CISTER - Research Center in Real-Time & Embedded Computing Systems

IoT and Middleware for Industrial Applications

Michele Albano

isep Instituto Superior de Engenharia do Porto

INESCTEC

Outline

- IoT and Middleware
- 6LoWPAN
- CoAP
- MQTT
- Arrowhead

The Industrial Revolutions 1/2

- First Industrial Revolution (1781):
  - Invention of the (patented) steam engine by James Watt
  - Mechanical production
- Second Industrial Revolution / Technological Revolution (1874):
  - Invention of the incandescent light bulb
  - Electricity, moving assembly line, division of labour
  - Mass Production
- Third Industrial Revolution (1969):
  - Invention of the microprocessor
  - Electronics, IT
  - Automated production

The Industrial Revolutions 2/2

- Fourth Industrial Revolution / Industrie 4.0 / Digitizing Industry (today)

- Its focus is on:
  - Sensing / acting on the environment by means of Cyber-Physical Systems
  - Ubiquitous fruition of information by means of IoT
  - Computation on the Cloud
  - Machine learning

IoT

IoT $\rightarrow$ Use of IP

- All devices have an IP address
- Devices are accessible through the Internet

IIoT

- The Industrial Internet of Things (IIoT) is the use of Internet of Things (IoT) technologies in the industrial context (e.g.: manufacturing).

- IIoT incorporates:
  - machine and product sensoring
  - machine learning and big data technology crunching sensor data
  - machine-to-machine (M2M) communications and automation technologies on a global scale

Main Application Areas

- IIoT has a great potential for:
  - quality control
  - sustainable and green practices
  - supply chain traceability
  - overall supply chain efficiency
  - user in the loop

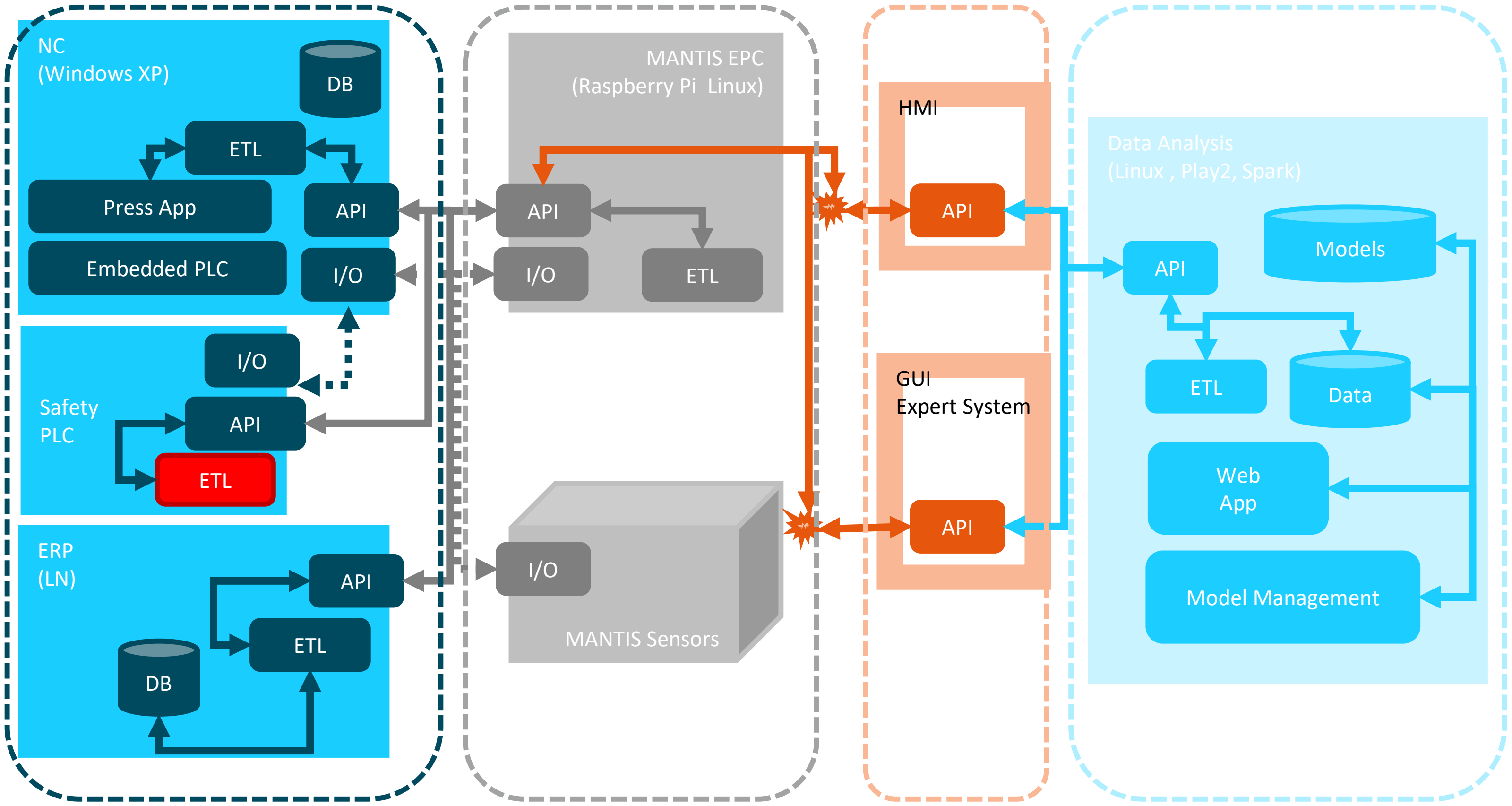
IIOT - another perspective

- The Industrial Internet of Things will **transform** companies and countries, opening up **a new era of economic growth** and competitiveness.
- A future where the **intersection of people**, **data** and **intelligent machines** will have far-reaching impacts on the **productivity**, **efficiency** and **operations** of industries around the world.

Accenture, "https://www.accenture.com/us-en/labs-insight-industrial-internet-of-things.aspx"

Press brake example

Industrial maintenance



Outline

- IoT and Middleware
- 6LoWPAN
- CoAP
- MQTT
- Arrowhead

Many Advantages of IP

- Extensive interoperability
  - Other wireless embedded 802.15.4 network devices
  - Devices on any other IP network link (WiFi, Ethernet, GPRS, Serial lines, ...)
- Established security
  - Authentication, access control, and firewall mechanisms
  - Network design and policy determines access, not the technology
- Established naming, addressing, translation, lookup, discovery
- Established proxy architectures for higher-level services
  - NAT, load balancing, caching, mobility
- Established application level data model and services
  - HTTP/HTML/XML/SOAP/REST, Application profiles
- Established network management tools
  - Ping, Traceroute, SNMP, ... OpenView, NetManager, Ganglia, ...
- Transport protocols
  - End-to-end reliability in addition to link reliability
- Most “industrial” (wired and wireless) standards support an IP option

IoT (IP) and 802.15.4

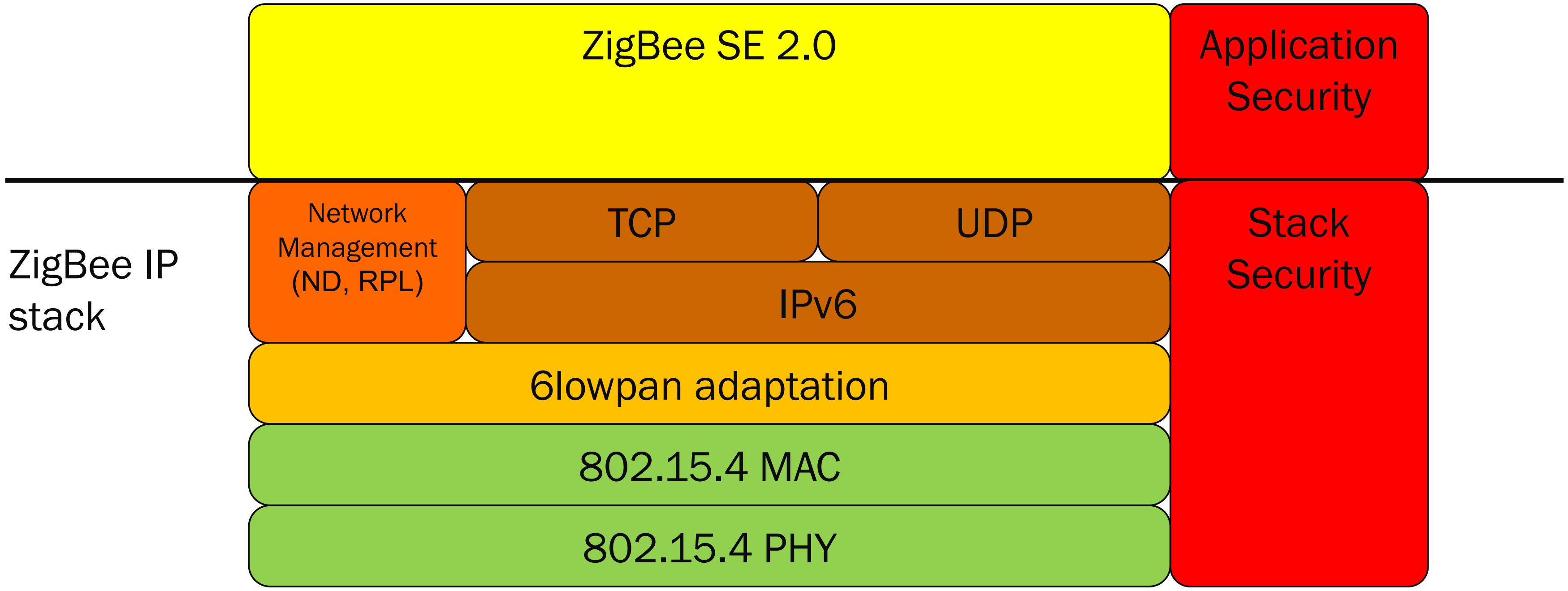
- 802.15.4 has small PDUs
  - Maximum PHY PDU is 127 bytes
- IPv6 header is 40 octets, UDP header is 8 octets
- 802.15.4 MAC header can be up to 25 octets (null security) or 25+21=46 octets (AES-CCM-128)
- With the 802.15.4 frame size of 127 octets, we have
  - 127-25-40-8 = 54 octets (null security)
  - 127-46-40-8 = 33 octets (AES-CCM-128)
- of space left for application data

