWPAN Protocol Stack



6LoWPAN Key Elements

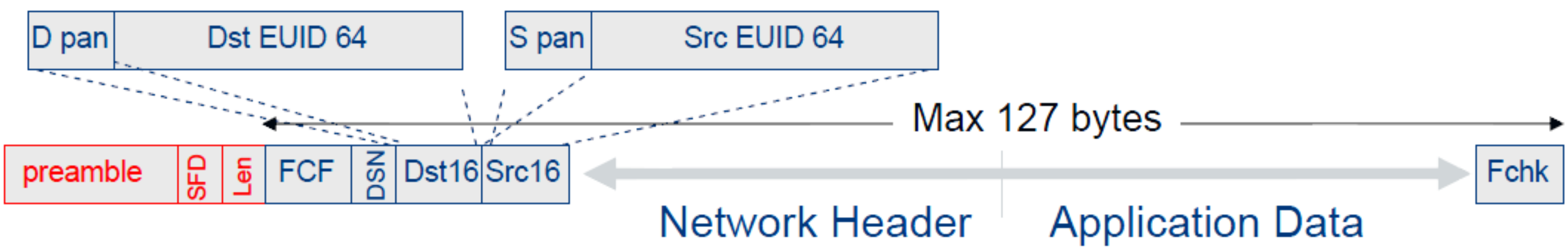
- 6LoWPAN introduces an adaptation layer between the IP stack's link and network layers to enable efficient transmission of IPv6 datagrams over 802.15.4 links
  - Provides header compression to reduce transmission overhead
  - Fragmentation to support the IPv6 minimum MTU requirement
  - Support for layer-two forwarding to deliver and IPv6 datagram over multiple radio hops

Key Concept

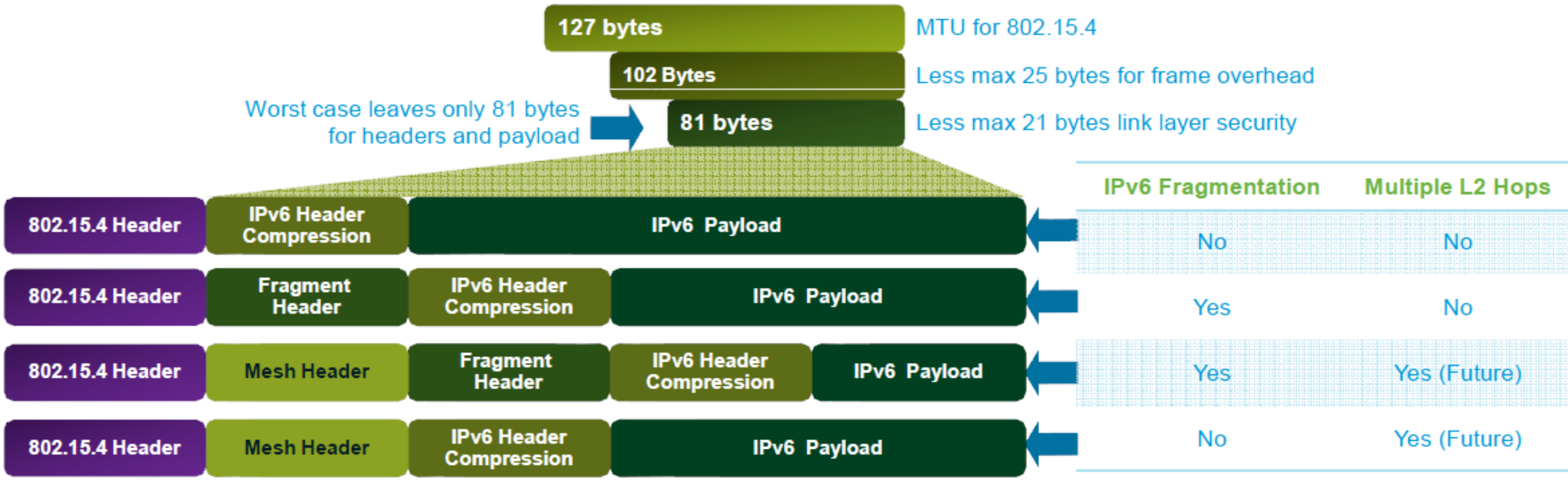
- Use of stateless or shared-context compression to elide adaptation, network, and transport layer header fields
  - Compressing all three layers down to a few bytes.
- It's possible to compress header fields to a few bits when we observe that they often carry common values, reserving an escape value for when less-common ones appear.

IEEE 802.15.4 Frame Format

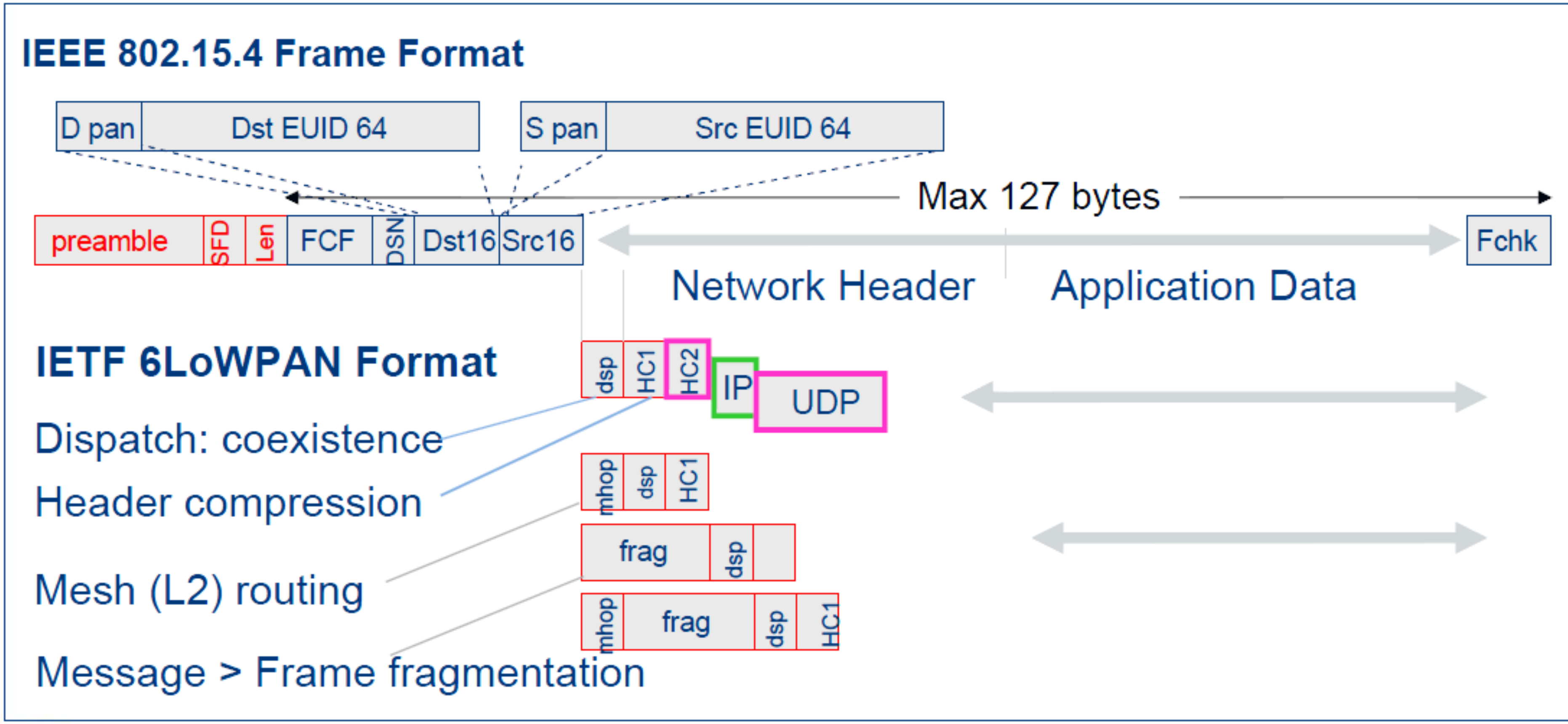
Octets:2	**1**	**0/2**	**0/2/8**	**0/2**	**0/2/8**	**0/5/6/10/14**	**Variable**	**2**
Frame Control	Sequence Number	Dest. PAN Identifier	Dest. Address	Source PAN Identifier	Source Addr.	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields					***MAC Payload***	**MFR**
MHR								



Typical 6LoWPAN Header Stacks



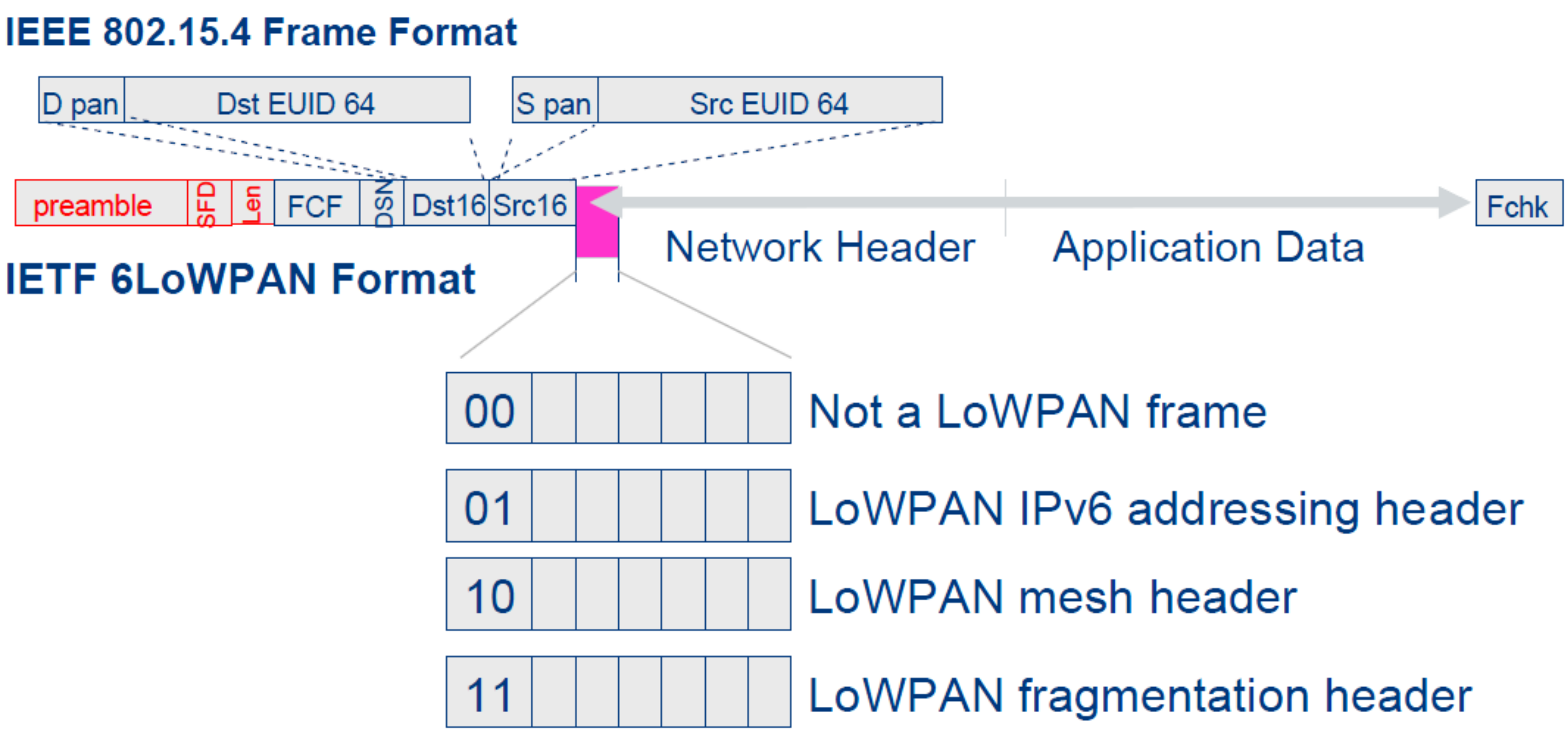
6LoWPAN Format Design



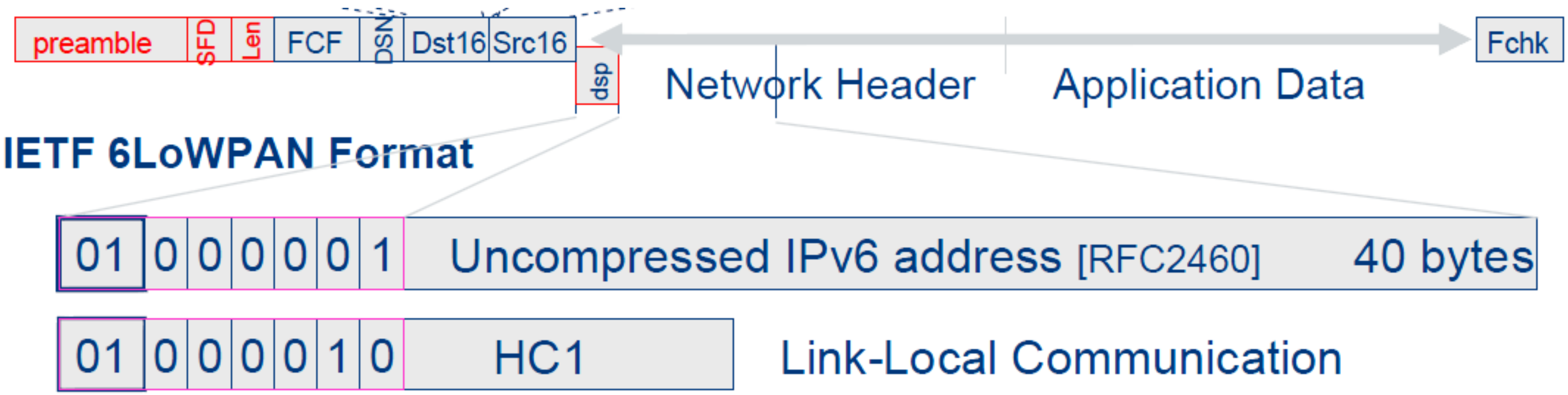
- Orthogonal stackable header format
- Almost no overhead for the ability to interoperate and scale.
- Pay for only what you use

6LoWPAN – The First Byte

- Coexistence with other network protocols over same link
- Header dispatch - understand what's coming



6LoWPAN – IPv6 Header



➔ *Fully compressed: 1 byte remains from uncompressed header*

Source address	: derived from link address or common prefix
Destination address	: derived from link address or common prefix
Traffic Class & Flow Label	: zero
Next header	: UDP, TCP, or ICMP
Hop Limit	: uncompressed

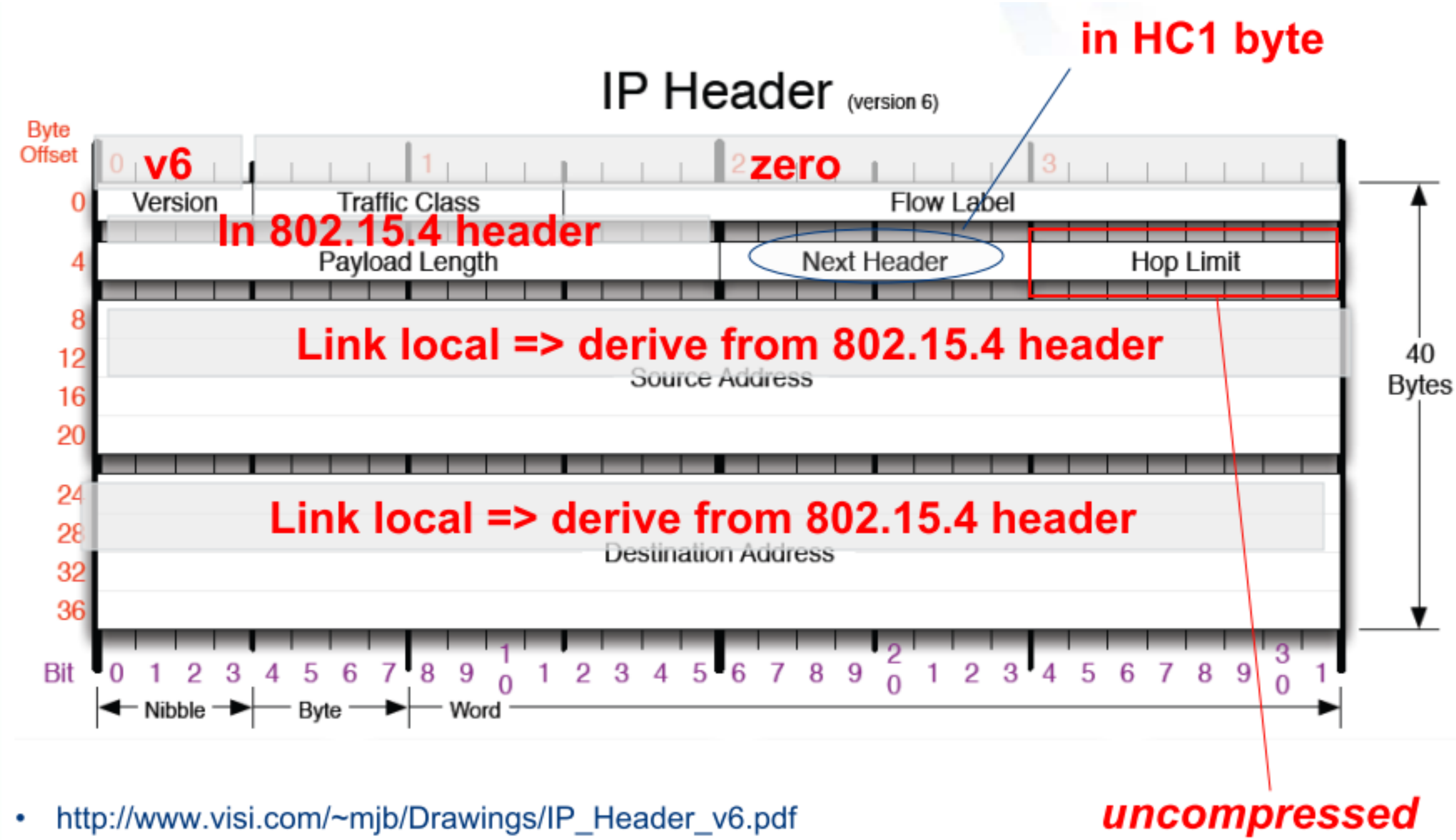
Dispatch Value Bit Pattern

Bit Pattern	Header Type	Reference
00 xxxxxx	NALP - Not a LoWPAN frame	[RFC4944]
01 000000	Reserved as a replacement value for ESC	[RFC6282]
01 000001	IPv6 - uncompressed IPv6 Addresses	[RFC4944]
01 000010	LOWPAN_HC1 - LOWPAN_HC1 compressed IPv6	[RFC4944]
01 000011	LOWPAN_DFF	[RFC6971]
01 000100 through 01 001111	reserved for future use	
01 010000	LOWPAN_BC0 - LOWPAN_BC0 broadcast	[RFC4944]
01 010001 through 01 011111	reserved for future use	
01 1xxxxx	LOWPAN_IPHC	[RFC6282]
10 xxxxxx	MESH - Mesh header	[RFC4944]
11 000xxx	FRAG1 -- Fragmentation Header (first)	[RFC4944]
11 001000 through 11 011111	reserved for future use	
11 100xxx	FRAGN -- Fragmentation Header (subsequent)	[RFC4944]
11 101000 through 11 111111	reserved for future use	

Dispatch Value Bit Pattern

```
   Pattern    Header Type
 +------------+-----------------------------------------------+
 |  00  xxxxxx | NALP        - Not a LoWPAN frame              |
 |  01  000001 | IPv6        - Uncompressed IPv6 Addresses     |
 |  01  000010 | LOWPAN_HC1  - LOWPAN_HC1 compressed IPv6      |
 |  01  000011 | reserved    - Reserved for future use         |
 |    ...      | reserved    - Reserved for future use         |
 |  01  001111 | reserved    - Reserved for future use         |
 |  01  010000 | LOWPAN_BC0  - LOWPAN_BC0 broadcast            |
 |  01  010001 | reserved    - Reserved for future use         |
 |    ...      | reserved    - Reserved for future use         |
 |  01  111110 | reserved    - Reserved for future use         |
 |  01  111111 | ESC         - Additional Dispatch byte follows |
 |  10  xxxxxx | MESH        - Mesh Header                     |
 |  11  000xxx | FRAG1       - Fragmentation Header (first)    |
 |  11  001000 | reserved    - Reserved for future use         |
 |    ...      | reserved    - Reserved for future use         |
 |  11  011111 | reserved    - Reserved for future use         |
 |  11  100xxx | FRAGN       - Fragmentation Header (subsequent)|
 |  11  101000 | reserved    - Reserved for future use         |
 |    ...      | reserved    - Reserved for future use         |
 |  11  111111 | reserved    - Reserved for future use         |
 +------------+-----------------------------------------------+
```

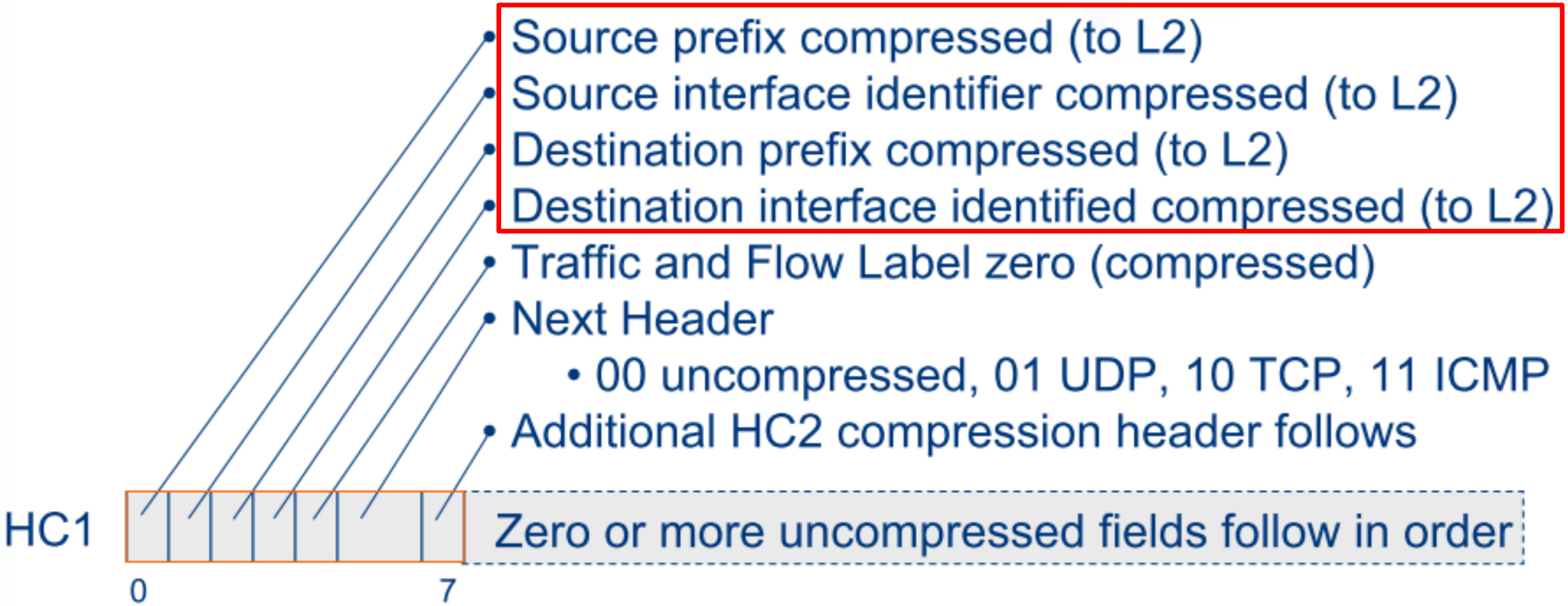
IPv6 Header Compression



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

HC1 Compressed IPv6 Header [4944]

For Link-Local Communication

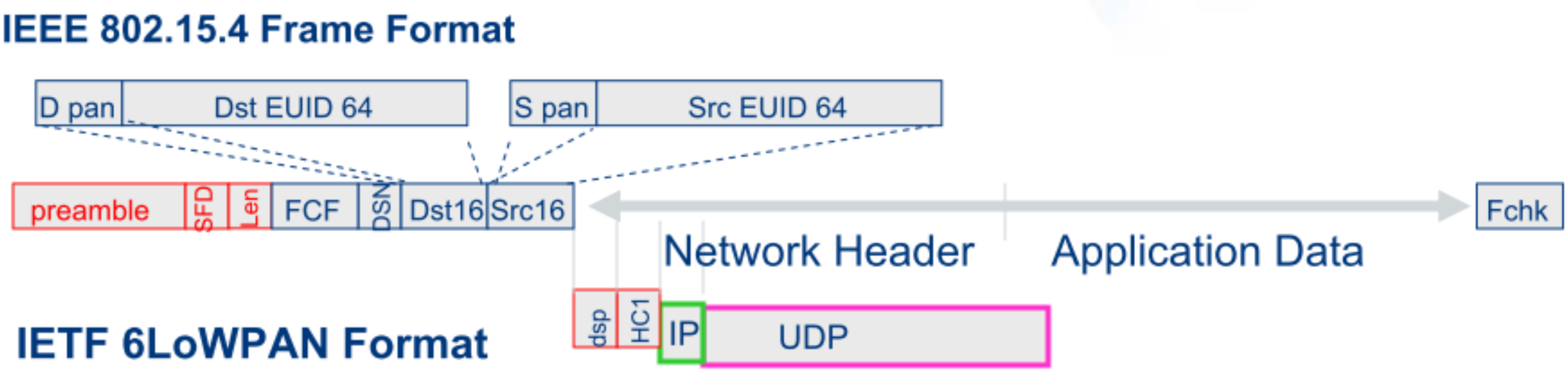


- Efficient communication with link-local IPv6 addresses
- IPv6 address <prefix64 || interface id> for nodes in 802.15.4 subnet derived from the link address.
  - PAN ID maps to a unique IPv6 prefix
  - Interface identifier generated from EUI-64 or short address
- Hop Limit is the only incompressible IPv6 header field

LOWPAN_HC1 (common compressed header encoding)

- The address fields encoded by "HC1 encoding" are interpreted as follows:
  - Source/Destination Prefix compression
    - PI(0): Prefix carried in-line (Section 10.3.1).
    - PC(1): Prefix compressed (link-local prefix assumed).
  - Source/Destination Interface ID compression
    - II(0): Interface identifier carried in-line (Section 10.3.1).
    - IC(1): Interface identifier elided (derivable from the corresponding link-layer address).

