Design and Development of IoT Applications

Dr. –Ing. Vo Que Son

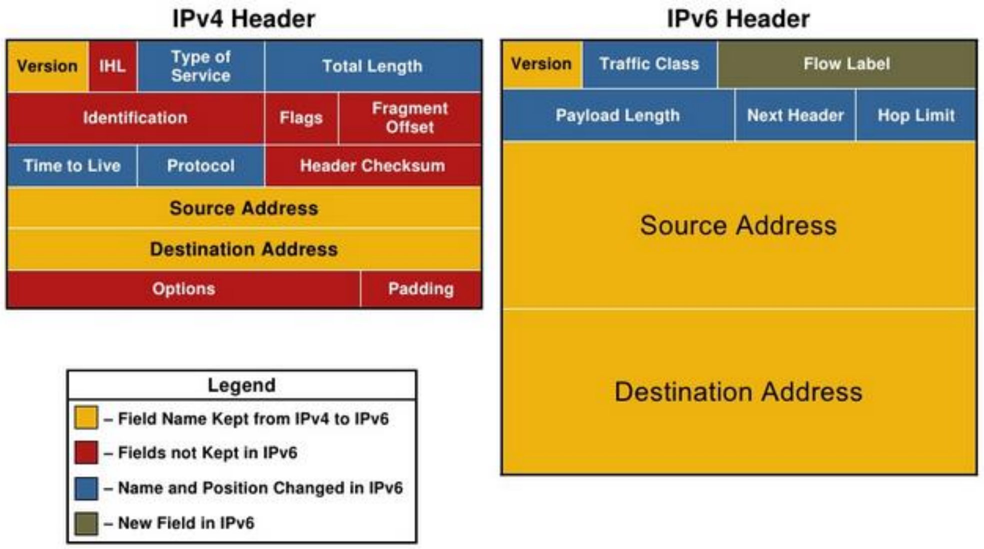
Email: sonvq@hcmut.edu.vn

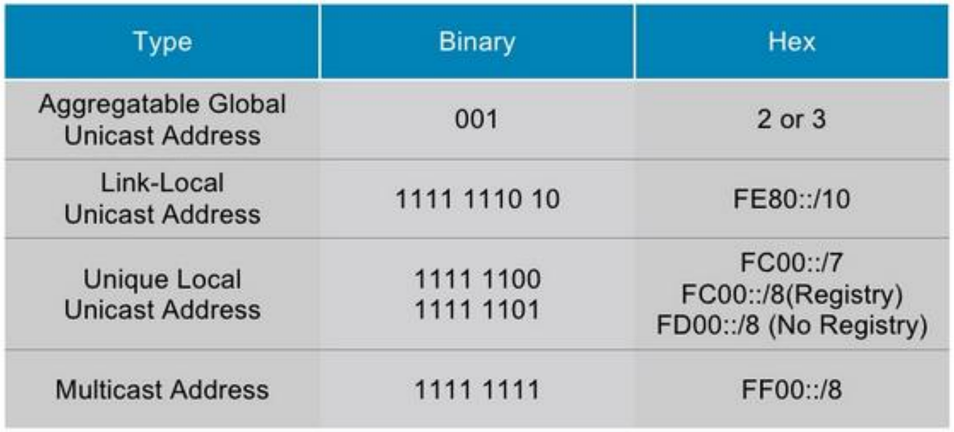
Content

- Chapter 7: Industrial IoT standards
  - ZigBee Technology
  - Dynamic Network protocols
  - Backhaul networks for Home Automation
  - Sub-1GHz technology
  - Thread protocol stack
- Chapter 8: Wireless Embedded Internet
  - ICMPv6
  - Auto-configuration & Neighbor Discovery
  - IP routing in WSNs: RPL
  - Embedded web – REST/CoAP
  - MQTT-SN
  - Huma-Machine-Interface

IPv6 review

- ❑ IPv4 and IPv6 comparison
- ❑ 6LoWPAN: IPv6 Header Compression: support L3 routing (Route-Over)
- ❑ 6LoWPAN Neighbor Discovery

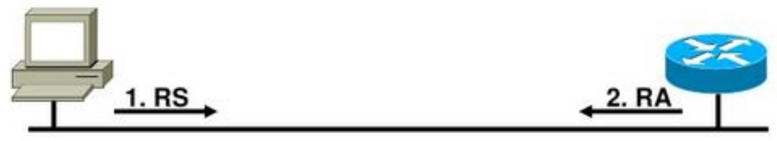




Type	Binary	Hex
Aggregatable Global Unicast Address	001	2 or 3
Link-Local Unicast Address	1111 1110 10	FE80::/10
Unique Local Unicast Address	1111 1100 1111 1101	FC00::/7 FC00::/8(Registry) FD00::/8 (No Registry)
Multicast Address	1111 1111	FF00::/8

ICMPv6

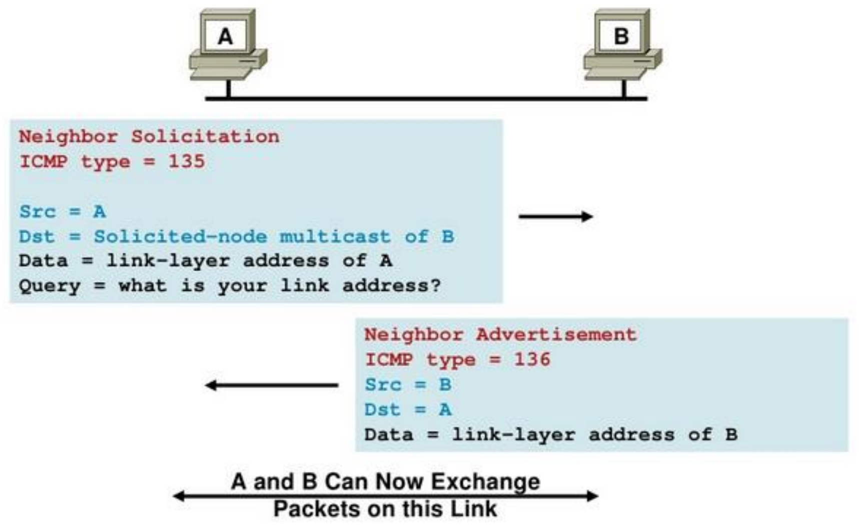
- ❑ Internet Control Message Protocol v6
  - ❖ Modification from ICMP for IPv4
- ❑ Neighbor Discovery (ND)
  - ❖ Replace ARP, ICMP
  - ❖ Reachability of Neighbors
  - ❖ Hosts use to discovery routers, auto-configuration of addresses
  - ❖ Duplicate Address Detection (DAD)
  - ❖ 5 ND messages:
    - Router Solicitation (type 133)
    - Router Advertisement (type 134)
    - Neighbor Solicitation (type 135)
    - Neighbor Advertisement (type 136)
    - Redirect (type 137)



1—ICMP Type = 133 (RS)
Src = link-local address (FE80::1/10)
Dst = all-routers multicast address (FF02::2)
Query = please send RA

2—ICMP Type = 134 (RA)
Src = link-local address (FE80::2/10)
Dst = all-nodes multicast address (FF02::1)
Data = options, subnet prefix, lifetime, autoconfig flag

- Router Solicitations (RS) are sent by booting nodes to request RAs for configuring the interfaces
- Routers send periodic Router Advertisements (RA) to the all-nodes multicast address

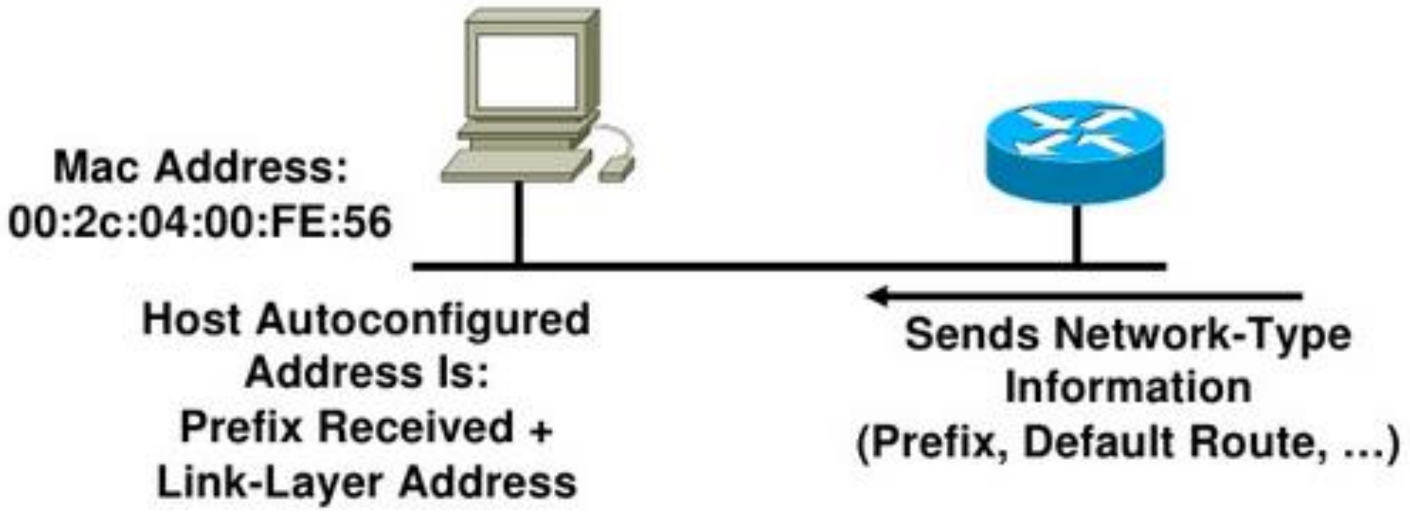


```
Neighbor Solicitation
ICMP type = 135

Src = A
Dst = Solicited-node multicast of B
Data = link-layer address of A
Query = what is your link address?
```

```
Neighbor Advertisement
ICMP type = 136
Src = B
Dst = A
Data = link-layer address of B
```

A and B Can Now Exchange Packets on this Link

Auto-configuration

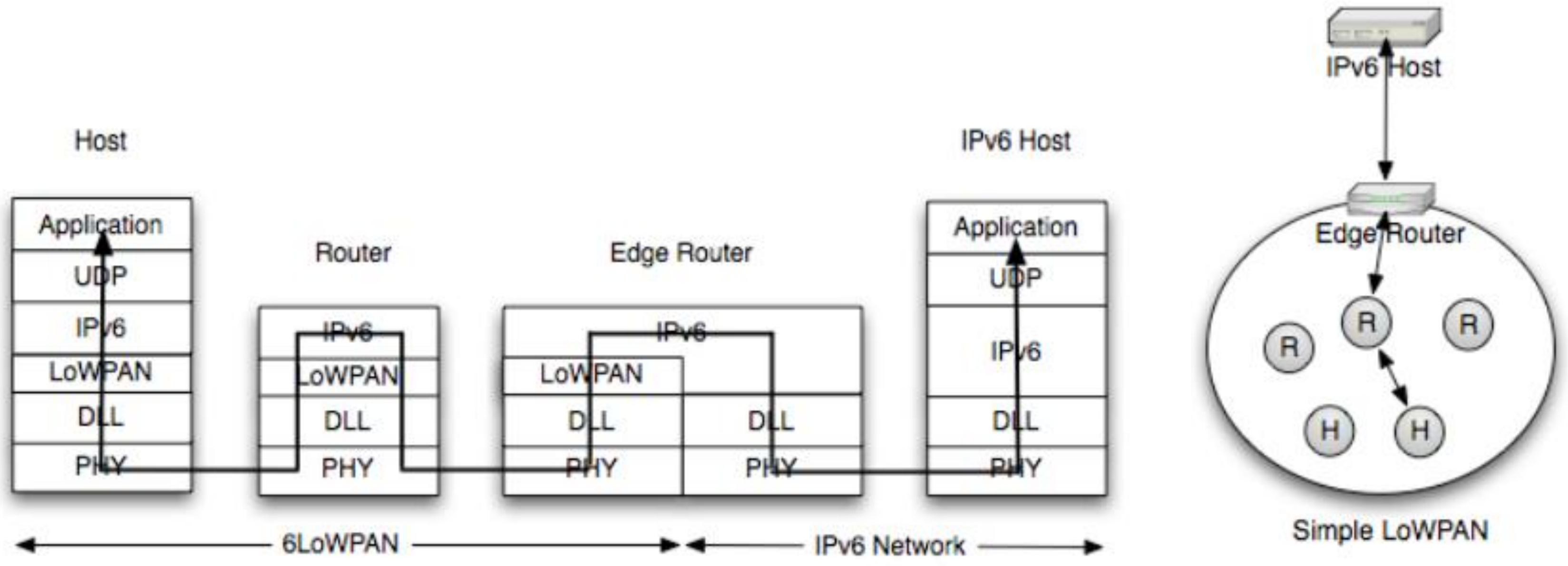

Mac Address:
00:2c:04:00:FE:56
Host Autoconfigured Address Is: Prefix Received + Link-Layer Address
Sends Network-Type Information (Prefix, Default Route, ...)

Larger Address Space Enables:

- The use of link-layer addresses inside the address space
- Autoconfiguration with "no collisions"
- Offers "plug-and-play"

IP Routing in WSNs

- Low power and Lossy Network (LLN) consists of an edge router (also called as LLN Border Router LBR), Router(R) and Host(H):
  - H chooses only the default router
  - R forwards traffic
  - ROLL operates only within LoWPAN and terminates at LBR



RPL: new trend in WSN

- Routing Protocol for Low-power and Lossy Network (LLNs)
- LLN: *"Low power and Lossy networks (LLNs) are typically composed of many embedded devices with limited power, memory, and processing resources interconnected by a variety of links, such as IEEE 802.15.4 or Low Power WiFi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (HVAC, lighting, access control, fire), connected home, healthcare, environmental monitoring, urban sensor networks, energy management, assets tracking and refrigeration"*

LLNs: Fact

- LLNs have not to be confused with Wireless Sensor Networks
  - A WSN is an LLN, but not the opposite !
- LLNs are not necessarily wireless.
  - Industrial automation networks are wired.
- LLNs will use IPv6. No discussions about this.
  - IPv4 is a dead horse.
- LLN nodes can not be easily configured.
  - Typically, they're cheap, small and with very limited memory/CPU

LLNs: Open Issues

- Transmission efficiency (6LoWPAN)
  - Nodes run on battery. Saving energy is a premium. IPv6 Headers must be compressed.
- Node bootstrap (6lowpan-nd)
  - Configuration is a nightmare, and ‘normal’ IPv6 auto configuration is too chatty, NS/NA as well, etc,.
- Routing (ROLL)
  - Routing requirements are scenario-based. Mainly:
    - one-to-one (configuration, data reading)
    - many-to-one (data gathering)
    - one-to-many (data dissemination)

DAG and DODAG

- ❑ DAG: Directed Acyclic Graph (Fig. B)
  - ❖ **DAG root**: a node within the DAG that has no outgoing edge
  - ❖ All paths **terminated** at a **DAG root**
  - ❖ All edges are oriented in such a way that no cycles exist (Fig. A is not a DAG)
- ❑ DODAG: Destination Oriented DAG (Fig. C)
  - ❖ Defines a DAG that forms paths to a **single logical root**

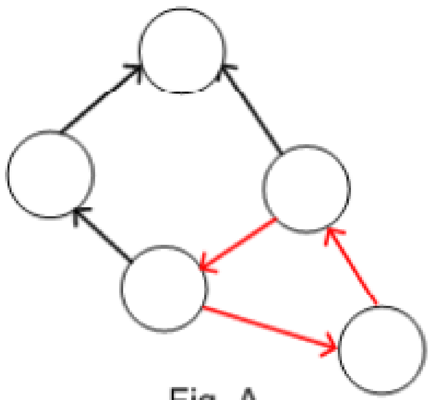


Fig. A

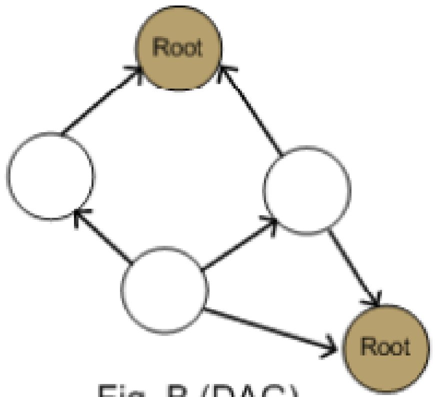


Fig. B (DAG)

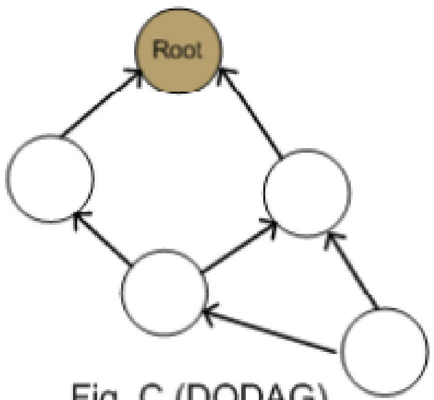


Fig. C (DODAG)

Why DODAG for RPL?

- ❑Traffic in LLN
  - ❖Mainly for P2MP and MP2P traffic flows
- ❑As opposed to a tree topology, DODAG enables (re)route around loss/interference and easily adapts transient changes
- ❑RPL organizes a topology as a Directed Acyclic Graph (DAG) that is partitioned into one or more Destination Oriented DAGs (DODAGs)
  - ❖The root in DAG acts as a sink – P2MP and MP2P traffic
    - DODAG root -> LBR
    - Information stored and processing at other nodes can be reduced

RPL Control Messages

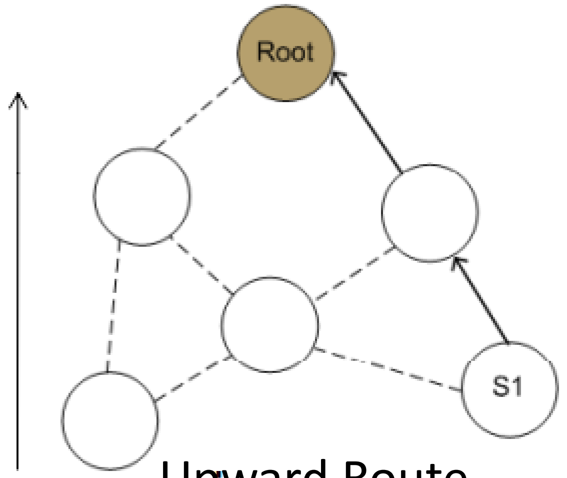
- RPL defines a new ICMPv6 message with 3 possible types:
  - *DAG Information Object* (DIO) - carries information that allows a node to discover an RPL Instance, learn its configuration parameters and select DODAG parents
  - *DAG Information Solicitation* (DIS) - solicit a DODAG Information Object from a RPL node
  - *Destination Advertisement Object* (DAO) - used to propagate destination information upwards along the DODAG.