- … and IP datagrams have a typical MTU of 1280 bytes

IPv6 over IEEE 802.15.4

- 6LoWPAN introduces an adaptation layer between the IP stack's link and network layers
- The adaptation layer enables efficient transmission of IPv6 datagrams over 802.15.4

ZigBee IP stack diagram

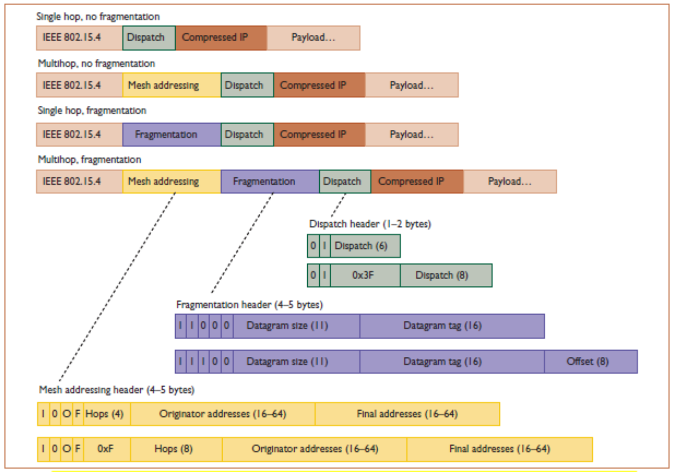


6LoWPAN Adaptation Layer

- Introducing the idea of stacked header:
  - You only pay for what you use
- This layer provides :
  - Header compression
  - Fragmentation
  - Support for layer-two forwarding

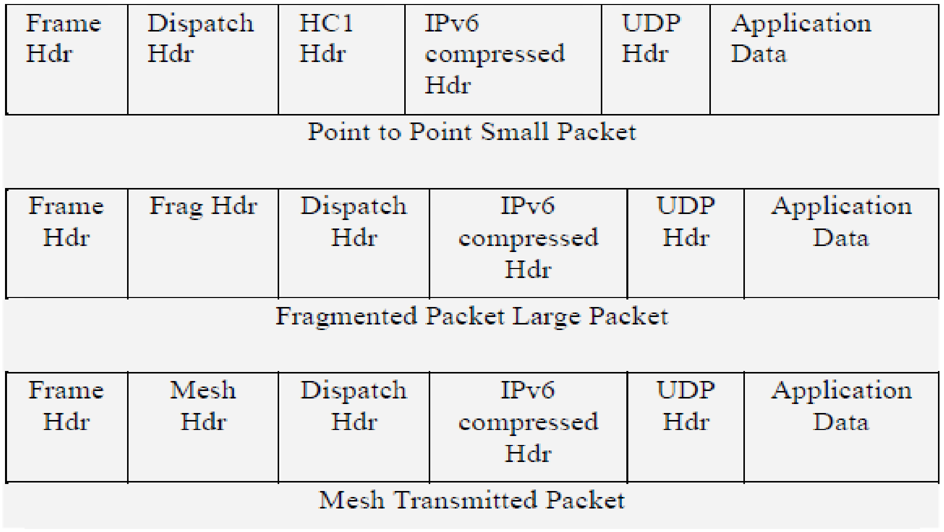
6LoWPAN Stacked Headers

- All header formats are identified using dispatch header
- The mesh header is used to encode the hop limit
- The fragmentation header supports the fragmentation and reassembly of payloads


Single hop, no fragmentation
IEEE 802.15.4
Dispatch
Compressed IP
Payload…
Multihop, no fragmentation
IEEE 802.15.4
Mesh addressing
Dispatch
Compressed IP
Payload…
Single hop, fragmentation
IEEE 802.15.4
Fragmentation
Dispatch
Compressed IP
Payload…
Multihop, fragmentation
IEEE 802.15.4
Mesh addressing
Fragmentation
Dispatch
Compressed IP
Payload…
Dispatch header (1–2 bytes)
0 1 Dispatch (6)
0 1 0x3F Dispatch (8)
Fragmentation header (4–5 bytes)
1 1 0 0 0 Datagram size (11) Datagram tag (16)
1 1 1 0 0 Datagram size (11) Datagram tag (16) Offset (8)
Mesh addressing header (4–5 bytes)
1 0 O F Hops (4) Originator addresses (16–64) Final addresses (16–64)
1 0 O F 0xF Hops (8) Originator addresses (16–64) Final addresses (16–64)

Utilizing Separate Headers

- 1)Point to Point Small Packet
- 2)Fragmented Large Packet
- 3)Mesh Transmitted Packet



Frame Hdr	Dispatch Hdr	HC1 Hdr	IPv6 compressed Hdr	UDP Hdr	Application Data

Point to Point Small Packet

Frame Hdr	Frag Hdr	Dispatch Hdr	IPv6 compressed Hdr	UDP Hdr	Application Data

Fragmented Packet Large Packet

Frame Hdr	Mesh Hdr	Dispatch Hdr	IPv6 compressed Hdr	UDP Hdr	Application Data

Mesh Transmitted Packet

Routing in 6LoWPAN

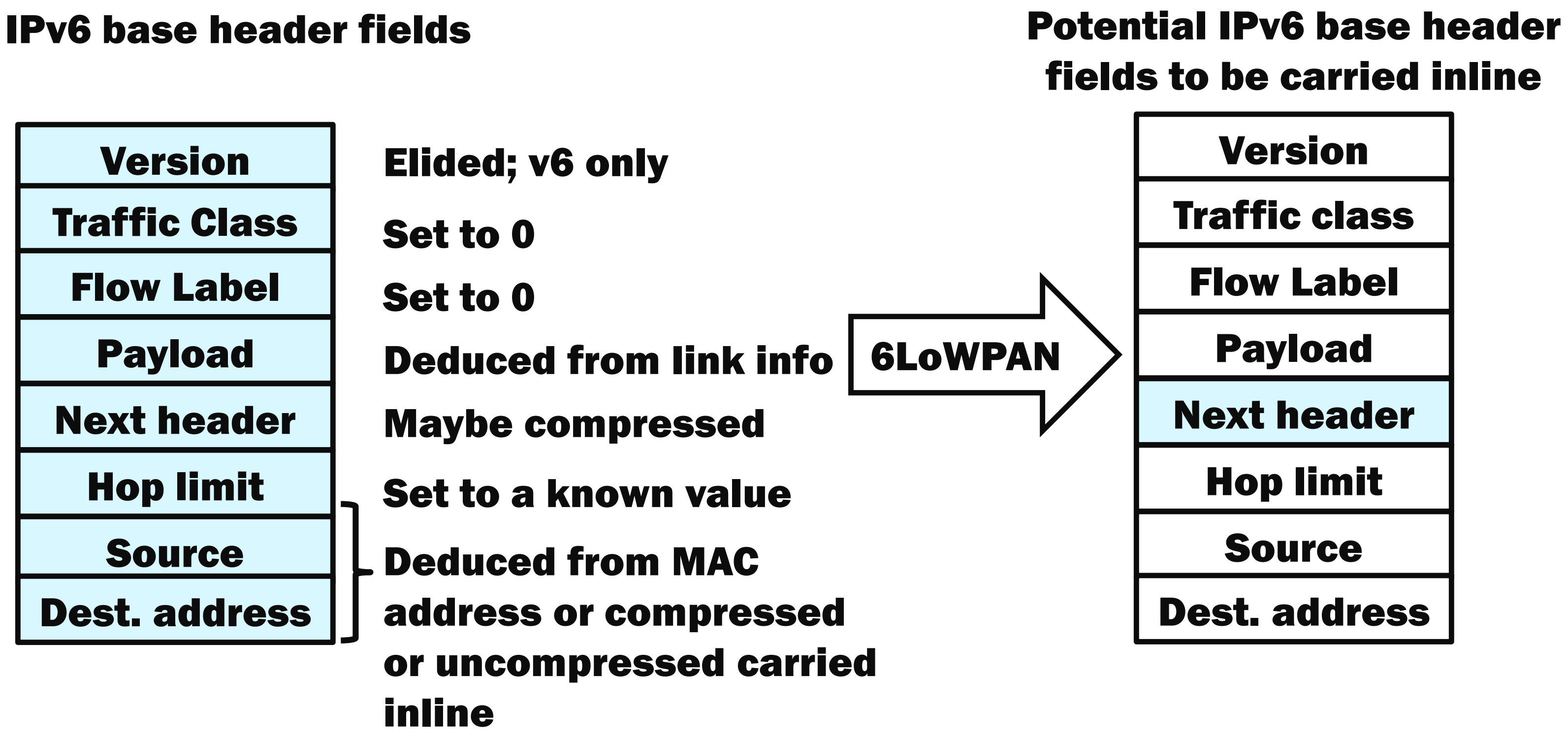
- Mesh Under Routing
  - No IP routing
  - Routing within the LoWPAN
- Route Over Routing
  - Routing at the IP layer
  - Utilizing network-layer capabilities defined by IP

Compression Principles 1/2

- 6LoWPAN compresses the header reducing redundancy
  - Some information is deducted from underlying link layer
- This achieves an efficient transport of IPv6 headers and next headers
  - In some cases, IPv6 addresses can be deduced from MAC addresses
  - IP payload length can be deduced from L2/L1 length information
  - Traffic Class and Flow Label values are set to zero
  - Version field value is IPv6
  - Hop limit can be set to predefined values