6LoWPAN – Compressed / UDP or ICMP



Dispatch: Compressed IPv6

HC1: Source & Dest Local, next hdr=UDP

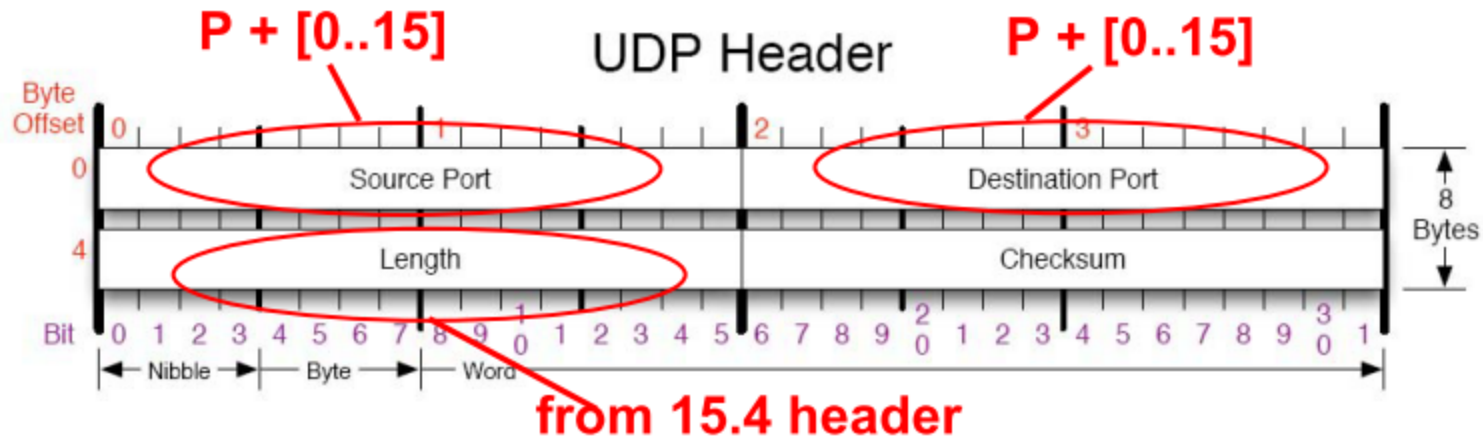
IP: Hop limit

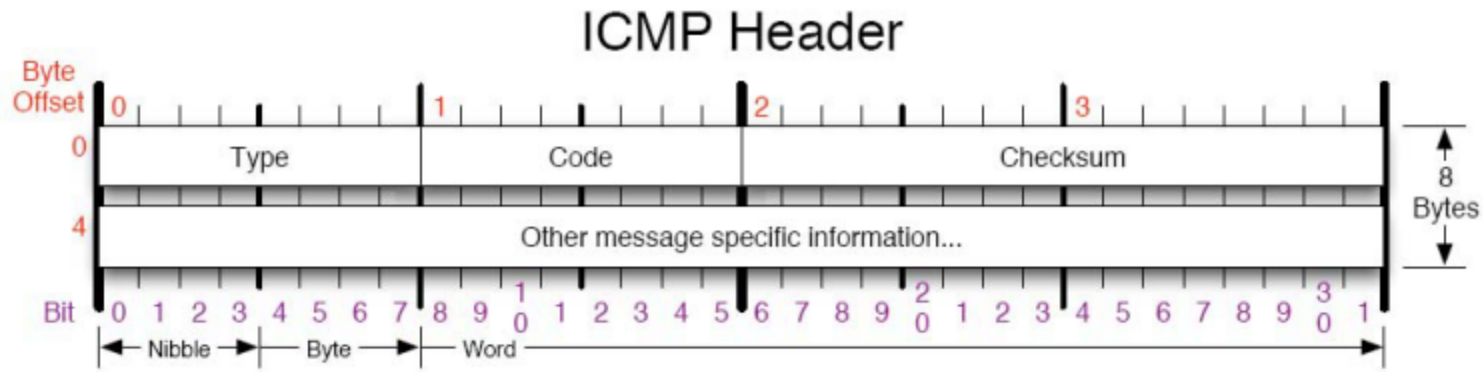
UDP: 8-byte header (uncompressed)

HC2 bit is not set

ICMP: 8-byte header (uncompressed)

L4 – UDP/ICMP Headers (8 bytes)

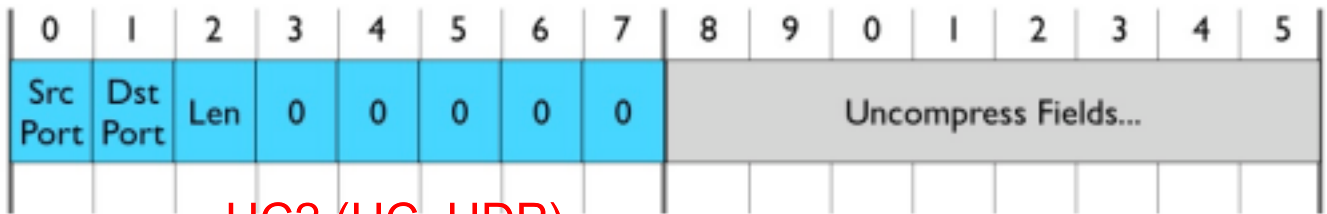




6LoWPAN – Compressed / Compressed UDP

IP: 3 bytes
UDP: 4 bytes (compressed indicator, ports, checksum)

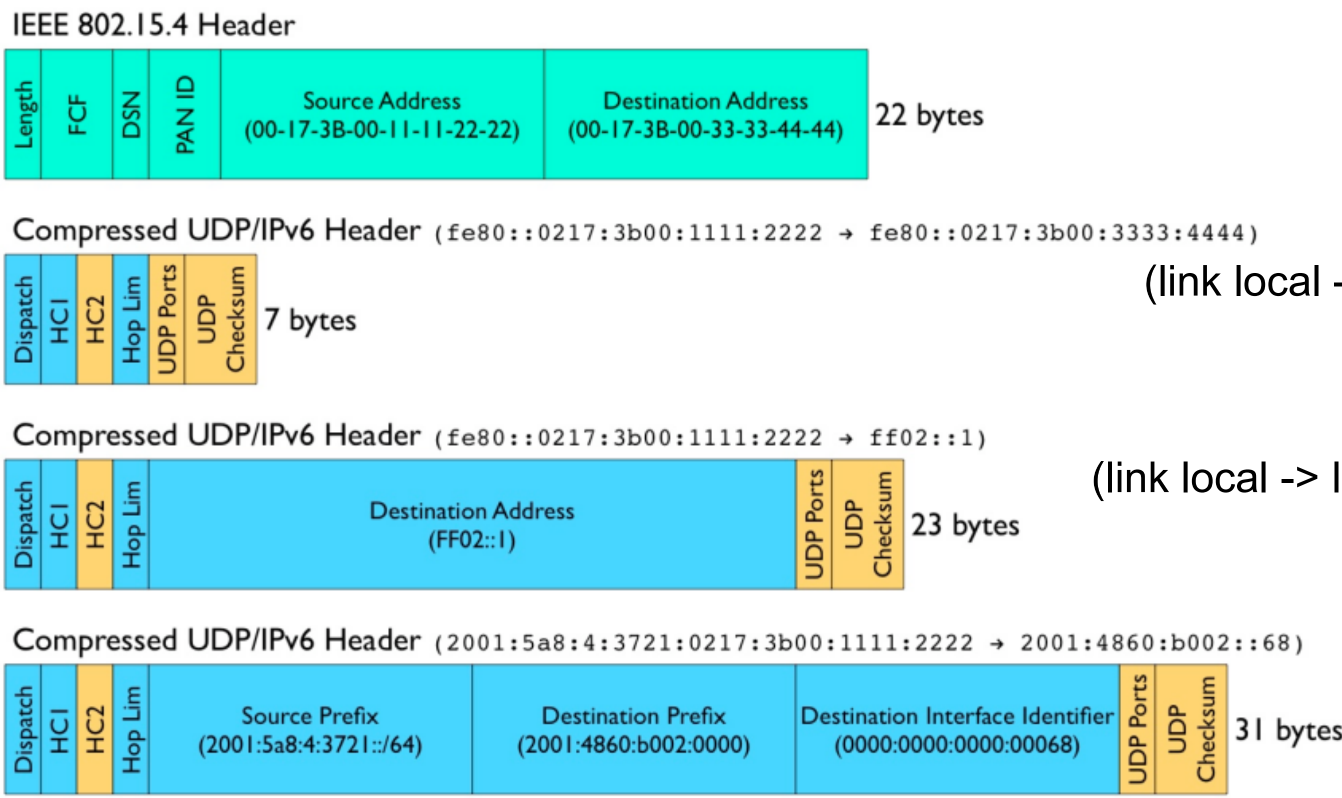
HC2 bit is set



HC2 (HC_UDP)

6LoWPAN introduces a range of well-known ports (61616 – 61631).
When ports fall in the well-known range, the upper 12 bits may be elided.
HC2 also allows elision of the UDP Length, as it can be derived from the IPv6 Payload Length field.

6LoWPAN IPv6/UDP Compression Examples

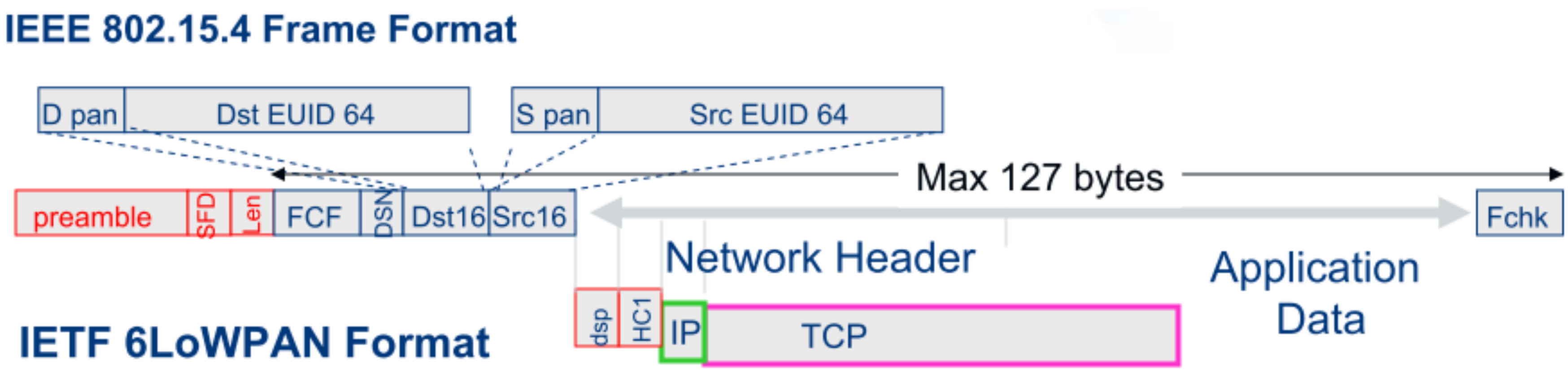


(link local -> link local)

(link local -> local multicast)

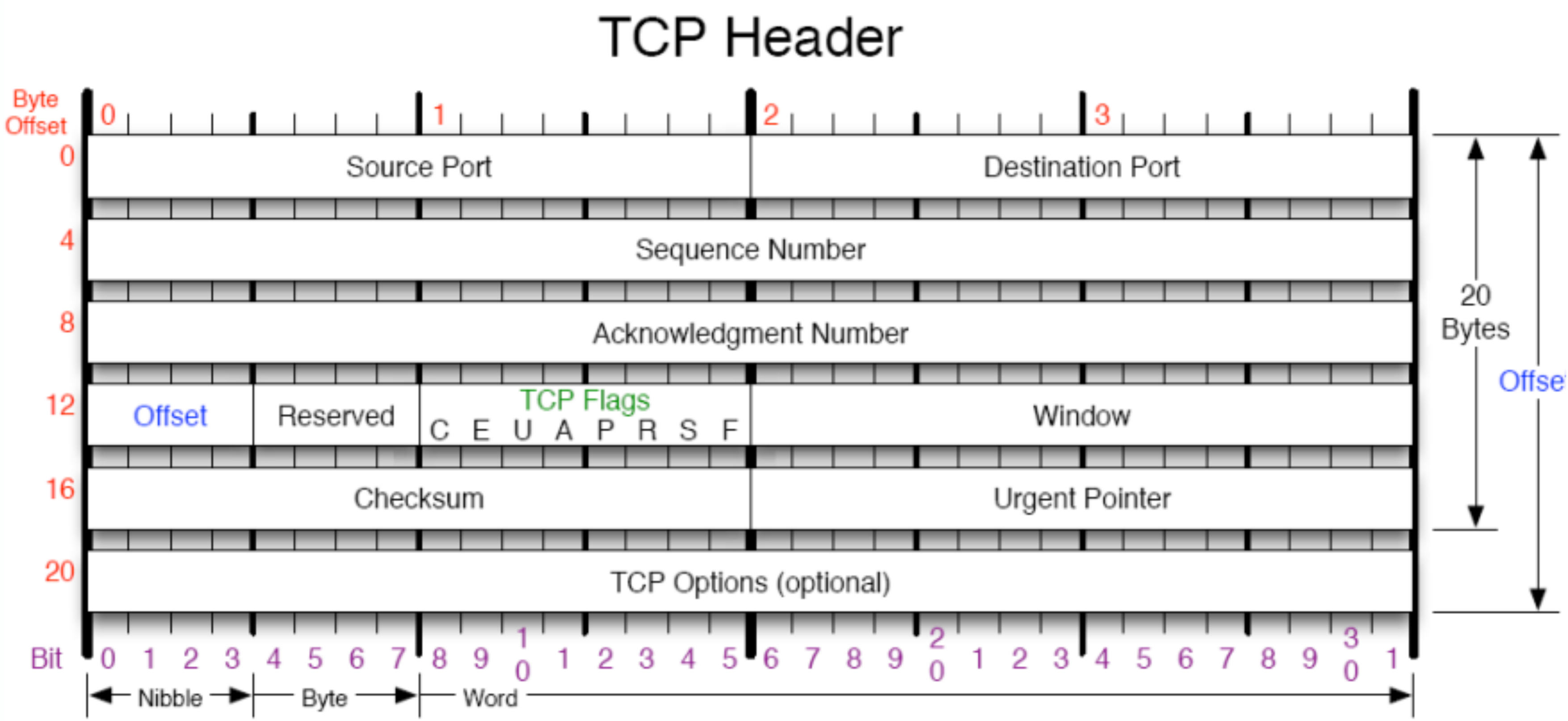
(link global -> link global)

6LoWPAN – Compressed / TCP



Dispatch: Compressed IPv6
HC1: Source & Dest Local, next hdr=TCP
IP: Hops Limit
TCP: 20-byte header

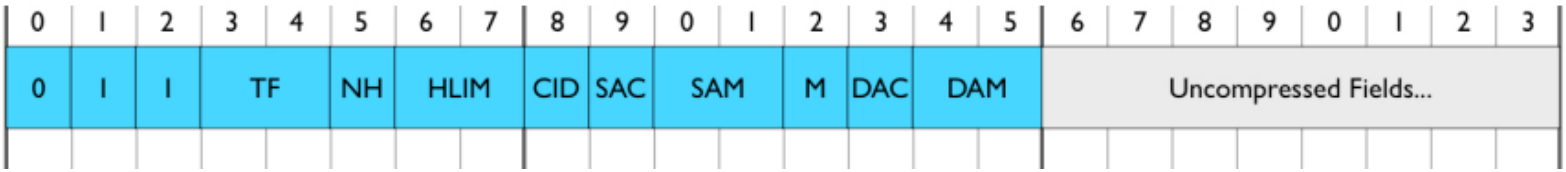
TCP Header


TCP Header
Byte Offset
0
4
8
12
16
20
0
1
2
3
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Offset
Reserved
TCP Flags
C E U A P R S F
Window
Checksum
Urgent Pointer
TCP Options (optional)
20 Bytes
Offse
Bit
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
Nibble
Byte
Word

LOWPAN_IPHC, NHC [RFC 6282]

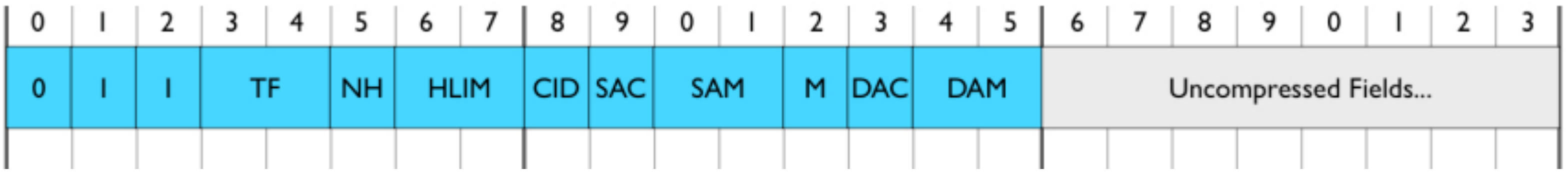
- Common case assumption (IPv6 header)
  - Version is 6
  - Traffic Class and Flow Label are both zero
  - Payload Length can be inferred from lower layers
  - Hop Limit will be set to a well-known value
  - Source addresses is formed using the link-local prefix or a small set of routable prefixes
  - Addresses assigned to 6LoWPAN interfaces are formed with an IID derived directly from either the 64-bit extended or the 16-bit short IEEE 802.15.4 addresses.

6LoWPAN Improved IPv6 Header Compression [RFC 6282]



- IPHC
  - TF: Traffic Class and Flow Label to be individually compressed
    - 2-bit Explicit Congestion Notification (ECN) and 6-bit Differentiated Services Code Point (DSCP)
    - 00: ECN + DSCP + 4-bit Pad + Flow Label (4 bytes)
    - 01: ECN + 2-bit Pad + Flow Label (3 bytes), DSCP is elided.
    - 10: ECN + DSCP (1 byte), Flow Label is elided.
    - 11: Traffic Class and Flow Label are elided.
  - NH: Next Header
    - 0: Full 8 bits for Next Header are carried in-line.
    - 1: The Next Header field is compressed and the next header is encoded using LOWPAN_NHC

6LoWPAN Improved IPv6 Header Compression [RFC 6282]

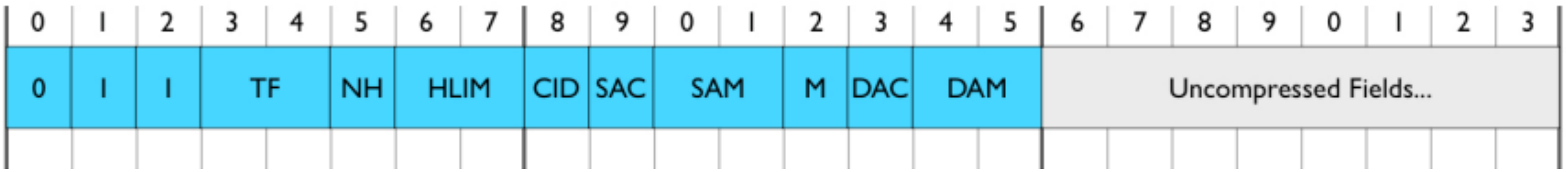


- IPHC
  - HLIM: Hop Limit compression when common values
    - 00: The Hop Limit field is carried in-line.
    - 01: The Hop Limit field is compressed and the hop limit is 1.
    - 10: The Hop Limit field is compressed and the hop limit is 64.
    - 11: The Hop Limit field is compressed and the hop limit is 255.
  - Context Identifier(CID): Makes use of shared-context to elide the prefix from IPv6 addresses (two additional 4-bit fields)
    - 0: No additional 8-bit Context Identifier Extension is used (either Source Address Compression (SAC) or Destination Address Compression (DAC)).
    - 1: An additional 8-bit Context Identifier Extension field immediately follows the Destination Address Mode (DAM) field.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|      SCI      |      DCI      |
+---+---+---+---+---+---+---+---+
```

Figure 3: LOWPAN_IPHC Encoding

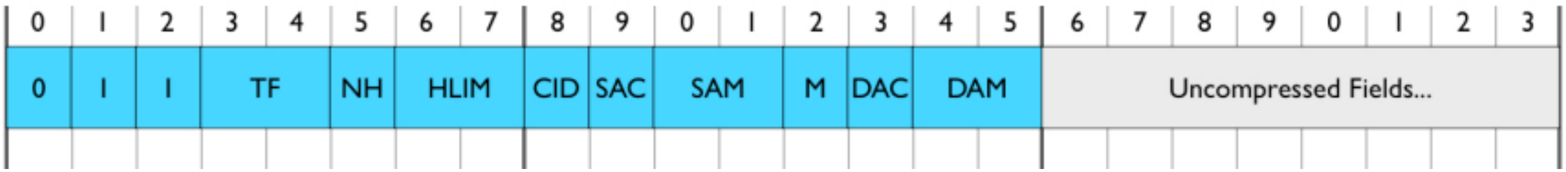
6LoWPAN Improved IPv6 Header Compression [RFC 6282]



- IPHC
  - Source Address Compression (SAC) indicates whether stateless compression is used
    - 0: Source address compression uses stateless compression.
    - 1: Source address compression uses stateful, context-based compression.
  - Source Address Mode (SAM) indicates whether the full Source Address is carried inline, upper 16 or 64-bits are elided, or the full Source Address is elided.
    - If SAC=0:
      - 00: 128 bits
      - 01: 64 bits (network prefix elided, use link-local prefix)
      - 10: 16 bits (first 112 bits elided); 0000:00ff:fe00:XXXX
      - 11: 0 bits.

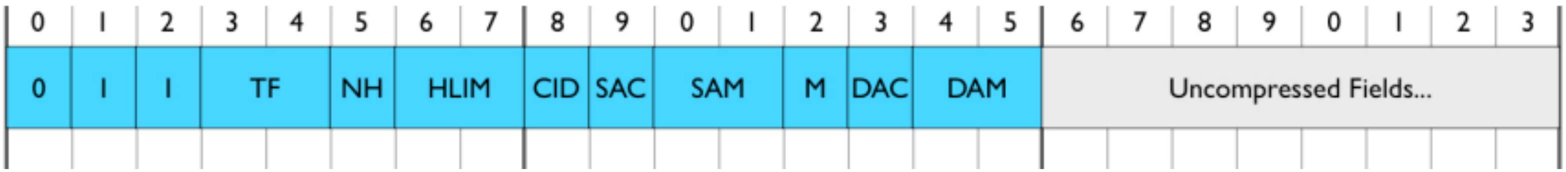
Zigbee short address

6LoWPAN Improved IPv6 Header Compression [RFC 6282]



- IPHC
  - Source Address Mode (SAM)
    - If SAC=1:
      - 00: The UNSPECIFIED address, ::
      - 01: 64 bits. (other address bits are derived using context information)
      - 10: 16 bits. (using context information + 0000:00ff:fe00:XXXX)
      - 11: 0 bits. The address is fully elided and is derived using context information and the encapsulating header (e.g., 802.15.4 or IPv6 source address)
  - M: Supports multicast addresses most often used for IPv6 ND and SLAAC (StateLess Address AutoConfiguration).
    - 0: Destination address is not a multicast address.
    - 1: Destination address is a multicast address.

6LoWPAN Improved IPv6 Header Compression [RFC 6282]



- IPHC
  - M:
    - If M=1 and DAC=0, DAM:
      - 00: 128 bits. The full address is carried in-line.
      - 01: 48 bits. The address takes the form ffFS::00GGGGGG:GGGG. (F: Flag, S: Scope)
      - 10: 32 bits. The address takes the form ffFS::00GG:GGGG. (G: Group id)
      - 11: 8 bits. The address takes the form ff02::00GG.
    - If M=1 and DAC=1, DAM:
      - 00: 48 bits. This format is designed to match Unicast-Prefix-based IPv6 Multicast Addresses as defined in [RFC3306] and [RFC3956]. The multicast address takes the form ffXX:XXLL:PPPP:PPPP:PPPP:PPPP:XXXX:XXXX. (Prefix Length and Network Prefix can be taken from a context)
      - 01, 10, 11: reserved

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Flags | Scope | Rsvd / RIID   |         Group Identifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Group Identifier       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header Compression (NHC)

- NH=1 in IPHC indicates the use of LOWPAN_NHC

```
+-------------+-------------+-------------+-----------------+--------
| LOWPAN_IPHC | In-line     | LOWPAN_NHC  | In-line Next    | Payload
|   Encoding  |   IP Fields |   Encoding  |   Header Fields |
+-------------+-------------+-------------+-----------------+--------
```

```
+----------------+---------------------------
| var-len NHC ID | compressed next header...
+----------------+---------------------------
```

LOWPAN_NHC Format

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 0 |    EID    |NH |
+---+---+---+---+---+---+---+---+
```

IPv6 Extension Header Encoding

EID: IPv6 Extension Header ID:

0: IPv6 Hop-by-Hop Options Header [RFC2460]
1: IPv6 Routing Header [RFC2460]
2: IPv6 Fragment Header [RFC2460]
3: IPv6 Destination Options Header [RFC2460]
4: IPv6 Mobility Header [RFC6275]
5: Reserved
6: Reserved
7: IPv6 Header (NH must be 0; MUST be encoded using LOWPAN_IPHC)

NH: Next Header:

0: Full 8 bits for Next Header are carried in-line.
1: Next Header is elided, next header is encoded using LOWPAN_NHC

UDP NHC Format

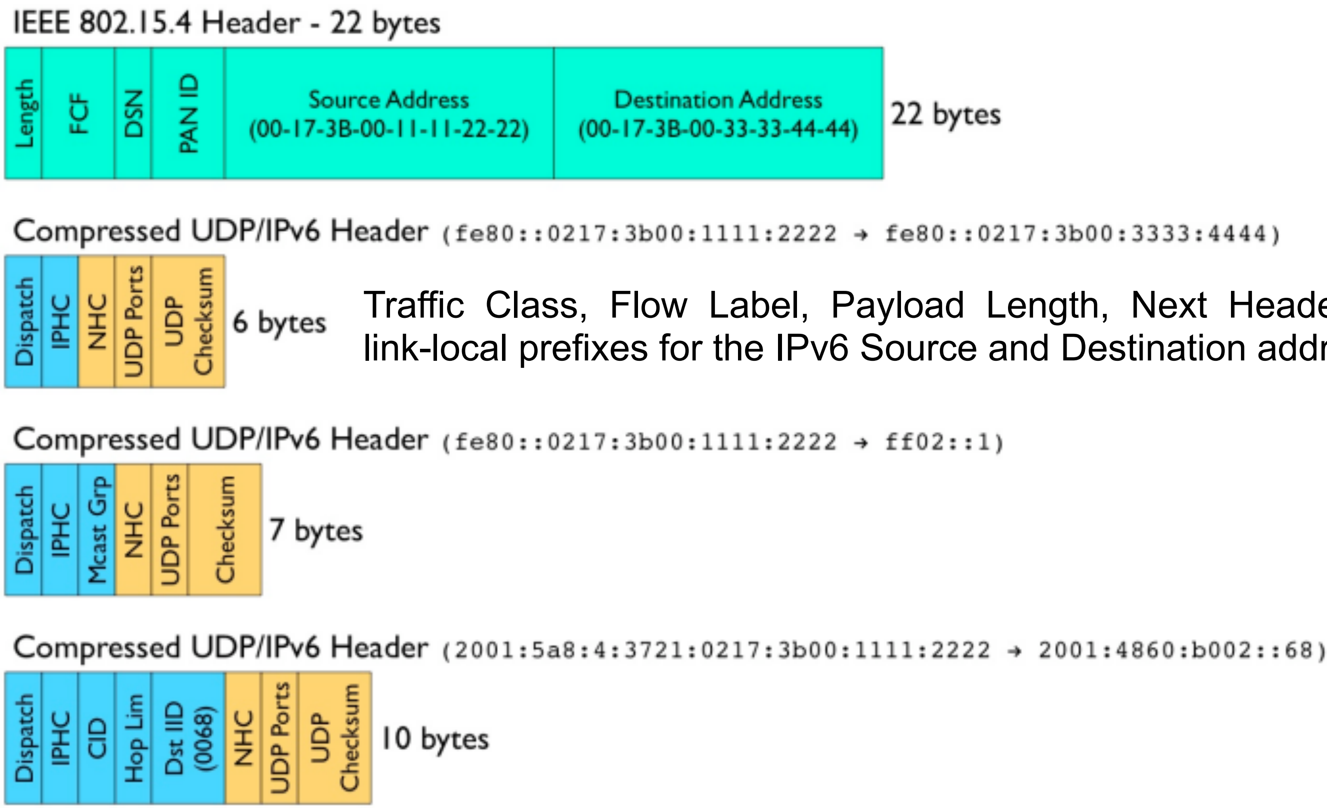
```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 1 | 0 | C |   P   |
+---+---+---+---+---+---+---+---+
```

- C: Checksum:
  - 0: All 16 bits of Checksum are carried in-line.
  - 1: All 16 bits of Checksum are elided. The Checksum is recovered by re-computing it on the 6LoWPAN termination point.
- P: Ports:
  - 00: All 16 bits for both Source Port and Destination Port are carried in-line.
  - 01: All 16 bits for Source Port are carried in-line. First 8 bits of Destination Port is 0xf0 and elided. The remaining 8 bits of Destination Port are carried in-line.
  - 10: First 8 bits of Source Port are 0xf0 and elided. The remaining 8 bits of Source Port are carried in-line. All 16 bits for Destination Port are carried in-line.
  - 11: First 12 bits of both Source Port and Destination Port are 0xf0b and elided. The remaining 4 bits for each are carried in-line.