Upward and Downward Routes

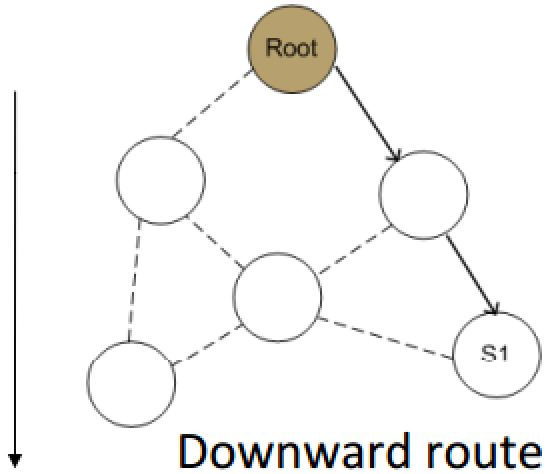
- **Upward**: the direction from leaf nodes towards the DODAG root
  - Data flow towards the DAG root - MP2P traffic flows

- **Downward**: the direction from the DODAG root towards leaf nodes
  - Data flow away from the DAG root – P2MP traffic flows

- **Point-to-point** data flow via up and down route



Upward Route



Downward route

RPL Metrics

- A metric defines how a node (or link) will affect the path. A metric is used to compute the node's *rank* and to decide about preferred Parents.
- Each metric is carried in a DAG Metric Container, embedded in the relevant RPL control messages.
- *Node metrics*: Node-related information, such as: Node State and Attributes, Node Energy, Hop-Count
- *Link metrics*: Link-related information, such as: Throughput, Latency, Link Reliability, Link Color
- Routing metric:
  - RFC 6551: path calculations (node energy, hop count, throughput, latency, etc…

Rank vs OF

- Rank: node's individual position relative to other nodes with respect to a DODAG root
  - Increases in the Down direction and decreases in the Up direction
  - Rank computed depends on the **Objective Function** (OF) - e.g. calculated as a simple topological distance (hop counts) or may also be calculated as a function of link metrics
- Objective Function (OF): Defines the optimization objectives to compute the Rank
  - Optimization objectives : minimizing energy, minimizing latency, or satisfying some constraints, etc…
  - Dictates how routers (parents) inside the DODAG are selected.

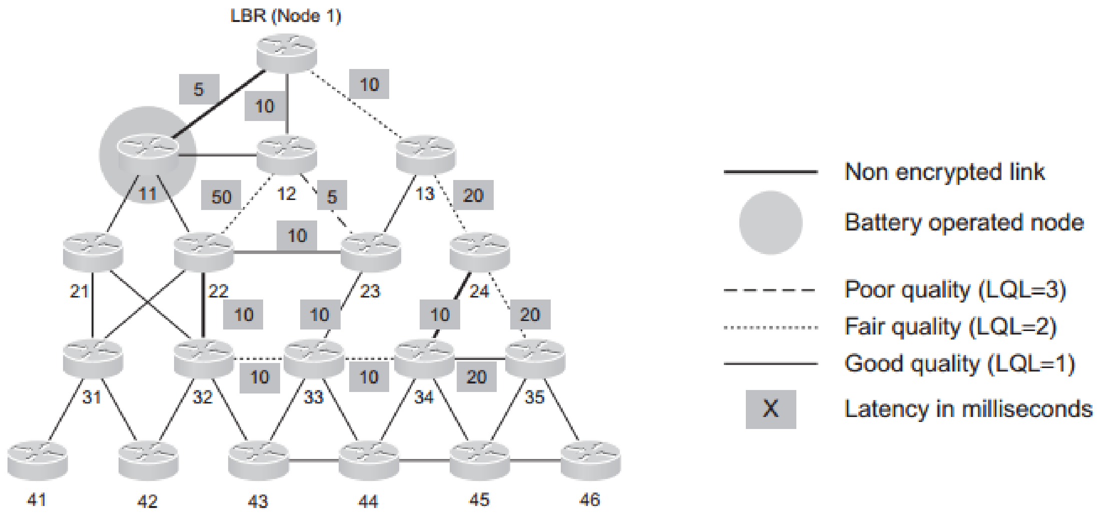
Objective Functions (OF)

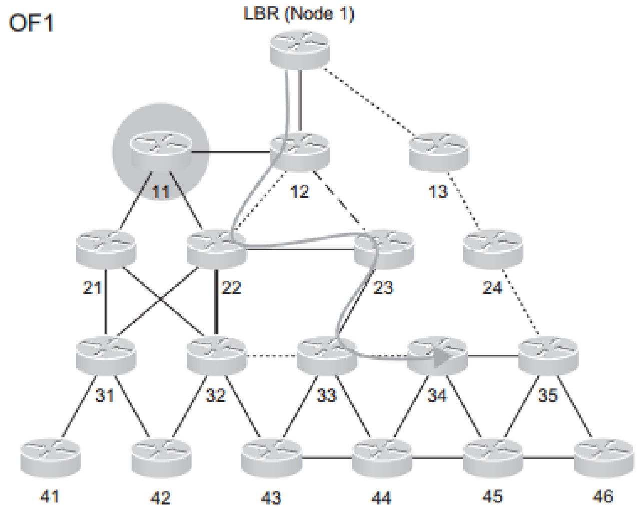
- RPL was designed as a generic protocol, can be adapted by OFs. There exists several OFs in operation on the same node and 6LoWPAN network - to carry traffic with very different requirements of path quality. For example, several DODAGs may be used with:
  - **OF1**: ‘Find the best path in terms of ETX [Expected Transmissions] values (metric) and avoid non-encrypted links (’constraint)’ or
  - **OF2**: ‘Find the best path in terms of latency (metric) while avoiding battery- operated nodes (constraint)’
- RPL (generic core):
  - Parent (feasible successor) selection rules
  - Loop avoidance based on Rank
- The OF functions:
  - Selects the Preferred Parent in parents set
  - Computes Rank

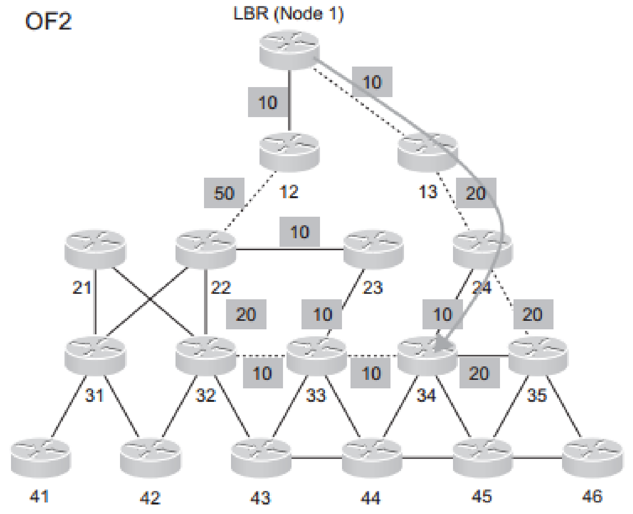
Objective Functions (OF)

- Uses a combination of metrics (ETX, hop counts, latency, etc) and constraints (avoiding battery powered node, avoiding non-encrypted nodes) to compute the ‘best’ path
- OF does not necessarily specify the metric/constraints but does dictate some rules to form the DODAG (for example, the number of parents, back-up parents, use of load-balancing, etc)
- The OF accounts for
  - Statistical information (ETX, packet losses, etc)
  - Boolean information (battery operated)
  - Physical information (max bandwidth)
  - Configuration (preference, security level, …
- Two proposed OFs
  - RPL Objective Function 0, **OF0** (RFC 6552)
  - The Minimum Rank Objective Function with Hysteresis, **MRHOF** (draft-ietf-roll-minrank-hysteresis-of-11)
- The Trickle Algorithm
  - RFC 6206: to exchange information in a highly robust, energy efficient, simple, and scalable manner

Example of 2 different OFs

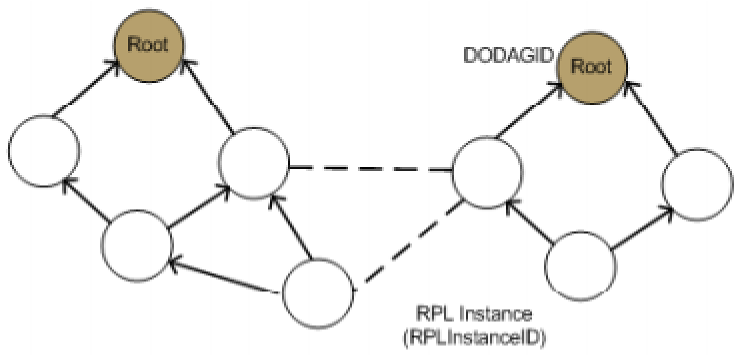






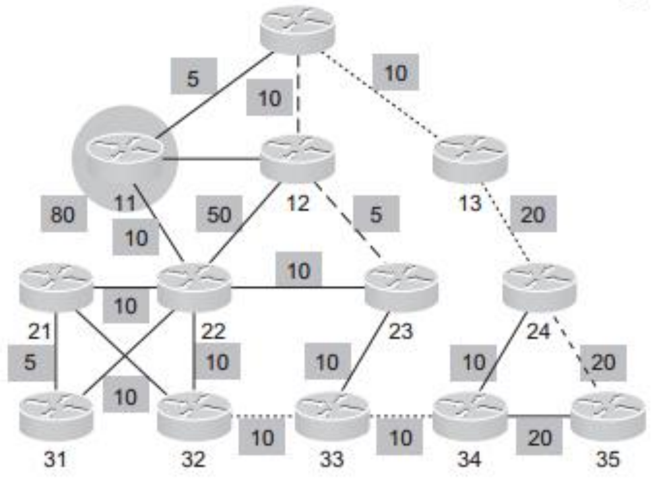
RPL Instance

- **RPL Instance**: Composed of one or more disjoint DODAGs, each of which has an unique DODAGID
  - All DODAGs in the same **RPL Instance** use the same OF (Objective Function)



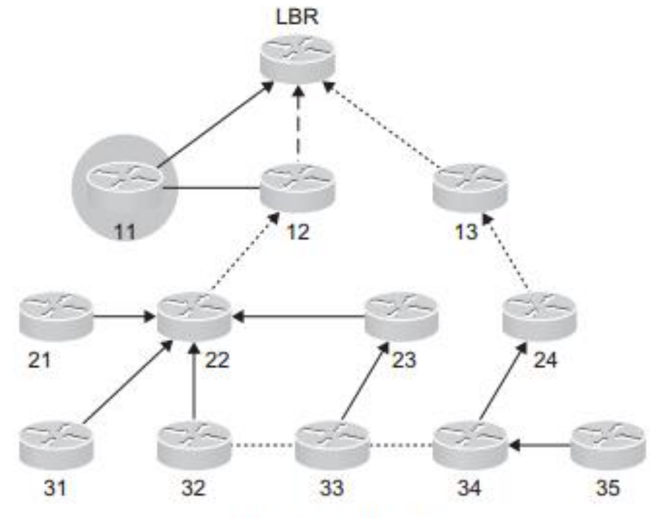
- **RPLInstanceID**: identifies a set of one or more Destination Oriented DAGs (DODAGs)
  - **RPLinstanceID** can be used to identify different OFs
- **RPLInstanceID** and **DODAGID** uniquely identifies a single DODAG in the network
  - different DODAGID = different root node

RPL Instance Example: Multiple DAGs

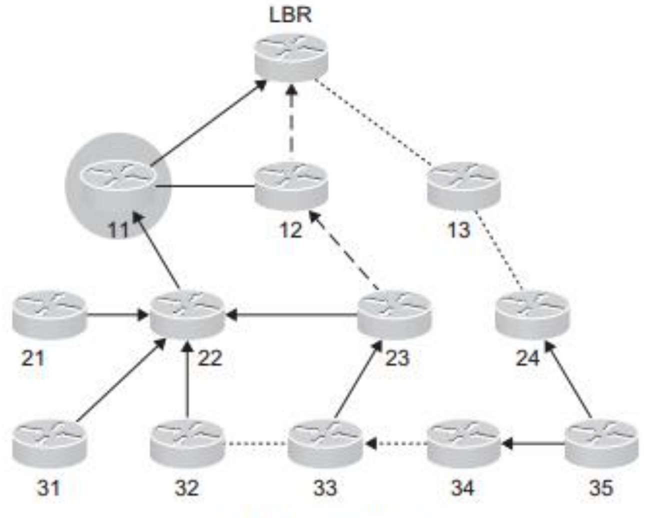


●	Battery operated node
- - - - -	Poor quality (LQL=3)
············	Fair quality (LQL=2)
———	Good quality (LQL=1)
X	Latency in milliseconds

DAG instance 1: High quality – no battery operated nodes
DAG instance 2: Low latency



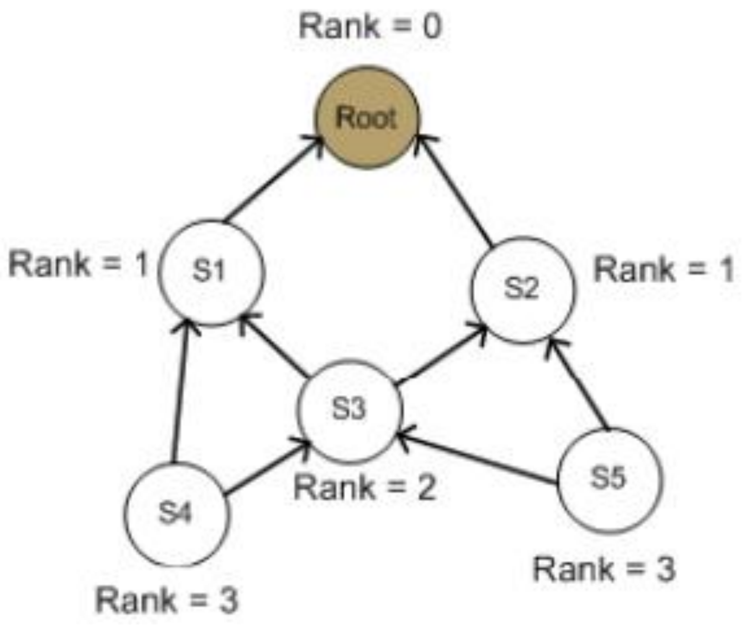
DAG instance 1



DAG instance 2

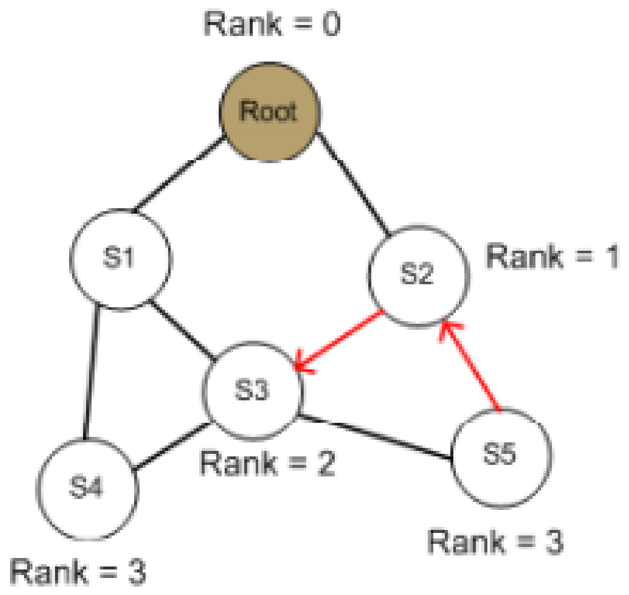
DAG Construction

- Distance-Vector
  - advertise path cost (routing metric)
  - choose a parent (default router/ next hop) that minimize path cost
  - Avoids loops & count-to-infinity
- Assign every node a rank
  - Node Rank: Relative position within a DODAG
  - Rank strictly decreasing towards the root
- Parents & Siblings
  - Parents –nodes with lower ranks
  - Siblings –nodes with same ranks



Data Path Validation & Loop Detection

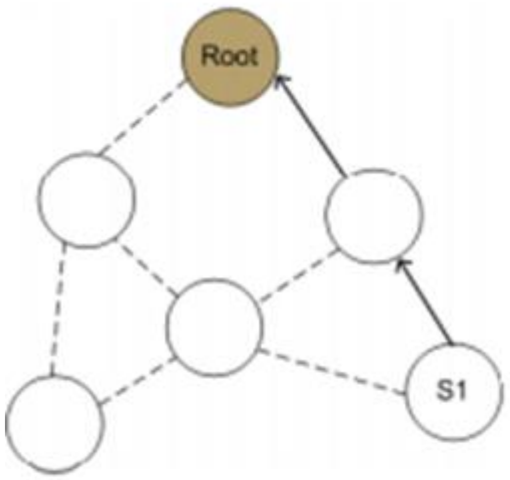
- *RPL packet information* (Up/Down route) is transported inside a data packet
- On-demand loop detection using data packets
  - maintaining an up-to-date routing topology can waste energy for LLNs with infrequent data
- An inconsistency between the routing decision for a packet (upward or downward) and the Rank relationship between the two nodes indicates a ***possible loop***
  - E.g. if S3 receives a packet flagged as moving in the *upward direction*, and if that packet records that the transmitter (S2) is of a *lower Rank* than the receiving node (S3)
  - S3 should initiate a local repair