Compression Principles 2/2

- Example of header compression

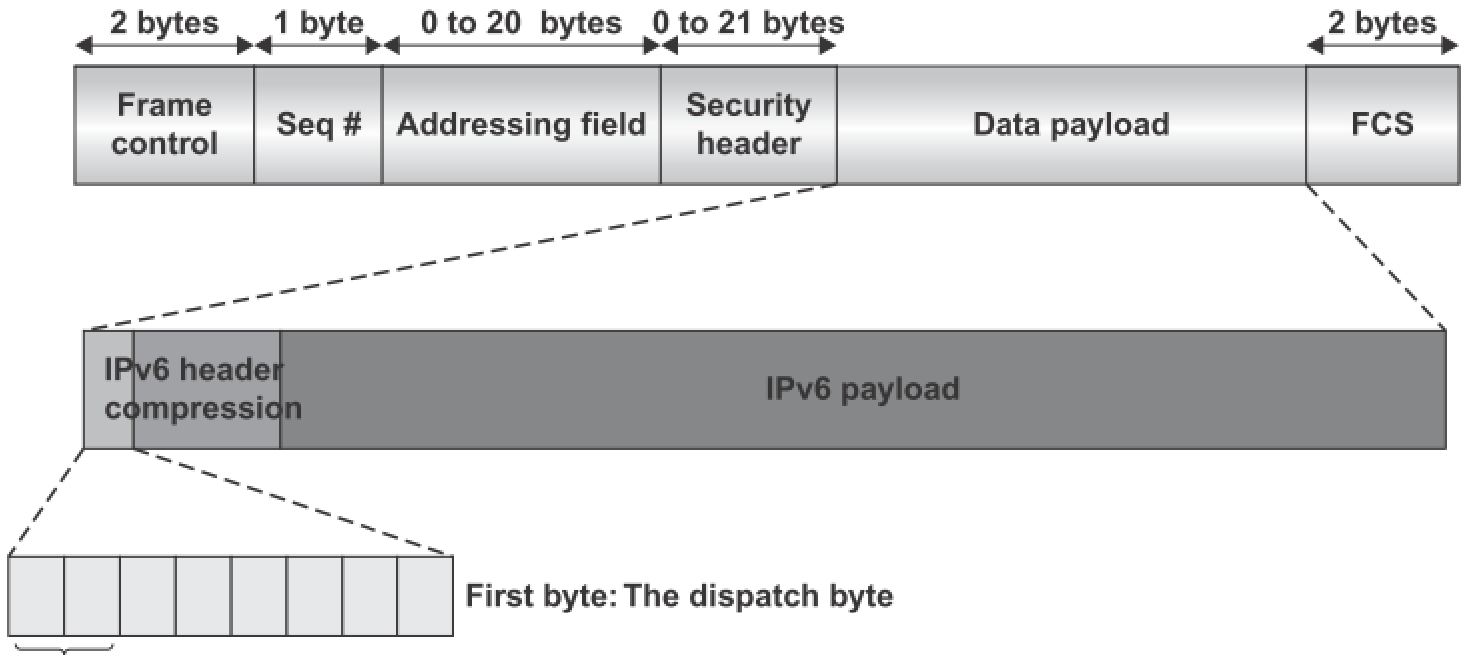


6LoWPAN

- Comprises a Dispatch
  - Identifies the type of header immediately following the Dispatch Header.
  - Similar to a Frame Control field.
- It follows a LOWPAN IP Header Compression (LOWPAN_IPHC) field

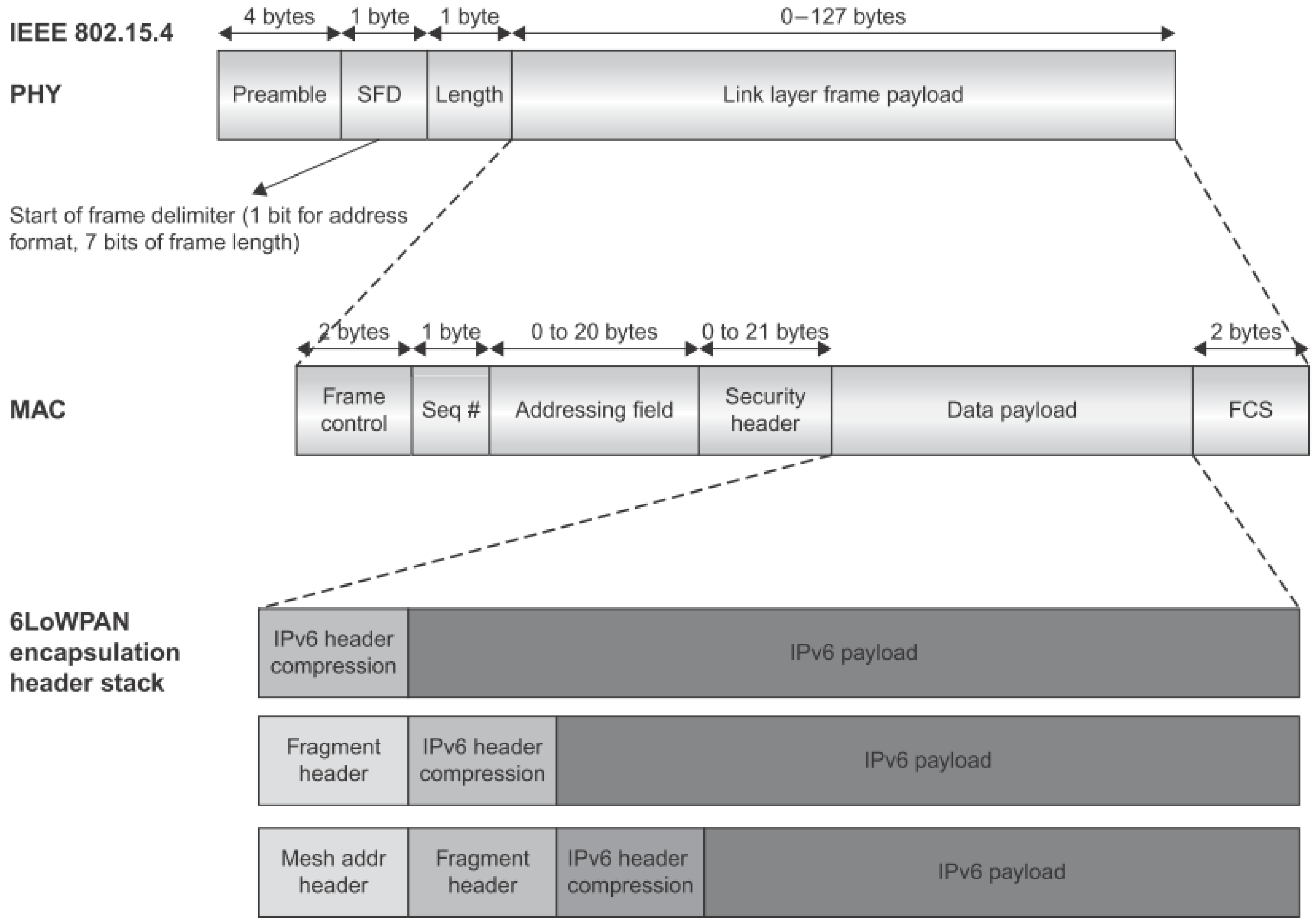
Header Dispatch Byte

The 6LoWPAN dispatch byte (first byte)



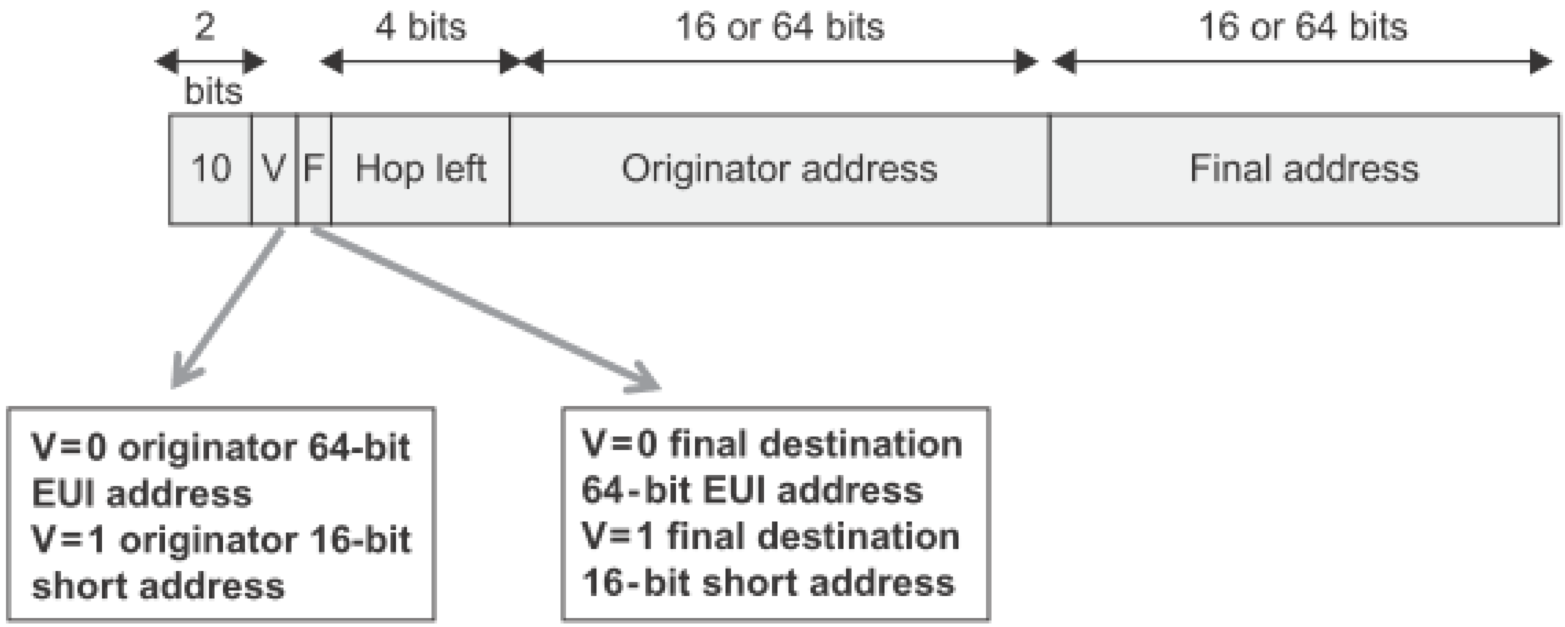
00	**Not a 6LoWPAN frame**
01	IPv6 addressing header
10	Mesh header
11	Fragmentation header (6 lower bits are 100xxx)

6LowPAN Header Stack



Mesh Address Header (1)