Compressing UDP Checksum

- UDP checksum is mandatory with IPv6
- In RFC 6282, an endpoint MAY elide the UDP Checksum if it is authorized by the upper layer.
  - Tunneling: tunneled Protocol Data Unit (PDU) possesses its own addressing, security and integrity check
  - Message Integrity Check: e.g., IPsec Authentication Header
- A decompressor that expands a 6LoWPAN packet with the C bit set MUST compute the UDP Checksum on behalf of the source node and place that value in the restored UDP header as specified in the incumbent standards [RFC0768], [RFC2460].

Improved UDP/IPv6 Header Compression Examples



Traffic Class, Flow Label, Payload Length, Next Header, Hop Limit, and link-local prefixes for the IPv6 Source and Destination addresses are all elided.

NHC header defines a new variable length Next Header identifier, allowing for future definition of arbitrary next header compression encodings.

Fragmentation

- Datagram Size(11): total size of the unfragmented payload
- Datagram Tag(16): ID of the fragmented packet
- Datagram Offset(8): in units of 8-byte chunks

The header type is only two bits.

The third bit is used to compress the datagram offset on the first fragment as it is always zero.

The fragment header is 4 bytes for the first fragment and 5 bytes for all subsequent fragments.

Mesh Addressing Header

- Hop Limit: 4 bits
- Source Address, and Destination Address: IEEE 802.15.4 link addresses and may carry either a short or extended address.
  - S/D: short or full address of source/destination address

Conclusion

- 6LoWPAN turns IEEE 802.15.4 into the next IP-enabled link
- Provides open-systems based interoperability among low-power devices over IEEE 802.15.4
- Provides interoperability between low-power devices and existing IP devices, using standard routing techniques
- Paves the way for further standardization of communication functions among low-power IEEE 802.15.4 devices

RPL: The IP routing protocol designed for low power and lossy networks [RFC 6550]

黃仁竑 教授
國立中正大學資工系

What is RPL?

- The IETF Routing Over Low-power and Lossy networks (ROLL) Working Group was formed in 2008
  - to create an IP level routing protocol adapted to the requirements of mesh networking for IoT/M2M
- The first version of RPL (Routing Protocol for Low-power and lossy networks) was finalized in April 2011
- Current standard: RFC 6550 (March 2012)
  - based on distance vector algorithms

Working Items of ROLL WG

- Protocol work
  - http://datatracker.ietf.org/doc/draft-ietf-roll-rpl/
  - RPL is designed to support different LLN application requirements
    - RFC 5548 - Routing requirements for Urban LLNs
    - RFC 5673 - Routing requirements for Industrial LLNs
    - RFC 5826 - Routing requirements for Home Automation LLNs
    - RFC 5867 - Routing requirements for Building Automation LLNs
- Routing metrics
  - http://tools.ietf.org/id/draft-ietf-roll-routing-metrics/
  - RFC 6551: Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks (2012/3)
- Security Framework
  - http://tools.ietf.org/id/draft-ietf-roll-security-framework/
- The Trickle Algorithm (RFC 6206): adjustable transmission window scheme
- Terminology
  - http://tools.ietf.org/id/draft-ietf-roll-terminology/
- Applicability statement
  - http://tools.ietf.org/id/draft-ietf-roll-applicability-ami/

Functionality of RPL

- RPL specifies a routing protocol specially adapted for the needs of IPv6 communication over "low-power and lossy networks" (LLNs), supporting
  - peer to peer traffic (point to point) (P2P)
  - point to multipoint (P2MP) communication: from a central server to multiple nodes on the LLN
  - multipoint to point (MP2P) communication
- The base RPL specification is optimized only for MP2P traffic or P2MP, and P2P is optimized only through use of additional mechanisms.

Functionality of RPL

- RPL expects an external mechanism to access and transport some control information, referred to as the "RPL Packet Information", in data packets. (ICMP)
- RPL provides a mechanism to disseminate information over the dynamically formed network topology.
  - To reduce the number of messages sent on the network, a trickle algorithm may limit the number of periodic messages that are sent. [RFC6206]

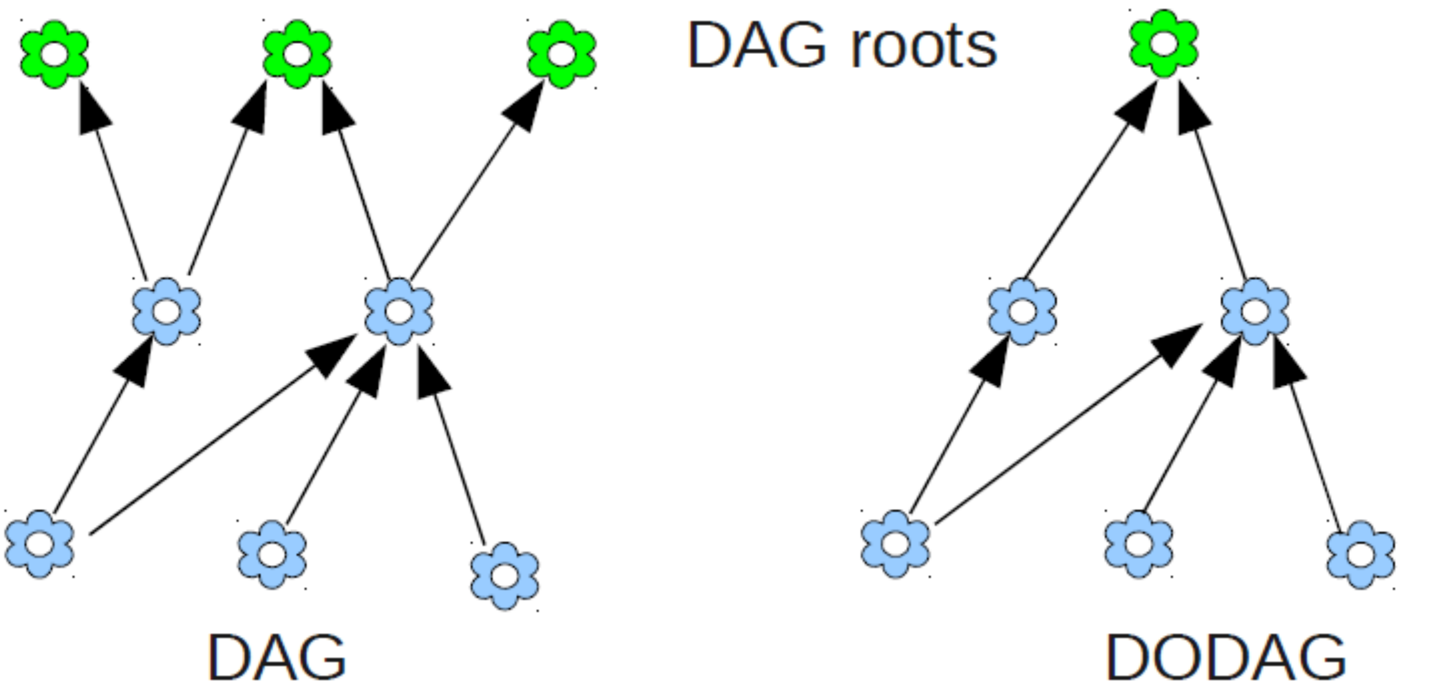
Functionality of RPL

- In some applications, RPL assembles topologies of routers that own independent prefixes.
- RPL also introduces the capability to bind a subnet together with a common prefix and to route within that subnet.
- RPL may disseminate IPv6 Neighbor Discovery (ND) information such as the [RFC4861] Prefix Information Option (PIO) and the [RFC4191] Route Information Option (RIO).

Terminology

- DAG: Directed Acyclic Graph
- DAG root: A DAG root is a node within the DAG that has no outgoing edge.
- Destination-Oriented DAG (DODAG): A DAG rooted at a single destination
- DODAG root: A DODAG root is the DAG root of a DODAG; it may act as a border router for the DODAG.

DAG and DODAG



Terminology

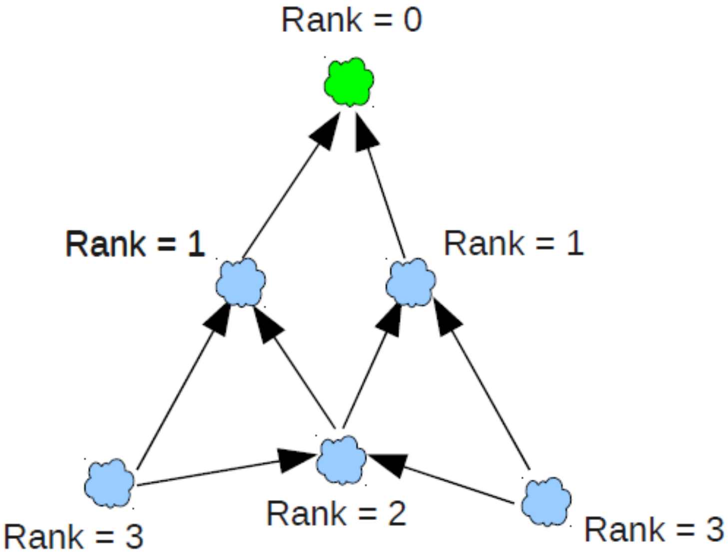
- Virtual DODAG root: A Virtual DODAG root is the result of two or more RPL routers, for instance, 6LoWPAN Border Routers (6LBRs), coordinating to synchronize DODAG state and act in concert as if they are a single DODAG root (with multiple interfaces), with respect to the LLN.

Terminology

- Up: Up refers to the direction from leaf nodes towards DODAG roots, following DODAG edges.
- Down: Down refers to the direction from DODAG roots towards leaf nodes, in the reverse direction of DODAG edges.

Terminology

- Rank: A node's Rank defines the node's individual position relative to other nodes with respect to a DODAG root. Rank strictly increases in the Down direction and strictly decreases in the Up direction. The exact way Rank is computed depends on the DAG's Objective Function (OF).



Terminology

- Objective Function (OF): An OF defines how routing metrics, optimization objectives, and related functions are used to compute Rank.
  - Currently, two objective functions are defined
    - OF0: based on hop counts (no routing metrics)
    - Minimum rank with hysteresis objective function (MRHOF)
      - The rank computation is based on metrics (e.g. link quality) contained in DIO messages.
      - MRHOF works only for additive metrics

Objective Function (Contiki OS)

- OF0
  - Cooja uses a rank with a minimum of 256 units (min_hoprankinc) that allows a maximum of 255 hops
- Minimum rank with hysteresis objective function (MRHOF)
  - Cooja uses a rank with a minimum unit of 128. The ETX metric starts with a unit of 256 with a fixed-point divisor of 128. (ETX: expected number of TX)

$$ETX = \frac{1}{D_f \times D_r}$$

D_f indicates the probability of packets being received by the neighboring node.
D_r is the probability that the acknowledgment is received successfully.

Terminology

- Routing Metrics and constraints
  - LLN requires a sophisticated routing metric strategy driven by type of data traffic.
  - A metric is a scalar quantity used as input for best path selection.
  - A constraint is used to prune links or nodes that do not meet the set of constraints.
  - Metrics and constraints can be node or link based.
    - Examples of node level metrics are node state attribute, node energy state etc., while link level metrics can be latency, reliability, etc.

Terminology

- Objective Code Point (OCP): An OCP is an identifier that indicates which Objective Function the DODAG uses.
- RPLInstanceID: A RPLInstanceID is a unique identifier within a network. DODAGs with the same RPLInstanceID share the same Objective Function. multi-topology routing (MTR)

Terminology

- RPL Instance: A RPL Instance is a set of one or more DODAGs that share a RPLInstanceID.
- DODAGID: identifier of a DODAG root.
- DODAG Version: a specific iteration ("Version") of a DODAG with a given DODAGID
- DODAGVersionNumber: a sequential counter that is incremented by the root to form a new Version of a DODAG. A DODAG Version is identified uniquely by the (RPLInstanceID, DODAGID, DODAGVersionNumber) tuple

Terminology

- Grounded: A DODAG is grounded when the DODAG root can satisfy the Goal. (DAG's root is a border router)
- Floating: A DODAG is floating if it is not grounded. (a subDAG's root may not be a border router)
- DODAG parent: one of the immediate successors of the node on a path towards the DODAG root.
- Sub-DODAG: The sub-DODAG of a node is the set of other nodes whose paths to the DODAG root pass through that node.

Terminology

- DIO: DODAG Information Object
- DAO: Destination Advertisement Object
- DIS: DODAG Information Solicitation
- CC: Consistency Check

RPLinstanceID

- Multiple concurrent instances of RPL may operate in a given network, each of them is characterized by a unique RPLinstanceID.
  - Below, we describe the behavior of an individual RPL instance.
  - A RPL instance defines Optimization Objective when forming paths towards roots based on one or more metrics
    - Metrics may include both Link properties (Reliability, Latency) and Node properties (Powered on not)

Topology

- RPL organizes a topology as a Directed Acyclic Graph (DAG) that is partitioned into one or more Destination Oriented DAGs (DODAGs), one DODAG per sink.
- A RPLInstanceID identifies a set of one or more Destination Oriented DAGs (DODAGs).
- The set of DODAGs identified by a RPLInstanceID is called a RPL Instance. All DODAGs in the same RPL Instance use the same OF.

DODAG Construction (1st view)

- The root starts advertising the information about the graph using the DIO message.
- The neighboring nodes of the root will receive and process DIO message potentially from multiple nodes and makes a decision based on certain rules (according to the objective function, DAG characteristics, advertised path cost and potentially local policy) whether to join the graph or not.
- Once the node has joined a graph it has a route toward the graph (DODAG) root.
- The graph root is termed as the ‘parent’ of the node.

DODAG Construction (1^{st} view)

- The node computes the ‘rank’ of itself within the graph, which indicates the “coordinates” of the node in the graph hierarchy.
- The neighboring peers will repeat this process and do parent selection, route addition and graph information advertisement using DIO messages.
  - If configured to act as a router, it starts advertising the graph information with the new information to its neighboring peers.
  - If the node is a “leaf node”, it simply joins the graph and does not send any DIO message.
- This rippling effect builds the graph edges out from the root to the leaf nodes where the process terminates.

DODAG Construction (1st view)

- In this graph, each node has a routing entry towards its parent (or multiple parents depending on the objective function) in a hop-by-hop fashion and the leaf nodes can send a data packet all the way to root of the graph by just forwarding the packet to its immediate parent.
- This model represents a MP2P (Multipoint-to-point) forwarding model where each node of the graph has reach-ability toward the graph root. This is also referred to as UPWARD routing.

DODAG Construction (1st view)

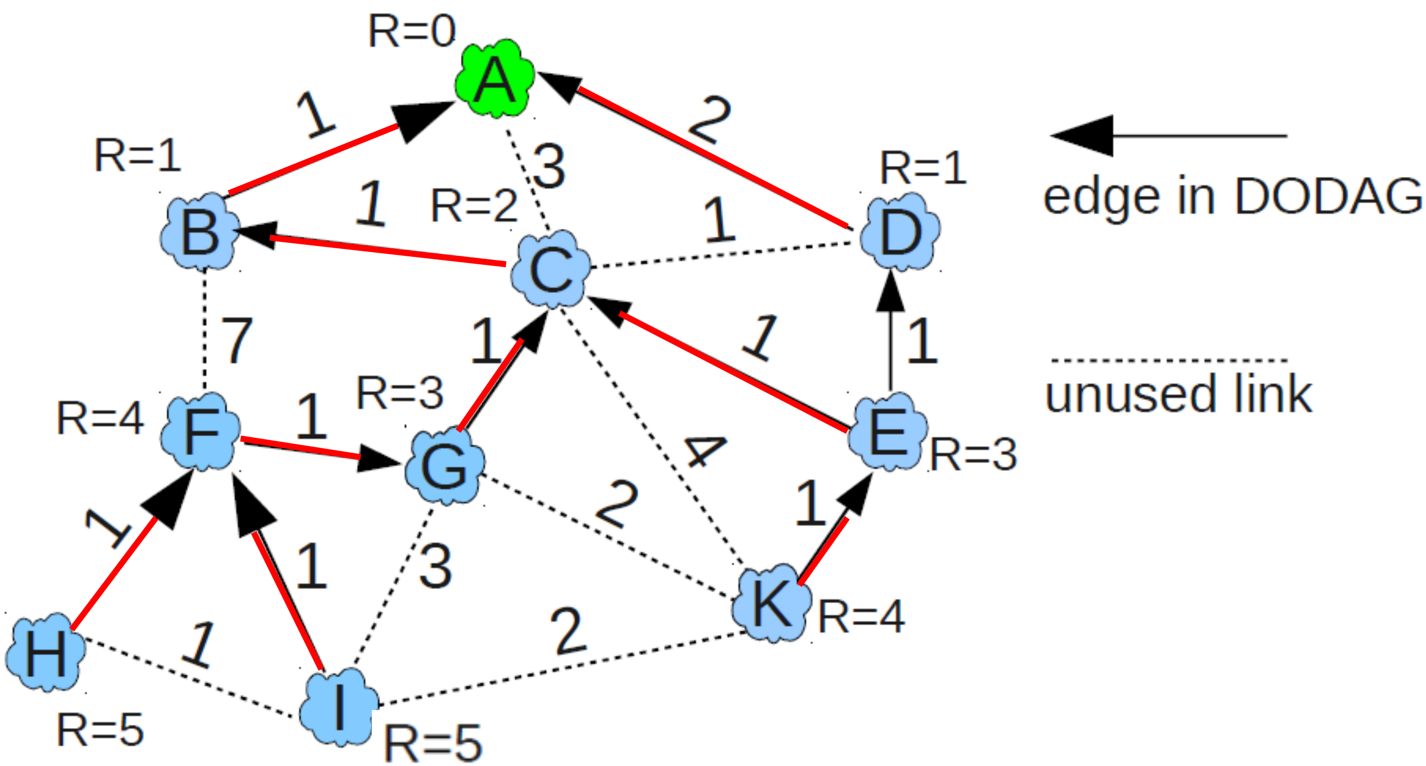
- Each node in the graph has a ‘rank’ that is relative and represents an increasing coordinate of the relative position of the node with respect to the root in graph topology.
- The notion of “rank” is used by RPL for various purposes including loop avoidance. The MP2P flow of traffic is called the ‘up’ direction in the DODAG.

DODAG Construction (2nd view)

- Some nodes are configured to be DODAG roots, with associated DODAG configurations.
- Nodes advertise their presence, affiliation with a DODAG, routing cost, and related metrics by sending link-local multicast DIO messages to all-RPL-nodes.
- Nodes listen for DIOs and use their information to join a new DODAG (thus, selecting DODAG parents), or to maintain an existing DODAG, according to the specified Objective Function and Rank of their neighbors.
- Nodes provision routing table entries, for the destinations specified by the DIO message, via their DODAG parents in the DODAG Version. Nodes that decide to join a DODAG can provision one or more DODAG parents as the next hop for the default route and a number of other external routes for the associated instance.
  - chooses parents that minimize path cost to the DODAG root

DODAG Example

- Each node has a set of parent nodes
- A node has no knowledge about children → ONLY upward routes



Routing Loop Detection

- If a node receives a packet flagged as moving in the Upward direction, and if that packet records that the transmitter is of a lower (lesser) Rank than the receiving node, then the receiving node is able to conclude that the packet has not progressed in the Upward direction and that the DODAG is inconsistent.

Downward Routes

- RPL uses Destination Advertisement Object (DAO) messages to establish Downward routes. (for P2MP or P2P) Two modes:
  - Storing (fully stateful)
    - packet may be directed Down towards the destination by a common ancestor of the source and the destination
  - Non-Storing (fully source routed)
    - packet will travel all the way to a DODAG root before traveling Down.(因為只有root存routing info.)

A mixed mode of operation is not allowed.

Downward Routing

- Each RPL instance supporting download traffic selects one of the two models
  - "storing" model: nodes maintain routing tables
    - DAO message, including the prefixes and addresses reachable by the sending node, are sent to the parents.
    - Parents store the preferred downward routes and propagate aggregated DAOs upward.
  - "nonstoring" model: nodes use default routing upward and source routing downward
    - All downward traffic includes a source routing header specifying each hop along the path.
    - Intermediary routers don't store any routing information .
    - The DAG root calculate an optional hop by hop source routing path for each advertised destination. (IPv6 routing extensions)
    - Messages will be much longer.
    - P2P traffic is always routed to the DAG root.

Storing Mode

- DAO messages are used to advertise prefix reachability towards the leaf nodes in support of the 'down' traffic.
- DAO carries prefix information, valid lifetime and other information about the distance of the prefix.
- As each node joins the graph it will send DAO message to its parent set. Alternately, a node or root can poll the sub-DAG for DAO message through an indication in the DIO message.
- As each node receives the DAO message, it processes the prefix information and adds a routing entry in the routing table.

Storing Mode

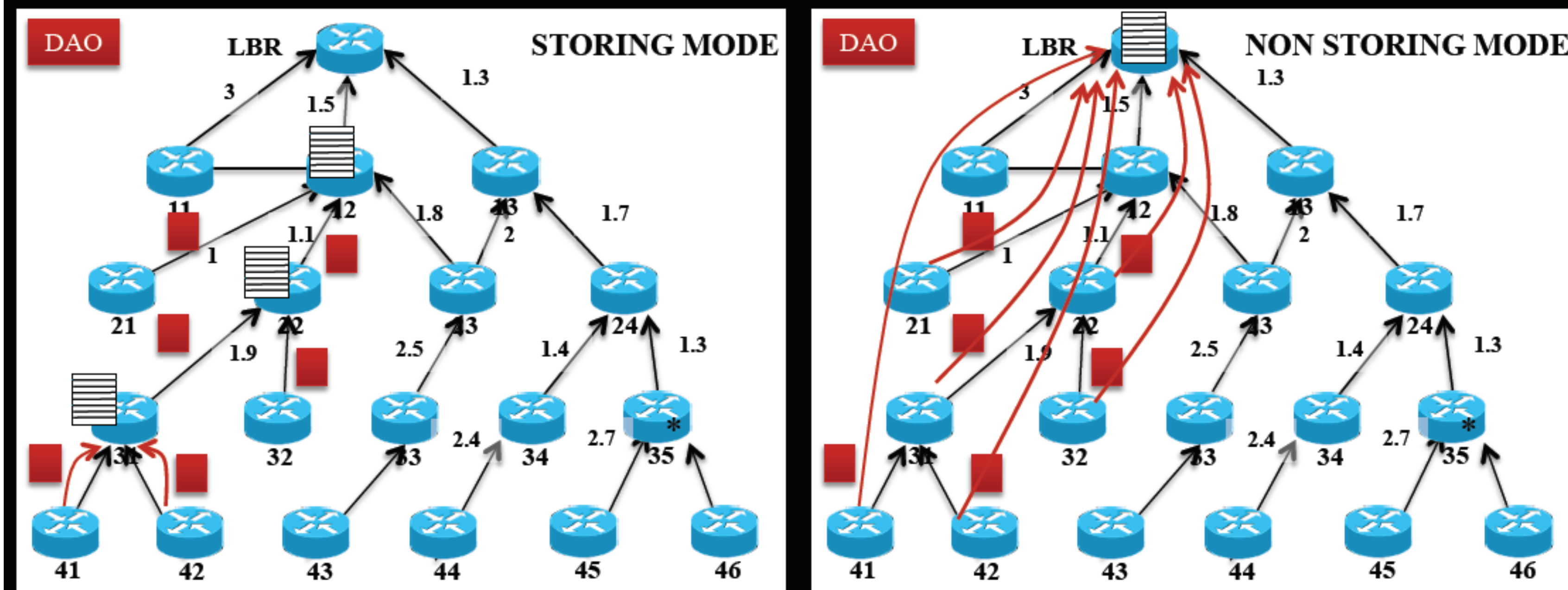
- It optionally aggregates the prefix information received from various nodes in the sub-DAG and sends a DAO message to its parent set.
- This process continues until the prefix information reaches the root and a complete path to the prefix is setup.