RPL does not guarantee the absence of loops but rather tries to avoid them and specifies mechanisms to detect loops via data path validation

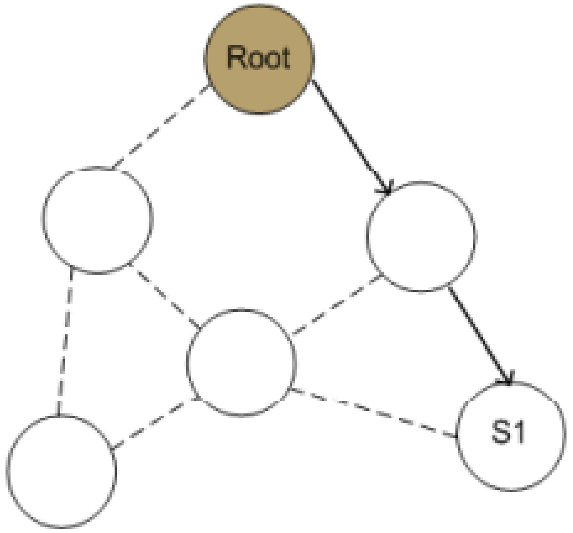
Creation of Upward routes – DIO messages

- Some nodes are configured to be DODAG roots
- DODAG roots 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
  - to join a new DODAG (thus selecting DODAG parents),
  - or to maintain an existing DODAG, according to the specified Objective Function
- Nodes update routing table entries, for the destinations specified by the DIO message
  - Nodes can decide to join one or more DODAG parents as the next-hop for the default route



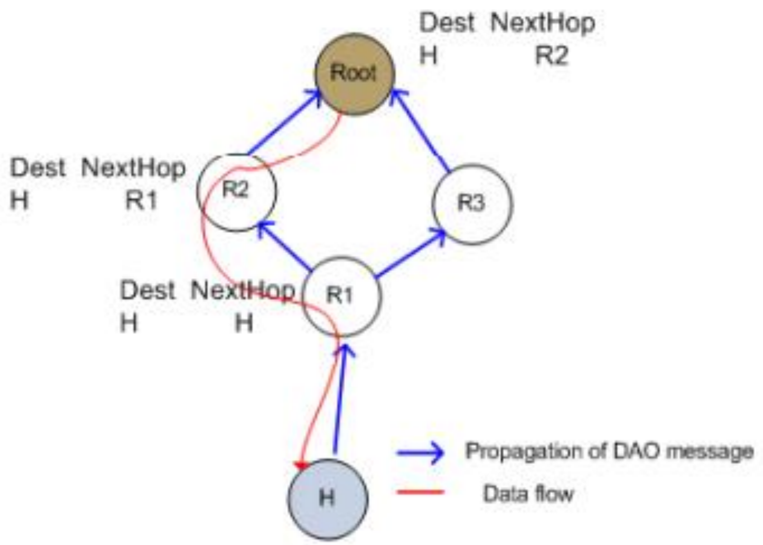
Creation of Downward Routes: DAO Messages

- RPL uses DAO messages to establish downward routes: nodes inform parents of their presence and reachability to descendants by sending a DAO message
- DAO messages are an optional feature for applications that require P2MP or P2P traffic
- RPL supports two modes of downward traffic
  - Storing (fully stateful): aggregate routes
  - Non-storing (fully source routed): attach a next-hop address to the reverse route stack contained within the DAO message

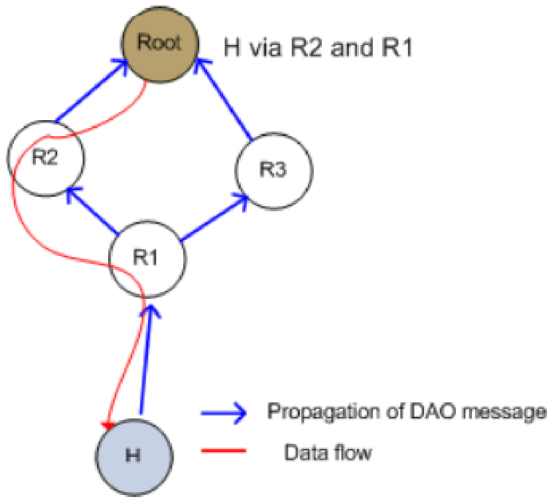


Storing Mode vs Non-Storing mode

- Storing mode
  - Creation of routes
    - DAO propagates towards the root via the routers
    - Each router maintains routes to each router in WSNs
  - Data goes hop by hop (each router forwards the packet to the right next hop)



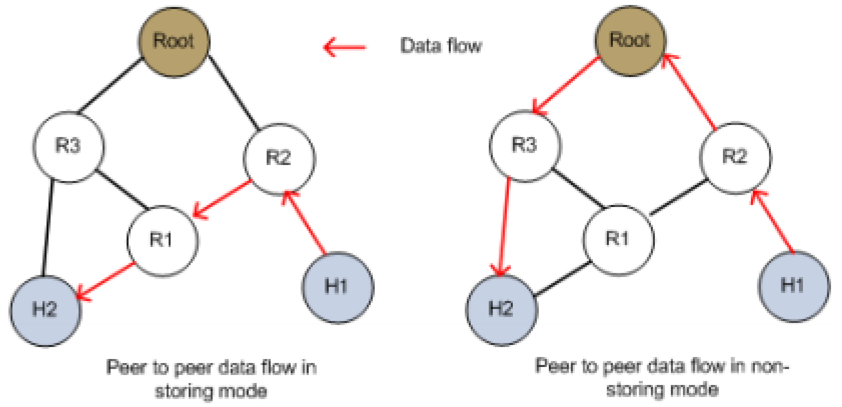
- Non-storing mode
  - Only the root stores routes to the all routers in WSNs
  - Calculates routes to all destination by piecing together the info (address of the routers) collected from DAO messages
  - Data forwards using *source routing*



Storing Mode vs Non-Storing mode

- Storing mode:
  - Requires more storage capacities and memory on each router
  - Efficient communications
  - Extra mechanism to avoid loops (e.g. use of RPL packet information)
- Non-storing mode:
  - Does not require additional state on the routers, but increases the message overhead
  - Traffic through the root node increases for internal traffic (e.g. P2P traffic within a 6LoWPAN)
  - Loop can be avoided by checking headers?

The data packet travels 'up' to a common ancestor at which point it is forwarded in the 'down' direction to the destination

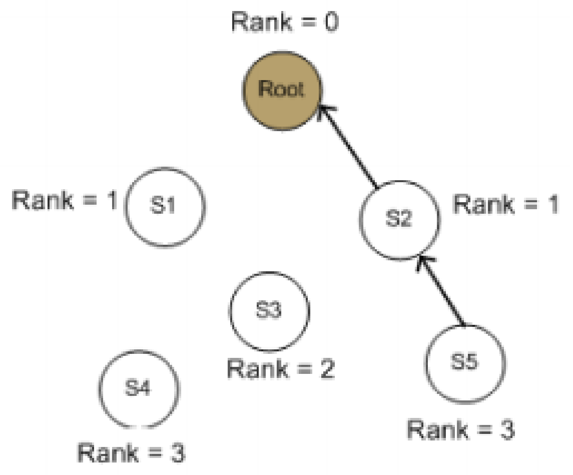


Peer to peer data flow in storing mode

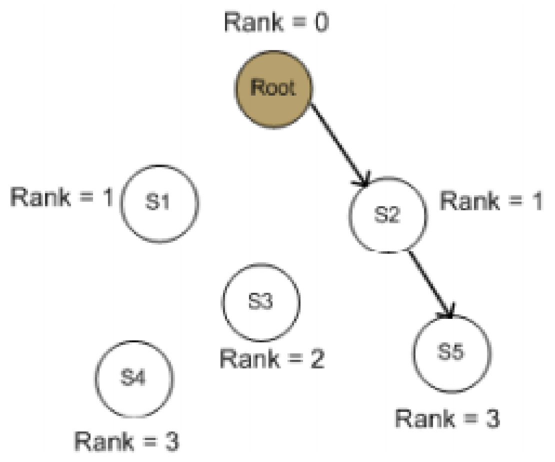
Peer to peer data flow in non-storing mode

Route Construction and Forwarding

- Up routes towards nodes of decreasing rank
  - Via DAG parents (always forward to lower rank when possible)
  - may forward to sibling if no lower rank exists

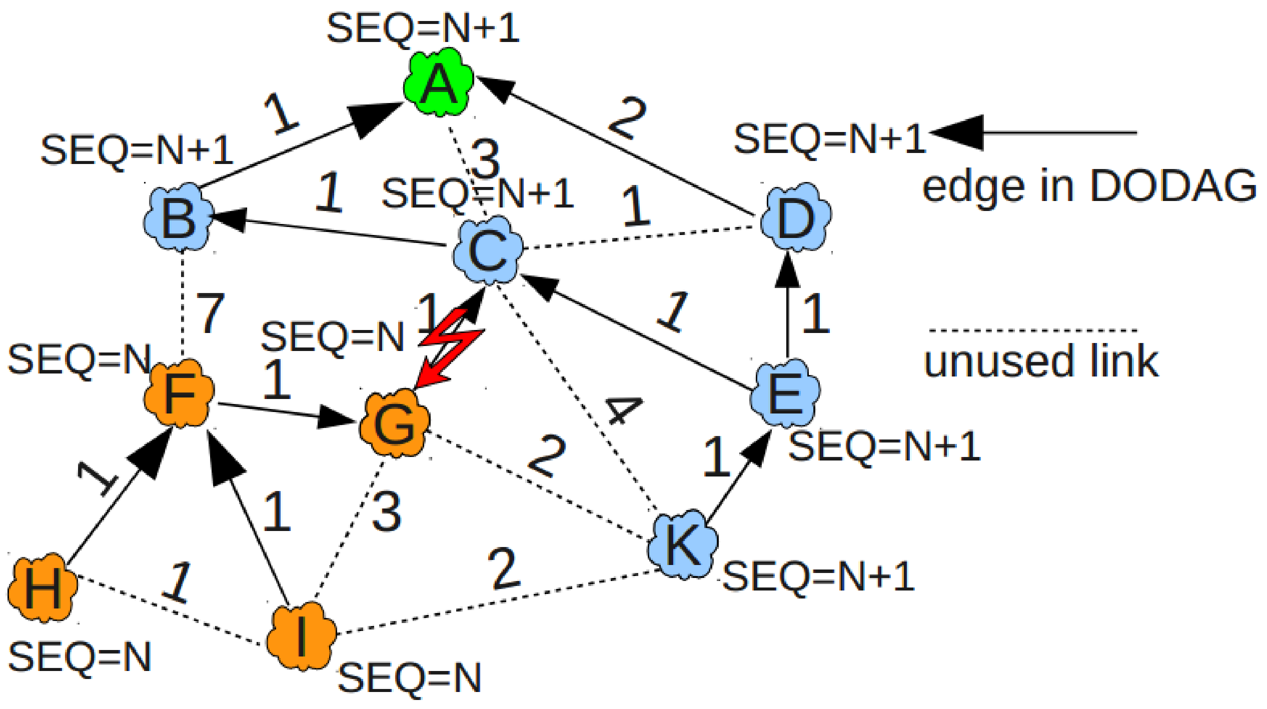


- Down routes towards nodes of increasing rank
  - Nodes inform parents of their presence and reachability
  - Forward hop by hop (storing mode) or using source routing (non-storing mode)



RPL Repair

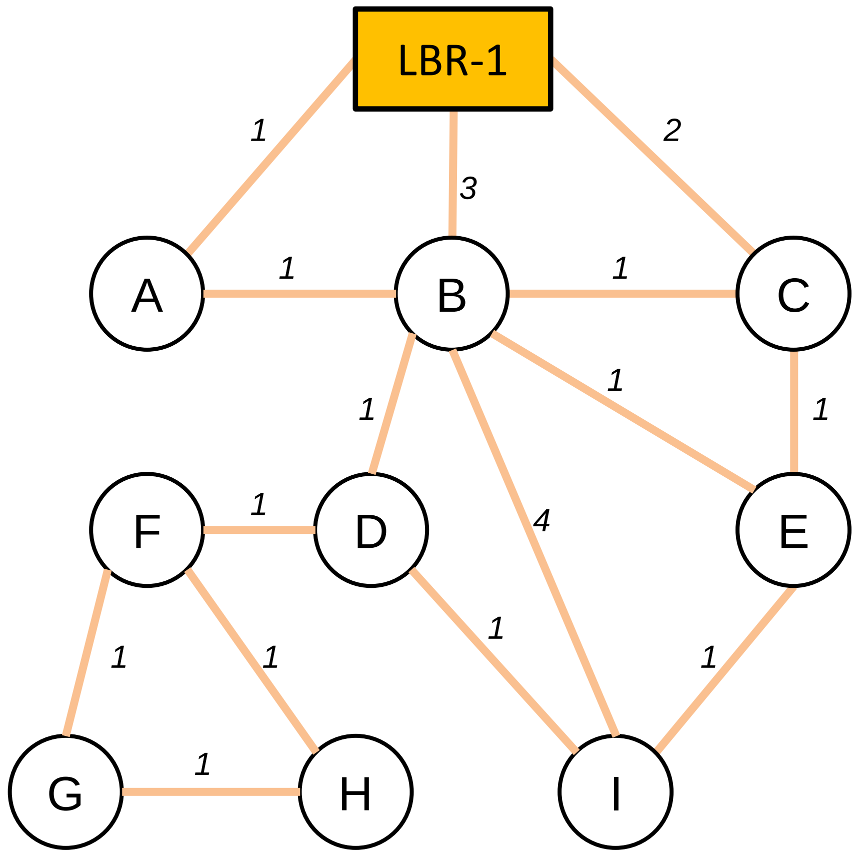
- Global repair: makes use of DODAG Sequence Numbers
- Local repair - poison the sub-DODAG by advertising the rank of INFINITY



Summary: Building DODAG

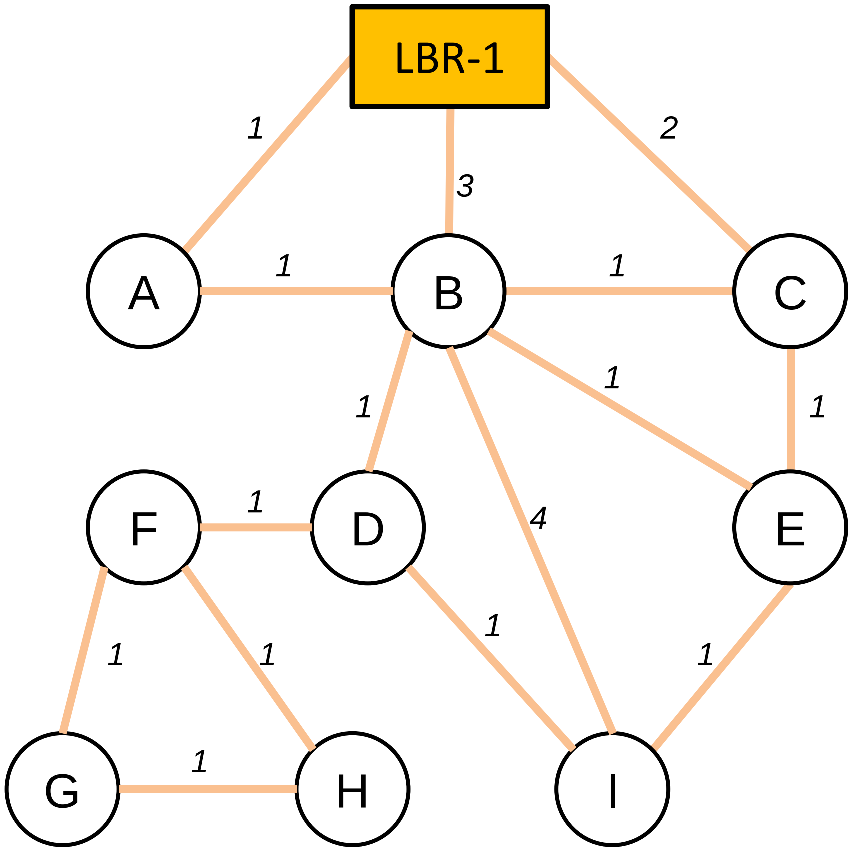
- Routers sends DIO (DODAG Information Object) messages
  - Uses link local multicasting
  - Indicates a respective rank (= position), according to a metric (e.g. hop count) w.r.t. the root
  - Upward routes (paths towards the root) are created
  - Defines the optimization objective when forming upward route
    - Link properties: (Reliability, Latency), Node properties: (Powered or not)
    - Objective: optimize paths based on one or more metrics
- Node issues DAO (Destination Advertisement Object) messages
  - Propagates via DAO selected parents discovered during DIO propagation
  - DAO can be acknowledged
  - Downward routes are created