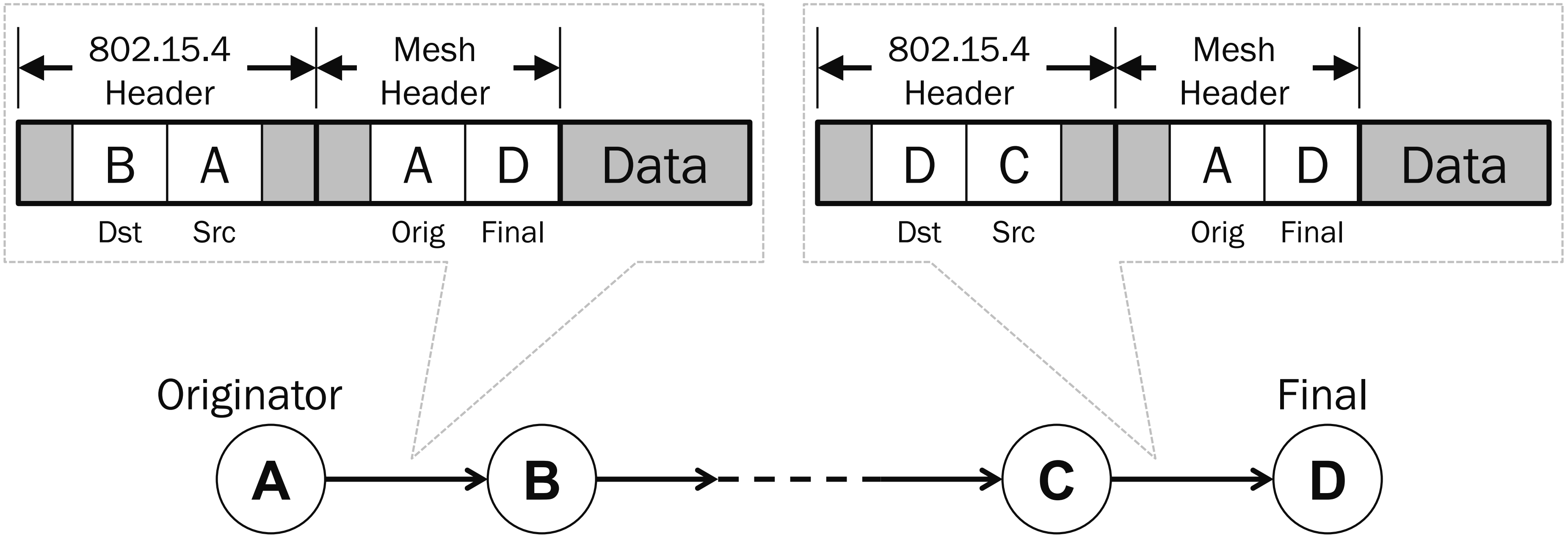
- Used with mesh-under routing approach
  - Only performed by FFDs



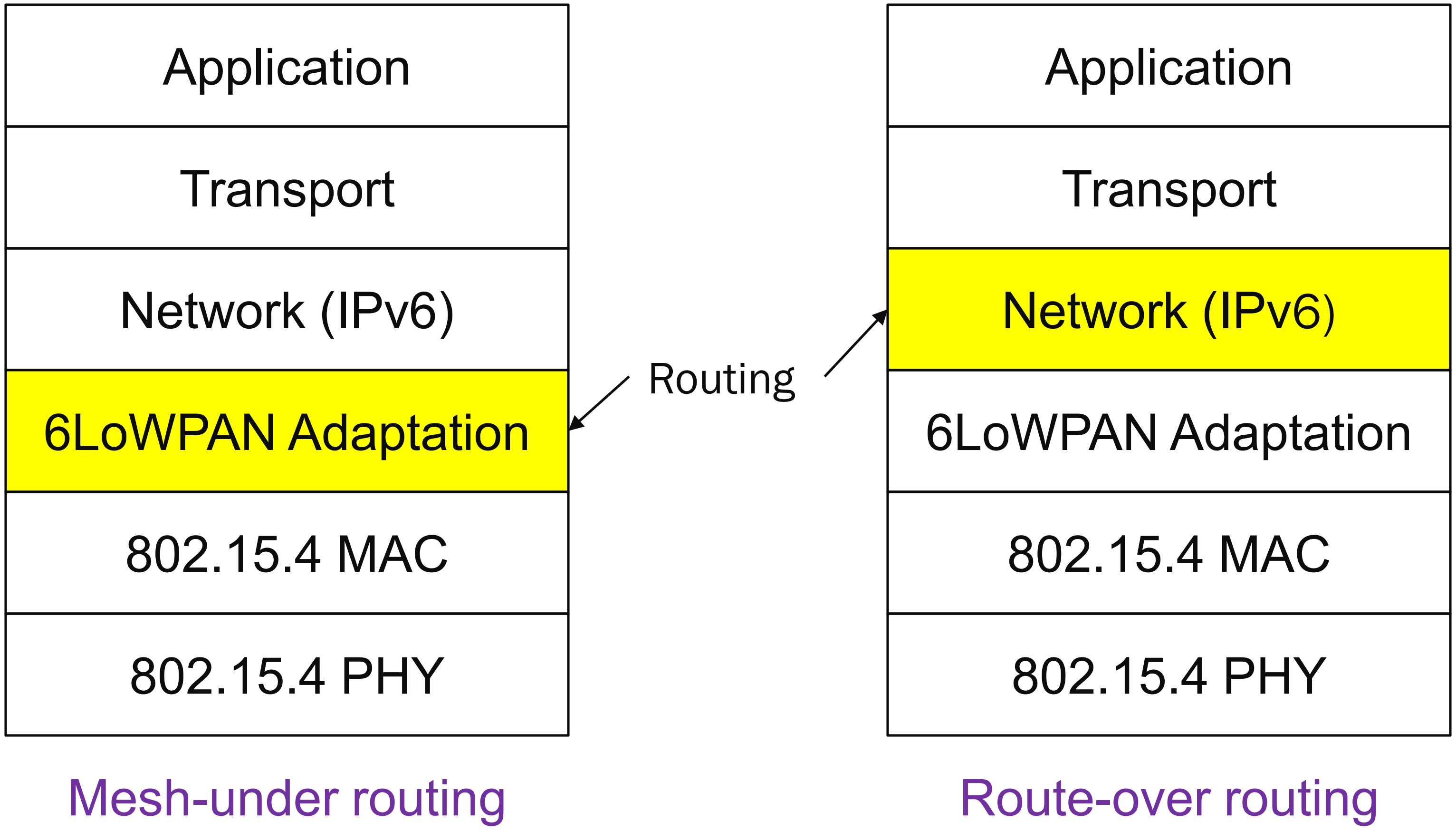
2 bits
4 bits
16 or 64 bits
16 or 64 bits
10
V
F
Hop left
Originator address
Final address
V=0 originator 64-bit EUI address
V=1 originator 16-bit short address
V=0 final destination 64-bit EUI address
V=1 final destination 16-bit short address

Mesh Address Header (2)

- Hop left field is decremented by one every hop
  - Frame is discarded when hop left is 0
- Address fields are unchanged

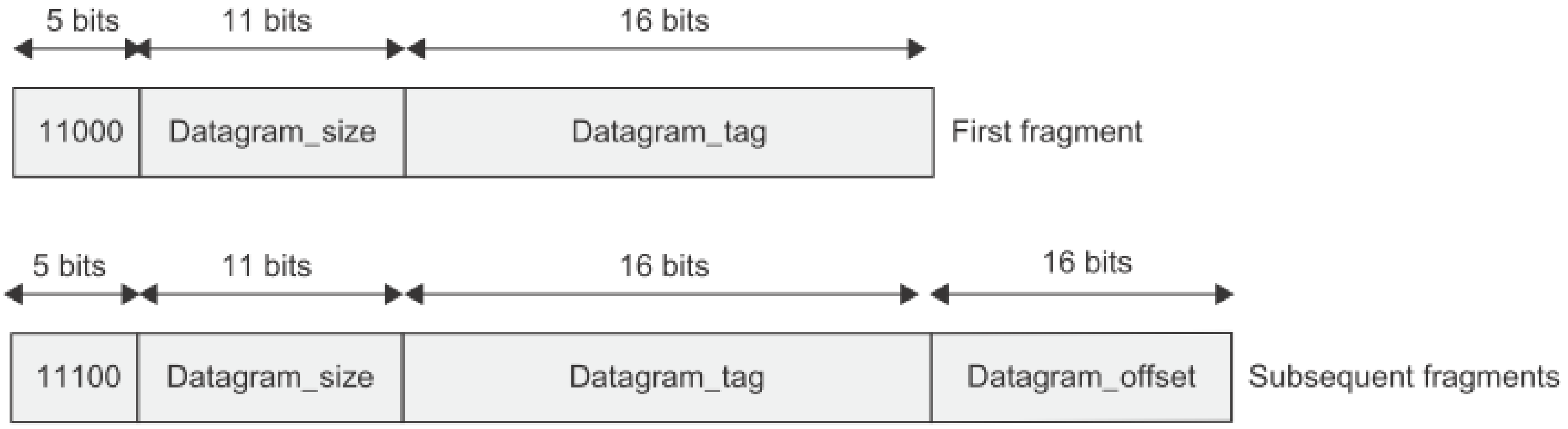


Mesh-under vs. Route-over Routing



Fragment Header

- Fragmentation is required when IPv6 payload size exceeds that of IEEE 802.15.4 payload limit
- All fragments are in units of 8 bytes



(in 8-byte units)

Outline

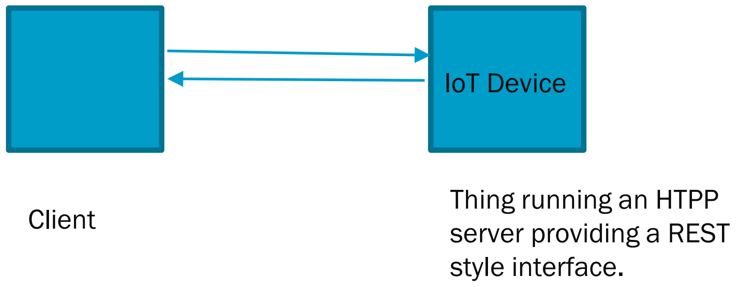
- IoT and Middleware
- 6LoWPAN
- CoAP
- MQTT
- Arrowhead