Non-storing Mode

- When a node A sends a packet to a node B within the RPL domain, the packet first follows the graph up to the root where the routing information is stored.
- At this point, the graph root inspects the destination, consults its routing table that contains the path to the destination (obtained from DAO messages received).
- The root “source -routes” the packet to its destination using a specific routing header for IPv6 (called RH4).

Two modes of Operation

- Two modes of operations: storing mode and non storing modes



Unicast to DAO parents

Unicast to DODAG Root (not processed by intermediate nodes)

RPL Control Messages

- RPL Control message are ICMPv6 messages
  - DAG Information Object (DIO) - carries information that allows a node to discover an RPL Instance, learn its configuration parameters and select DODAG parents
  - DAG Information Solicitation (DIS) - solicit a DODAG Information Object from a RPL node
  - Destination Advertisement Object (DAO) - used to propagate destination information upwards along the DODAG.
  - DAO-ACK: Destination Advertisement Object Acknowledgement

+ The 4 secured versions

Control Message Exchange

- Each DODAG, uniquely identified by RPLInstanceID and DODAGID, is incrementally built from the root to the leaf nodes.
  - RPL nodes send DIOs periodically via link-local multicasts.
  - Joining nodes may request DIOs from their neighbors by multicasting DIS (DODAG Information Solicitation) .
  - DTSN (Destination Advertisement Trigger Sequence Number) is a 8-bit unsigned integer set by the issuer of the message. In the storing mode, increasing DTSN is to request updated DAOs from child nodes.

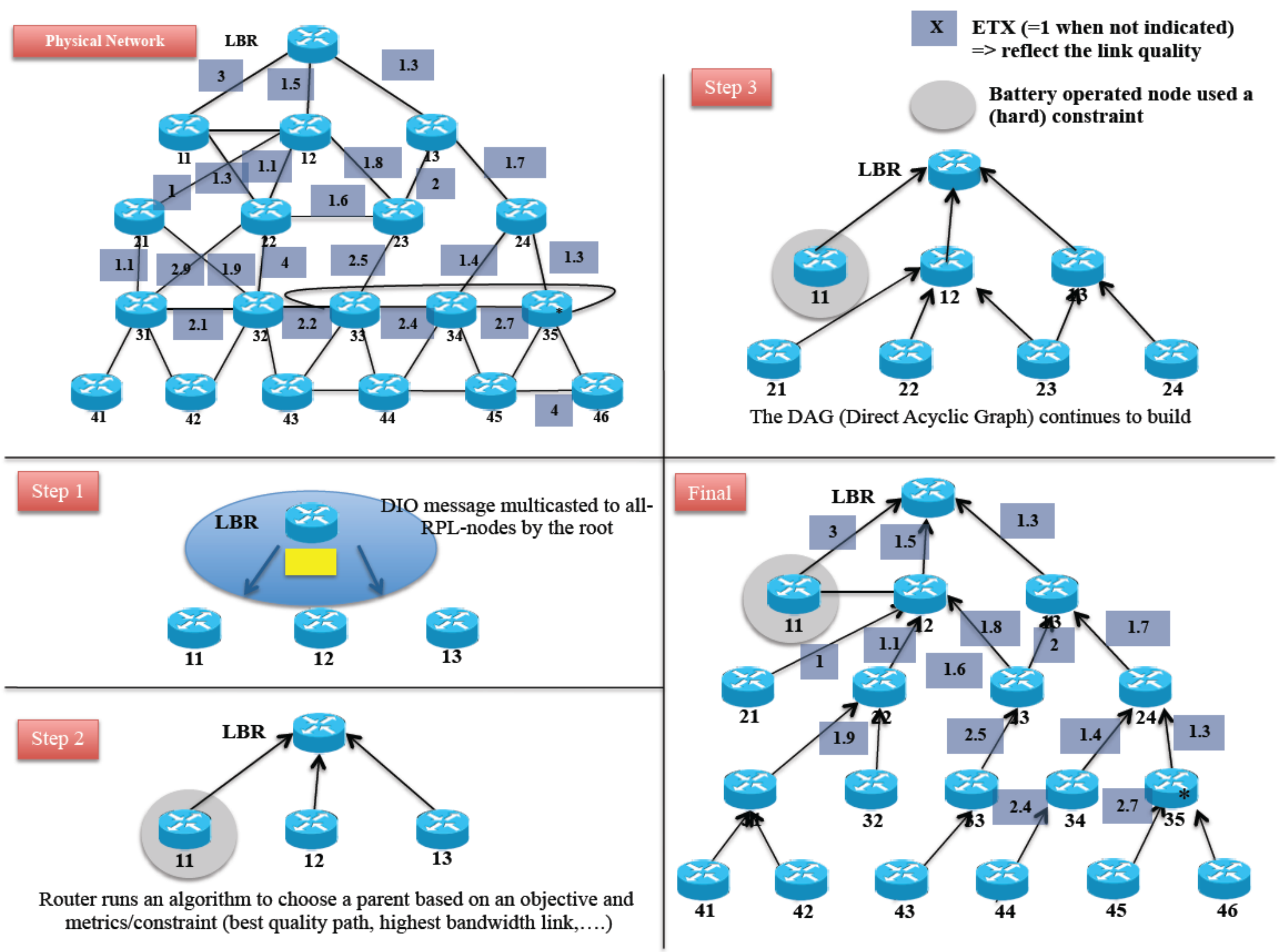
Routing Metrics in LLNs

Node Metrics	Link Metrics
Node State and Attributes Object Purpose is to reflects node workload (CPU, Memory…) "O" flag signals overload of resource "A" flag signal node can act as traffic aggregator	**Throughput Object** Currently available throughput (Bytes per second) Throughput range supported
Node Energy Object "T" flag: Node type: 0 = Mains, 1 = Battery, 2 = Scavenger "I" bit: Use node type as a constraint (include/ exclude) "E" flag: Estimated energy remaining	**Latency** Constraint - max latency allowable on path Metric - additive metric updated along path
Hop Count Object Constraint - max number of hops that can be traversed Metric - total number of hops traversed	**Link Reliability** Link Quality Level Reliability (LQL) 0=Unknown, 1=High, 2=Medium, 3=Low Expected Transmission Count (ETX) (Average number of TX to deliver a packet)
Object can be used as metric and/or constraint - metric can be additive/max/..	**Link Colour** Metric or constraint, arbitrary admin value

Specified in draft-ietf-roll-routing-metrics

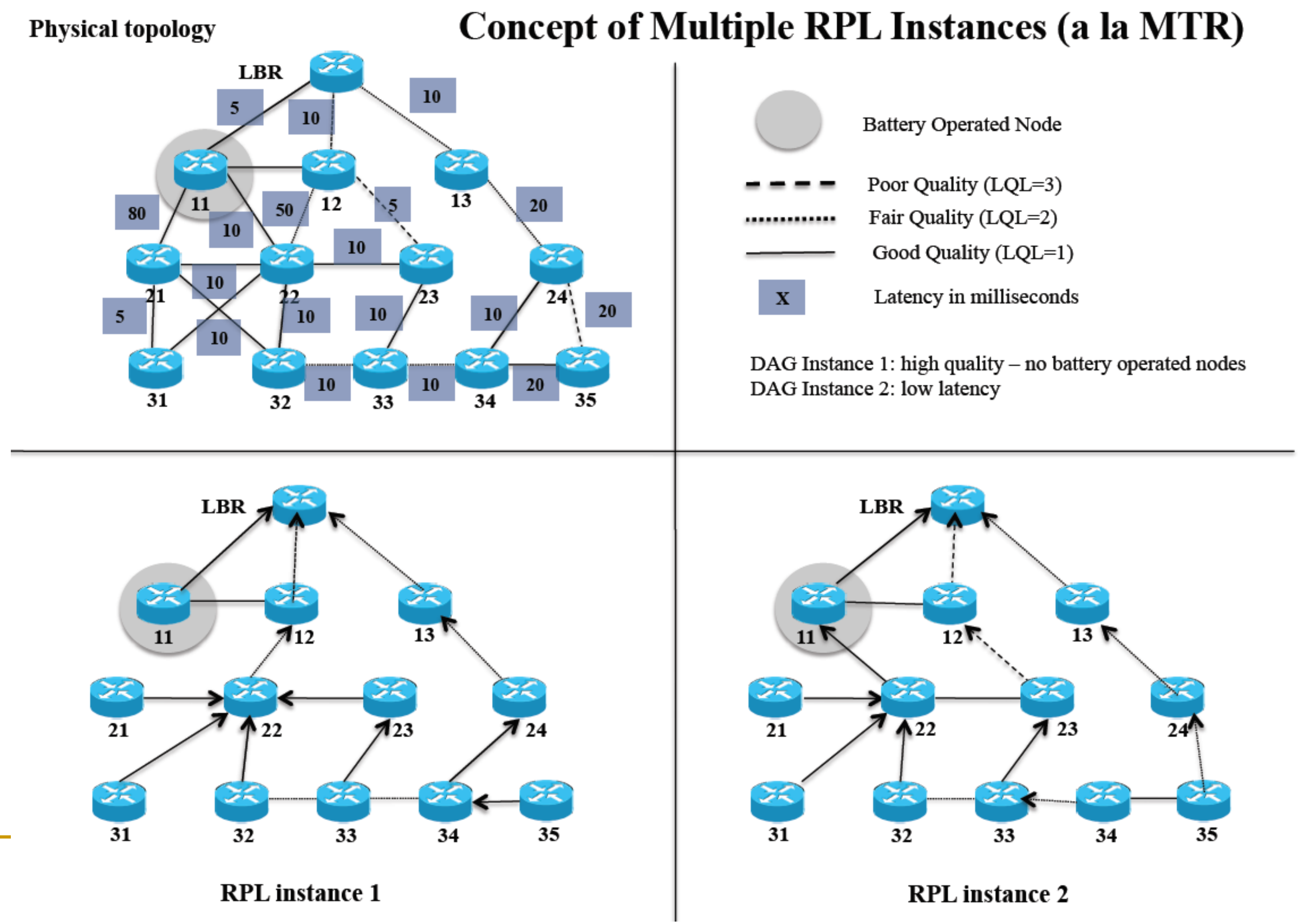
Reference: IoT Workshop RPL Tutorial, Cisco Systems

Building a DAG-Upward Routing

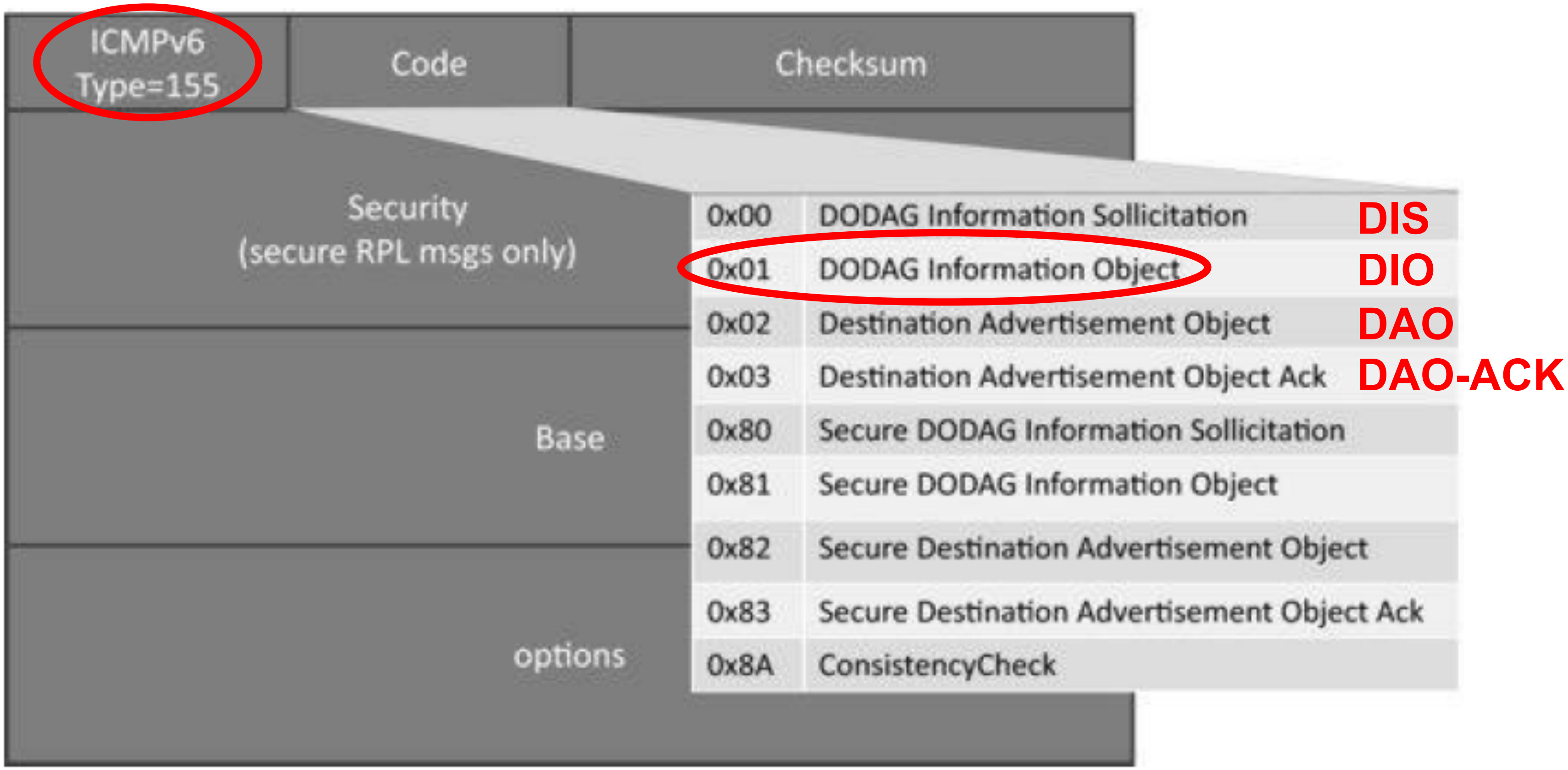


Reference: IoT Workshop RPL Tutorial, Cisco Systems

Multiple Instances of RPL



ICMPv6 RPL Control Message

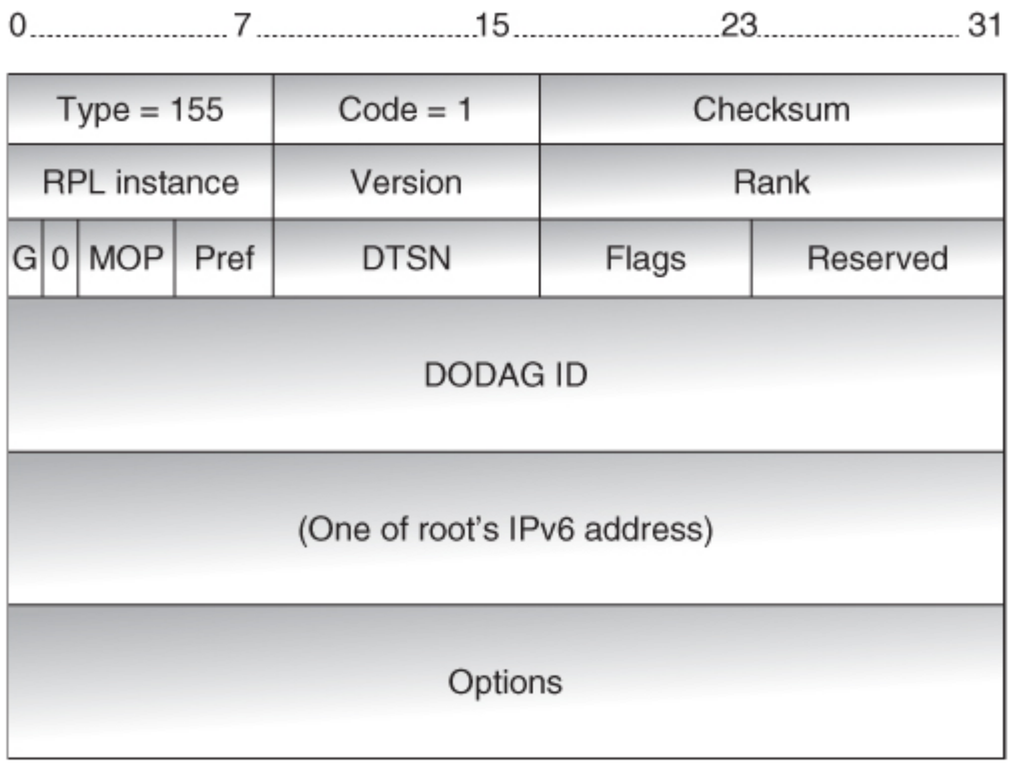


Link-local scope: source is link-local unicast and destination=link-local unicast or all-RPL-nodes(FF02::1) (for all RPL messages except DAO/ DAO-ACK in non storing mode, DIO replies to DIS)

Reference: Figure 12.7: Structure of ICMPv6 RPL control message

DODAG Information Object (DIO)

All nodes except "leaves" generate DIO periodically (controlled by Trickle).
A node uses the DIO messages received from its neighbor to determine their rank.
A node will select a set of possible parents and a preferred parent.



Trickle Timer

- RPL uses an adaptive timer mechanism called the “trickle timer”
  - The algorithm treats building of graphs as a consistency problem and makes use of trickle timers to decide when to multicast DIO messages.
  - The interval of the trickle timer increases as the network stabilizes
  - Inconsistency events: loop, join, move, etc
    - As inconsistencies are detected, the nodes reset the trickle timer and send DIOs more often.

Trickle Algorithm

- Configuration parameters
  - Imin: minimum interval size (in some unit of times)
  - Imax: maximum interval size (the base-2 log(max/min)
    - $interval = 2^{Imax} \times Imin$
  - K: redundancy constant
    - a protocol SHOULD set k and Imin such that Imin is at least two to three times as long as it takes to transmit k packets

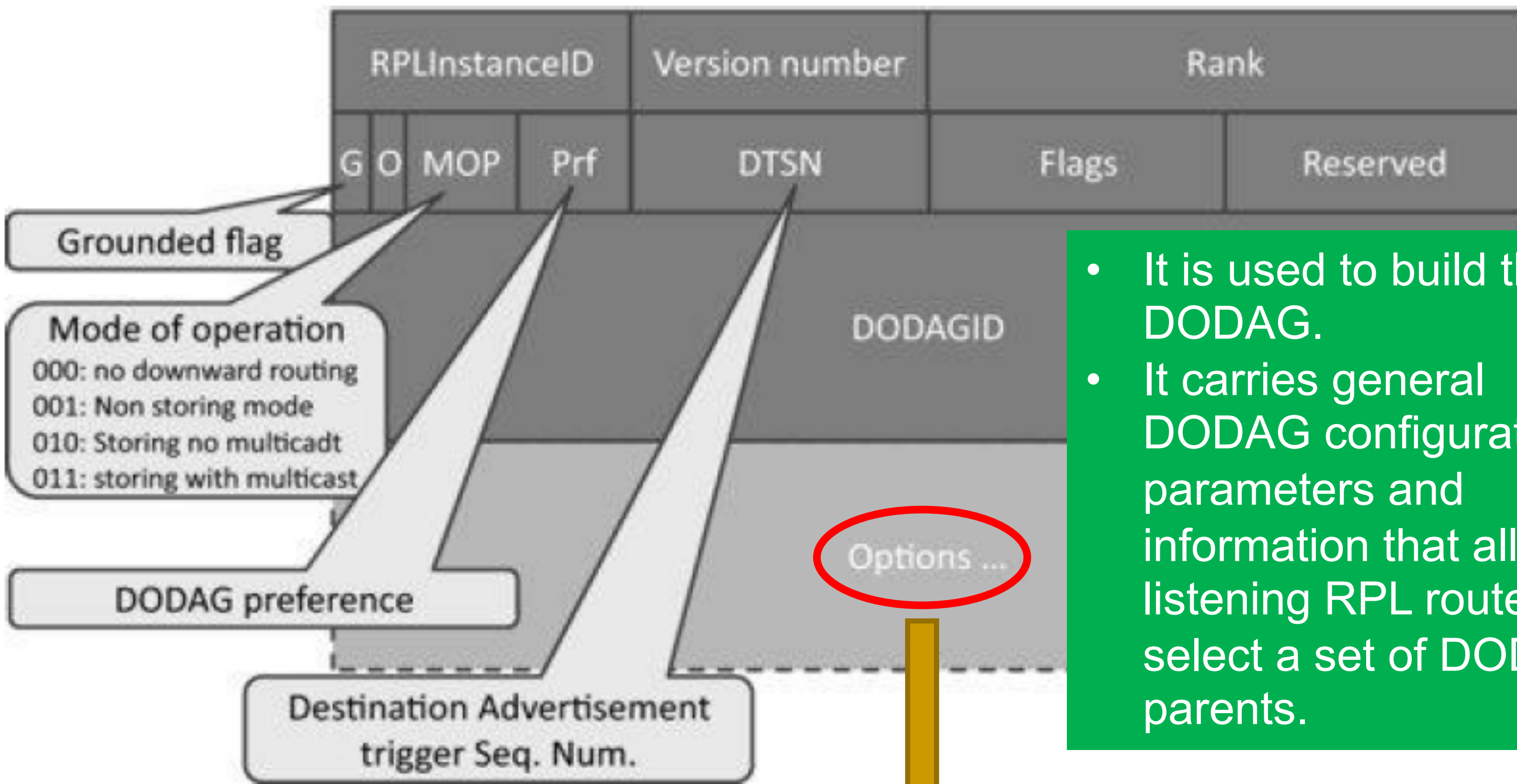
Trickle Algorithm

- Node operation parameters
  - I: the current interval size
  - t: a time within the current interval; time to send a control packet
  - c: a consistent counter

Trickle Algorithm

1. Sets I to a value in the range of [Imin, Imax]
2. Starts an interval; resets c to 0; sets t to a random number from the range [I/2, I)
3. If receives a consistent transmission, c++
4. At time t, transmits a control packet iff c<k
5. When I expires, I=Max(2xI, Imax)
6. If receives an inconsistent transmission and I>Imin, resets I=Imin, starts a new interval; resets c to 0; t=random[I/2, I)

DODAG Information Object (DIO)



- It is used to build the DODAG.
- It carries general DODAG configuration parameters and information that allows listening RPL routers to select a set of DODAG parents.