DAG Construction



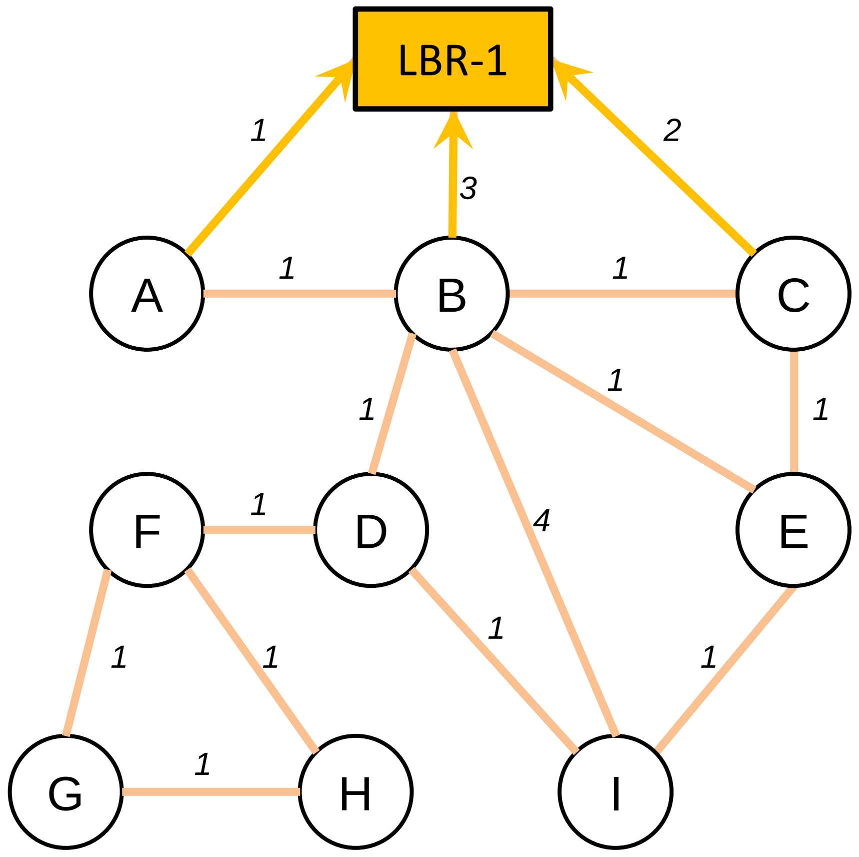
- LLN links are depicted
- LBR is configured by the system administrator (There could be several LBR)
- Links are annotated w/ ETX
- RPL uses new ICMPv6 messages
  - **DIO**: DODAG Information Object
  - **DAO**: Destination Advertisement Object
  - **DIS**: DODAG Information Solicitation

DAG Construction



- Objective Code Point for example
  - Metric: ETX
  - Objective: Minimize ETX
  - Depth computation: Depth ~ ETX
    - Note that a practical computation may be more coarse

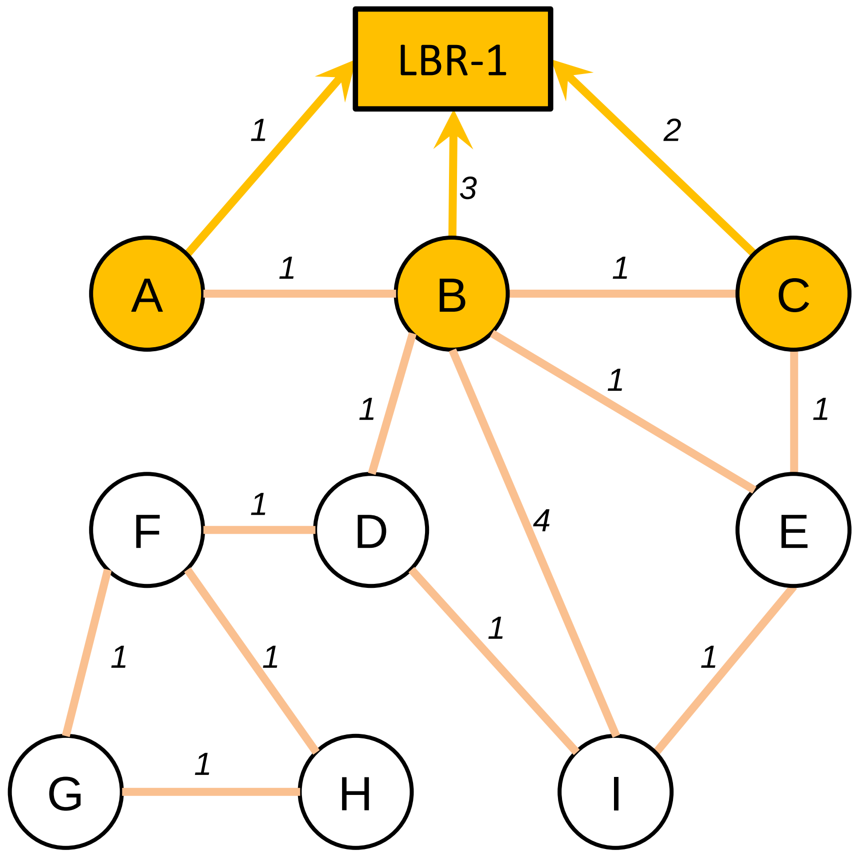
DAG Construction



- LBR-1 multicasts RA-DIO
- Nodes A, B, C receive and process RA-DIO
- Nodes A, B, C consider link metrics to LBR-1 and the optimization objective
- The optimization objective can be satisfied by joining the DAG rooted at LBR-1
- Nodes A, B, C add LBR-1 as a DAG parent and join the DAG

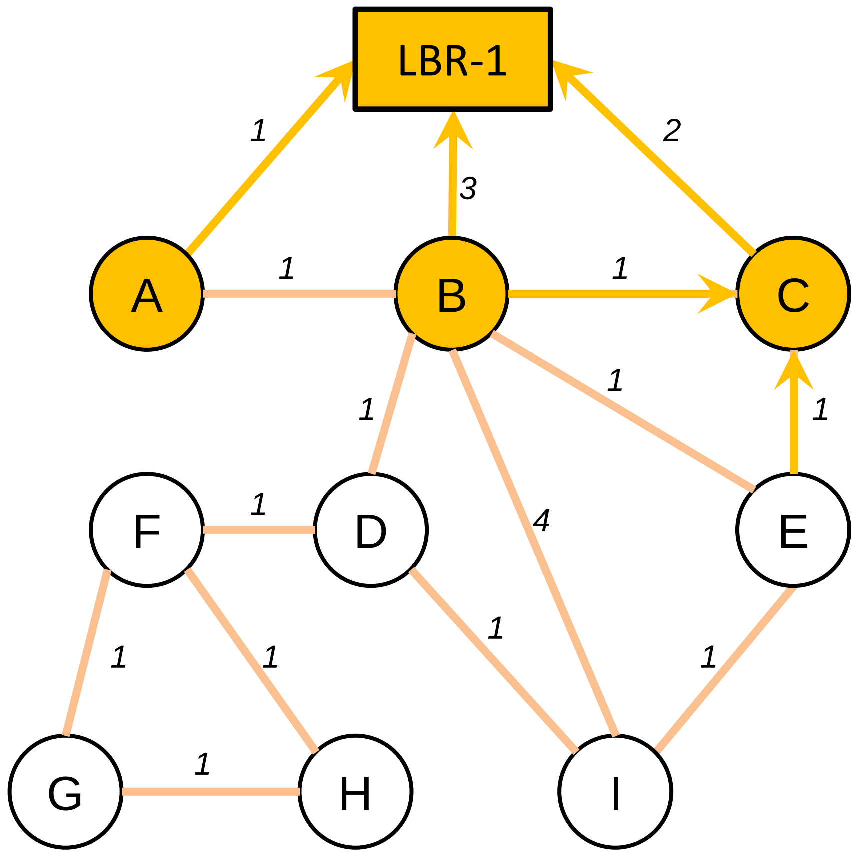
RA: Route Advertisement

DAG Construction



- Node A is at Depth 1 in the DAG, as calculated by the routine indicated by the example OCP (Depth ~ ETX)
- Node B is at Depth 3, Node C is at Depth 2
- Nodes A, B, C have installed default routes (::/0) with LBR-1 as successor

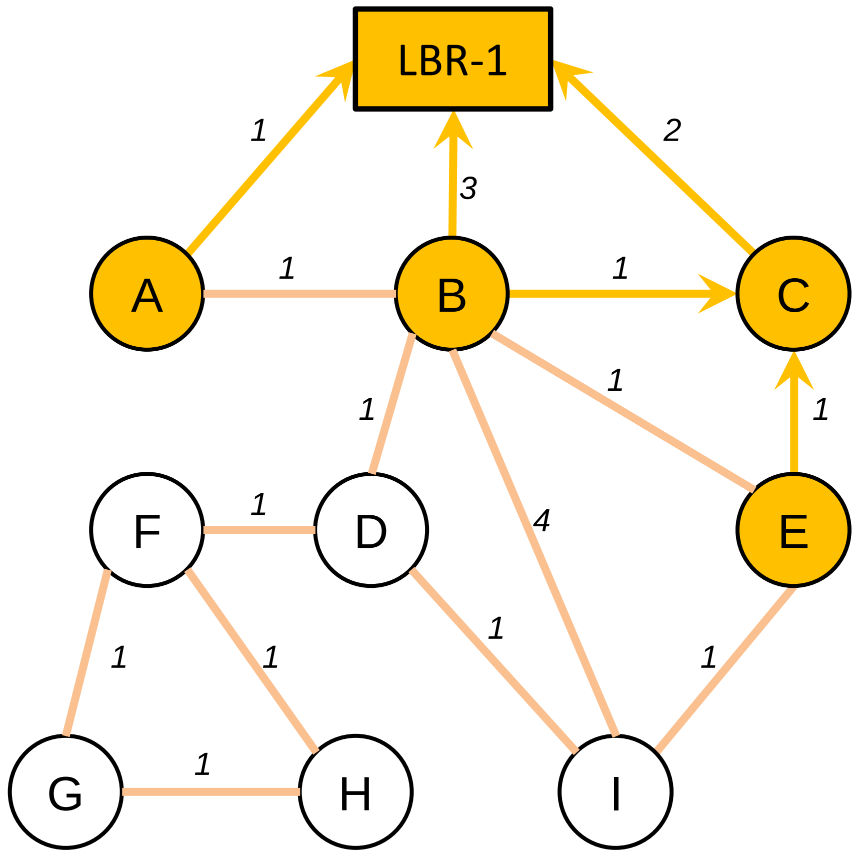
DAG Construction



- The RA timer on Node C expires
- Node C multicasts RA-DIO
- LBR-1 ignores RA-DIO from deeper node
- Node B can add Node C as *alternate* DAG Parent, remaining at Depth 3
- Node E joins the DAG at Depth 3 by adding Node C as DAG Parent

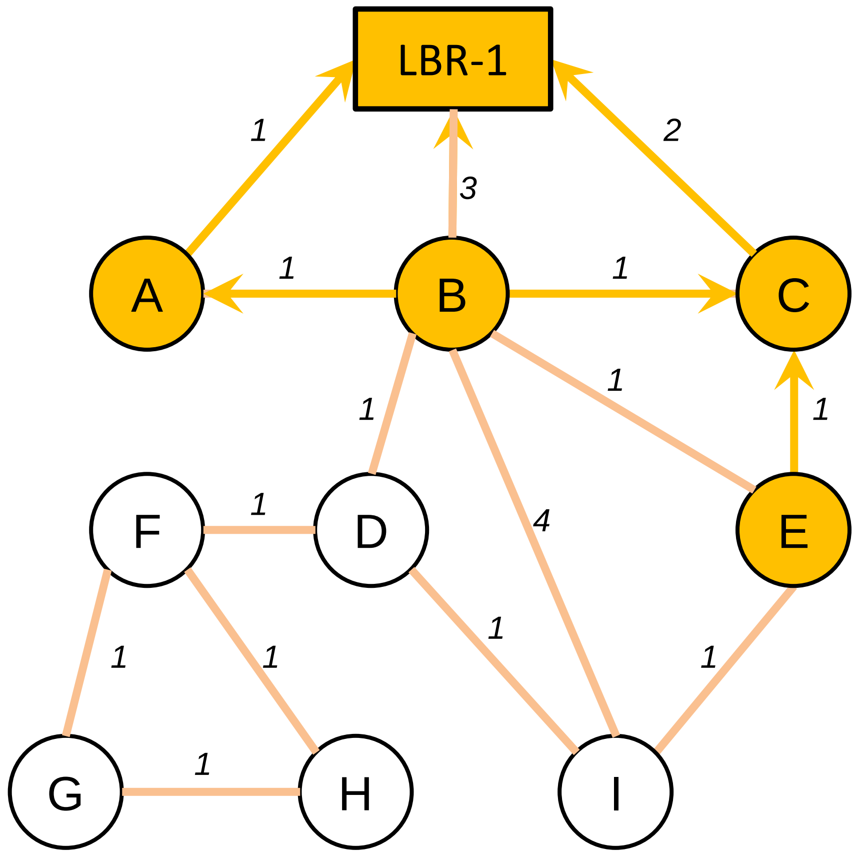
If a node is configured to act as a router, it starts advertising the graph information to its neighbors
A leaf node, it simply joins the graph and does not send any DIO message

DAG Construction



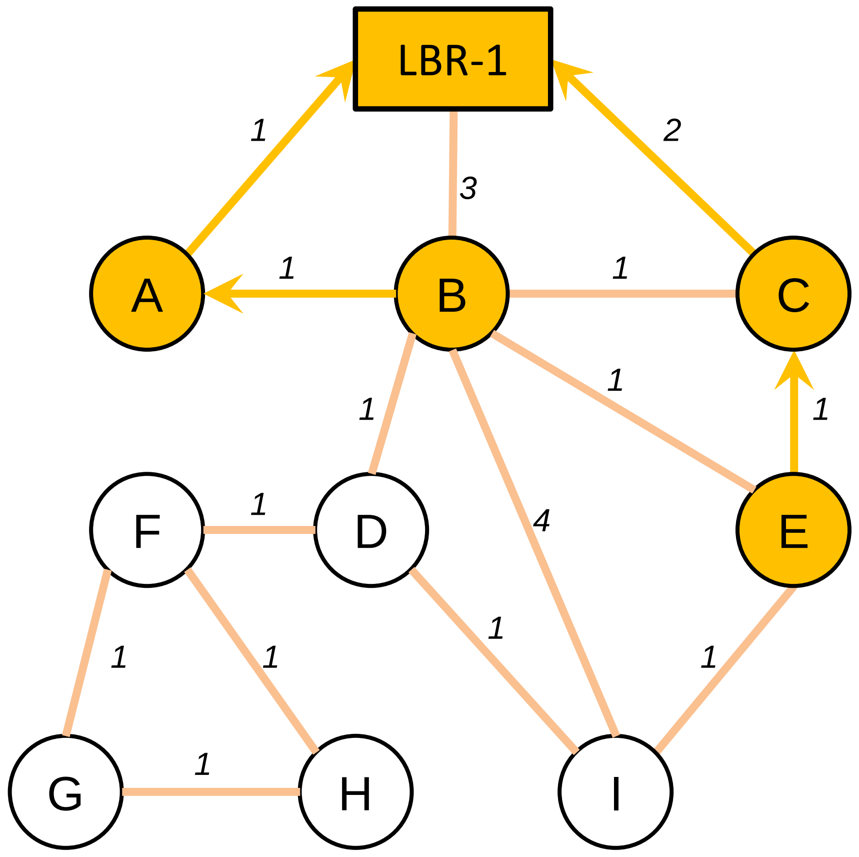
- LBR-1 multicasts DIO
- Node A is at Depth 1, and can reach ::/0 via LBR-1 with ETX 1
- Node B is at Depth 3, with DAG Parents LBR-1, and can reach ::/0 via LBR-1 or C with ETX 3
- Node C is at Depth 2, ::/0 via LBR-1 with ETX 2
- Node E is at Depth 3, ::/0 via C with ETX 3

DAG Construction



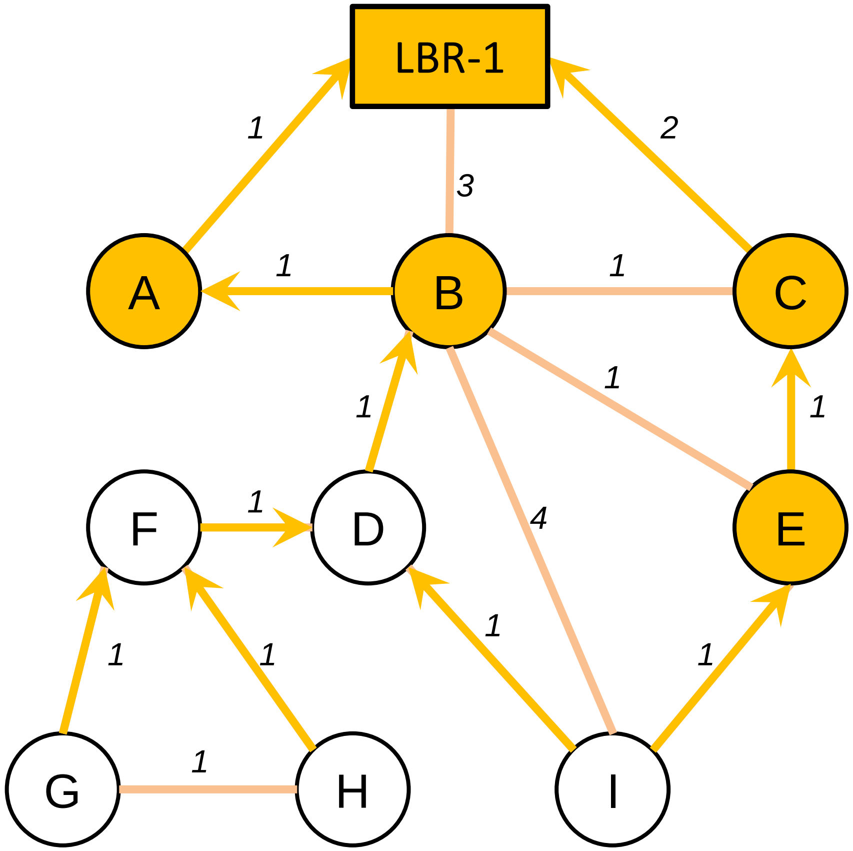
- The RA timer on Node A expires
- Node A multicasts RA-DIO
- LBR-1 ignores RA-DIO from deeper node
- Node B adds Node A
- Node B can improve to a more optimum position in the DAG
- Node B *removes* LBR-1, Node C as DAG Parents
- Node B *removes* (keeps as backup)
  - ::/0 via LBR-1 with ETX 3
  - ::/0 via C with ETX 3
- Adds:
  - ::/0 via A with ETX 2