IoT Integration Patterns, REST, and CoAp

- Direct integration Pattern
- Gateway Integration Pattern
- Cloud Integration Pattern
- REST
- CoAp

Direct Integration Pattern

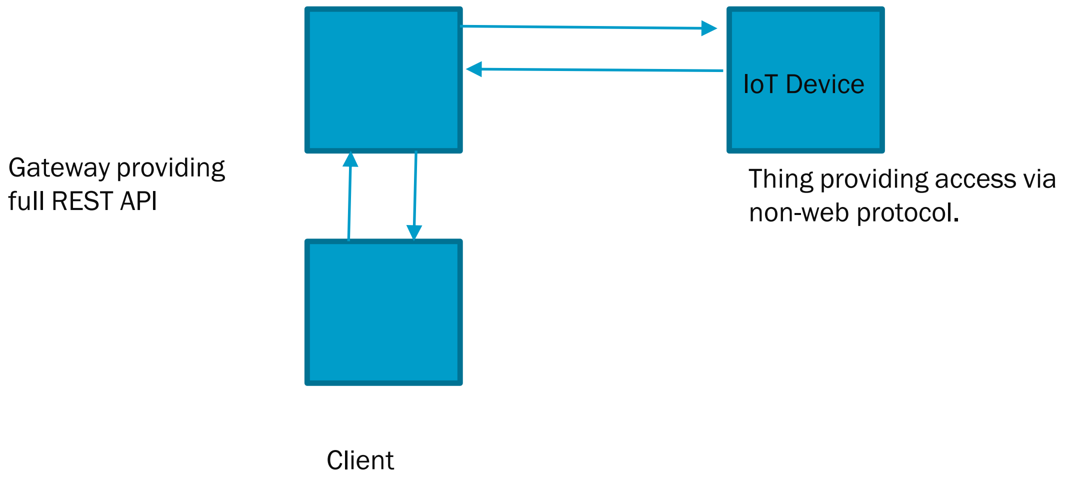
- Some IoT devices have full internet access.
  - May provide an HTTP server running over TCP/IP, and
  - connect direcly to the internet (WiFi, Ethernet, cellular, etc).
  - may be used to implement a Direct Integration Pattern – REST on devices.


IoT Device
Client
Thing running an HTPP server providing a REST style interface.

- Typical use case: The Thing is not battery powered and communicates with low latency to a local device like a phone.
- Example: Use a phone to communicate via WiFi (with WiFi router) to an HTTP server on a device. Use web sockets for publish/subscribe.

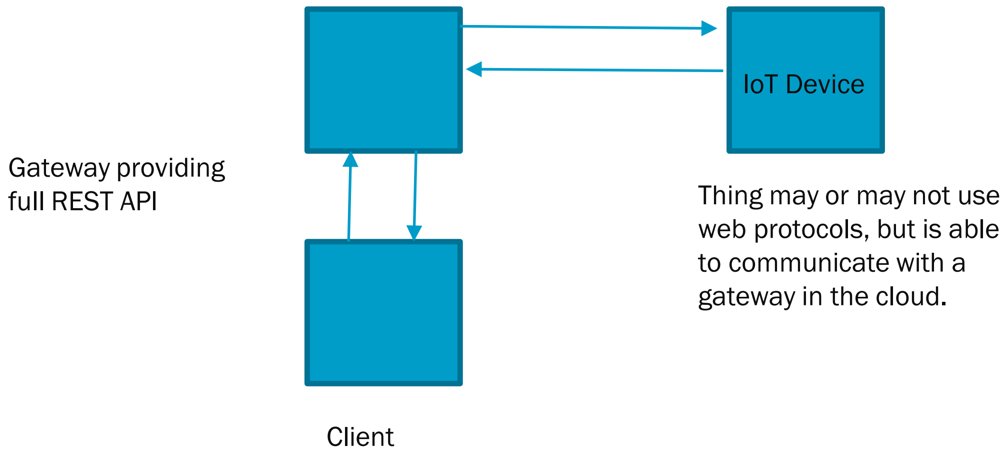
Gateway Integration Pattern

- Some IoT devices do not have full internet access
  - May support only Zigbee or Bluetooth or 802.15.4
  - We are not sending IP packets to these devices – they are constrained. This is the Gateway Integration Pattern.


IoT Device
Gateway providing full REST API
Thing providing access via non-web protocol.
Client

Cloud Integration Pattern

- Some IoT devices have access to the cloud and need powerful and scalable cloud support. This is the Cloud Integration Pattern.


IoT Device
Gateway providing full REST API
Thing may or may not use web protocols, but is able to communicate with a gateway in the cloud.
Client

REST API Design Principles

Principle	Implementation
Constrained user interface	HTTP GET, POST, DELETE, PUT
Standard status codes	HTTP codes
Well designed and standard URI's	Naming. Well Designed URI's representing a resource. Protocol and location. Identification of resources.
Standard representations	JSON or XML messages
HATEOS	Messages returning pointers or links for further discovery
Statelessness	Simple request/response required no conversational state. Easy to scale.

A REST style request in IoT

- GET /basement/water/temperature 200 OK
  application/text
  40.5 F

- GET /basement/water/volume 200 OK
  application/text
  200 G

HTTP codes reused as return values.

Constrained Application Protocol (CoAP)

- A key IoT standard
- Open IETF standard since June 2014
- Based on web standards, easily integrates with HTPP. Is not simply a compressed version of HTTP.
- Built for small, constrained, imbedded, occasionally sleeping devices
- Some built-in reliability
- May run over 6LoWPan (IP-like layer over IEEE 802.15.4)
- Use on low power, low bandwidth, lossy networks
- Over UDP or SMS on cellular networks
- DTLS for security
- Asynchronous subscriptions and notifications over UDP
- Built-in resource discovery
- Peer to peer or client server and multi-cast requests

Compare with HTTP over internet

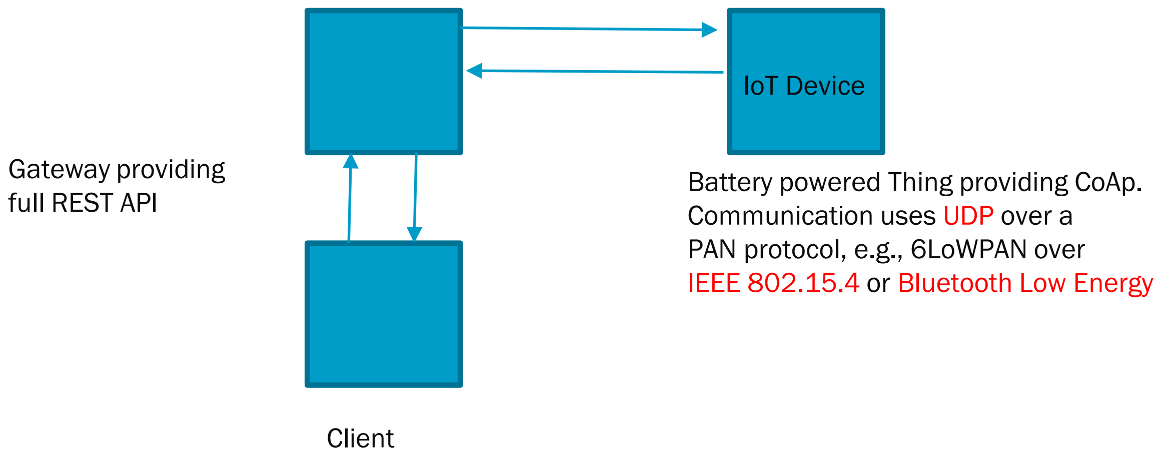
- Connection oriented and synchronous
- TCP 3 way handshake with server
- HTTP GET /kitchen/light
- HTTP response
- TCP 2 way termination
- Too much work for simple IoT applications
- CoAP does not support all features of HTTP

CoAP is based on REST

- Interaction: request/response RESTful similar to HTTP
- Messages: smaller than HTTP, much lower overhead

Example: BLE nodes

- Sensors and actuators on BLE nodes are simply CoAP resources
- To obtain a current temperature, send a GET request
- To turn on/off or toggle LEDs, use a PUT request


IoT Device
Gateway providing full REST API
Battery powered Thing providing CoAp. Communication uses UDP over a PAN protocol, e.g., 6LoWPAN over IEEE 802.15.4 or Bluetooth Low Energy
Client

CoAP

- Has a scheme coap://
- Has a well known port.
- GET, POST, PUT, DELETE encoded in binary (1 == GET)
- Block transfer support.
- Confirmable messages require an ack with message ID.
- Non-confirmable messages do not require an ack.
- Example:

```
CoAP Client                          CoAP Server

   ---->   CON  {id} GET  /basement/light            Confirmable request

  <----   ACK  {id} 200 Content {“status” : “on”}    Piggy back response
                                                     (likely in binary)
```

CoAP Uses Timeouts over UDP

```
CoAP Client                              CoAP Server
            ---->   CON  {id} GET  /basement/light
  lost request timeout!!
            ---->   CON  {id} GET /basement/light
  this time it gets through
            <----    ACK  {id} 200 Content {“status” : “on”}
```

The {id} allows us to detect duplicates.

CoAP Request/Acknowledge/Callback

```
CoAP Client                              CoAP Server
    ---->    CON  {id} PUT  /basement/cleanFloor   Token:  0x22 Needs time

    <----    ACK  {id}

    <-----   CON {newID} 200 Content /basement/cleanFloor Token: 0x22 Done

    ---->     ACK {newID}
```

Token to recognize which request was satisfied

CoAP Publish/Subscribe

```
CoAP Client                              CoAP Server
    ---->   CON  {id} GET  /basement/light Observe: 0   Token:  0x22

    <----   ACK  200 {id}  Observer: 27 Token 0x22

    <----   CON 200 Observe: 28 Token: 0x22 {“light” : ”off”}

    ----->  ACK Token: 0x22

    <----   CON 200 Observe: 28 Token: 0x22 {“light” : ”on”}
             :
             :
             etc.
```

CoAP Resource Discovery

- Implementing by means of REST (POST to register, GET to discover, PUT to update, DELETE to remove)
- Different (higher level) than service discovery.
- We register a device as web resources using a discovery service, to find it later on the fly.
- Links are returned (coap:// ……)
  - Links may include a rel attribute.
  - The rel attribute specifies the relationship between the current document and the linked document.
- A well known resource can be used to discover other resources.
  - Perform a GET on the well known resource. Returned content is a list of links with REL attributes.
- Resource directories may be used to register services.