Next Slide

Reference: Figure 12.8: RPL DIO base object (followed by options)

Options of RPL DIO

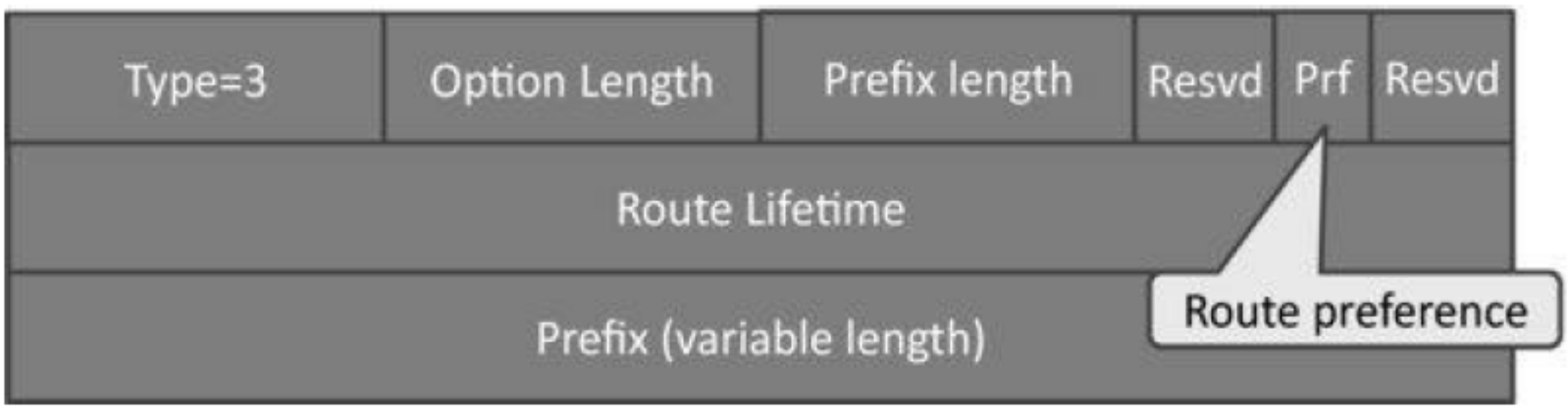
Option:

Type	Length	Data …

- Option types
  - 0x02: metric container option
    - Estimate the cost to reach destinations
  - 0x03: routing information option
    - Contains the same fields as the IPv6 neighbor discovery route information option
  - 0x04: DODAG information option
    - Constrain the rank a node can advertise when reattaching to a DODAG, or
    - The default lifetime of all RPL routes
  - 0x08: prefix information option
    - Contains the same fields as the IPv6 neighbor discovery prefix option

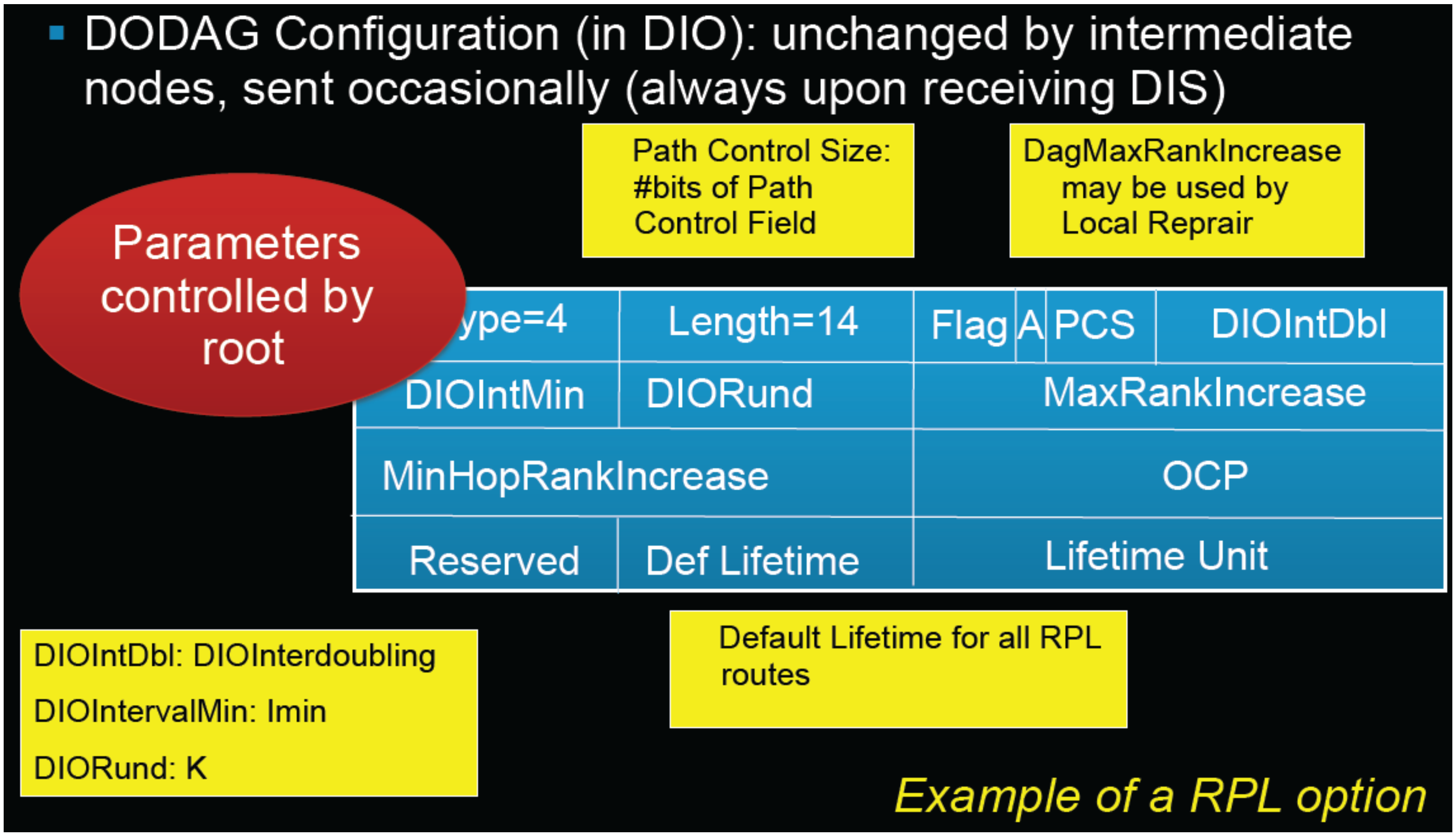
Routing Information Option (Type=3)

- RPL nodes send DIOs periodically via link-local multicasts
- Joining nodes may request DIOs from their neighbors by multicasting DIS



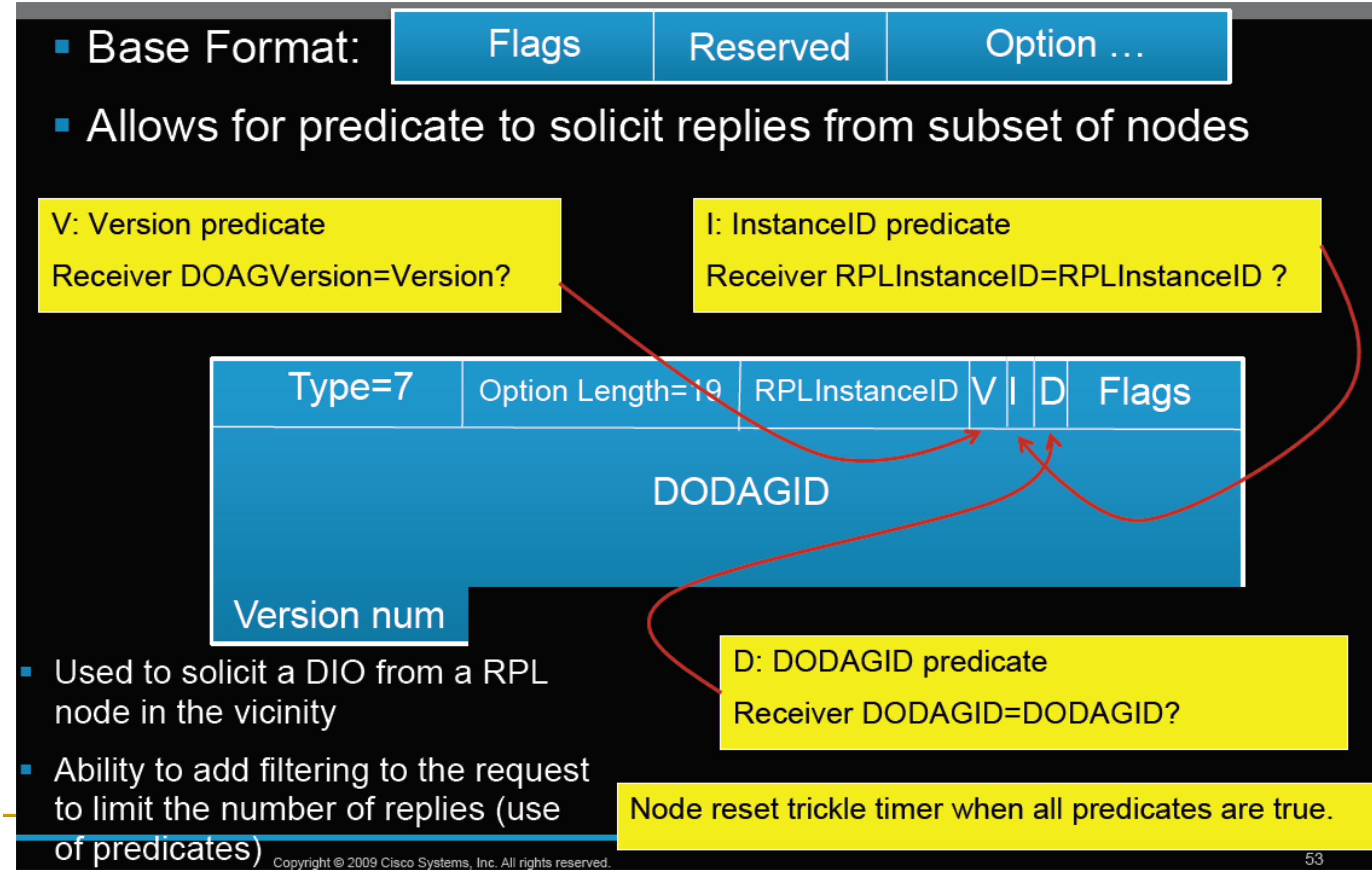
Reference: Figure 12.9: RPL Route Information option (Type=3)

DODAG Information Option (Type=4)



Reference: Figure 12.8: RPL DIO base object (followed by options)

RPL DIS Message

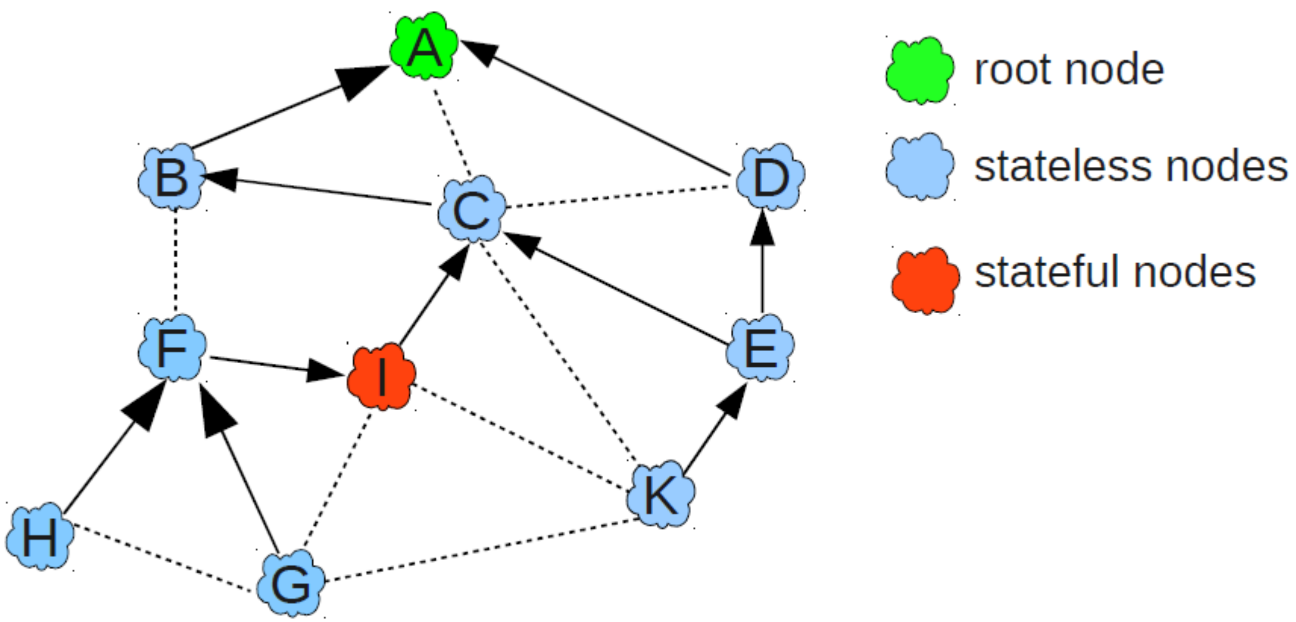


Downward Routes and Destination Advertisement

- Nodes inform parents of their presence and reachability to descendants by sending a DAO message
- Node capable of maintaining routing state →aggregate routes
- Node incapable of maintaining routing state → attach a next-hop address to the reverse route stack contained within the DAO message

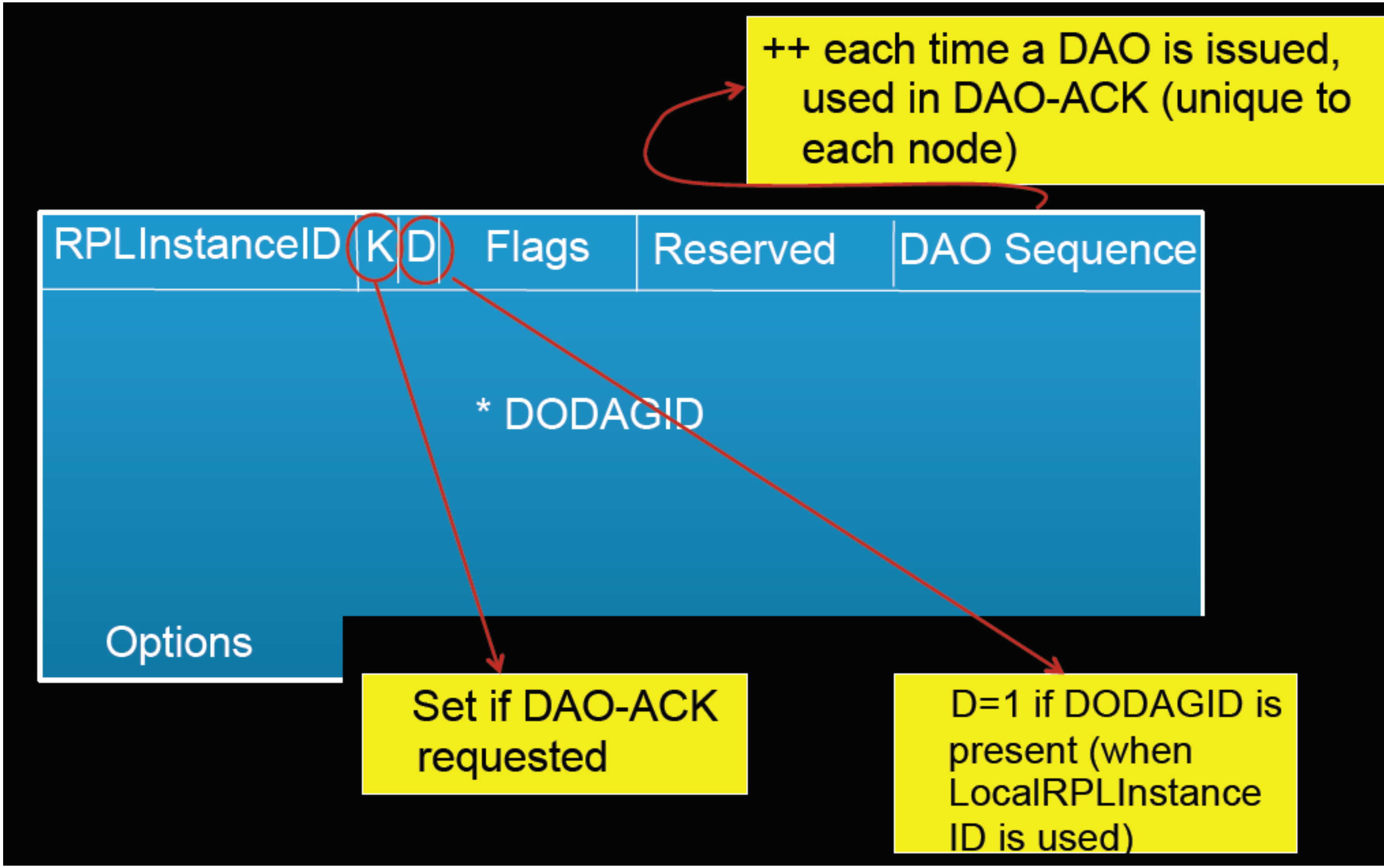
Destination Advertisement - Example

- H sends a DAO message to F indication the availability of H, F adds the next-hop and forwards the message to I
- G sends a DAO message to F indication the availability of G, F adds the next-hop and forwards the message to I
- F sends a DAO message to I indication the availability of F
- I aggregates the routes and sends a DAO advertising (F-I)



Source: Siarhei Kuryla, Networks and Distributed Systems seminar, March 2010

DAO Message



++ each time a DAO is issued, used in DAO-ACK (unique to each node)
RPLInstanceID
K
D
Flags
Reserved
DAO Sequence
* DODAGID
Options
Set if DAO-ACK requested
D=1 if DODAGID is present (when LocalRPLInstance ID is used)

Conclusion

- Optimized for many-to-one and one-to-many traffic patterns
- Routing state is minimized: stateless nodes have to store only instance(s) configuration parameters and a list of parent nodes
- Takes into account both link and node properties when choosing paths
- Link failures does not trigger global network re-optimization

CoAP: Constrained Application Protocol

黃仁竑 教授
國立中正大學資工系

CoAP

- The Constrained Application Protocol (CoAP) is defined by IETF CoRE WG for the manipulation of resources on a device that is on the constrained IP networks.

What CoAP is (and is not)

- CoAP is
  - A RESTful protocol
  - Both synchronous and asynchronous
  - For constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP
- CoAP is not
  - A replacement for HTTP
  - General HTTP compression
  - Separate from the web

CoRE Documents

Number	Title	Date	Status
RFC 7252 RFC 7959	The Constrained Application Protocol (CoAP)	June 2014 August 2016	Proposed Standard
RFC 7390	Group Communication for the Constrained Application Protocol (CoAP)	October 2014	Experimental
RFC 7641	Observing Resources in the Constrained Application Protocol (CoAP)	September 2015	Proposed Standard
RFC 7650	A Constrained Application Protocol (CoAP) Usage for REsource LOcation And Discovery (RELOAD)	September 2015	Proposed Standard
RFC 8323	CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets	February 2018	Proposed Standard

Constrained IP Networks

- A constrained IP network has limited packet sizes, may exhibit a high degree of packet loss, and may have a substantial number of devices that may be powered off at any point in time but periodically "wake up" for brief periods of time.
- These networks and the nodes within them are characterized by severe limits on throughput, available power, and particularly on the complexity that can be supported with limited code size and limited RAM per node.
- Low-Power Wireless Personal Area Networks (LoWPANs) are an example of this type of network. Constrained networks can occur as part of home and building automation, energy management, and the Internet of Things.

Source: IETF CoRE WG

Devices on Constrained Networks

- The general architecture consists of nodes on the constrained network, called Devices, that are responsible for one or more Resources that may represent sensors, actuators, combinations of values or other information.
- Devices send messages to change and query resources on other Devices.
- Devices can send notifications about changed resource values to Devices that have subscribed to receive notification about changes.
- A Device can also publish or be queried about its resources.

Source: IETF IPv6 WG

Application Scope of CoAP

- CoAP targets the type of operating environments defined in the ROLL and 6LOWPAN working groups which have additional constraints compared to normal IP networks, but the CoAP protocol will also operate over traditional IP networks.
- This includes applications to monitor simple sensors (e.g. temperature sensors, light switches, and power meters), to control actuators (e.g. light switches, heating controllers, and door locks), and to manage devices.

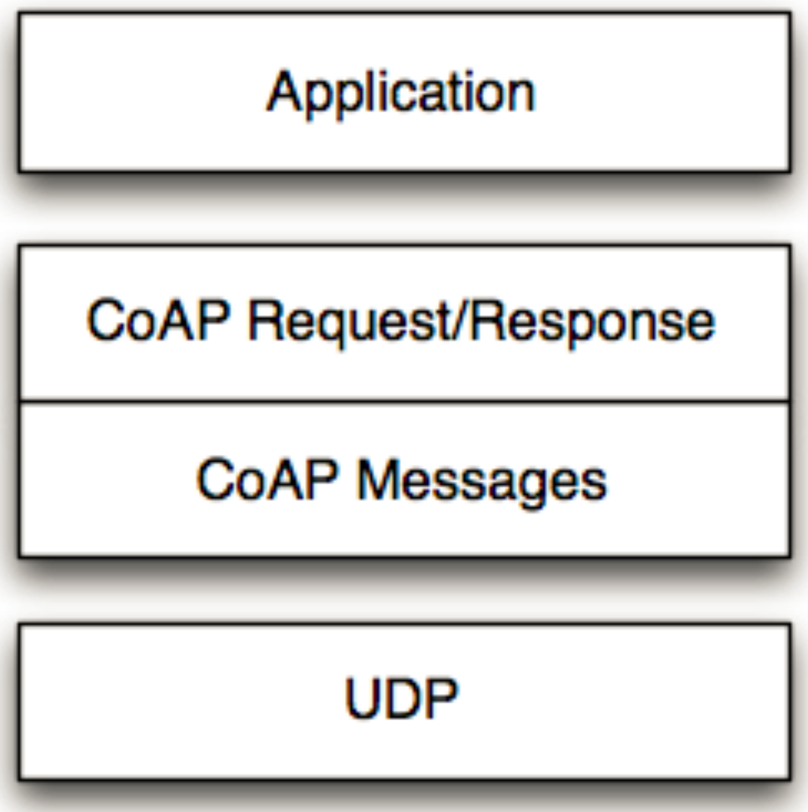
Source: IETF IPv6 WG

CoAP vs. HTTP

- Like HTTP, the CoAP is a way of structuring REST communications but optimized for M2M applications.
- TCP and HTTP are considered too heavy for 6LowPAN devices such as sensors. CoAP is thus based on UDP and a compressed simplified message exchange.
  - NB: RFC 8323 extends CoAP over TCP/TLS

```
+--------------------------------+
|          Application           |
+--------------------------------+
+--------------------------------+
|  Requests/Responses/Signaling  |
|--------------------------------|
|        Message Framing         |
+--------------------------------+
|      Reliable Transport        |
+--------------------------------+
```

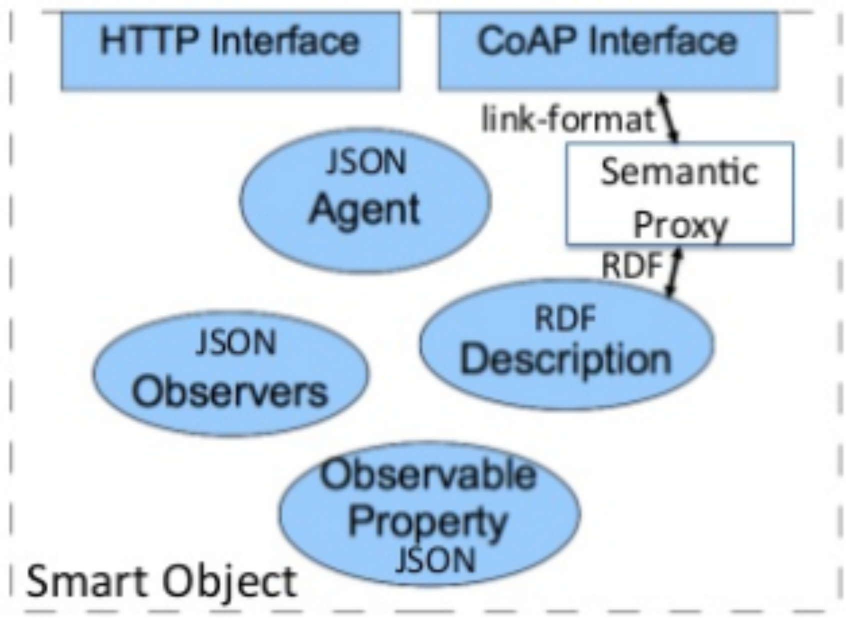
CoAP RESTful Applications



Source: IETF IPv6 WG

CoAP Server

CoAP Server Endpoint



- Compatible Object Models
- Observer pattern
- Standard JSON Resources
- Semantic Proxy exposes stored RDF via core-link-format GET/POST operations

https://www.slideshare.net/michaeljohnkoster/object-models-for-interoperability

CoAP Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Ver - Version (1) 2-bit

T - Transaction Type 2-bit

- CON (0) – Confirmable
- NON (1) – Non-Confirmable
- ACK (2) – Acknowledgement
- RST (3) – Reset

Token Length (TKL): 4-bit

Code – 3-bit class, 5-bit detail ("c.dd“)

Class: request (0), success response (2), client error response (4), server error response (5) (see next page for details)

Message ID - Identifier for matching responses

Token - used to correlate requests and responses.

CoAP Code and Message ID

- Code: compressed from HTTP text representation (3 numbers) into one byte
  - HTTP requests =>first 3 bits 000; next five bits 0~32 (1: GET; 2: POST; 3:PUT; 4:DELETE etc.)
  - HTTP responses=>first 3 bits 001-101 (1~5) representing the first number of 2xx: success, 4xx: client error, 5xx: server error; xx represented by next five bits 00001~01111 (1~15 used only; e.g. with HTTP response 201 is represented as 010-00001; HTTP response 400 is represented as 100-00000 etc.)
- Message ID: used in the acknowledgment process to tie a request with a response.

Method: 0.XX

0. EMPTY
1. GET
2. POST
3. PUT
4. DELETE
5. FETCH
6. PATCH
7. iPATCH

Success : 2.XX

1. Created
2. Deleted
3. Valid
4. Changed
5. Content

31. Continue

Signaling Codes : 7.XX

0. Unassigned
1. CSM
2. Ping
3. Pong
4. Release
5. Abort

CoAP Options

```
  0   1   2   3   4   5   6   7
+---------------+---------------+
|               |               |
|  Option Delta | Option Length |   1 byte
|               |               |
+---------------+---------------+
\                               \
/         Option Delta          /   0-2 bytes
\          (extended)           \
+-------------------------------+
\                               \
/         Option Length         /   0-2 bytes
\          (extended)           \
+-------------------------------+
\                               \
/                               /
\                               \
/         Option Value          /   0 or more bytes
\                               \
/                               /
\                               \
+-------------------------------+
```

Option Delta - Difference between this option type and the previo

Length - Length of the option value (0-270)

Value - The value of Length bytes immediately follows Length

Options

- CoAP defines a single set of options that are used in both requests and responses:
  - Content-Format
  - ETag
  - Location-Path
  - Location-Query
  - Max-Age
  - Proxy-Uri
  - Proxy-Scheme
  - Uri-Host
  - Uri-Path
  - Uri-Port
  - Uri-Query
  - Accept
  - If-Match
  - If-None-Match
  - Size1

Examples of Option types

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical
U=Unsafe
N=NoCacheKey
R=Repeatable

4 (Etag) entity tag: proxy can assign entity tags to responses it sends to a client

14 (Max-Age) gives the maximum duration in seconds for which the answer may be cached.

19 (Token) is used to match a response with a request.

6 (Observe) is used to receive regularly updated values from the server.

RFC 7641: Observing Resources in CoAP.

23, 27 (Block2, Block1) is used to transfer blocks of responses (Work in Progress)

Observe Option

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

CoAP Methods

- CoAP makes use of GET, PUT, POST, and DELETE methods in a similar manner to HTTP.
- New methods can be added, and do not necessarily have to use requests and responses in pairs.
  - For example: OBSERVE (embedded in GET method)

CoAP URI

- coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]
  - coap://example.com:5683/˜sensors/temp.xml
  - coap://EXAMPLE.com/%7Esensors/temp.xml
- coaps-URI = "coaps:" "//" host [":" port] path-abempty ["?" query]

Messaging Model

- Reliable Message Transmission:
  - Reliability is provided by marking a message as Confirmable (CON).
  - A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message.