DAG Construction



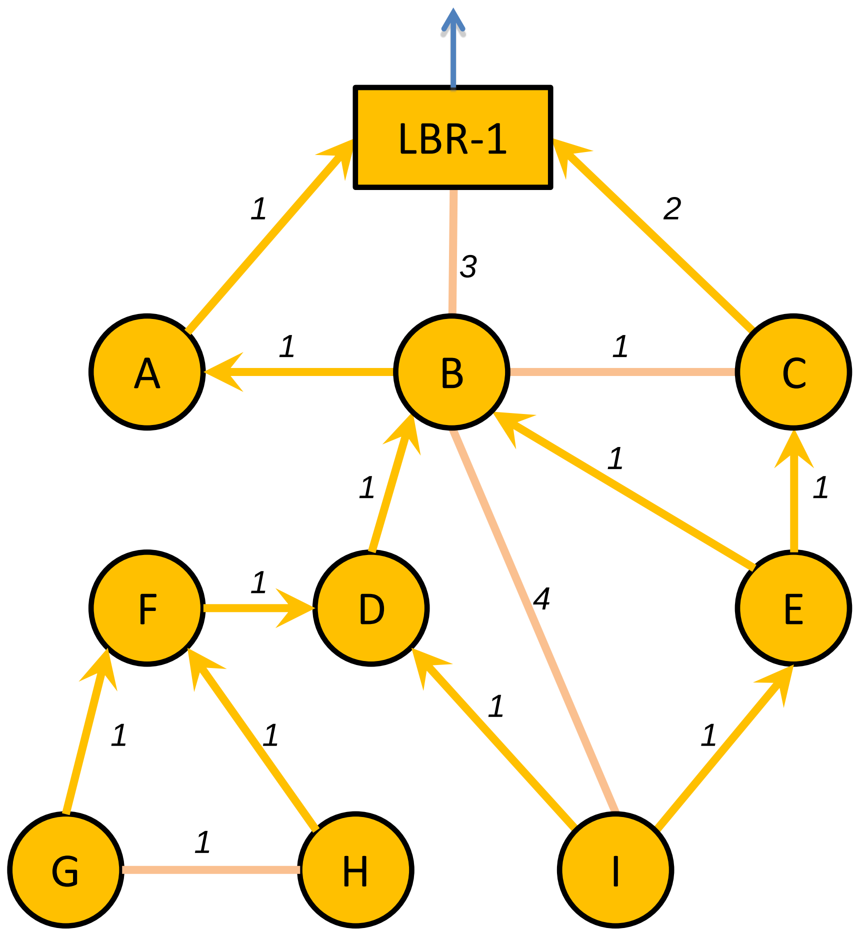
- Node A is at Depth 1, ::/0 via LBR-1 with ETX 1
- Node B is at Depth 2, ::/0 via A with ETX 2
- Node C is at Depth 2, ::/0 via LBR-1 with ETX 2
- Node E is at Depth 3, ::/0 via C with ETX 3

DAG Construction



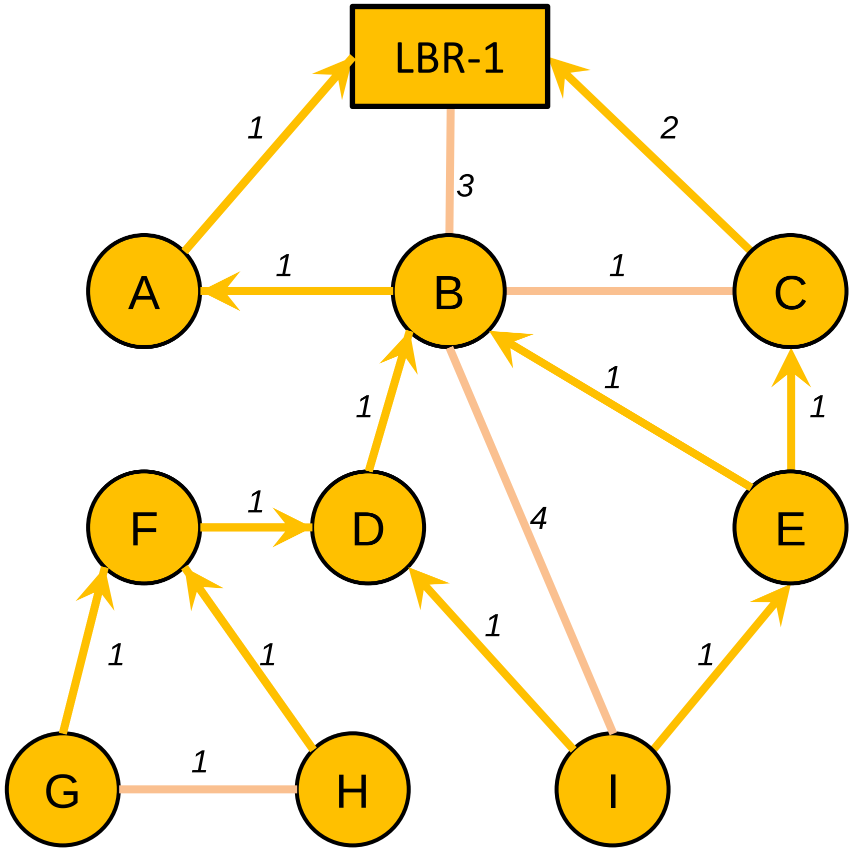
- DAG Construction continues…
- And is continuously maintained
  - *This rippling effect builds the graph edges out from the root to the leaf nodes where the process terminates*
  - *Each node of the graph has a routing entry towards its parent in a hop-by-hop fashion*
  - *The leaf node can send a packet all the way to root of the graph by just forwarding the packet to its immediate parent*

MP2P Traffic – via Upward Route



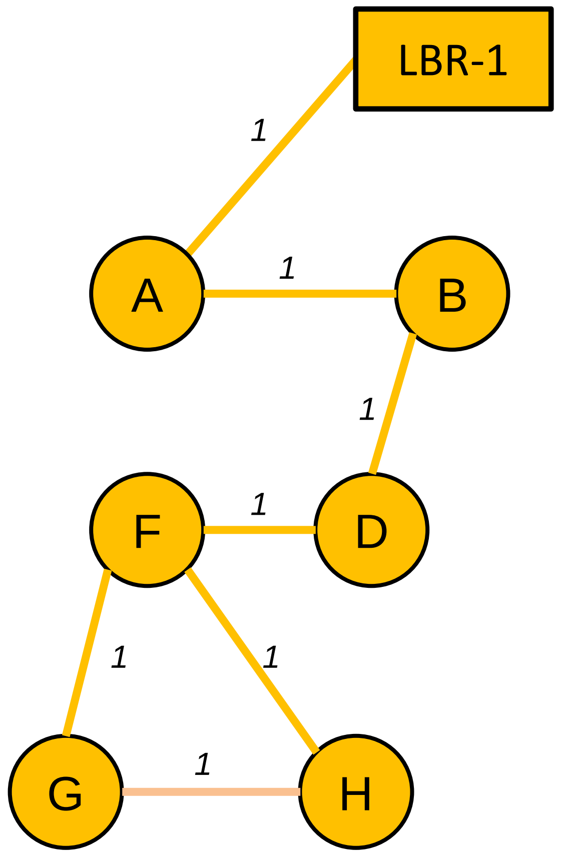
- MP2P traffic flows inwards along DAG, toward DAG Root
- DAG Root may also extend connectivity to other prefixes beyond the DAG root, as specified in the DIO
- Nodes may join multiple DAGs as necessary to satisfy application constraints

Downward Route and Destination Advertisements



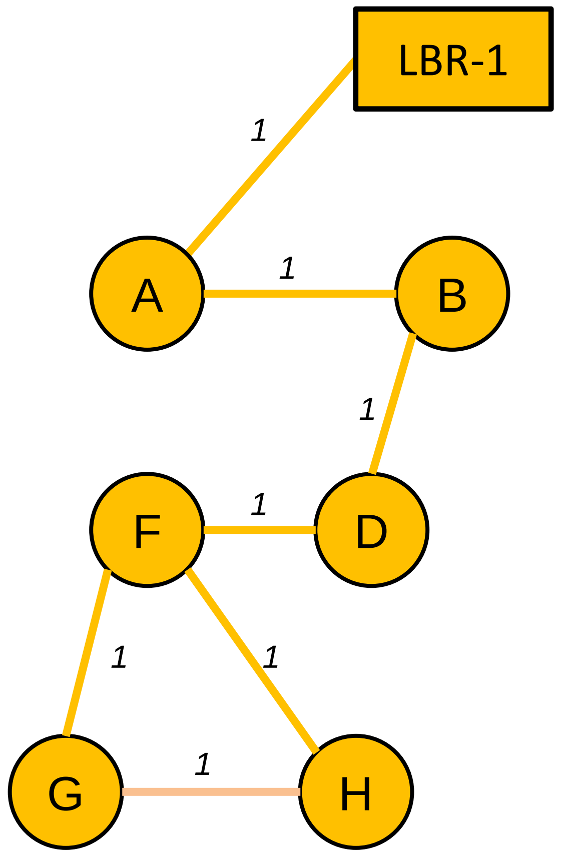
- Destination Advertisements build up routing state in support of P2MP traffic flows outward, from the sink to other nodes
- DA uses the same DAG
- For simplicity, we will focus on a subset of DA in the example
- Nodes inform parents of their presence and reachability to descendants by sending a DAO 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 Advertisements



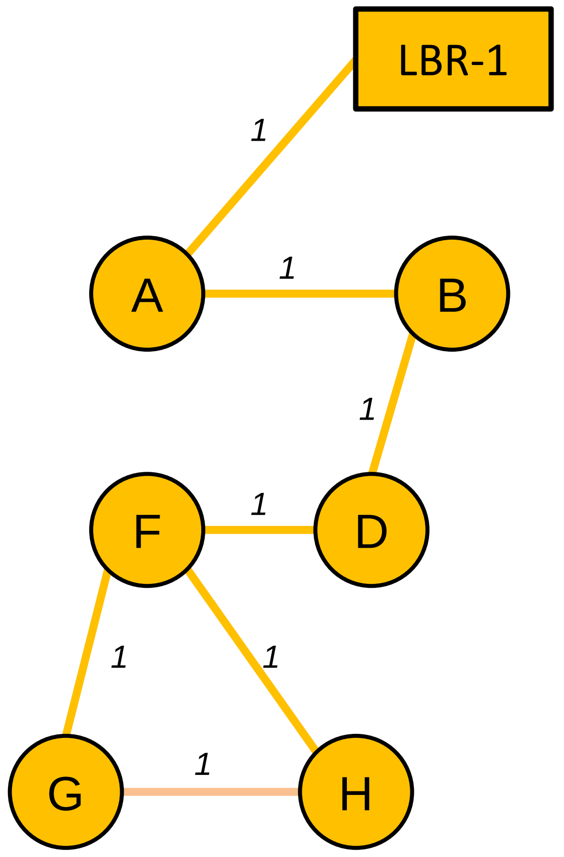
- Some nodes may be able to store routing state for outward flows (LBR-1, A, F)
- Some nodes may not (B, D)
- Some nodes may have a limited ability; DAs may indicate a priority for storage

Destination Advertisements



- ❑ DAs may be triggered by DAG root or node who detects a change
- ❑ DA timers configured such that DAs start at greater depth, and may aggregate as they move up

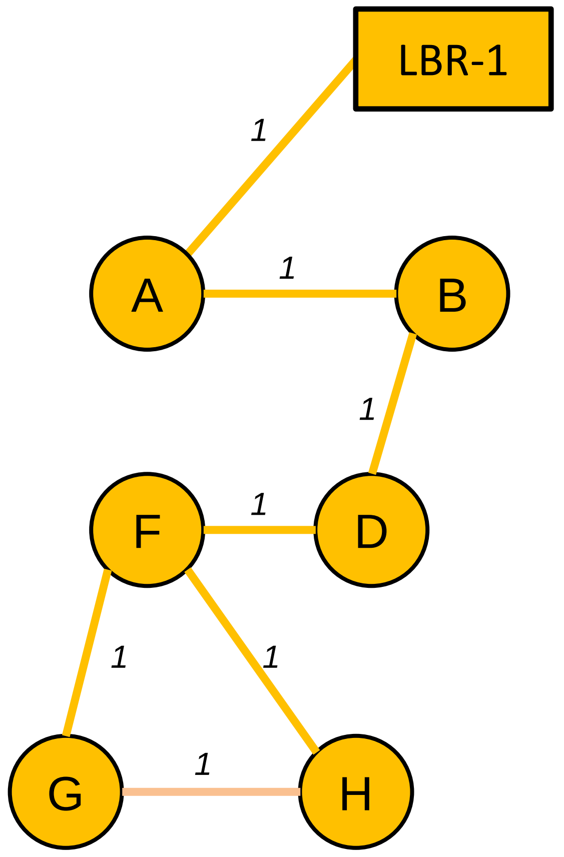
Destination Advertisements



- LBR-1 triggers Destination Advertisement mechanism in DIO
- G emits NA to F with DAO indicating reachability to destination prefix G::
- F stores G:: via G
- H emits NA to F for destination prefix H::
- F stores H:: via H

NA: Neighbor Advertisement

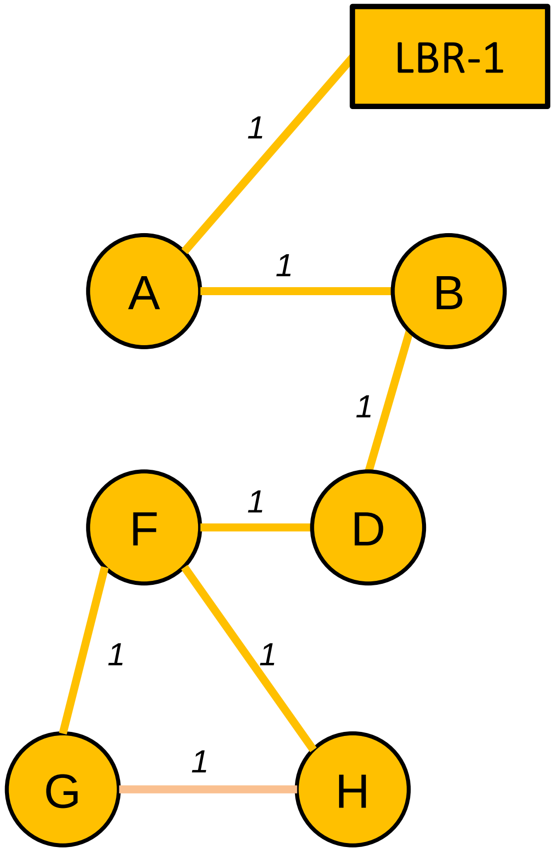
Destination Advertisements



- ❑ Suppose in this example F has a prefix F*:: capable of aggregating {F::, G::, H::}
  - ❖ The method to provision such a prefix is beyond the scope of RPL
- ❑ F emits NA to D with DAO indicating reachability to destination prefix F*::
- ❑ D cannot store…

(continued)

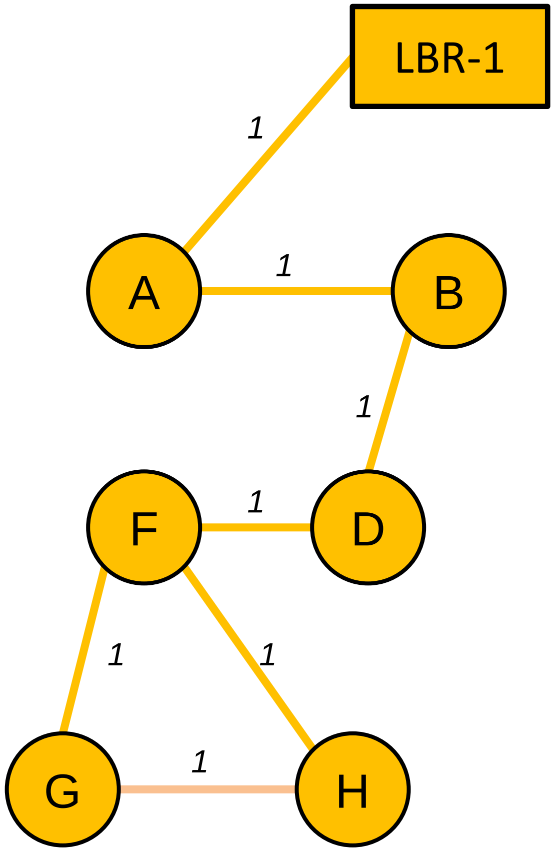
Destination Advertisements



- D adds F to the Reverse Route Stack in the DAO, and passes DAO on to B for F*:: [F]
- D also emits a DAO indicating prefix D:: to B
- B cannot store routing state…

(continued)

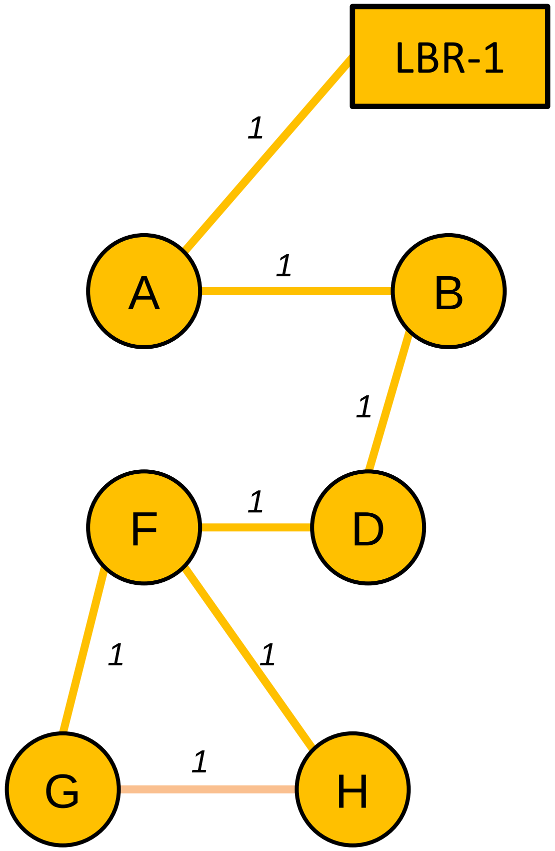
Destination Advertisements



- ❑ B adds D to the Reverse Route Stack in the DAO for D::, and passes DAO D:: [D] on to A
- ❑ A stores D:: via B, with the piecewise source route [D]
- ❑ B also emits a DAO indicating prefix B:: to A
- ❑ A stores B:: via B