Outline

- IoT and Middleware
- 6LoWPAN
- CoAP
- MQTT
- Arrowhead

Higher-level REST

- A "request" is sent to a server via URL
  - Eg. http://api.example.com/resources/user/1036721?name=something
- Response is usually text in HTML, XML, or JSON
- Great if your asking for something
  - What about "push"
  - Eg. Server wants to tell device to do something

The MQTT protocol is:

- Lightweight
- Publish/subscribe
- Over TCP/IP
- Aimed at remote sensors and control devices applications
- For communication through low bandwidth, unreliable or intermittent communications
- Available under a royalty free license

Sensor readings with pub/sub

Decoupled in space and time.
The clients do not need each others' IP address and port (space) and they do not need to be running at the same time (time).

The broker's IP and port must be known by clients.

Namespace hierarchy used for topic filtering.

It may be the case that a published message is never consumed by any subscriber.

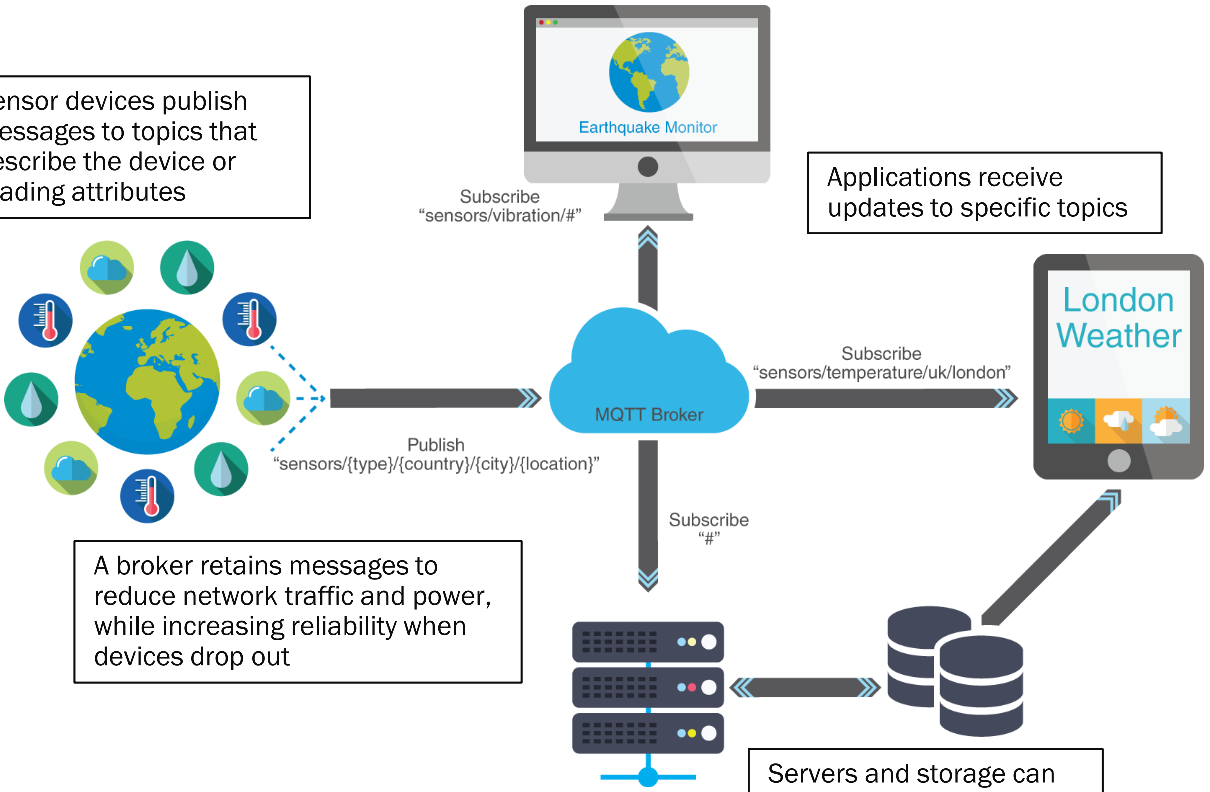
Architecture

Sensor devices publish messages to topics that describe the device or reading attributes

Applications receive updates to specific topics

A broker retains messages to reduce network traffic and power, while increasing reliability when devices drop out

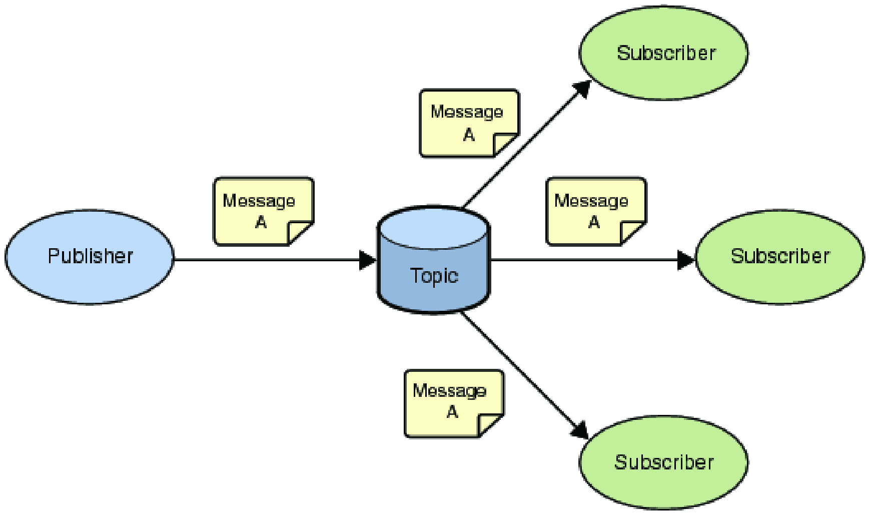
Servers and storage can be used to log all messages in a central location



CISTER - Research Center in Real-Time & Embedded Computing Systems

MQTT Concepts

- Topic
  - A string identifier of the form "a/b/c/d/e", where slashes separate subtopics
  - Clients _publish_ or _subscribe_ to topics
  - # is the glob symbol, i.e "a/#" means _all subtopics_ of "a"
- Broker
  - Server program that receives, buffers, and send messages based on topic
  - Each topic may contain up to 1 message value at a time, which may be retained
- Client
  - Anything that connects to the MQTT broker: serial devices, python scripts, java programs, etc.
  - Clients send a unique identifier when they connect to the broker, e.g. MAC address



A house publishes information about itself on:

<country>/<region>/<town>/<postcode>/<house>/energyConsumption
<country>/<region>/<town>/<postcode>/<house>/solarEnergy
<country>/<region>/<town>/<postcode>/<house>/alarmState
<country>/<region>/<town>/<postcode>/<house>/alarmState

And subscribes for control commands:

<country>/<region>/<town>/<postcode>/<house>/thermostat/setTemp

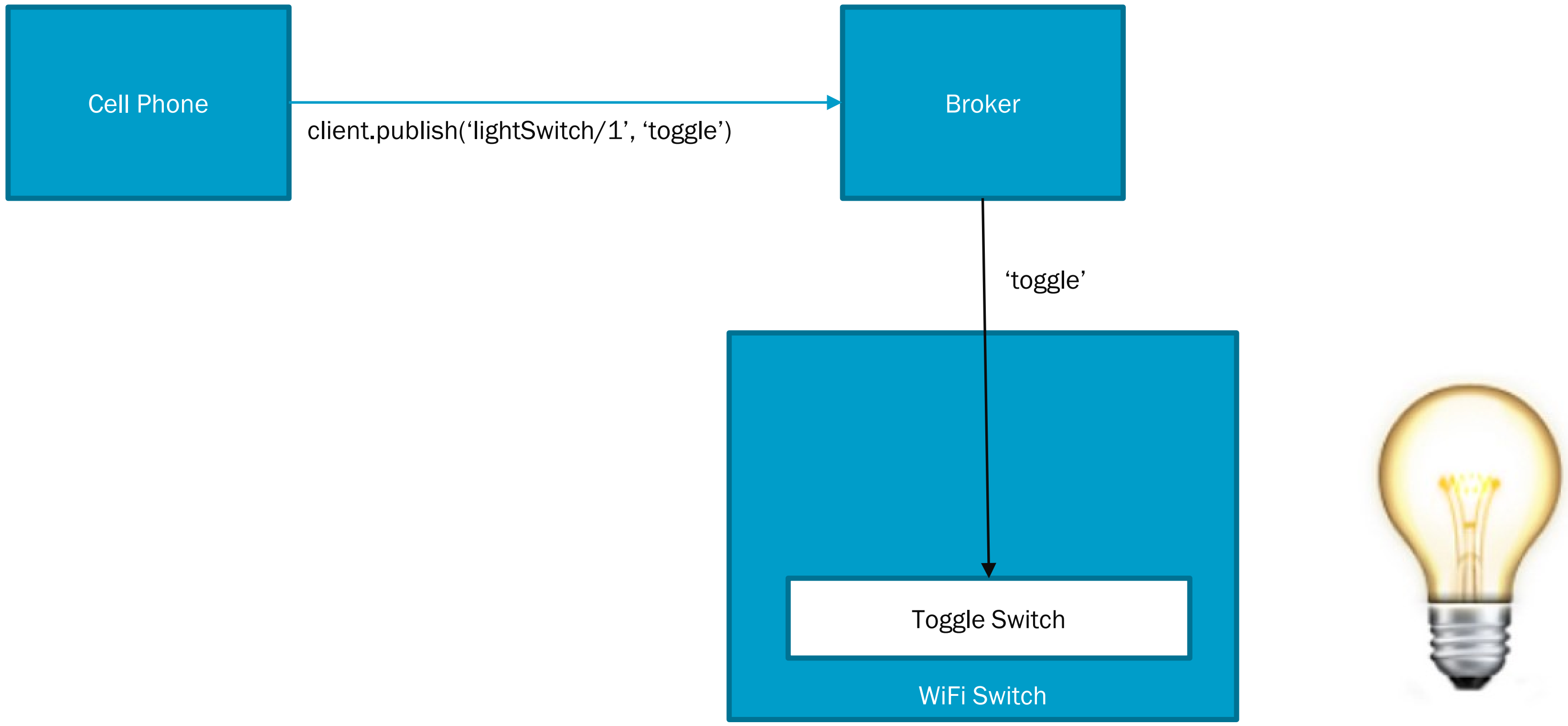
MQTT Concepts

- QoS
  - Level 0: Deliver message at most once (fire and forget)
  - Level 1: Deliver message at least once
  - Level 2: Deliver message exactly once
- Persistent Sessions
  - Client can connect to broker with the clean session flag set false
  - Messages are retained even if the client disconnects
  - Clients must also maintain messages when broker disconnects
  - Standard does not define how many or how long (until we run out of resources)
- Message Retention
  - Individual messages have a retention flag
  - Message is retained indefinitely