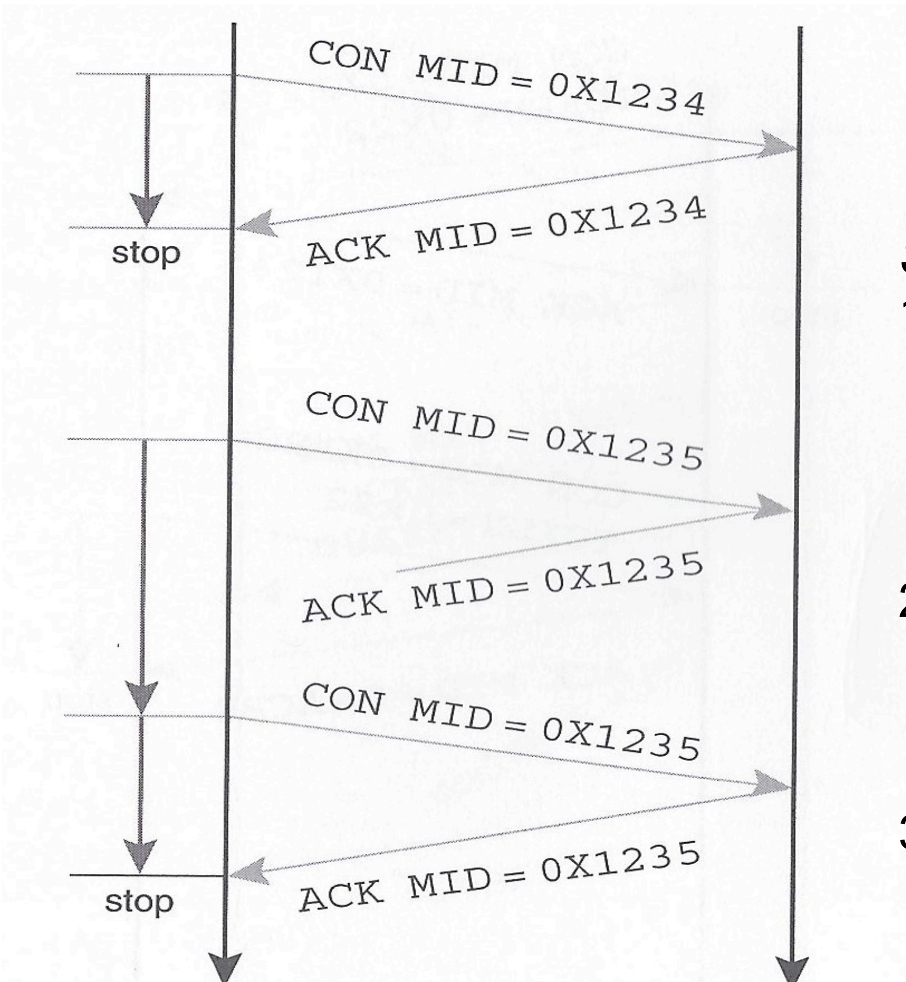
```
Client              Server
   |                   |
   |   CON [0x7d34]    |
   +------------------>|
   |                   |
   |   ACK [0x7d34]    |
   |<------------------+
   |                   |
```

Messaging Model

- Unreliable Message Transmission:
  - A message that does not require reliable transmission can be sent as a Non-confirmable message (NON).
  - These are not acknowledged.

```
Client              Server
   |                   |
   |   NON [0x01a0]    |
   +------------------>|
   |                   |
```

Example 1 of CoAP Requests



Synchronous Message Exchange

1. A CONfirmable message followed by ACKowledgement piggybacked with the response in the same Message ID (MID).

2. When ACKnowledgment was lost, Client's timer expires and it resends the message.

3. Exponential back-off between retransmissions.

Source: M2M Communications: A Systems Approach, Wiley, 2012

Example 2a of CoAP Requests

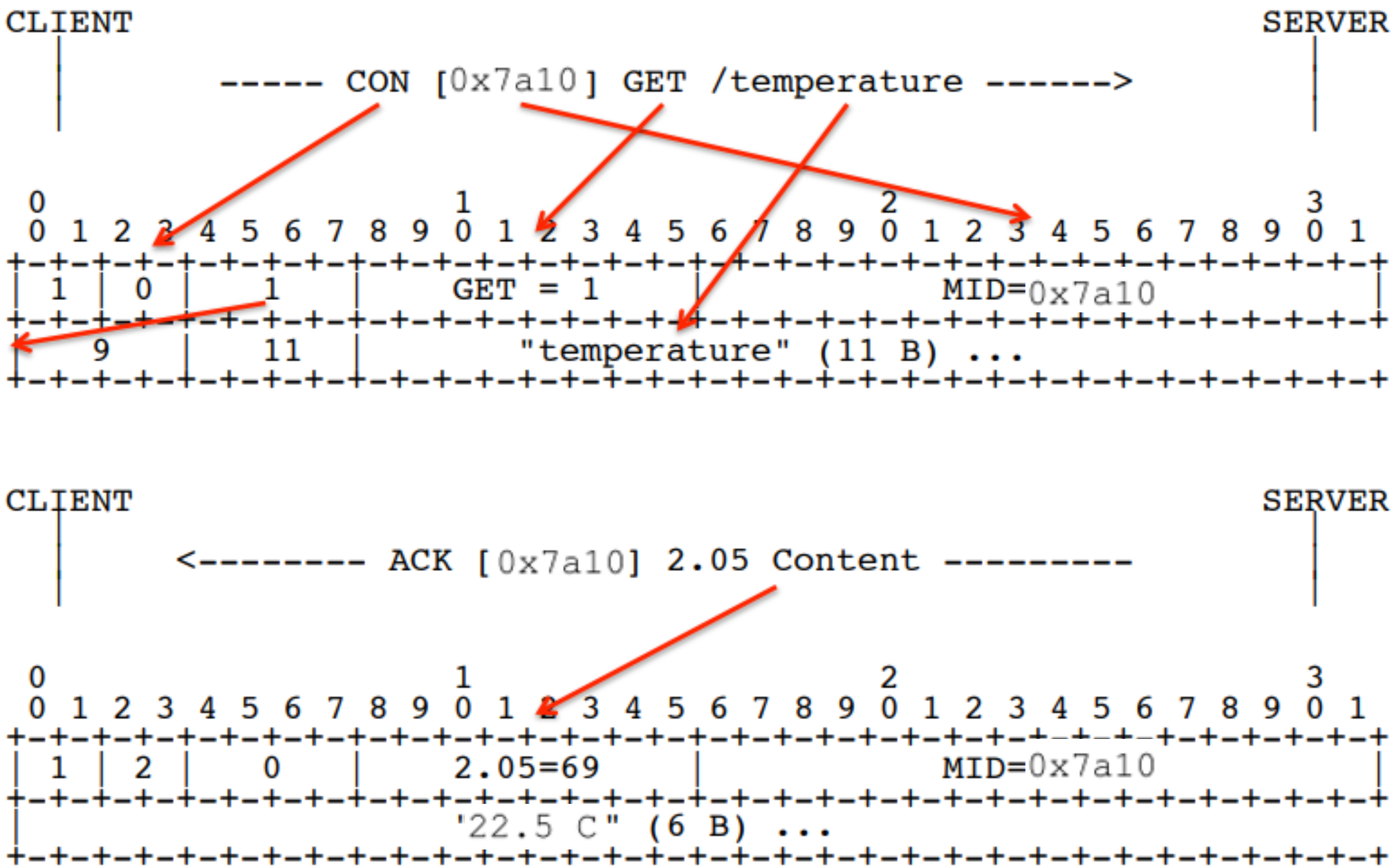
```
Client                  Server
   |                      |
   |   CON [0x7a10]       |
   | GET /temperature     |
   |   (Token 0x73)       |
   +------------------->|
   |                      |
   |   ACK [0x7a10]       |
   |<-------------------+
   |                      |
 ... Time Passes  ...
   |                      |
   |   CON [0x23bb]       |
   |   2.05 Content       |
   |   (Token 0x73)       |
   |    "22.5 C"          |
   |<-------------------+
   |                      |
   |   ACK [0x23bb]       |
   +------------------->|
   |                      |
```

Asynchronous Message Exchange

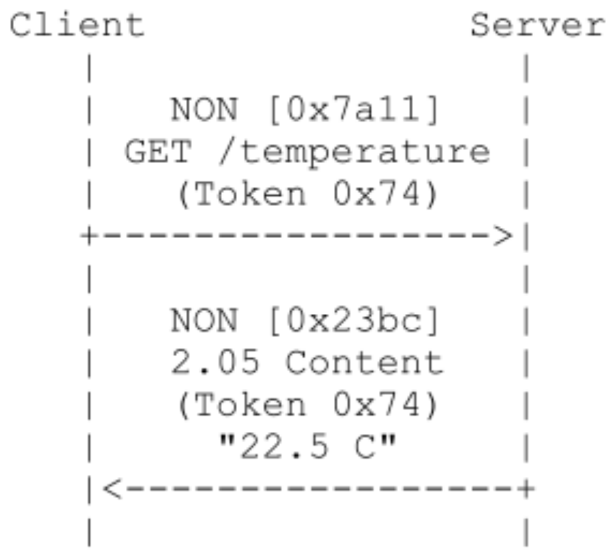
1. A CONfirmable message with TOKEN option can be acknowledged immediately with an Empty Acknowledgement.

2. When the response is available, it can be returned in a new CON message with the same TOKEN ID.

Message Details



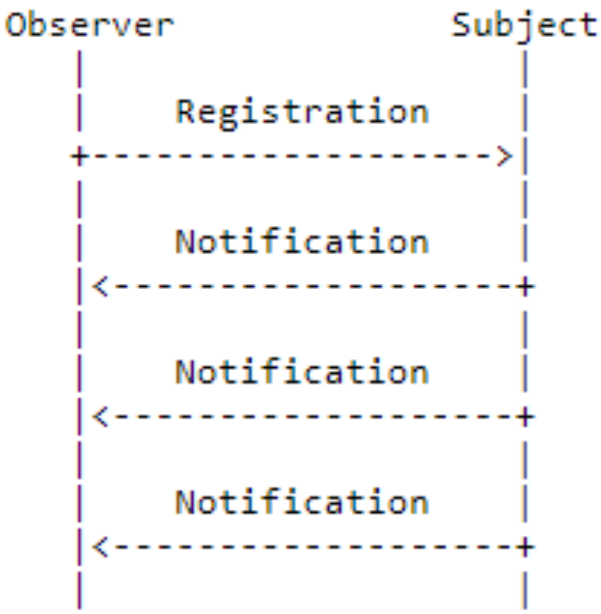
Example 2b of CoAP Requests



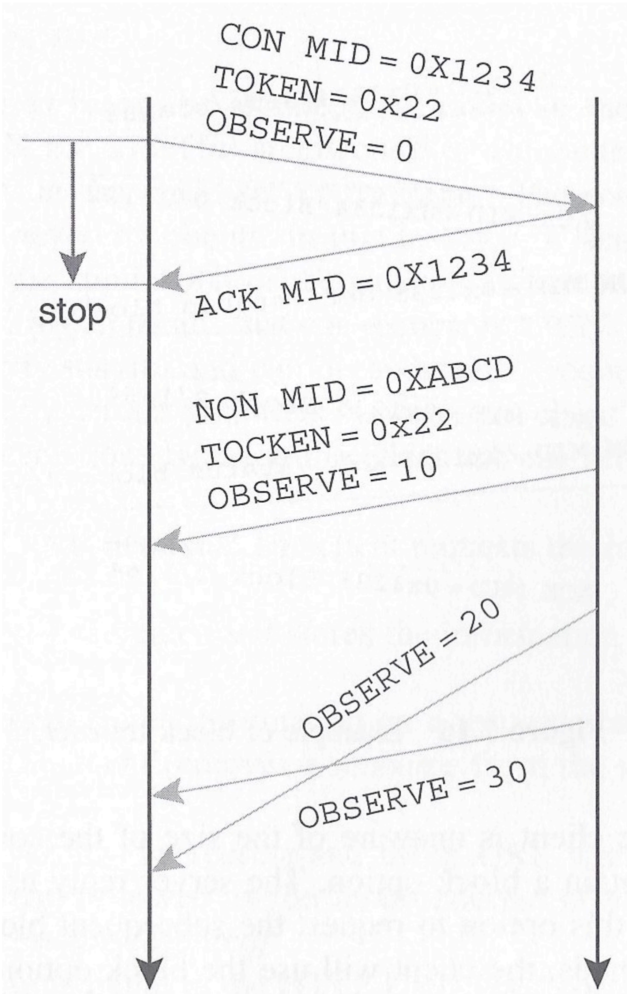
Asynchronous Message Exchange

1. A Non-confirmable message with TOKEN then the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message.

OBSERVE Design Pattern



Example 3a of CoAP Requests (OBSERVE)



Periodic response from a server

1. A CONfirmable message from the client contains OBSERVE option asking periodic responses from the server.

2. The server send NON responses with the same TOKEN ID.

3. OBSERVE is the response will be increased to indicate the order of the response.

4. The client will ignore OBSERVE=20 since it arrives later than OBSERVE=30.

5. Either client or server can terminate the process.

Source: M2M Communications: A Systems Approach, Wiley, 2012

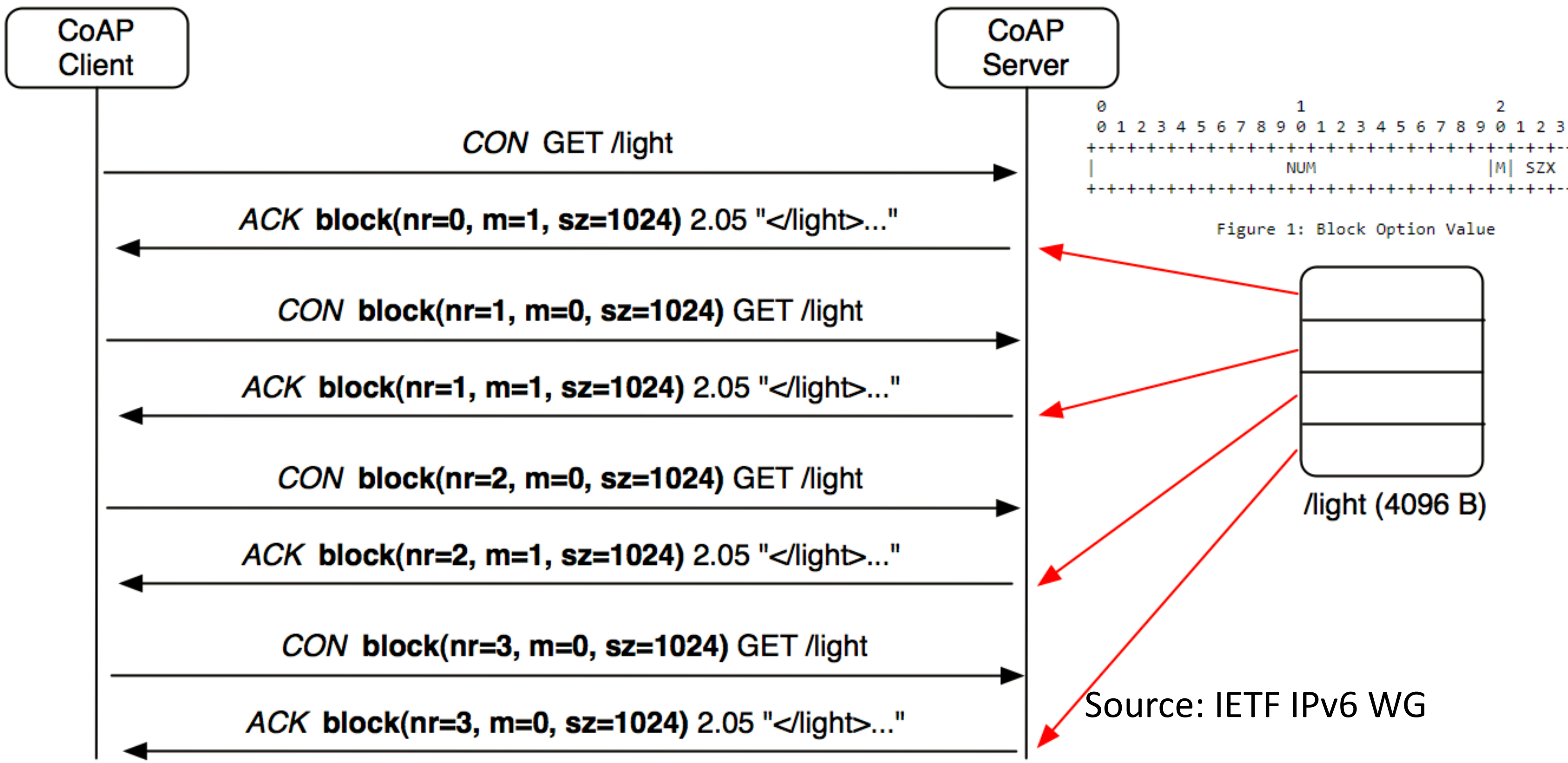
Example 3b of CoAP Requests (OBSERVE)

```
Client                 Server
   |                      |
   |  GET /temperature    |
   |    Token: 0x4a       |   Registration
   |  Observe: 0          |
   +--------------------->|
   |                      |
   |    2.05 Content      |
   |    Token: 0x4a       |   Notification of
   |  Observe: 12         |   the current state
   |  Payload: 22.9 Cel   |
   |<---------------------+
   |                      |
   |    2.05 Content      |
   |    Token: 0x4a       |   Notification upon
   |  Observe: 44         |   a state change
   |  Payload: 22.8 Cel   |
   |<---------------------+
   |                      |
   |    2.05 Content      |
   |    Token: 0x4a       |   Notification upon
   |  Observe: 60         |   a state change
   |  Payload: 23.1 Cel   |
   |<---------------------+
   |                      |
```

A notification can be confirmable or non-confirmable.

The server may send a notification in a confirmable CoAP message to request an acknowledgement from the client.

Example 4 of CoAP Requests

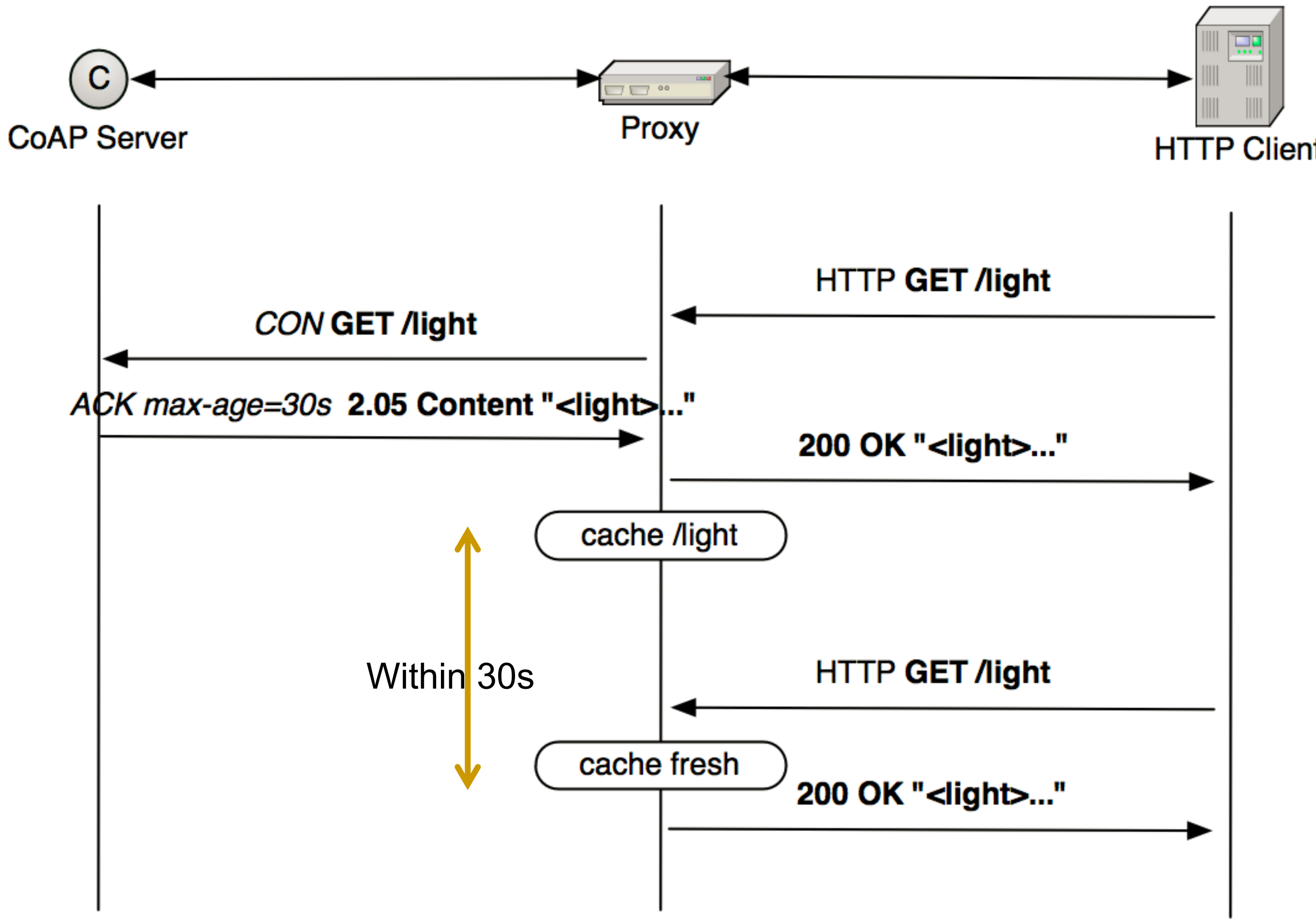


Source: IETF IPv6 WG

Block Transfer from Server to Client

1. A CONfirmable message from Client to get information.
2. Server indicates it has block of information to send.
3. Client then asks for more blocks of information.

Proxying and Caching



Source: IETF IPv6 WG

CoAP Caching Model

Cacheability determined by response code

- Freshness model
  - Max-Age option indicates cache lifetime
- Validation model
  - Validity checked using the Etag Option (http://en.wikipedia.org/wiki/HTTP_ETag)

Cacheability of CoAP responses depends on the Response Code.

CoAP Resource Discovery

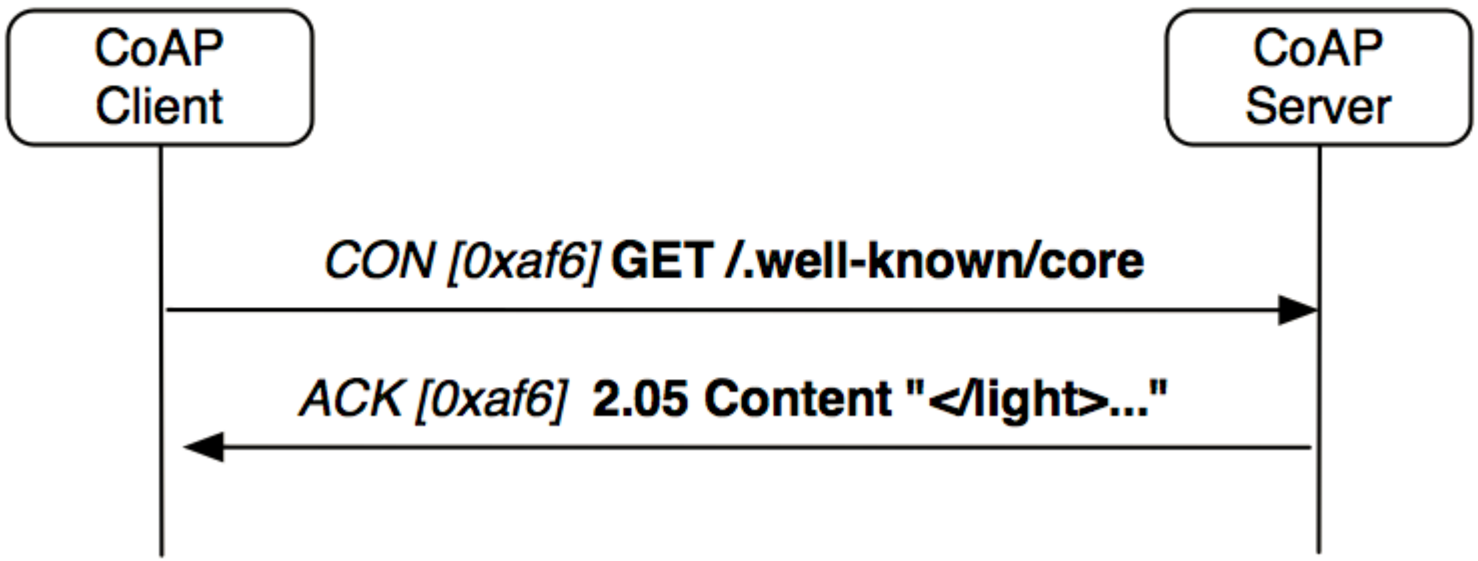
- Resource Discovery with CoRE Link Format
  - Discovering the links hosted by CoAP servers
  - GET /.well-known/core
  - Returns a link-header style format based on RFC5988 including URL, relation, type, interface, content-type etc.
  - See RFC 6690: Constrained RESTful Environments (CoRE) Link Format

coap://224.0.1.187/oc/core?rt=alpha.light

RFC 6690 Link Format

```
Link            = link-value-list
link-value-list = [ link-value *[ "," link-value ]]
link-value     = "<" URI-Reference ">" *( ";" link-param )
link-param     = ( ( "rel" "=" relation-types )
               / ( "anchor" "=" DQUOTE URI-Reference DQUOTE )
               / ( "rev" "=" relation-types )
               / ( "hreflang" "=" Language-Tag )
               / ( "media" "=" ( MediaDesc
                     / ( DQUOTE MediaDesc DQUOTE ) ) )
               / ( "title" "=" quoted-string )
               / ( "title*" "=" ext-value )
               / ( "type" "=" ( media-type / quoted-mt ) )
               / ( "rt" "=" relation-types )
               / ( "if" "=" relation-types )
               / ( "sz" "=" cardinal )
               / ( link-extension ) )
```

Example of Resource Discovery



</light>;rt="Illuminance";ct=0,
</s/maastr.xml>;title="Maastricht weather";ct=1,
</s/maastr/temp>;title="Temperature in Maastrich";ct=1,
</s/oulu.xml>;title="Oulu weather";ct=1,
</s/oulu/temp>;title="Temperature in Oulu";ct=1,
</s/temp>;rt="Temperature";ct=0

Source: IETF IPv6 WG

Resource Type 'rt' Attribute Interface Description 'if' Attribute

Example of Resource Discovery

REQ: GET /.well-known/core
RES: 2.05 Content
</sensors/temp>;if="sensor",
</sensors/light>;if="sensor“

REQ: GET /.well-known/core
RES: 2.05 Content
</sensors>;ct=40

REQ: GET /sensors
RES: 2.05 Content
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"

Source: IETF IPv6 WG

Summary

- CoAP is applicable to any IP networks.
- Open source software available for these protocols.
  - http://coapy.sourceforge.net/index.html
  - CoAPy: Constrained Application Protocol in Python

Q&A

MQTT

國立中正大學資工系 黃仁竑教授

What is MQTT ?