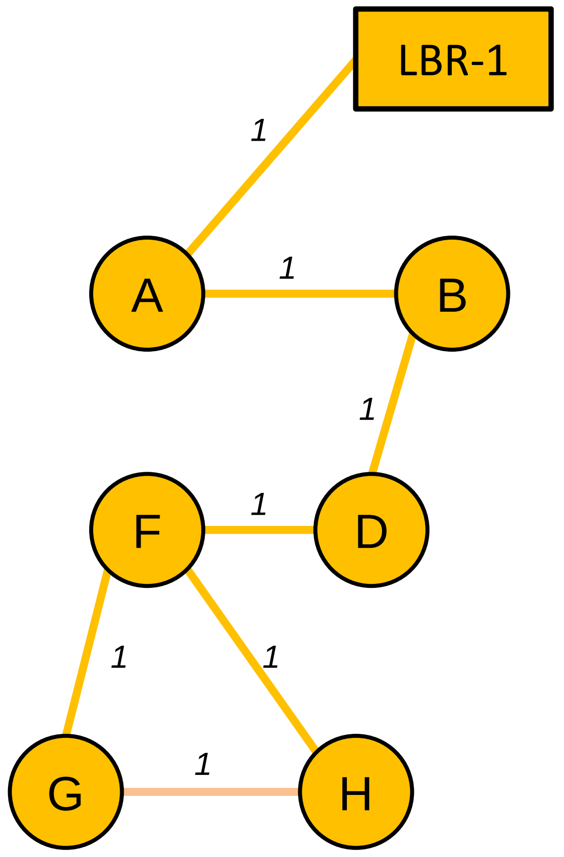
(continued)

Destination Advertisements



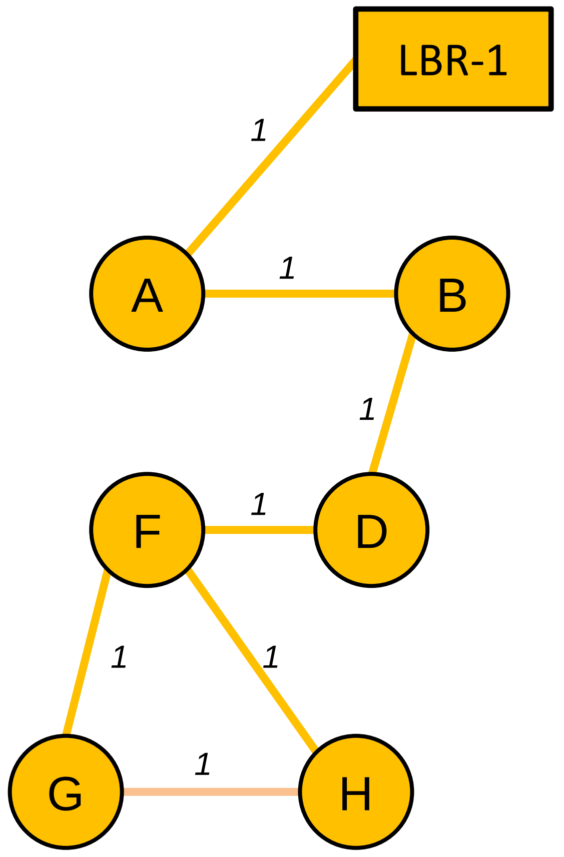
- A emits DAOs to LBR-1 for destination prefixes A::, B::, D::, and F*
- LBR-1 stores A:: via A, B:: via A, D:: via A, and F*:: via A
- A stored B:: via B, D:: via B [D], F* via B [D, F]
- B, D stored nothing
- F stored G:: via G, H:: via H

P2MP Traffic



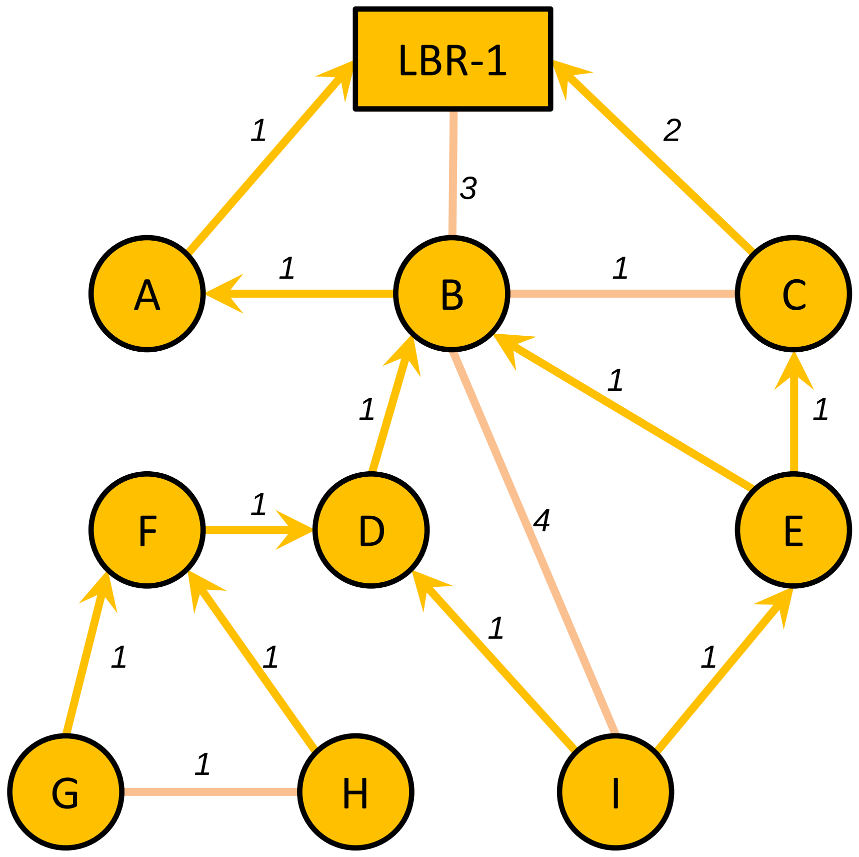
- The routing state setup by Destination Advertisement is used to direct P2MP traffic outward
- LBR-1 directs traffic for G (F*::) to A
- A adds source routing directive, [D, F], and forwards to B
- B uses source routing directive to forward to D...

P2MP Traffic



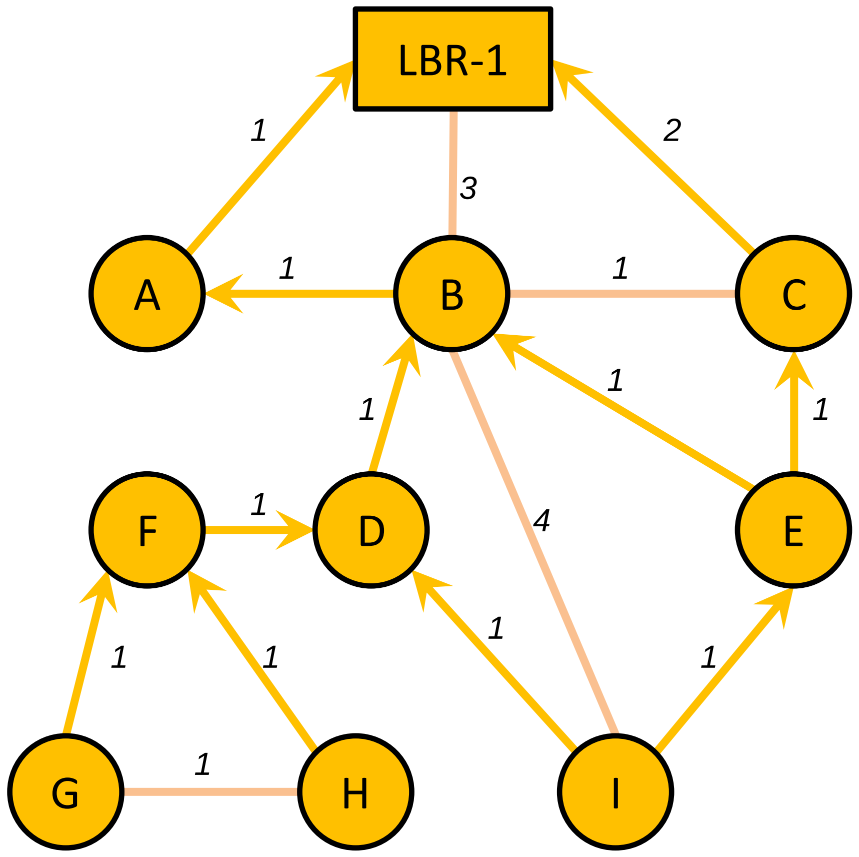
- ❑ D uses source routing directive to forward to F
- ❑ F uses routing state to forward to G
- ❑ Note the use of source routing to traverse the stateless region of the LLN
  - ❖ The details of the source routing mechanism are still to be defined

P2P Traffic



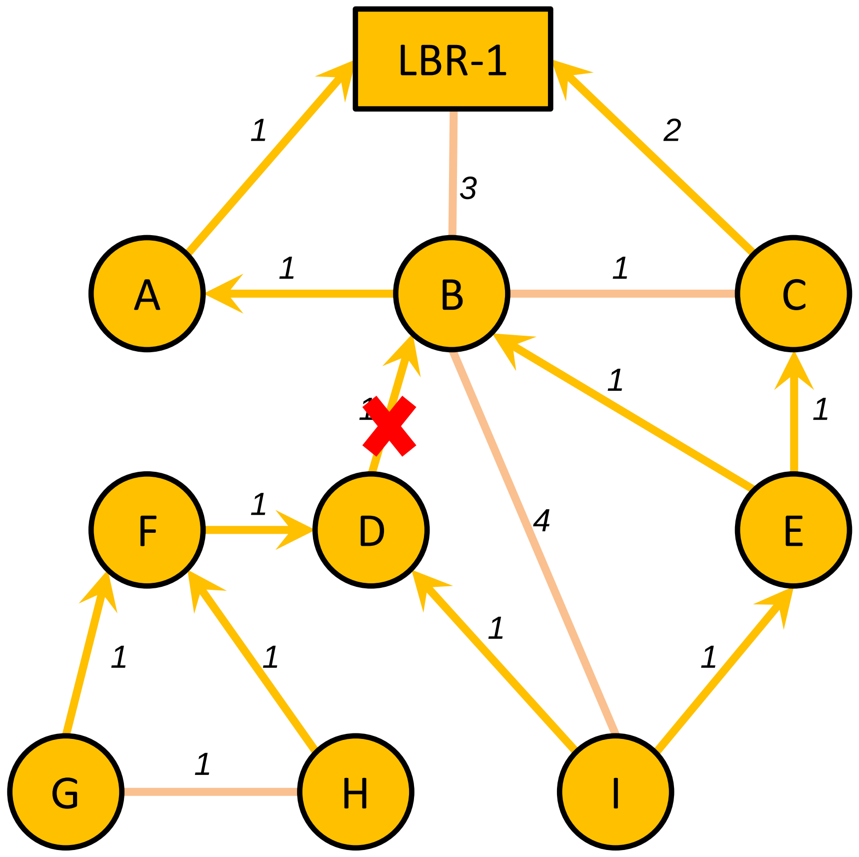
- ❑ By default, P2P traffic follows the default route until a node is encountered who has a more explicit route
- ❑ P2P traffic flows inwards along the DAG until a common node is reached to direct the traffic outwards

P2P Traffic



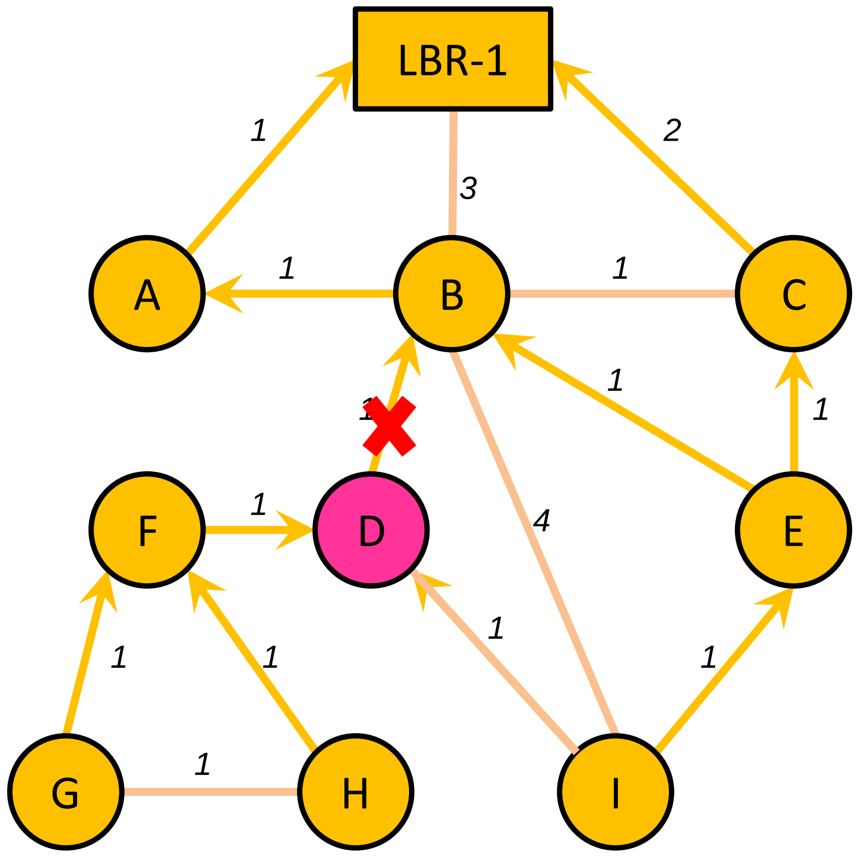
- ❑ The default scheme allows P2P traffic with minimal state required of nodes – but is not appropriate in all cases
- ❑ More optimal P2P routes may be computed and installed if the application requires
  - ❖ RPL does not specify nor preclude such a mechanism at this time

DAG Maintenance: Floating DAG



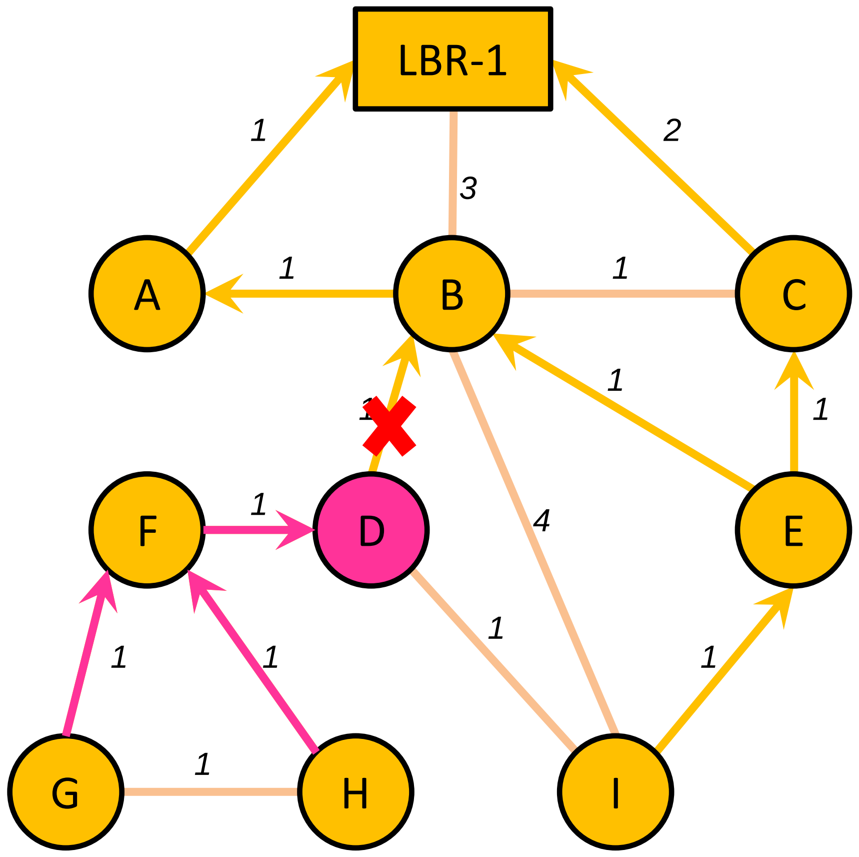
- Consider the case where the link B—D goes bad
- Node D will remove B from its DAG parent set
- Node D no longer has any DAG parents in the grounded DAG, so it will become the root of its own floating DAG