MQTT Concepts

- LWT (Last Will & Testament)
  - A device can indicate a final message to send when it disconnects from the broker
  - Often used to set status to “offline”

MQTT Hypothetical Light Switch



Details:

- MQTT always has a broker
- MQTT broker is the only "connection"
  - To detect whether device is connected to broker, use status topic
- MQTT messages are all plain text
  - No type information (e.g. double, string, table)
  - No meta-data fields (e.g. timestamp, alarm values)
- A subscriber can subscribe to an absolute topic or can use wildcards:
  - Single-level wildcards "+" can appear anywhere in the topic string
  - Multi-level wildcards "#" must appear at the end of the string
  - Wildcards must be next to a separator
  - Cannot be used wildcards when publishing
  - For example
    - UK/Hants/Hursley/SO212JN/1/energyConsumption
      - Energy consumption for 1 house in Hursley
    - UK/Hants/Hursley/+/+/energyConsumption
      - Energy consumption for all houses in Hursley
    - UK/Hants/Hursley/SO212JN/#
      - Details of energy consumption, solar and alarm for all houses in SO212JN

MQTT Message Format

- The message header for each MQTT command message contains a fixed header
  - Fixed length of 2 bytes
- Some messages also require a variable header and a payload.
- The format for each part of the message header:

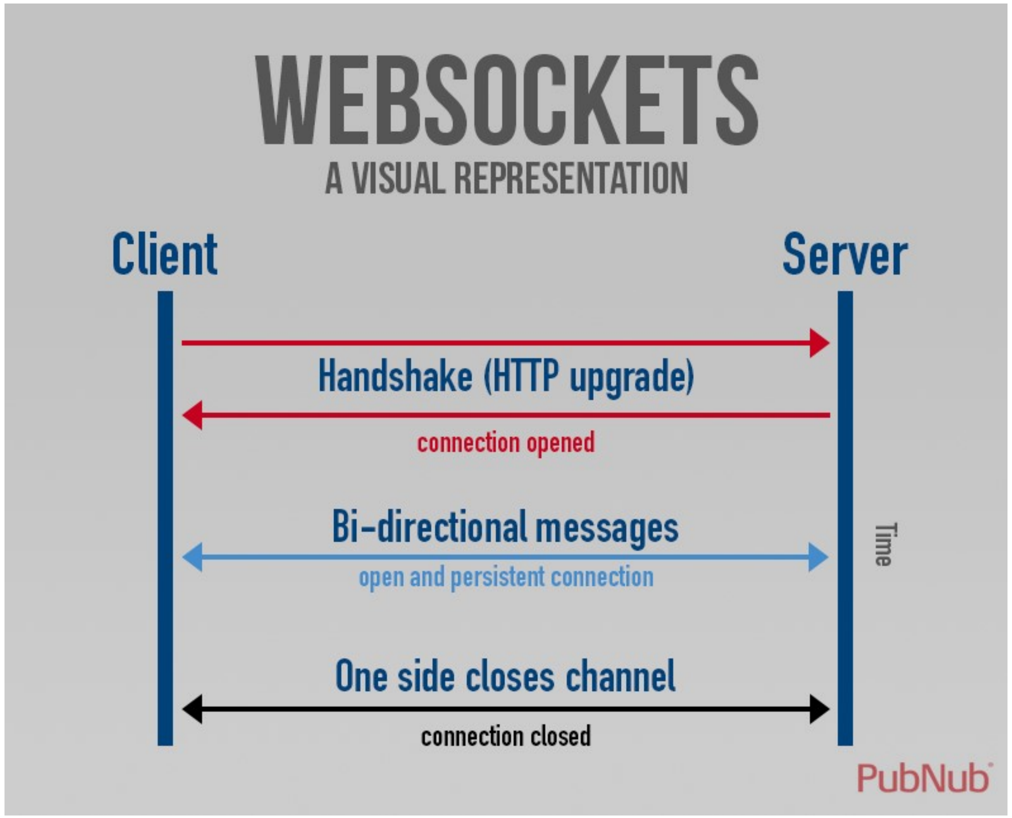
bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

— DUP: Duplicate delivery
— 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.

MQTT Communication

- Runs over TCP or TLS.
  - May use Websockets from within a browser.
  - MQTT–SN uses UDP packets or serial communication.
- As soon as you subscribe you receive the most recently published message



Outline

- IoT and Middleware
- 6LoWPAN
- CoAP
- MQTT
- Arrowhead

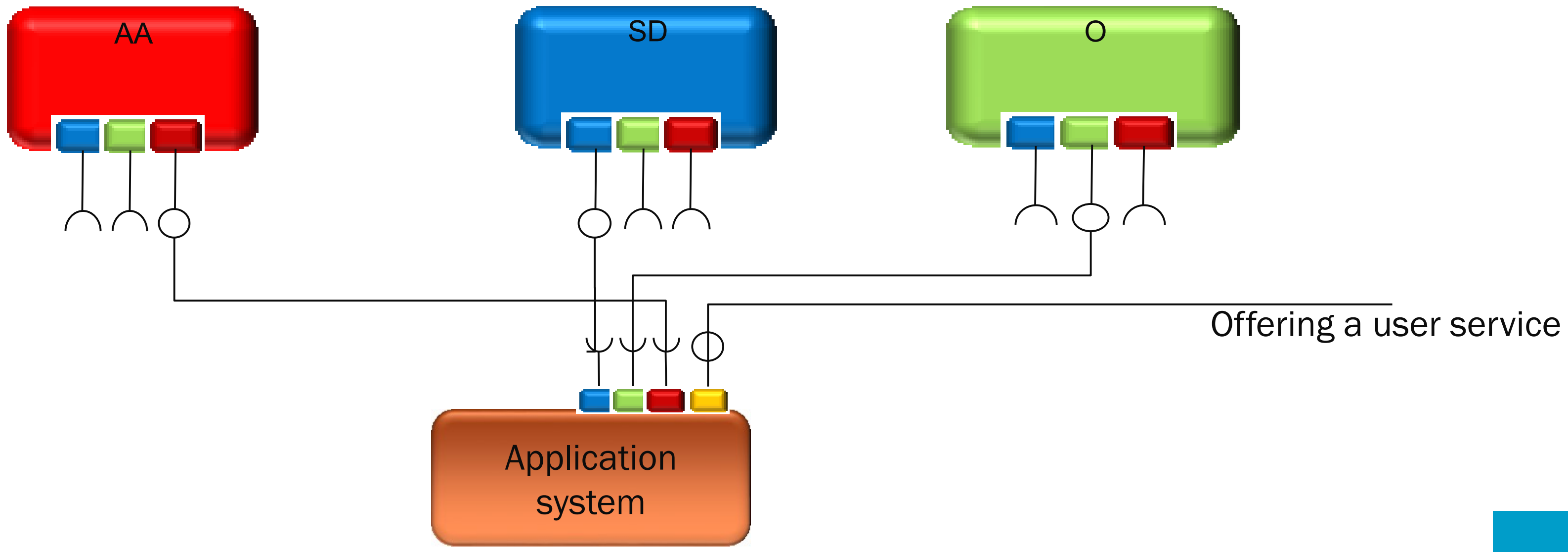
The Arrowhead Framework

- Service Oriented Architecture (SOA) approach supporting local cloud automation functionalities
- Local Clouds logically contain a set of application systems
- All interactions are mediated by **services**, which are produced / consumed by **systems**, which are run on **devices**

The Arrowhead Framework

Services are either

- User Services, providing the application functionalities on each particular scenario (business logic)
- Core Services, provided by the Arrowhead Framework and satisfying non-functional requirements (housekeeping)
  - At least: Authentication & Authorization (AA) Service for devices, Registration Service (SD) for devices and services, and Orchestration (O) Service to look-up for devices/services, and to create more complex services