- MQTT = MQ Telemetry Transport
- MQTT protocol is a lightweight publish/subscribe protocol flowing over TCP/IP for remote sensors and control devices through low bandwidth, unreliable or intermittent communications.
- MQTT was developed by Andy Stanford-Clark of IBM, and Arlen Nipper of Cirrus Link Solutions more than a decade ago.
- ISO/IEC 20922:2016 Message Queuing Telemetry Transport (MQTT) v3.1.1. (Draft of v5.0 published in July, 2017)
- Client libraries are available in almost all popular languages now.
  - https://eclipse.org/paho/
- The current MQTT specification is available at:
  - http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
  - http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Candidate OASIS Standard 01, 31 October 2018

Projects that Implement MQTT

- Amazon Web Services announced Amazon IoT based on MQTT in 2015
- Microsoft Azure IoT Hub uses MQTT as its main protocol for telemetry messages
- Node-RED supports MQTT nodes as of version 0.14
- Facebook has used aspects of MQTT in Facebook Messenger for online chat
- The EVRYTHNG IoT platform uses MQTT as an M2M protocol for millions of connected products.

Features of MQTT

- It supports publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications
- A messaging transport that is agnostic to the content of the payload
- Three qualities of service for message delivery:
  - "At most once" , where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
  - "At least once", where messages are assured to arrive but duplicates may occur.
  - "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.
- A small transport overhead and protocol exchanges minimized to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs. (Last Will and Testament)

Last Will and Testament

- The protocol provides a method for detecting when clients close their connections improperly by using keep-alive packets.
  - So when a client crashes or it's network goes down, the broker can take action.

- Clients can send a Last Will and Testament (LWT) message to the broker at any point. When the broker detects the client has gone offline (without closing their connection), it will send out the saved LWT message on a specified topic, thus letting other clients know that a node has gone offline unexpectedly.

MQTT Pub/Sub Protocol

- MQ Telemetry Transport (MQTT) is a lightweight broker-based publish/subscribe messaging protocol.
- MQTT is designed to be open, simple, lightweight and easy to implement.
  - These characteristics make MQTT ideal for use in constrained environments, for example in IoT.
    - Where the network is expensive, has low bandwidth or is unreliable
    - When run on an embedded device with limited processor or memory resources;
- A small transport overhead (the fixed-length header is just 2 bytes), and protocol exchanges minimized to reduce network traffic

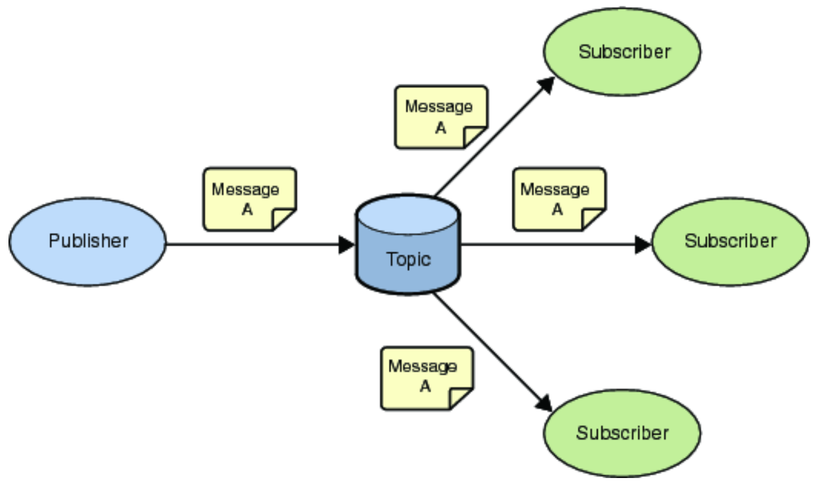
Source: MQTT V3.1 Protocol Specification, IBM, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

Suitable for Constrained Networks

- Protocol compressed into bit-wise headers and variable length fields.
- Smallest possible packet size is 2 bytes
- Asynchronous bidirectional "push" delivery of messages to applications (no polling)
- Client to server and server to client
- Supports always-connected and sometimes-connected models
- Provides Session awareness
- Configurable keep alive providing granular session awareness
- "Last will and testament" enable applications to know when a client goes offline abnormally
- Typically utilizes TCP based networks e.g. , Webscokets
- Tested on many networks

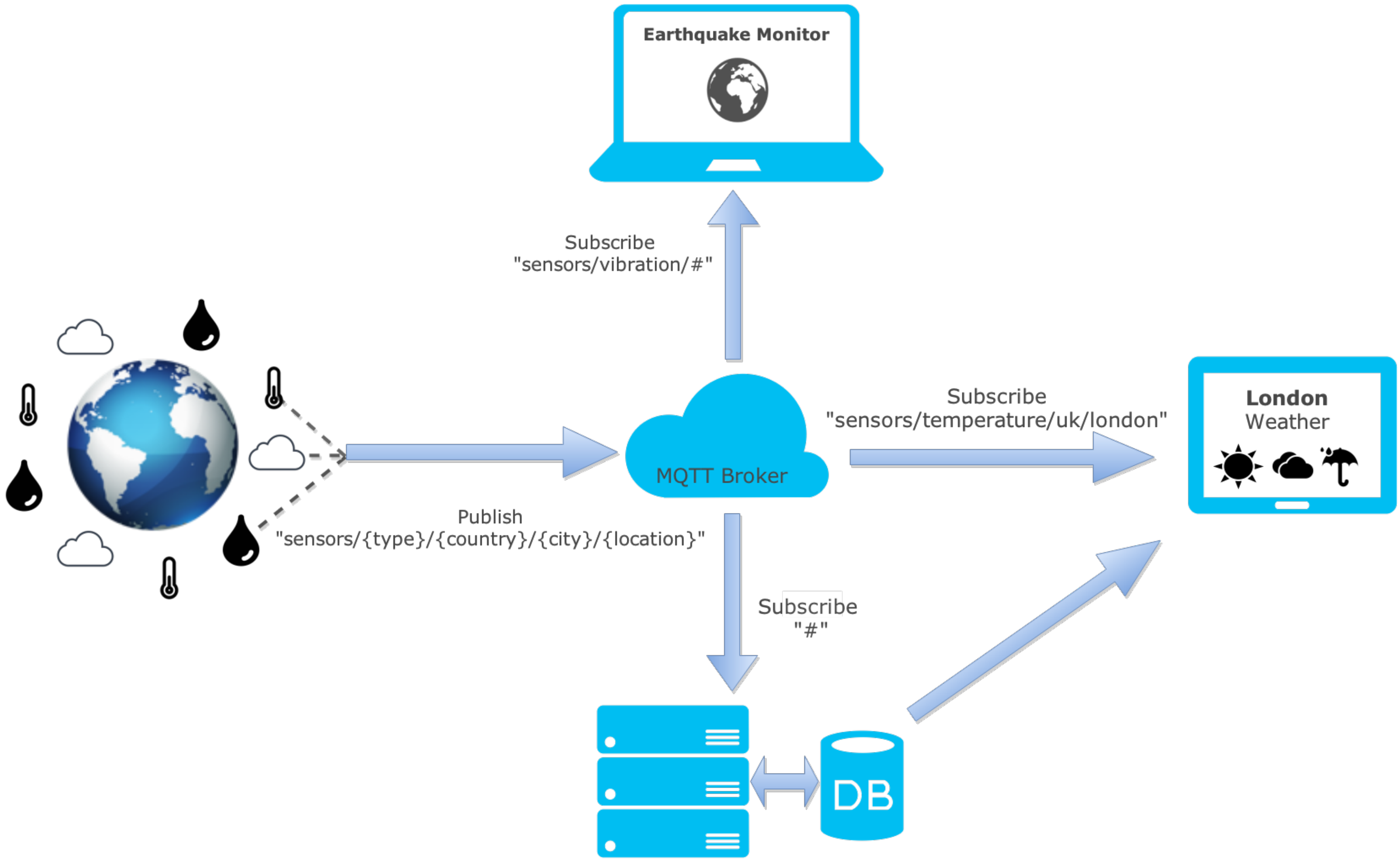
Publish Subscribe Messaging

- A Publish Subscribe messaging protocol allowing a message to be published once and multiple consumers (applications / devices) to receive the message providing decoupling between the producer and consumer(s)



  - A producer sends (publishes) a message (publication) on a topic (subject)
  - A consumer subscribes (makes a subscription) for messages on a topic (subject)
  - A message server/broker matches publications to subscriptions
    - If no matches the message is discarded
    - If one or more matches the message is delivered to each matching subscriber/consumer

Broker/Publish/Subscribe



https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot/

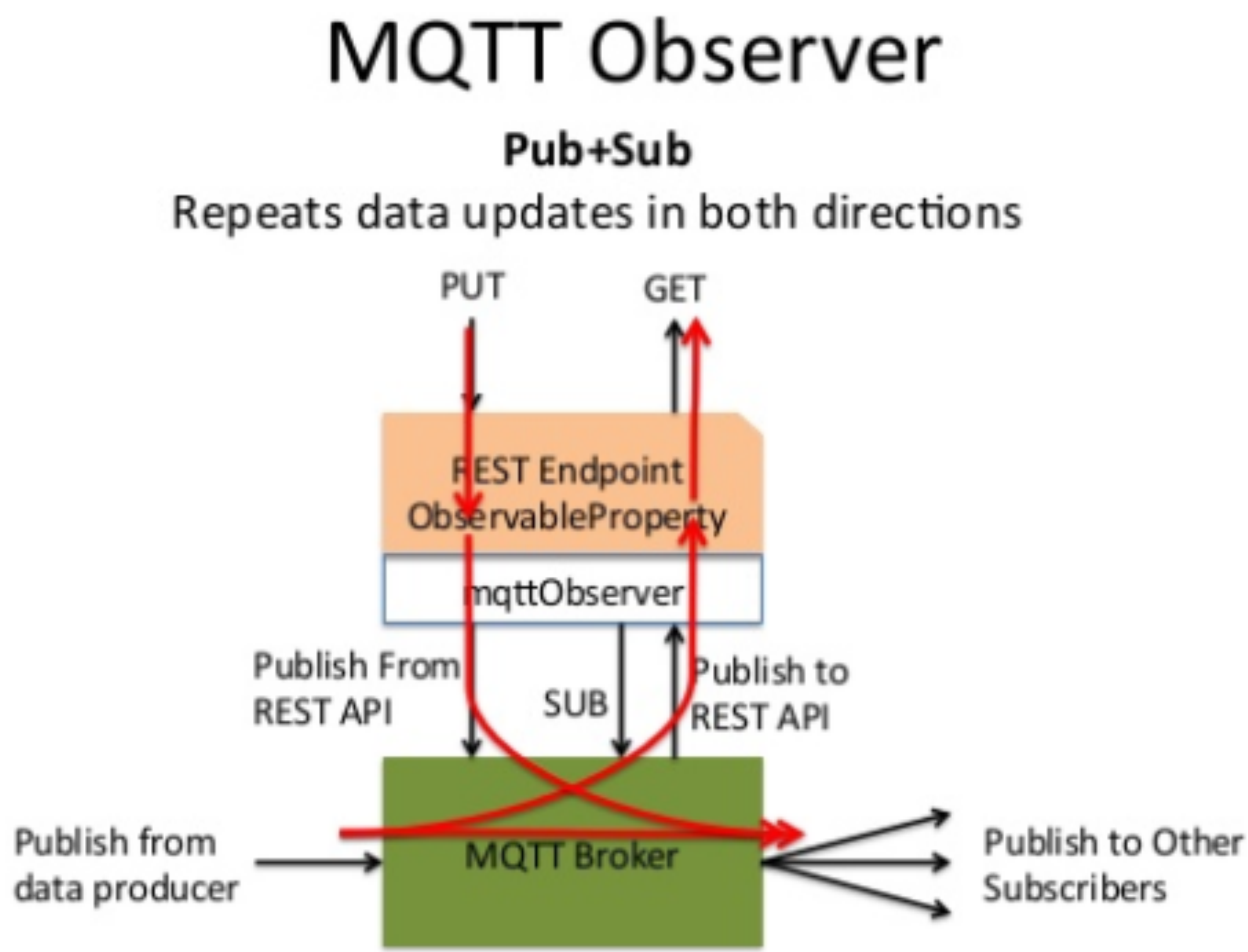
Publish Subscribe Messaging

- A topic forms the namespace
  - Is hierarchical with each "sub topic" separated by a /
    - <country>/<region>/<town>/<postcode>/<house>/alarmState
- A subscriber can subscribe to an absolute topic or can use wildcards:
  - Single-level wildcards "+" can appear anywhere in the topic string
    - sensors/+/uk/london/baker_street
    - get data from all sensor types in London
  - Multi-level wildcards "#" must appear at the end of the string
    - sensors/temperature/uk/#
    - Get temperature data from all locations in UK

Publish Subscribe Messaging

- A subscription can be durable or non durable
  - Durable:
    - Once a subscription is in place, a broker will forward matching messages to the subscriber:
      - Immediately if the subscriber is connected
      - If the subscriber is not connected, messages are stored on the server/broker until the next time the subscriber connects
  - Non-durable: The subscription lifetime is the same as the time the subscriber is connected to the server / broker
- A publication may be retained
  - A publisher can mark a publication as retained
  - The broker / server remembers the last known good message of a retained topic
  - The broker / server gives the last known good message to new subscribers
    - i.e. the new subscriber does not have to wait for a publisher to publish a message in order to receive its first message

Publish/Subscribe



https://www.slideshare.net/michaeljohnkoster/mqtt-rest-bridge

MQTT Message Format

- Structure of an MQTT Control Packet
  - The message header for each MQTT command message contains a fixed header.
  - Some messages also require a variable header and a payload.

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

MQTT Message Format

- The format for fixed header:

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

— DUP: Duplicate delivery of a PUBLISH Control Packet
— QoS: Quality of Service
— RETAIN: RETAIN flag

—This flag is only used on PUBLISH messages. When a client sends a PUBLISH to a server, if the Retain flag is set (1), the server should hold on to the message after it has been delivered to the current subscribers.

—This allows new subscribers to instantly receive data with the retained, or Last Known Good, value.

Control Packet types

- CONNECT: Client request to connect to Server
- CONNACK: Connect acknowledgment

- PUBLISH: Publish message
- PUBACK: Publish acknowledgment
- PUBREC: Publish received (assured delivery part 1)
- PUBREL: Publish release (assured delivery part 2)
- PUBCOMP: Publish complete (assured delivery part 3)

Control Packet types

- SUBSCRIBE: Client subscribe request
- SUBACK: Subscribe acknowledgment

- UNSUBSCRIBE: Unsubscribe request
- UNSUBACK: Unsubscribe acknowledgment

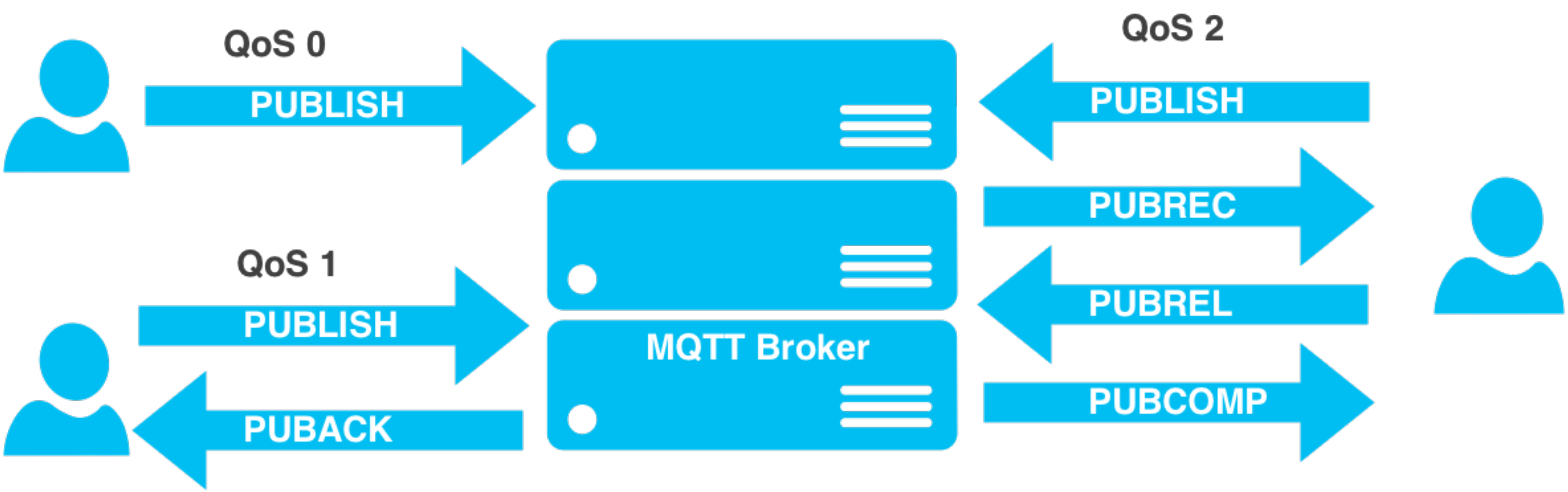
- PINGREQ: PING request
- PINGRESP: PING response

- DISCONNECT: Client is disconnecting

QoS

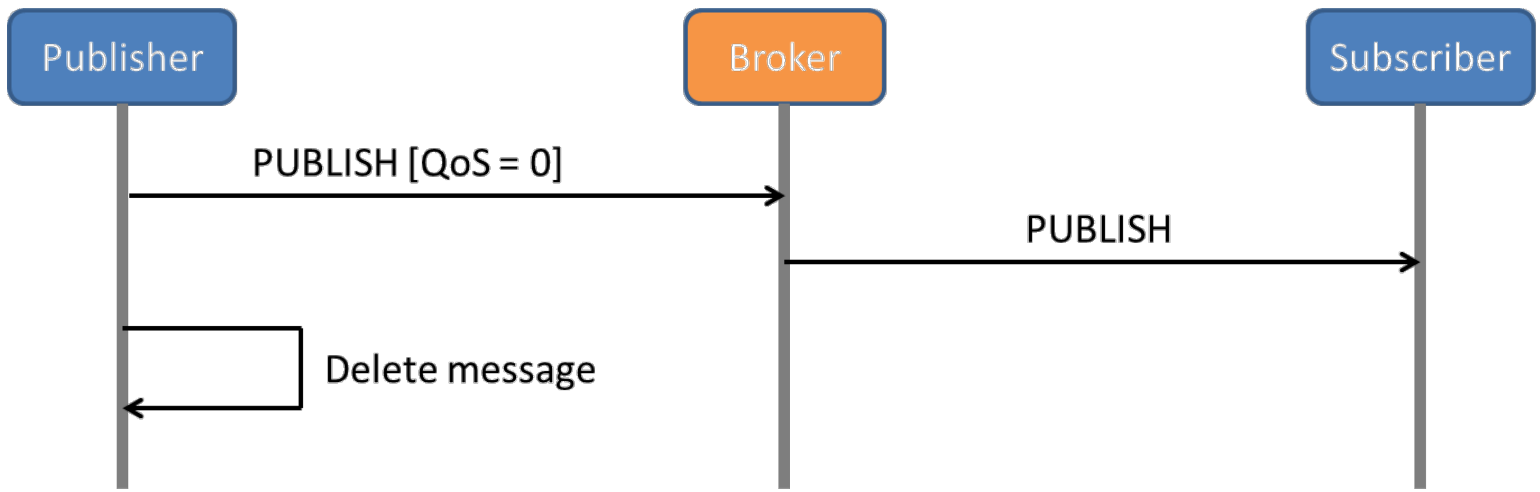
- QoS Level 0:
  - No acknowledgment from the client and the reliability will be the same as that of the underlying network layer, TCP/IP.
- QoS Level 1:
  - This ensures that the message is delivered to the client at least once, but it could be delivered more than once. It relies on the client sending an ACK packet when it receives a packet.
- QoS Level 2:
  - This ensures that a message is delivered once and only once. This method requires an exchange of four packets, and will decrease performance of the broker. This level is also often left out of MQTT implementations due to its relative complexity.

QoS



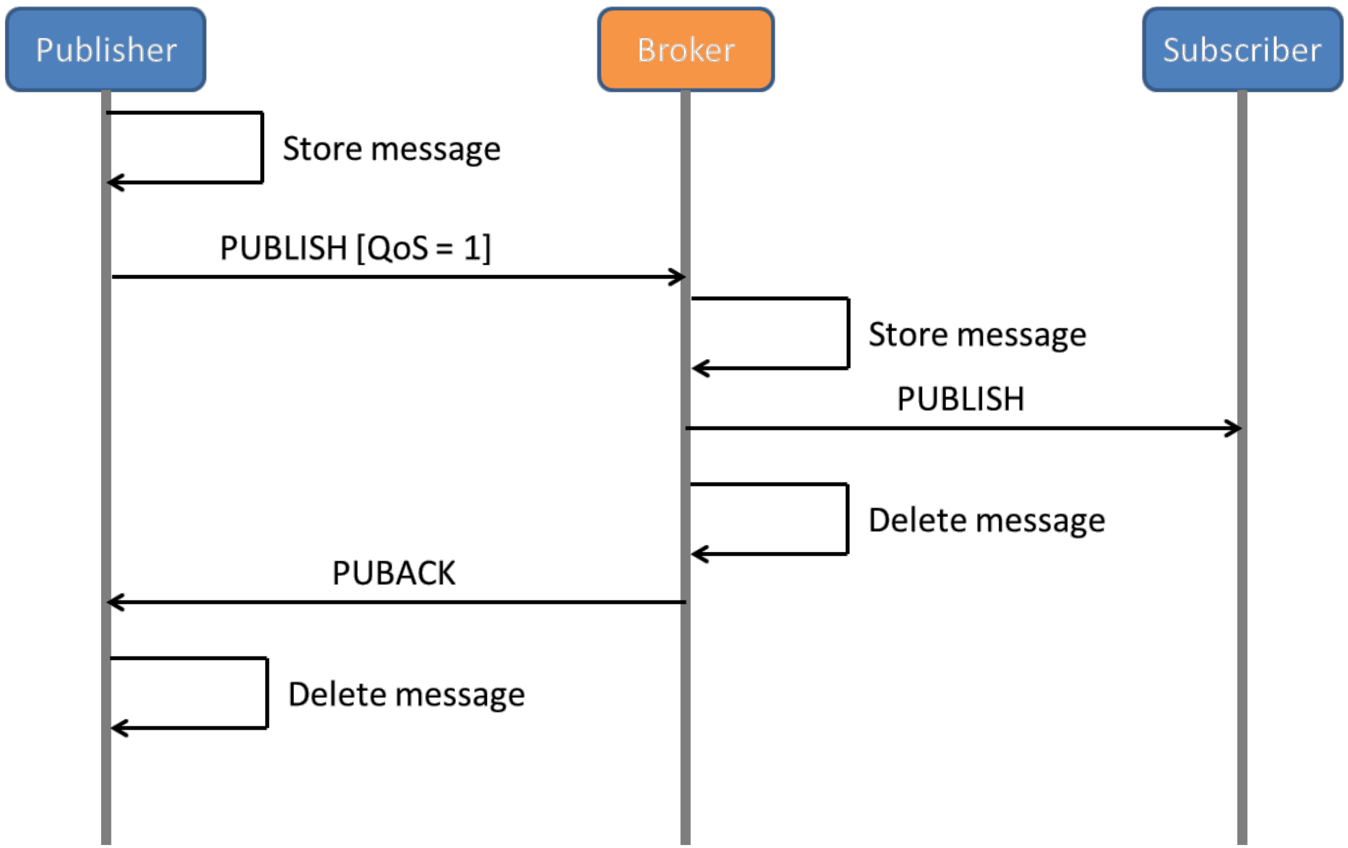
https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot/