DAG Maintenance: Floating DAG



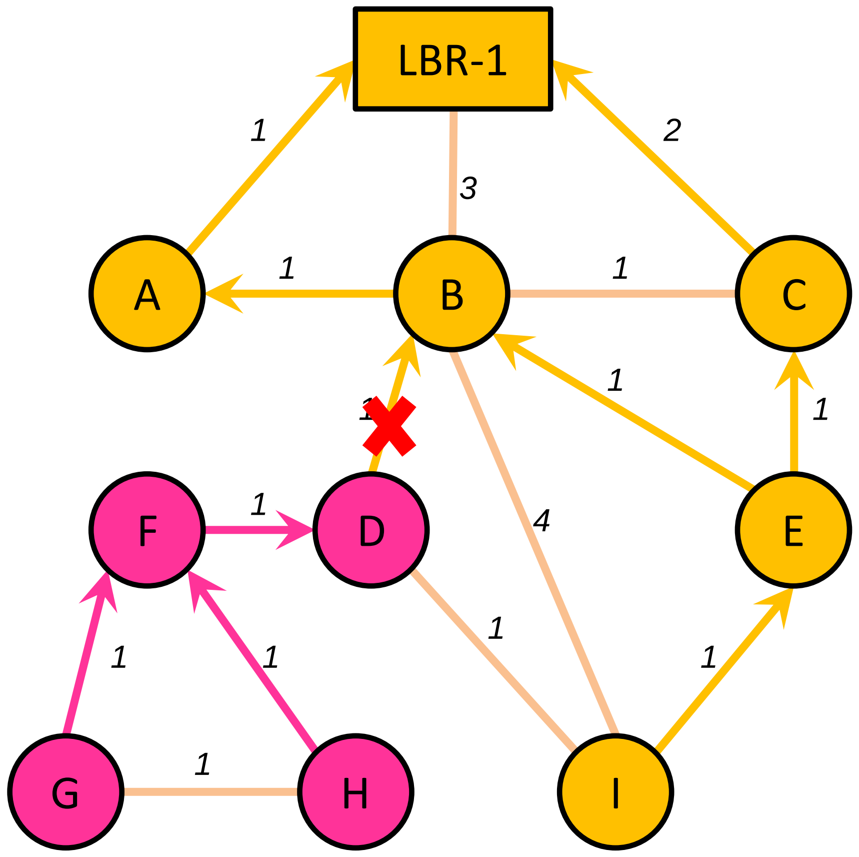
- ❑ Node D multicasts an RA-DIO to inform its sub-DAG of the change
- ❑ Node I has an alternate DAG Parent, E, and does not have to leave the DAG rooted at LBR-1.
- ❑ Node I removes Node D as a DAG Parent

DAG Maintenance: Local Repair



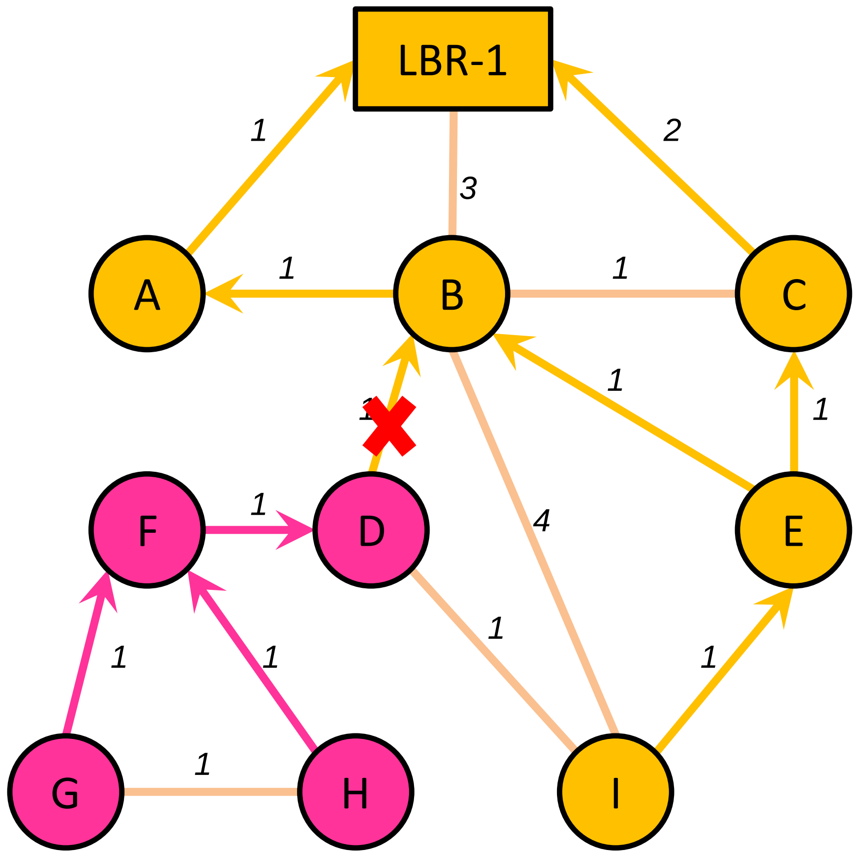
- Node F does not have an option to stay in the DAG rooted at LBR-1 (no alternate DAG Parents), so Node F follows Node D into the floating DAG
- Node F multicasts an RA-DIO
- Nodes G and H follow Node F into the floating DAG

DAG Maintenance: Local Repair



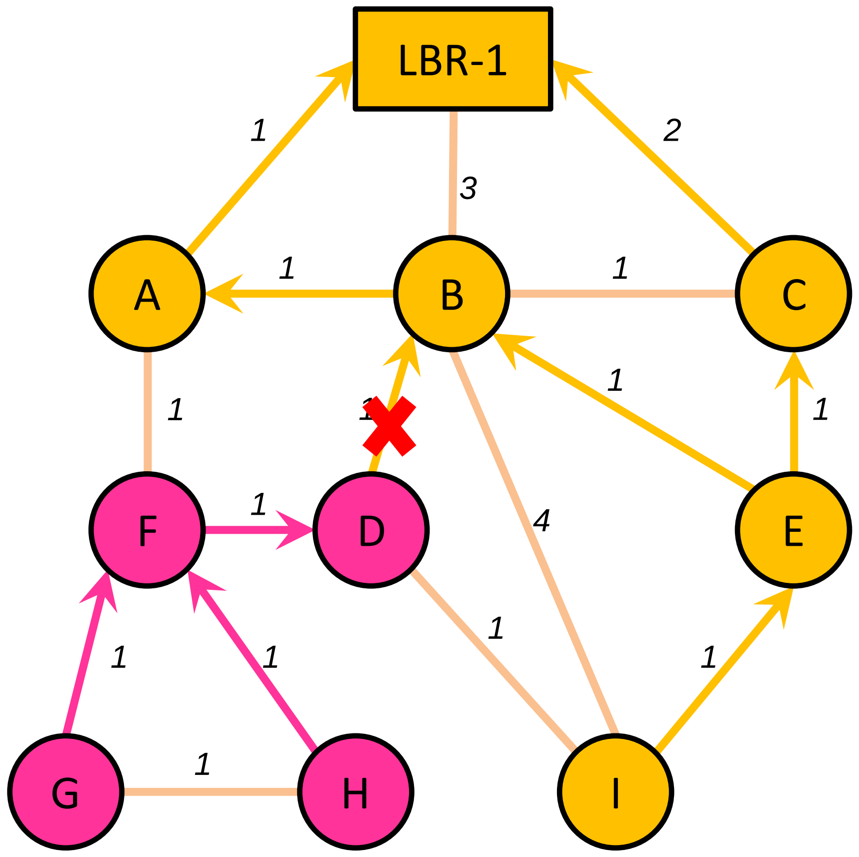
- ❑ The sub-DAG of node D has now been frozen
- ❑ Nodes contained in the sub-DAG have been identified, and by following node D into the floating DAG, all old routes to LBR-1 have been purged
- ❑ The floating DAG seeks to rejoin a grounded DAG…

DAG Maintenance



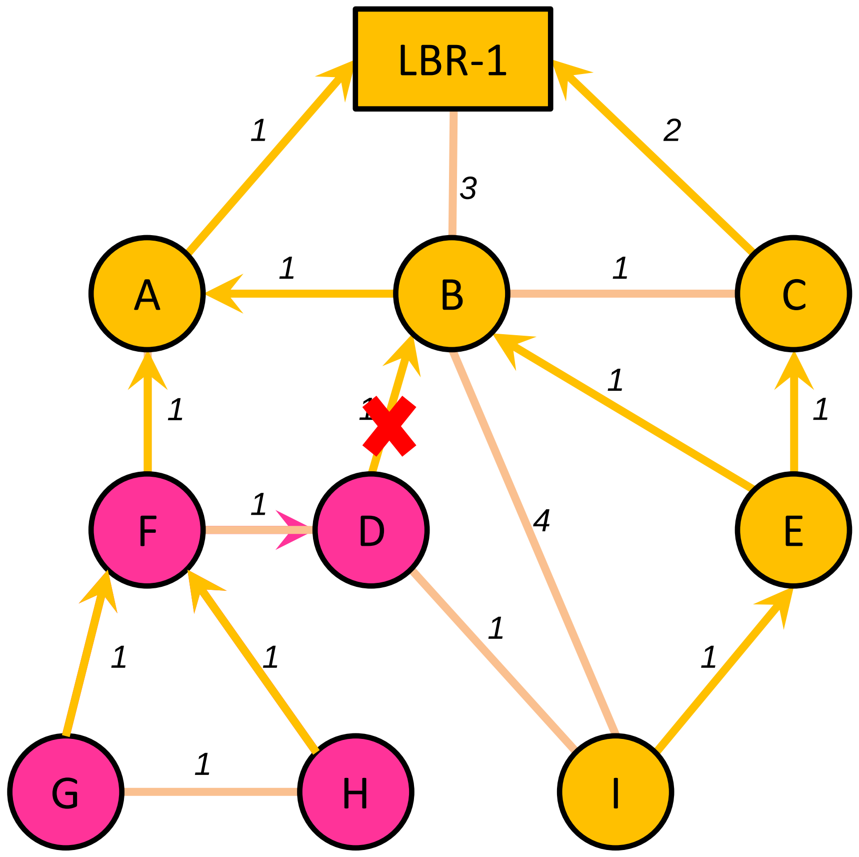
- ❑ Node I multicasts an RA-DIO
- ❑ Node D sees a chance to rejoin grounded DAG at depth 5 through Node I
- ❑ Node D starts a DAG Hop timer of duration α 4 associated with Node I

DAG Maintenance



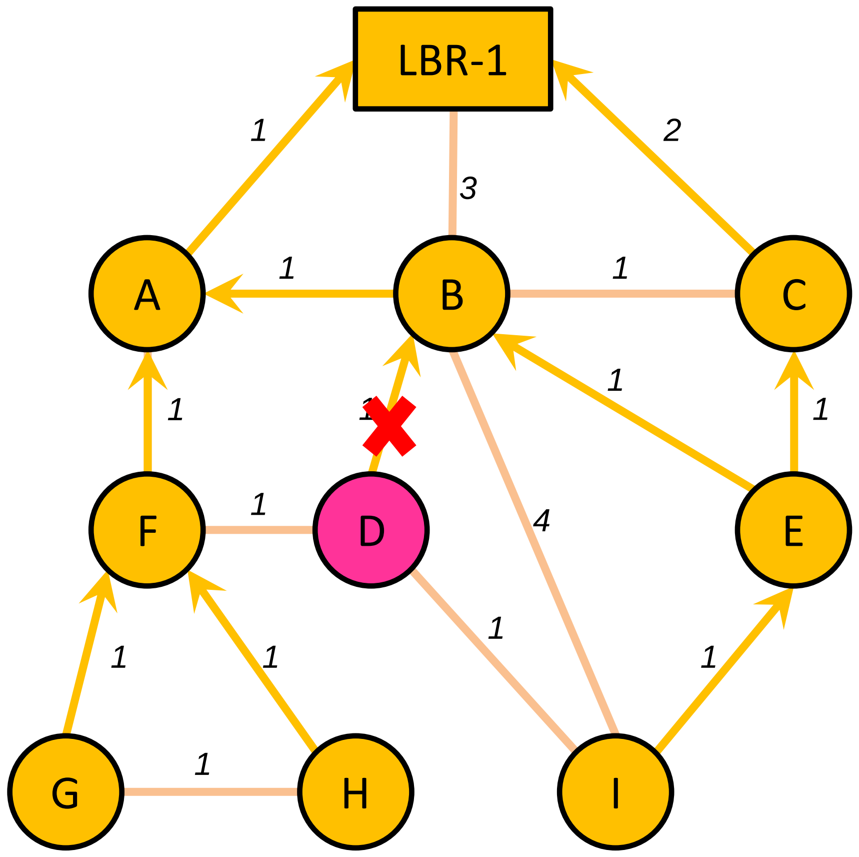
- ❑ Suppose a link A—F becomes viable
- ❑ Node A multicasts an RA-DIO
- ❑ Node F sees a chance to rejoin grounded DAG at depth 2 through Node A
- ❑ Node F starts a DAG Hop timer of duration α 1 associated with Node A

DAG Maintenance: Loop Avoidance



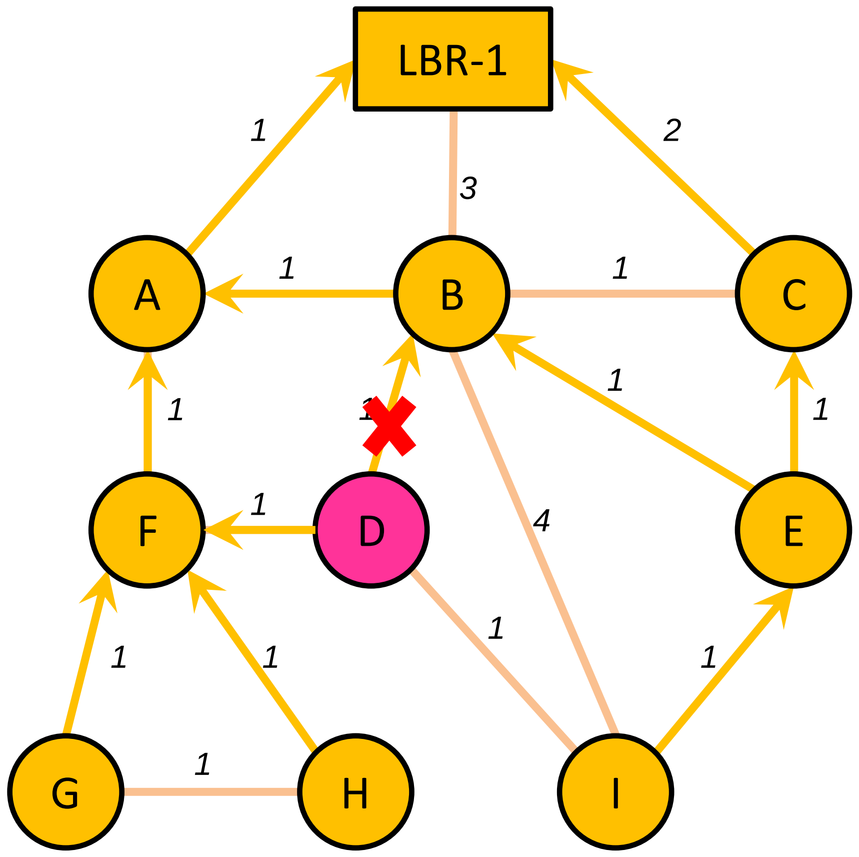
- ❑ Node F's DAG Hop Timer expires
- ❑ Node F joins to the grounded DAG at depth 2 by adding A as a DAG parent, and removing D
- ❑ Node F multicasts an RA-DIO
- ❑ Nodes G and H follow Node F to the grounded DAG

DAG Maintenance: Loop Avoidance



- Node D sees a chance to rejoin DAG LBR-1 at depth 3 through Node F
- Node D starts a DAG Hop timer of duration α 2 associated with Node F, in addition the DAG Hop timer already running with duration α 4 associated with Node I

DAG Maintenance: Loop Avoidance



- ❑ Node D's DAG Hop timer of duration α 2 tends to expire first
- ❑ Node D joins the grounded DAG at depth 3 by adding Node F as a DAG Parent
- ❑ The breaking-off and re-joining of the broken sub-DAG is thus coordinated with loop avoidance

Loop Avoidance

- Two mechanisms to avoid count-to-infinity

- Floating DAG
  - Leave DAG, color sub-DAG, then look for new routes
  - Operation local to nodes that must increase their depth
  - Does not guarantee loop freedom

- Sequence number change
  - Loop freedom, but expensive network-wide operation
  - Used infrequently if possible

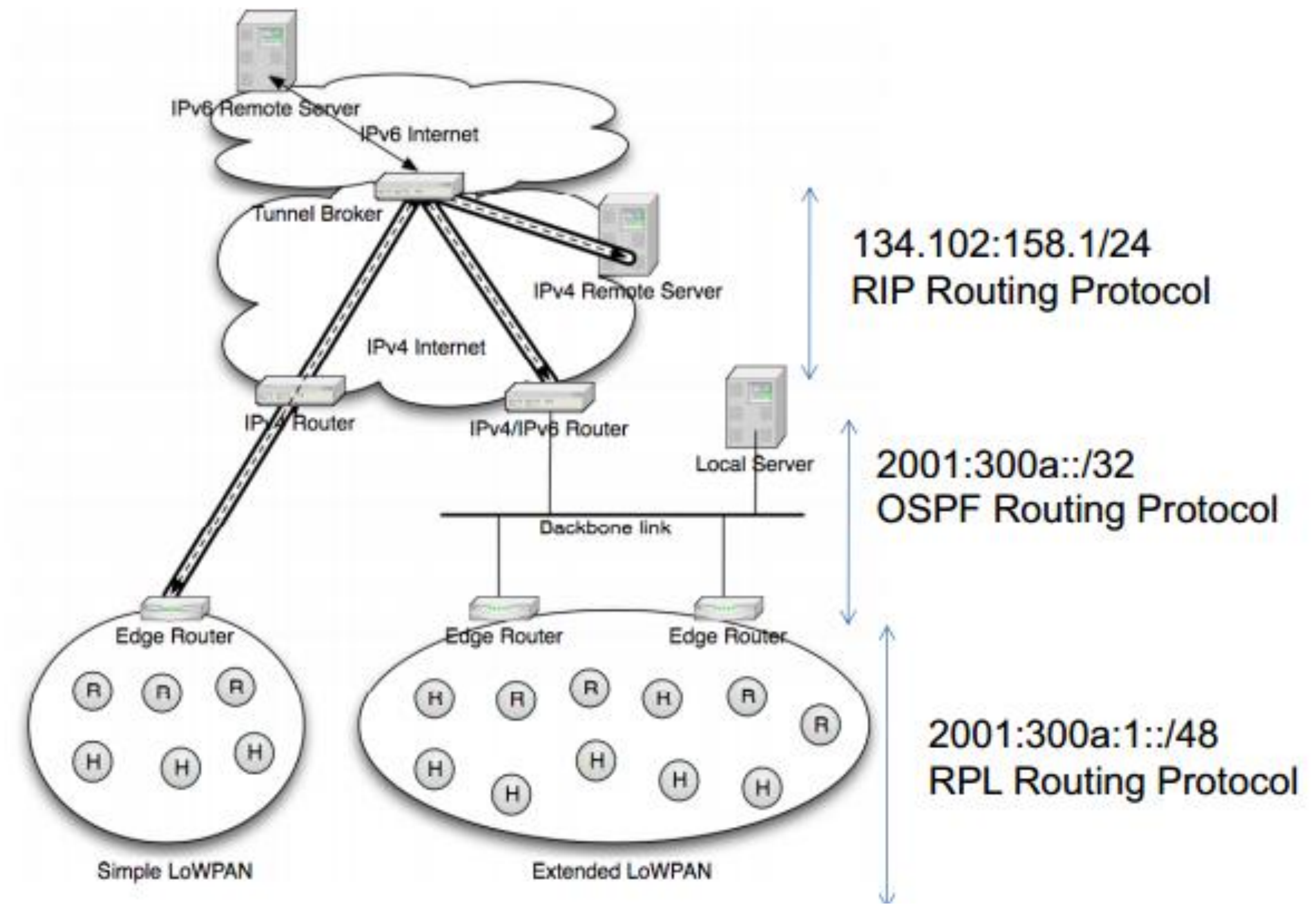
Joining RPL network

- ❑ When a node appears (it is switched on, or it moves the given area), it starts listening to DIO messages
  - ❖ If it receives such messages, it decides which DODAG to join
  - ❖ If it receives nothing, or if it wants to speed up the process, it sends a DIS message
    - DODAG Information Solicitation
    - Triggers the sending of DIO messages from its neighbors
  - ❖ If still no DIO is received, it has the choice to become the root of a floating DODAG
    - Starts sending its own multicast DIO messages