The Arrowhead Framework

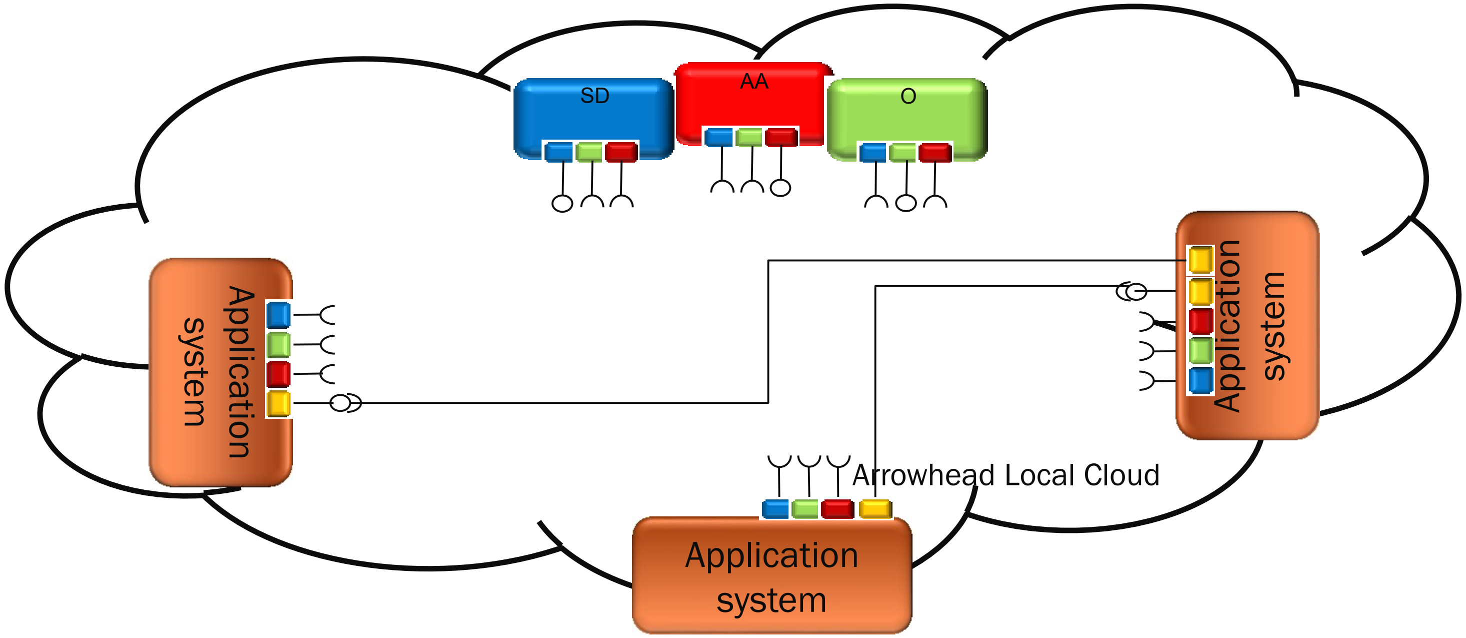
- Mandatory services:
  - Service Discovery;
  - Orchestration;
  - Authorization and Authentication
- A set of optional services:
  - QoS Manager
  - Configuration Manager
  - Event handler
- A detailed documentation system
- Several kinds of encryption protocols are supported

The Local Cloud

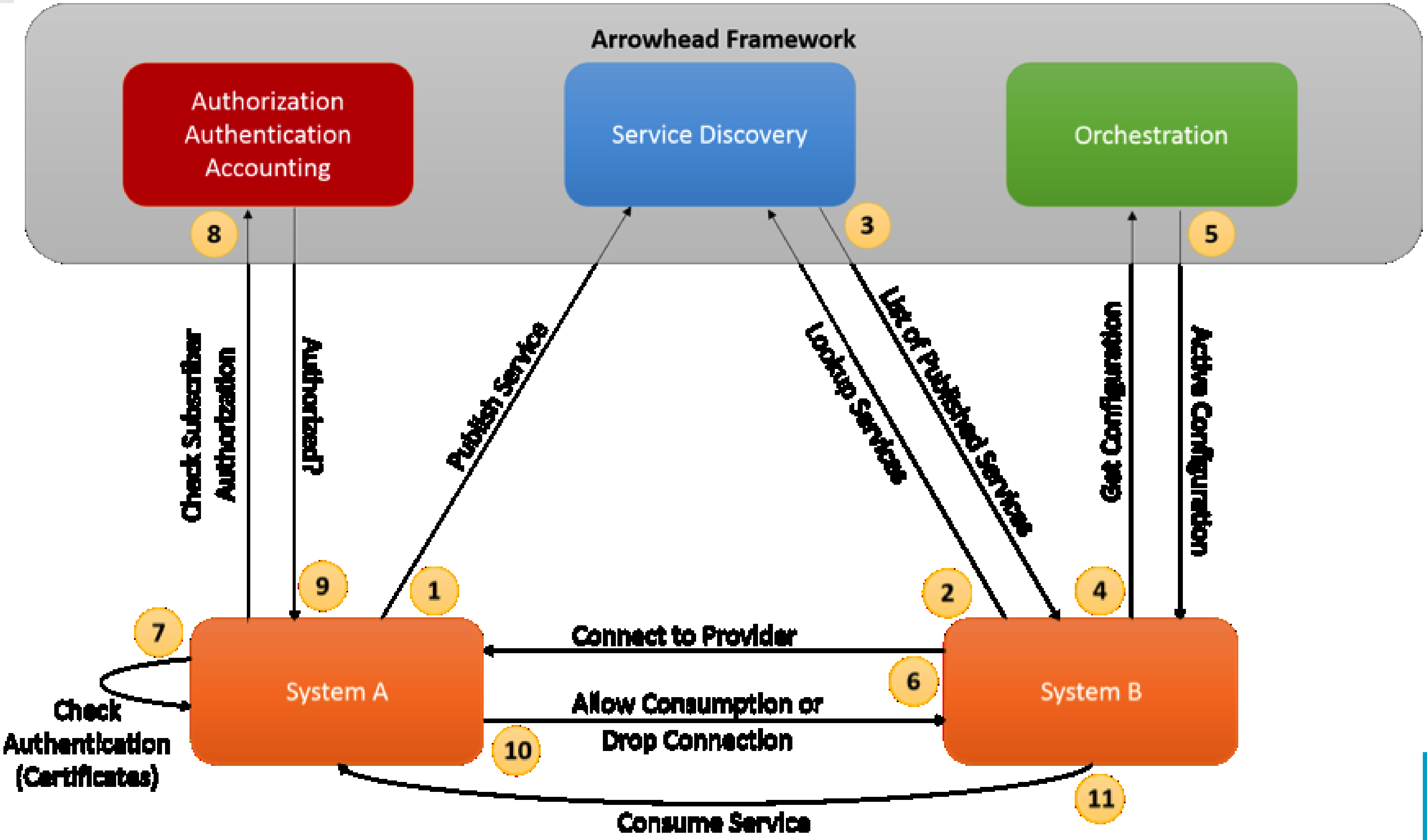
An Arrowhead-enabled SoS is based on the concept of Local Cloud:

- A bounded set of computational resources used by stakeholders to attain a goal
- Controls security
  - Restricts access to authenticated Devices/Systems/Services.
- Autonomous
  - Contains all core services needed to be able to function

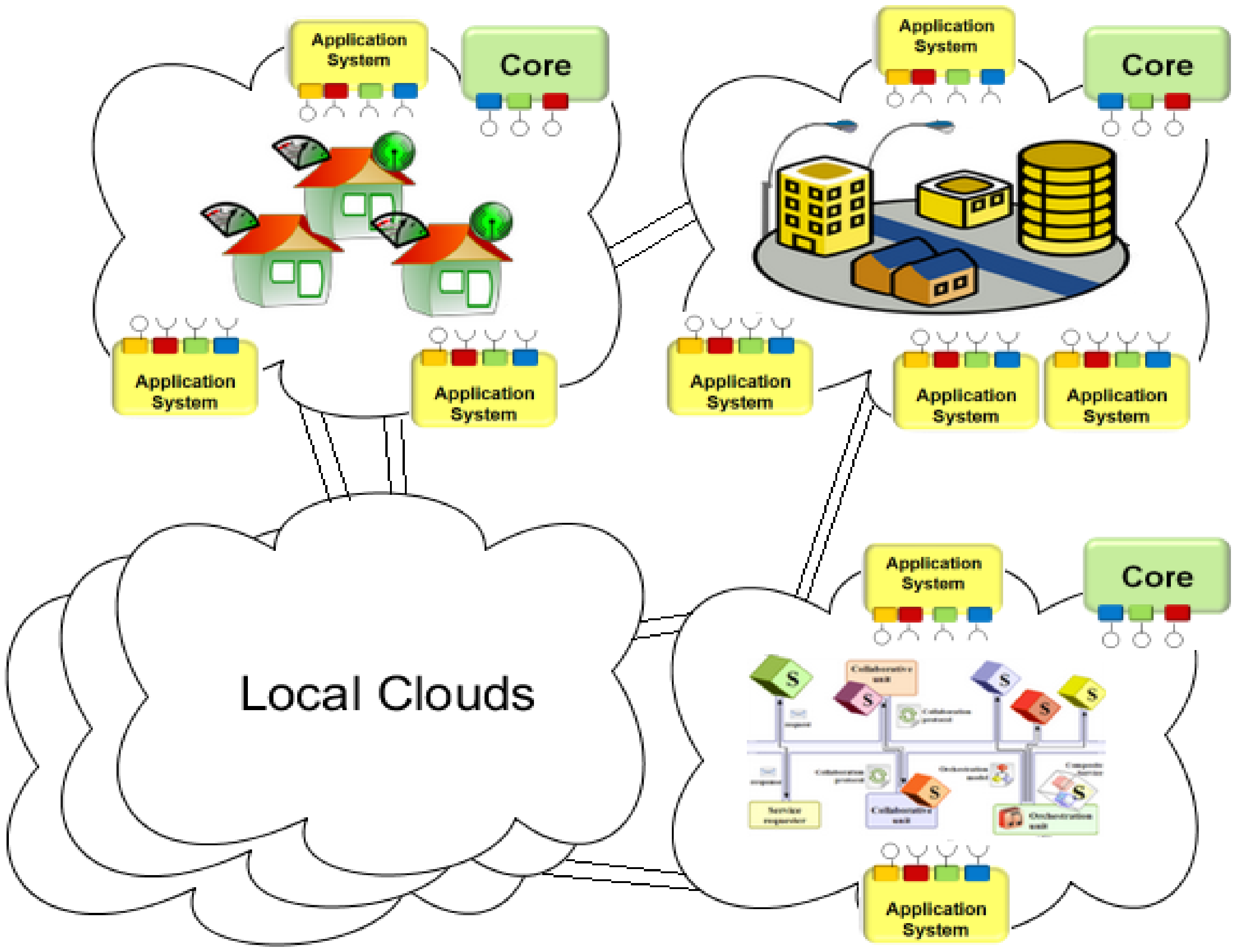
Relating QoS to a Local Cloud simplifies QoS monitoring and reduces the QoS managing complexity.



Arrowhead Architecture



Arrowhead Architecture


Application System
Core
Application System
Core
Application System
Application System
Application System
Application System
Application System
Local Clouds
Application System
Core
Application System