QoS 0



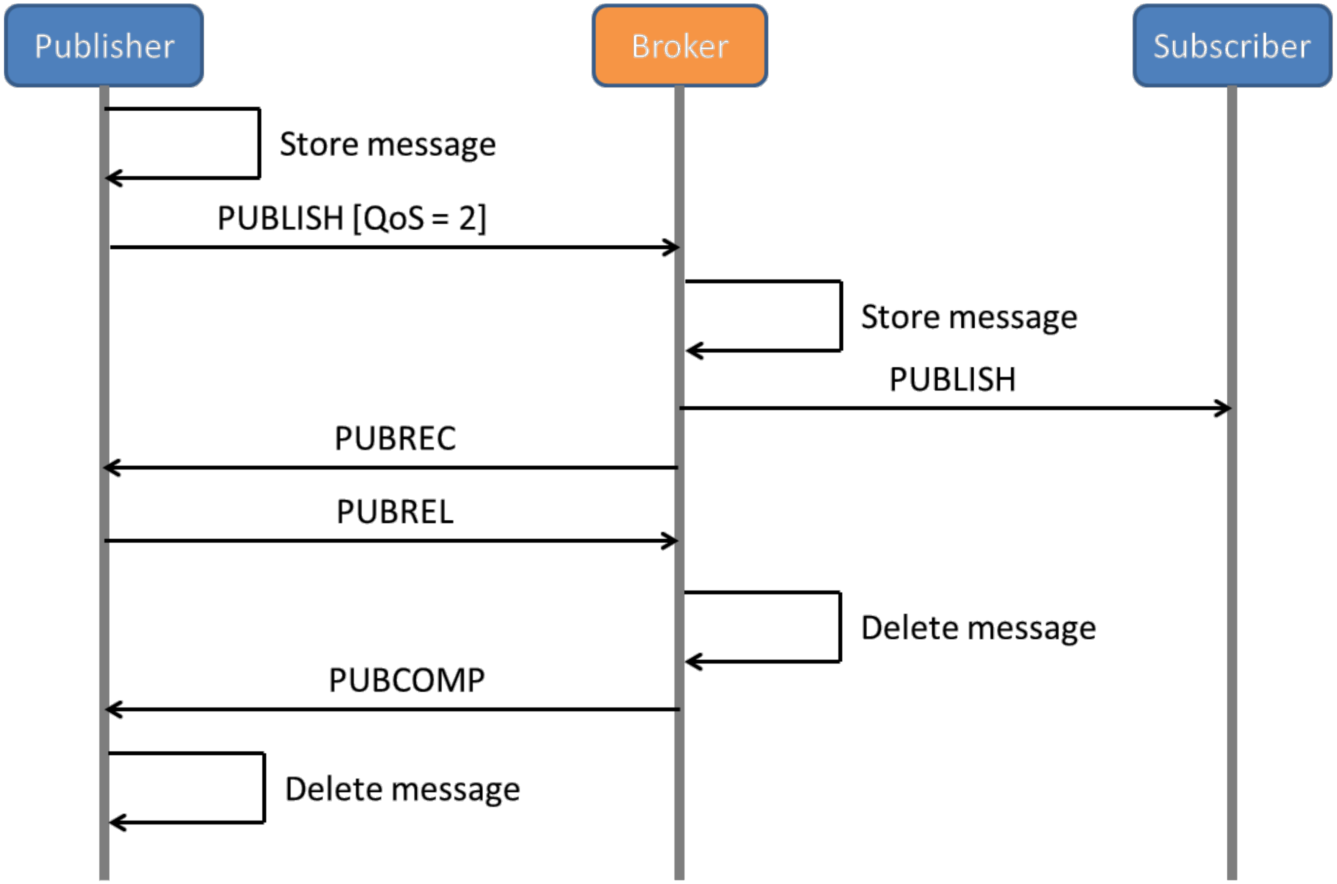
Publisher
Broker
Subscriber
PUBLISH [QoS = 0]
PUBLISH
Delete message

QoS 1



Publisher
Broker
Subscriber
Store message
PUBLISH [QoS = 1]
Store message
PUBLISH
Delete message
PUBACK
Delete message

QoS 2



Publisher
Broker
Subscriber
Store message
PUBLISH [QoS = 2]
Store message
PUBLISH
PUBREC
PUBREL
Delete message
PUBCOMP
Delete message

Variable Header

- CONNECT packet
  - Client requests a connection to a Server
  - Fixed header: packet type=1
  - Variable header: Protocol Name, Protocol Level (4), Connect Flags, Keep Alive

- Connect Flags

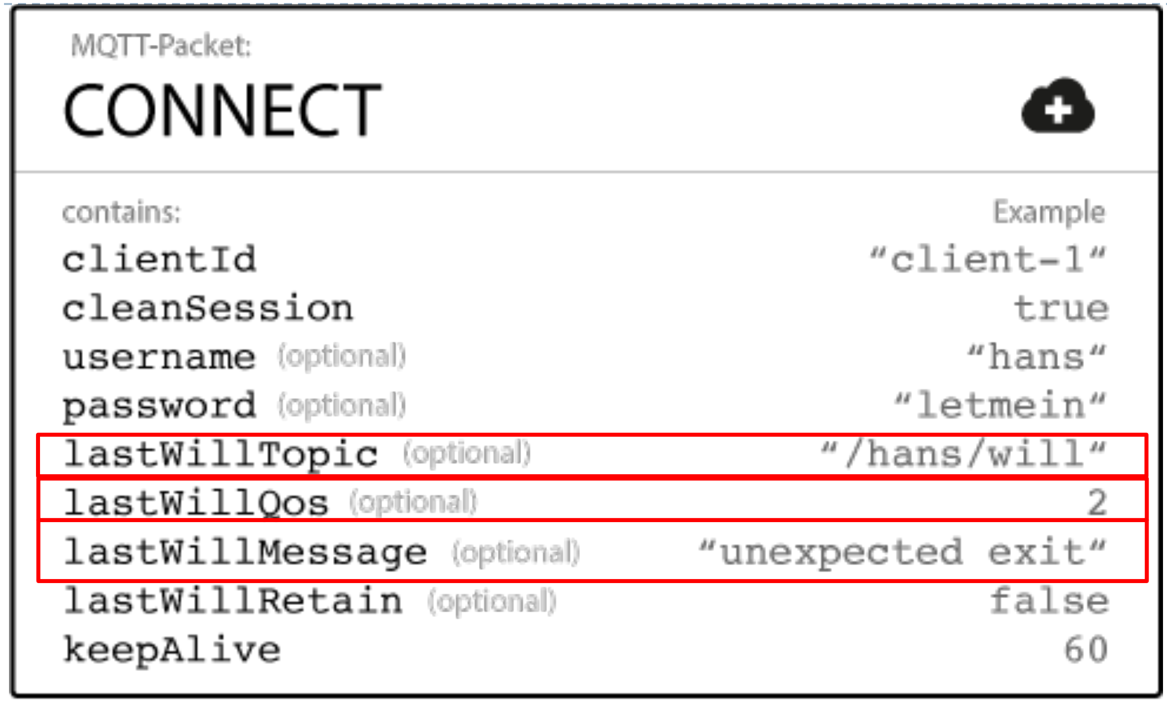
Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved

- If the Will Flag is set to 1 this indicates that, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Network Connection. The Will Message MUST be published when the Network Connection is subsequently closed.
  - Client crashes without sending a DISCONNECT Packet first

Payload

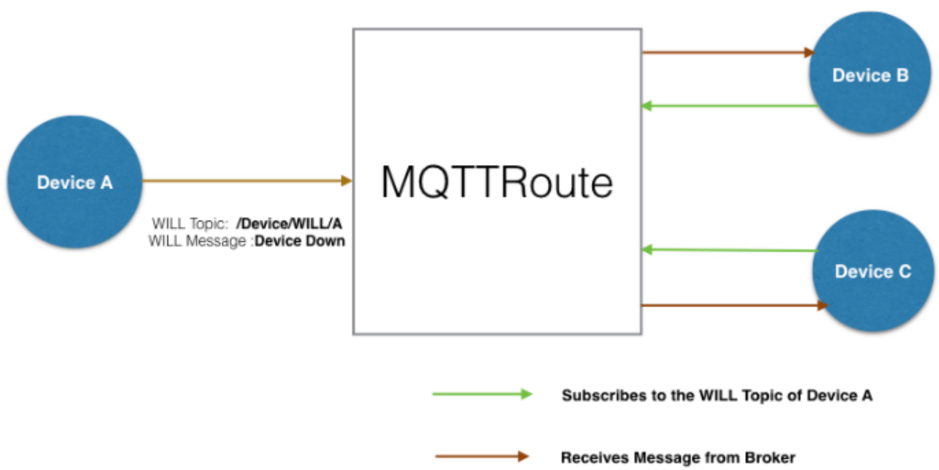
- CONNECT packet
  - Client Identifier
  - Will Topic
  - Will Message
  - User Name
  - User Password

How do you specify a LWT message for a client?


MQTT-Packet:

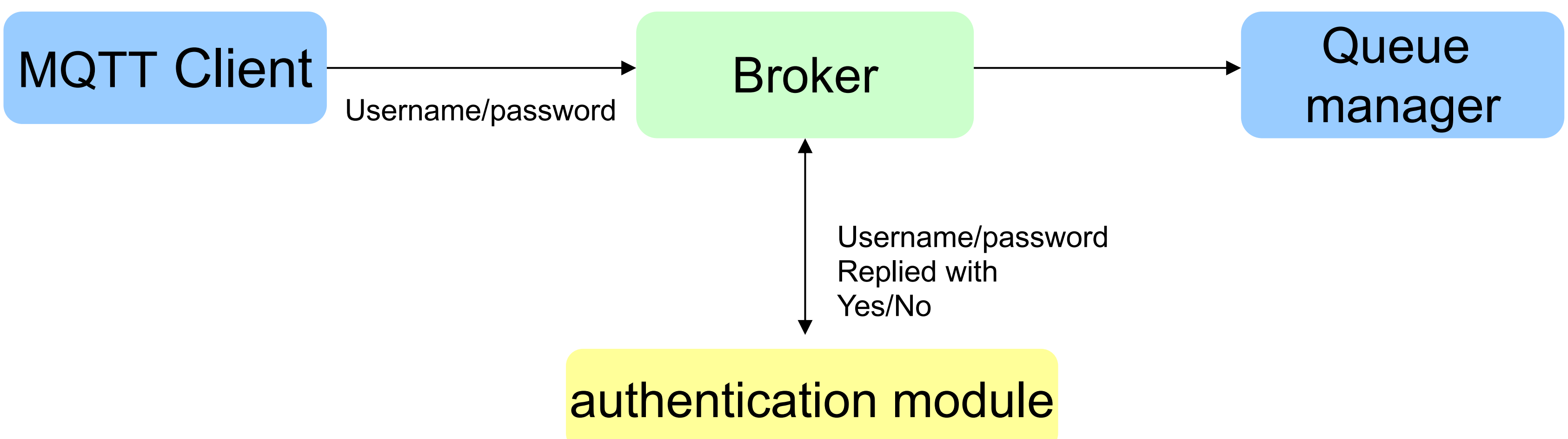
CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60



Security

- Authentication
  - Username and password as part of CONNECT action
- Encryption
  - SSL and plain text communication over TCP/IP



Broker Software

- Mosquitto - One of the earliest production ready brokers, Mosquitto is written in C and offers high performance with a lot of configurability.
- Mosca - Written in Node.js, this can be embedded in a Node application or run as a standalone executable. Easy configuration and extensibility, also very performant.
- RSMB - IBM's implementation of the MQTT protocol. This is one of the less popular options but is a mature system, written in C.
- HiveMQ - HiveMQ is a relatively new player, and is oriented towards enterprise environments.

Cisco webex

Introduction to XMPP

Joe Hildebrand

What is XMPP?

- eXtensible Messaging and Presence Protocol
- Bi-directional streaming XML
- Core: IETF RFC 6120, 7590, 6121
- Extensions: XMPP Standards Foundation (XSF)
  - Membership-based
  - Elected technical council
  - Unit of work: XMPP Extension Protocol (XEP)
  - Process: Experimental, Proposed, Draft, Final
- Goals:
  - Simple clients
  - Federate everything

Related RFCs

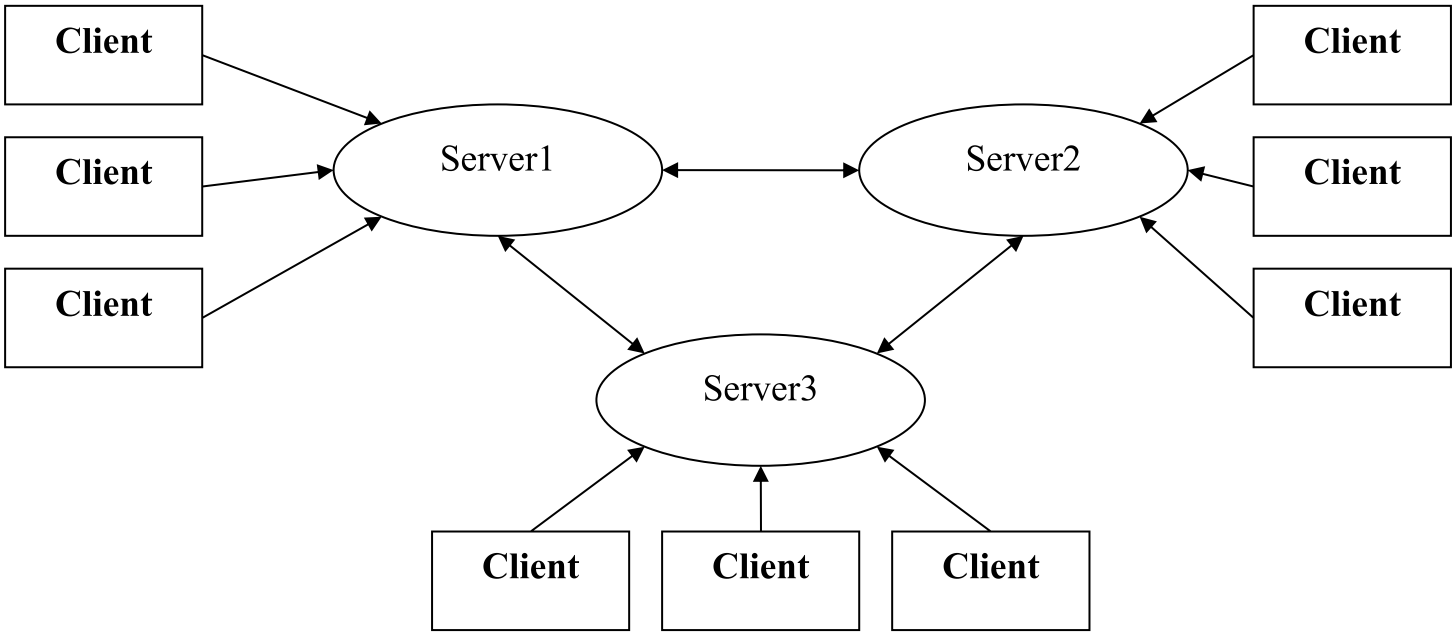
RFC	Name	Description
RFC 6120	XMPP Core	XMPP core features Updated by 7590 (TLS)
RFC 6121	XMPP IM	XMPP Instant Messaging and Presence
RFC 3922	XMPP CPIM	Mapping XMPP to Common Presence and Instant Messaging (CPIM)
RFC 3923	XMPP E2E	End-to-End Signing and Object Encryption for XMPP
RFC 5122	XMPP URI	Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for XMPP
RFC 4854	XMPP URN	Uniform Resource Name (URN) Namespace for XMPP

What is XMPP ?

- The eXtensible Messaging and Presence Protocol (XMPP) is a TCP communications protocol based on XML that enables near-real-time exchange of structured data between two or more connected entities.
- Out-of-the-box features of XMPP include presence information and contact list maintenance.
- Due in part to its open nature and XML foundation, XMPP has been extended for use in publish-subscribe systems
  - Perfect for IoT applications.

https://www.infoworld.com/article/2972143/internet-of-things/real-time-protocols-for-iot-apps.html

XMPP Architecture

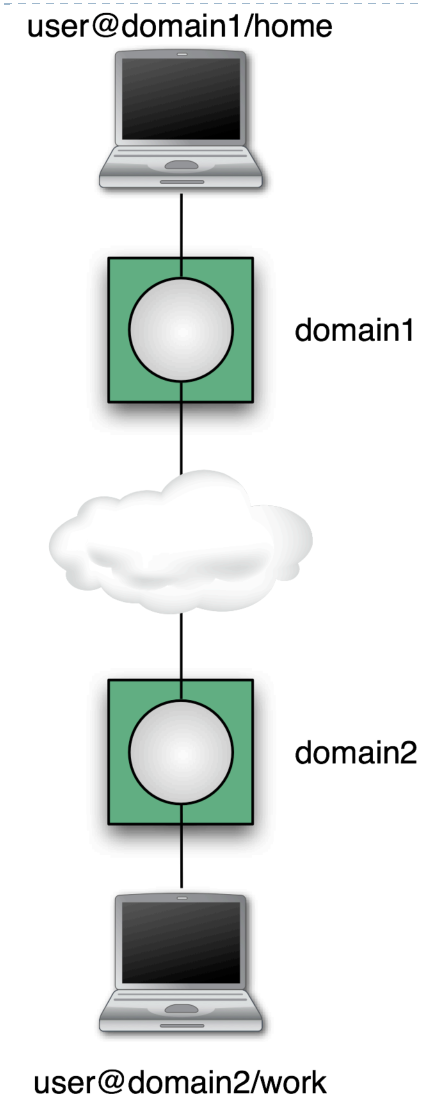


Server<->Server: Port 5269

Client<->Server: Port 5222

XMPP Architecture

- Addressing Scheme: `node@domain/resource`
  - JID = Jabber ID
  - Node: identity, e.g. user name
  - Domain: DNS domain name
  - Resource: device identifier
  - `node@domain` identifies a person
- Client talks to "local" server
  - Wherever the user account is hosted
  - Tied to directory if desired
  - Organizational policy enforced
- Servers talk to other servers
  - DNS lookup on domain portion of address
  - Dialback, MTLS for security
  - One connection for many conversations



XML Refresher

- Element
- Attribute
- Namespace
- Language
- Text

Attribute

```
<geoloc xmlns='http://jabber.org/protocol/geoloc'
        xml:lang='en'
        id='14'>
  <lat>38.9</lat>
  <lon>-77.1</lon>
  <locality>Arlington</locality>
  <region>VA</region>
</geoloc>
```

Element

XMPP Streams

- Client connects TCP socket to server
- Client sends stream start tag:

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               to='example.com'
               version='1.0'>
```

- Server sends stream start tag back:

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               from='example.com'
               id='someid'
               version='1.0'>
```

- Each child element of stream: a "*stanza*"

Stream features

- After stream start, server sends feature list:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
  </mechanisms>
  <compression xmlns='http://jabber.org/features/compress'>
    <method>zlib</method>
  </compression>
</stream:features>
```

- Client can negotiate any of these features

XML Stream Features

Feature	XML Element	Description	Documentation
amp	<amp xmlns='http://jabber.org/features/amp'>	Support for Advanced Message Processing	XEP-0079: Advanced Message Processing
compress	<compression xmlns='http://jabber.org/features/compress'>	Support for Stream Compression	XEP-0138: Stream Compression
iq-auth	<auth xmlns='http://jabber.org/features/iq-auth'>	Support for Non-SASL Authentication	XEP-0078: Non-SASL Authentication
iq-register	<register xmlns='http://jabber.org/features/iq-register'>	Support for In-Band Registration	XEP-0077: In-Band Registration
bind	<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>	Support for Resource Binding	RFC 6120: XMPP Core
mechanisms	<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>	Support for Simple Authentication and Security Layer (SASL)	RFC 6120: XMPP Core
session	<session xmlns='urn:ietf:params:xml:ns:xmpp-session'>	Support for IM Session Establishment	RFC 6121: XMPP IM
starttls	<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>	Support for Transport Layer Security (TLS)	RFC 6120: XMPP Core
bidi	<bidi xmlns='urn:xmpp:bidi'>	Support for Bidirectional Server-to-Server Connections	XEP-0288: Bidirectional Server-to-Server Connections
Server Dialback	<dialback xmlns='urn:xmpp:features:dialback'>	Support for Server Dialback with dialback errors	XEP-0220: Server Dialback
sm	<sm xmlns='urn:xmpp:sm:3'>	Support for Stream Management	XEP-0198: Stream Management

Security Stuff

- Start-TLS
  - Prove the identity of the server
  - Prove the identity of the user (optional)
  - Encryption
  - Data integrity
- SASL(Simple Authentication and Security Layer protocol) (RFC 4422)
  - Authentication
  - Optional encryption (rarely used)
  - Pluggable (e.g. passwords, Kerberos, X.509, SAML, etc.)

Stanzas

- All have `to='JID'` and `from='JID'` addresses
  - To gives destination
  - From added by local server
- Each stanza routed separately
- All contents of stanza passed along
- Extend with any XML from your namespace
- Different types for delivery semantics

  `<message/>`: one direction, one recipient

  `<presence/>`: one direction, publish to many

  `<iq/>`: "info/query", request/response

See details next page

Message

- Example:
```
<message xml:lang='en'
         to='romeo@example.net'
         from='juliet@example.com/balcony'
         type='chat'>
  <body>Wherefore art thou, Romeo?</body>
</message>
```
- Types: chat, groupchat, headline, error
- Body: plain text
- XHTML IM: XEP-0071

Presence

express an entity's current network availability

- Example:

```
<presence>
  <show>dnd</show>
  <status>Meeting</status>
  <priority>1</priority>
</presence>
```

- Show: chat, available, away, xa, dnd
- Status: Human-readable text
- Priority: Which resource "most available"?

IQ Request

a structured request-response mechanism

- Example:
```
<iq type='get'
    id='roster_1'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```
- Type: get, set, result, error
- ID: track the corresponding response
- Query/Namespace: what type of request?

IQ Response (Roster)

- Example:

```
<iq type='result'
    id='roster_1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          name='Romeo'
          subscription='both'>
      <group>Friends</group>
    </item>
  </query>
</iq>
```

- Type: response
- ID matches request
- Subscription state: none, to, from, both

Subscribing to Presence

- Send a subscription request:

```
<presence to='juliet@example.com'
          type='subscribe'/>
```

- Approving a request:

```
<presence to='romeo@example.net'
          type='subscribed'/>
```

- Every time you change a subscription, you get a "roster push":

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          subscription='from'/>
  </query>
</iq>
```

Extensibility Example: Message

- Use a new namespace
- Key: if you don't understand it, ignore it
- Example, CAP, XEP-0127:

```
<message to='weatherbot@jabber.org'
         from='KSTO@NWS.NOAA.GOV'>
  <alert xmlns='http://www.incident.com/cap/1.0'>
    <identifier>KSTO1055887203</identifier>
    <sent>2003-06-17T14:57:00-07:00</sent>
    <info>
      <category>Met</category>
      <event>SEVERE THUNDERSTORM</event>
...
    </info>
  </alert>
</message>
```

Common Alerting Protocol (CAP) Over XMPP

Extensibility Example: Presence

- Keep presence stanzas small
- Example: Entity Capabilities, XEP-0115:

```
<presence from='bard@shakespeare.lit/globe'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

- Ver attribute is hash of all features of this client
- Hash -> Feature list is cached

It defines an XMPP protocol extension for broadcasting and dynamically discovering client, device, or generic entity capabilities.

XMPP Extensions

- Many already exist: http://www.xmpp.org/extensions/
- Add new ones
  - Custom: use a namespace you control, make up protocol
  - Standardized: write a XEP.

Number	Name	Type	Status	Date
XEP-0001	XMPP Extension Protocols	Procedural	Active	2016-11-16
XEP-0002	Special Interest Groups (SIGs)	Procedural	Active	2002-01-11
XEP-0004	Data Forms	Standards Track	Final	2007-08-13
XEP-0009	Jabber-RPC	Standards Track	Final	2011-11-10
XEP-0012	Last Activity	Standards Track	Final	2008-11-26
XEP-0019	Streamlining the SIGs	Procedural	Active	2002-03-20
XEP-0030	Service Discovery	Standards Track	Final	2017-10-03
XEP-0047	In-Band Bytestreams	Standards Track	Final	2012-06-22
XEP-0049	Private XML Storage	Historical	Active	2004-03-01
XEP-0053	XMPP Registrar Function	Procedural	Active	2016-12-01
XEP-0054	vcard-temp	Historical	Active	2008-07-16
XEP-0055	Jabber Search	Historical	Active	2009-09-15

Federation: DNS

- Starts with: non-local domain in to address

```
# _service._proto.name.  TTL   class SRV priority weight port target.
_sip._tcp.example.com.   86400 IN    SRV 10       60     5060 bigbox.example.com.
_sip._tcp.example.com.   86400 IN    SRV 10       20     5060 smallbox1.example.com.
_sip._tcp.example.com.   86400 IN    SRV 10       20     5060 smallbox2.example.com.
_sip._tcp.example.com.   86400 IN    SRV 20       0      5060 backupbox.example.com.
```

- Look up this DNS SRV record: (service record)
  `_xmpp-server._tcp.`*`domain`*
- Example: jabber.com:
  `10 0 5269 jabber.com.`
  - Priority: Which one to try first if multiple
  - Weight: Within a priority, what percentage chance?
  - Port: TCP port number
  - Target: Machine to connect to

Federation: Security

- Old-style: dialback
  - Connect back to domain claimed by initiator
  - Check secret claimed by initiator
  - "Someone said they were example.com; was that you?"
- New-style: Mutual TLS
  - Initiator presents "client" certificate
  - Responder presents "server" certificate
  - Both certificates signed by trusted CA
- All stanzas *must* have from with correct domain

Bandwidth minimization

- TLS compression
  - Not implemented in all SSL/TLS stacks
  - Some want compression w/o encryption
- XEP-0138: Stream Compression
  - Defines zlib mechanism (2-3x or more compression)
  - Others can be added
  - Concern: battery drain vs. radio transmission
- XEP-0198: Stanza Acknowledgements
  - Quick reconnects
  - Avoid re-synchronizing state on startup
- Partial rosters
- Privacy lists
- Others being pursued

Latency

- Most critical on startup
  - Several handshakes and stream restarts
  - Can be minimized by client assuming server configuration
  - Example: don't wait for `<stream:features>`
- Once running
  - Stanza size matters: try to stay under 8kB, take larger blocks out of band if possible
  - Configure federation to keep links open, first stanza will be slow
  - Beware of DoS protection, "karma"

Reading List

- RFCs
  - 6120: Core
  - 6121: IM & Presence
  - 5122: XMPP URIs
- XEP highlights
  - 4: Forms
  - 30: Disco
  - 45: Chat rooms
  - 60: Pub/Sub
  - 71: XHTML
  - 115: Capabilities
  - 163: PEP

Advantages of XMPP

- The primary advantage is XMPP's decentralized nature.
- XMPP works similar to email, operating across a distributed network of transfer agents rather than relying on a single, central server or broker (as CoAP and MQTT do).
- As with email, it's easy for anyone to run their own XMPP server, allowing device manufacturers and API operators to create and manage their own network of devices.
- And because anyone can run their own server, if security is required, that server could be isolated on a company intranet behind secure authentication protocols using built-in TLS encryption.

Disadvantages of XMPP

- One of the largest flaws is the lack of end-to-end encryption. While there are many use cases in which encryption may not yet be necessary, most IoT devices will ultimately need it. The lack of end-to-end encryption is a major downside for IoT manufacturers.

https://wiki.xmpp.org/web/XMPP_E2E_Security

- Another downside is the lack of Quality of Service (QoS). Making sure that messages are delivered is even more important in the IoT world than it was in the instant messaging world. If your alarm system doesn't receive the message to turn itself on, then that vacation you've been planning could easily be ruined.

Q and A