RPL: summary

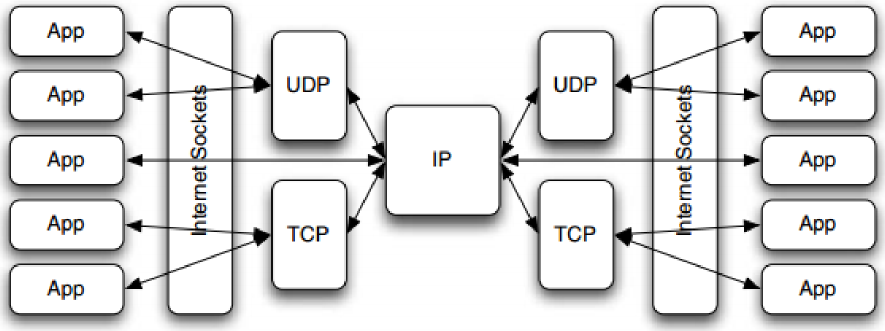
- ❑ Working with 6LoWPAN
- ❑ Support IP-based Application
- ❑ Support P2MP, MP2P traffic
- ❑ Routing state is minimized: stateless nodes have to store only instance(s) configuration parameters and a list of parent nodes
- ❑ Consider both link and node properties when choosing paths
- ❑ Link failures does not trigger global network re-optimization

IPv4 Interoperability



Protocol Stack for WSNs

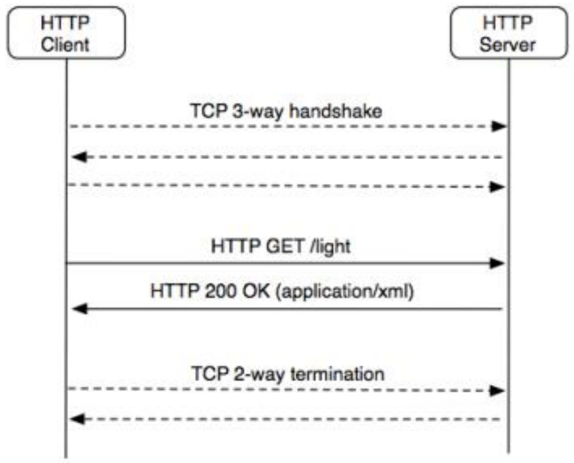
- What does missing in the protocol stack?
  - PHY/MAC 802.15.4
  - Adaptation: 6LoWPAN
  - Routing: RPL
  - Transport: UDP
  - Application Protocol?

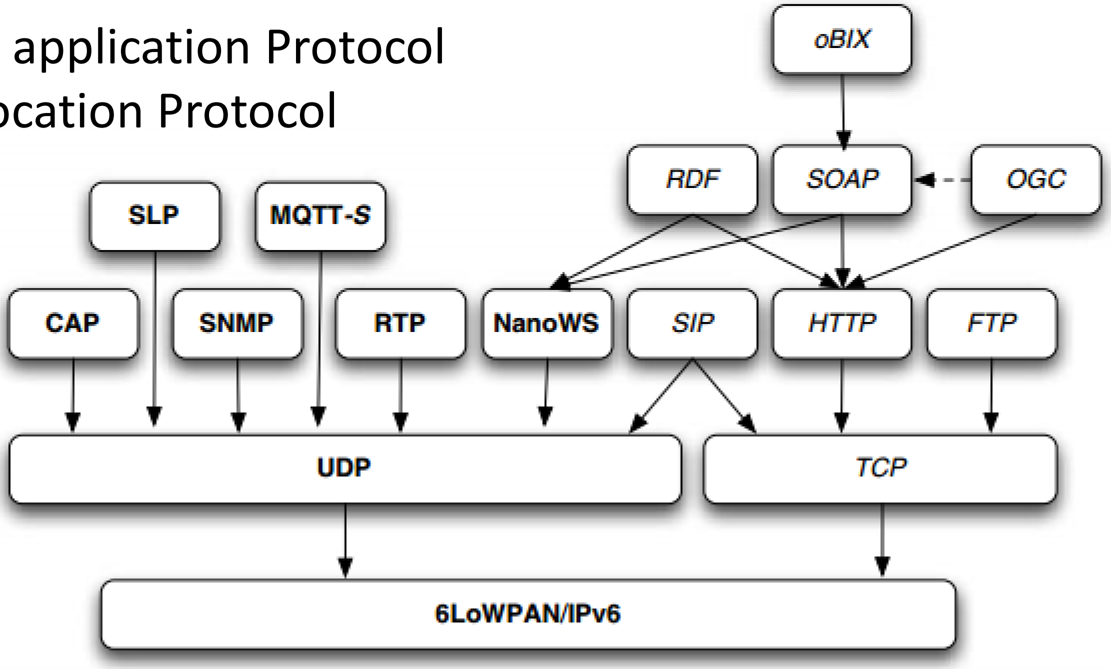


- IP provides packet networking over heterogeneous links
  - UDP: Best-effort (packets may be silently dropped, duplicated or delayed and may arrive out of order) - the fastest and most simple way of transmitting data
  - TCP: Reliable connection oriented
  - IP uses Socket Communication: end-points are defined by 16 bit source and destination ports and source and destination IP addresses

IP based Application Protocols

- Widely used internet protocols (TCP, HTTP, FTP; SIP and SOAP)
  - Considerable work has to be done for adapting to 6LoWPAN
- Existing protocols for WSNs (mainly based on UDP)
  - MQTT: Message Queue Telemetry Transport is developed by IBM
  - CAP: ZigBee Compact application Protocol
  - SLP: simple Service Location Protocol
  - Etc..





IP based Application Protocols

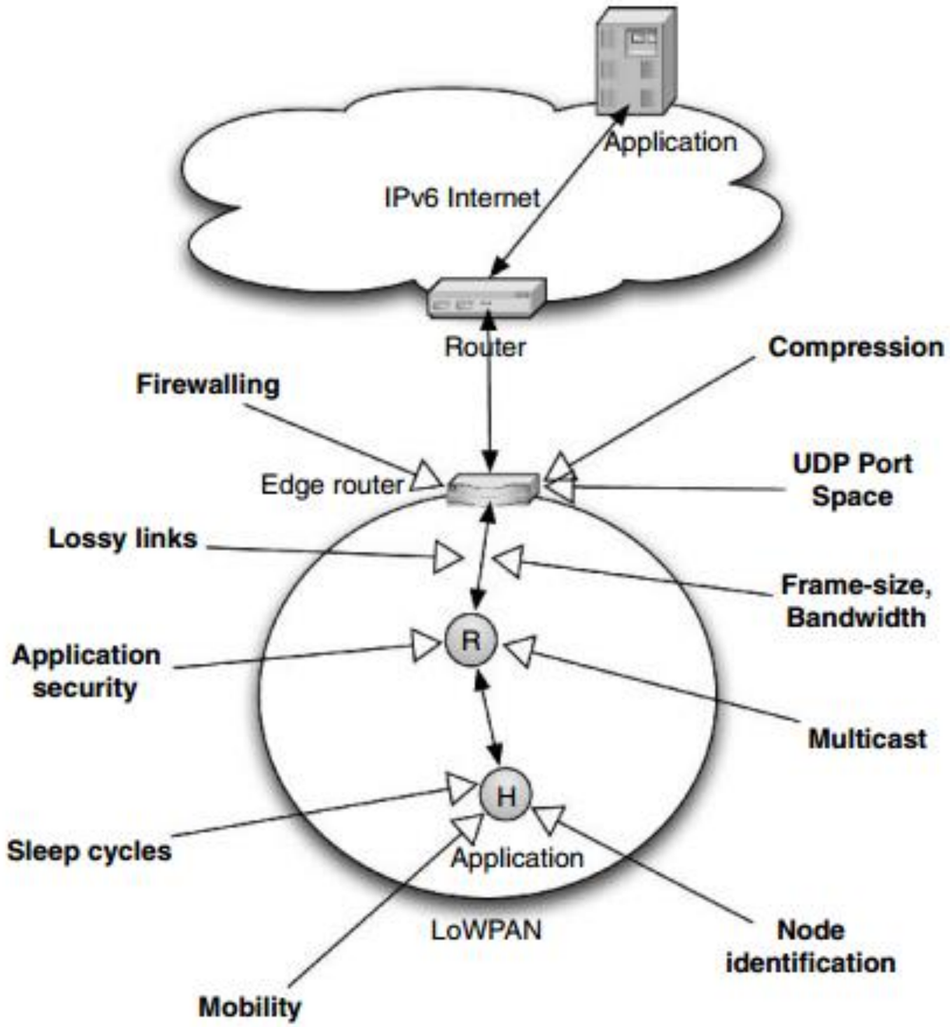
- IP based application protocols over 6LoWPAN
  - 6LoWPAN supports the compression of UDP ports down to a range of 16
    - should be enough for limited number of applications that can be run
  - TCP header is not easy to compress
  - TCP is not efficient over lossy wireless multi-hop networks
- Why UDP for 6LoWPAN?
  - Simple, compressible and suits most application protocols needs
  - Multicast support
  - ***Reliability***?

Application Protocols for WSNs

- MQTT (Message Queue Telemetry Transport) by IBM
  - Lightweight publish/subscribe protocol over TCP/IP
    - Fixed-length header (2 bytes)
  - One-to-many message distribution
  - 3 QoS for message delivery (At most once, at least once, exactly once)
  - Applications, e.g. Facebook Messenger
- XML-based protocols (SOAP)
  - M2MXML
    - Messages are strict ASCII
    - Devices are identified by 128-bit UUID
    - BiTXml
- RESTful protocols
  - HTTP
  - CoAP

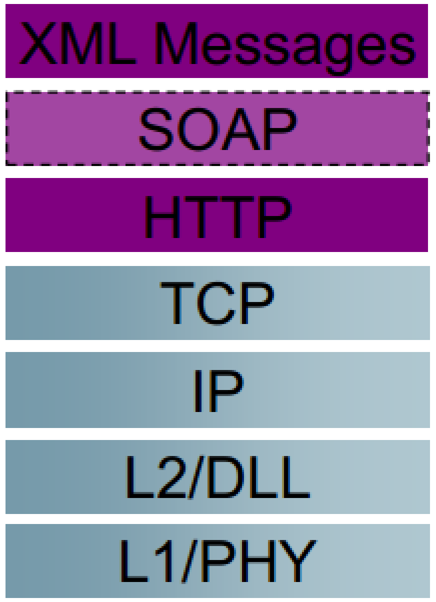
Design Issues

- Link Layer
  - Payload size, limited bandwidth, lossy
  - Multicast support
- Networking
  - Limited compressed UDP port
- Host issues
  - Sleeping cycles
  - Identification (64 bit, 16 bit addresses)
- Compression
  - Header and payload compression
  - End-to-end or by an intermediate node
- Security
  - Application layer security
  - Firewalling at the edge router



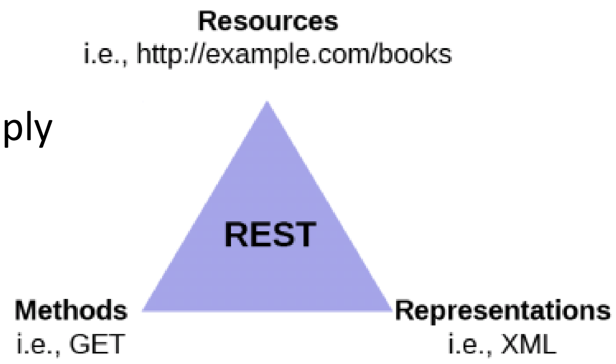
Web Service

- De-facto for inter-server communications
- All Internet servers speak HTTP/XML
- SOAP or REST architecture
- Advantages
  - Formal message sequences
  - Internet-wide support
  - Most Internet applications use web services, which depend on the basic Representational State Transfer (REST) architecture
- Disadvantages for embedded services
  - Inefficient, complex



REST Architecture

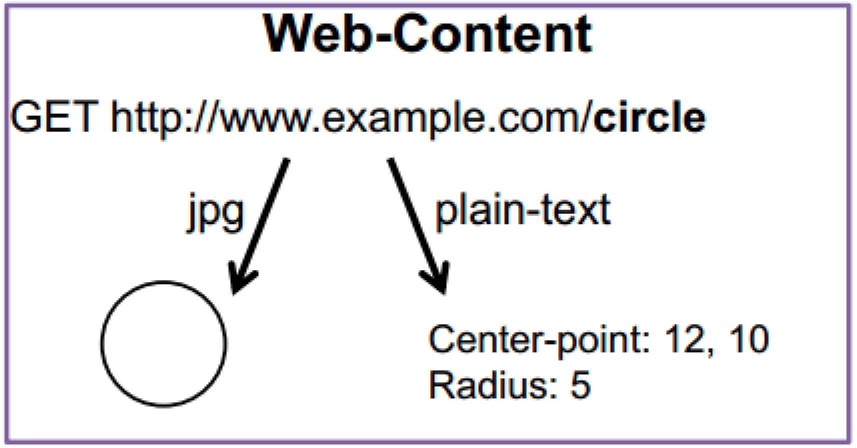
- Representational State Transfer (REST)
  - Describes client-server and cacheable communications protocol, such as HTTP, to make simple calls between network devices
- REST-style architectures consist of clients and servers
  - Clients initiate requests to servers
  - Servers process requests and return appropriate responses
  - Requests and responses are built around the transfer of representations of resources
- Resources: mostly identified by its URI (Uniform Resources Identifier)
  - Resources may be web-address, a physical resource or simply a document located somewhere on a server
- Representations: to manipulate resources
  - Representation defines the format of the resource
  - Typical representations are HTML, XML or JSON,
  - but may be also binary, plain-text, MIME types, etc
- Methods: allows clients to retrieve, modify or delete resources by use of their representations
  - The basic operations/methods on a resource are GET, PUT, POST and DELETE

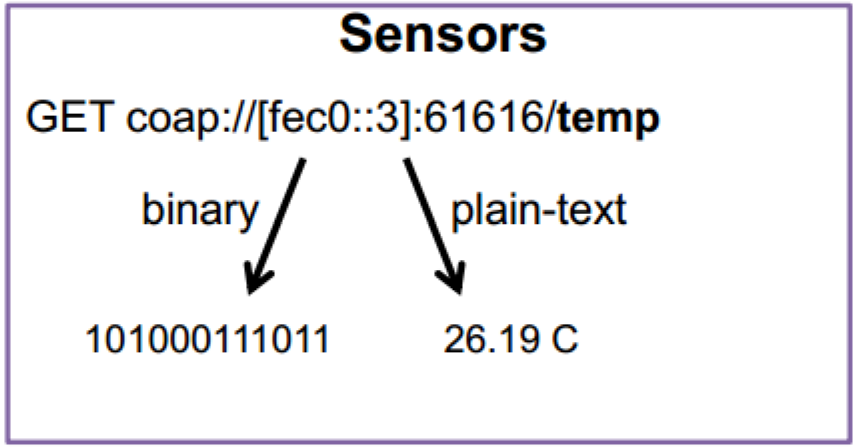


Resource

- Protocols (e.g. HTTP, CoAP)
  - Sources of specific information (e.g. sensors, web-content)
  - Identified by an Uniform Resource Identifier (URI)
  - Manipulation through Methods (e.g. GET, PUT)
  - Represented as jpg, plain-text, etc.

Resource	GET	PUT	Comments
/st	X		Temperature
/sh	X		Humidity
/sv	X		Voltage
/r	X		Temperature, humidity and voltage together
/l	X	X	LEDs
/ck	(X)	X	AES Encryption Key





Web Services for WSNs

- Today's web services are no good for embedded systems
  - FTP, HTTP, XML, SOAP
- Web services for embedded devices
  - RESTful architecture for good web integration
  - UDP based transport with multicast support
  - Overhead suitable for constrained networks
  - Complexity suitable for constrained nodes
  - Built-in web discovery and security
- The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture for constrained nodes and networks
  - e.g. 8-bit microcontrollers with limited RAM and ROM
  - networks (e.g. 6LoWPAN)

CoAP: Constrained Application Protocol

- A specialized web transfer protocol for use with constrained networks and nodes
  - M2M applications (smart energy, building automation, etc…)
  - 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)
  - To manage devices
- Embedded web transfer protocol (***coap://***)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple header < 10 bytes
- Subset of MIME types and HTTP-compatible response codes
- Optional observation, block transfer and discovery

What is CoAP (and is not)?

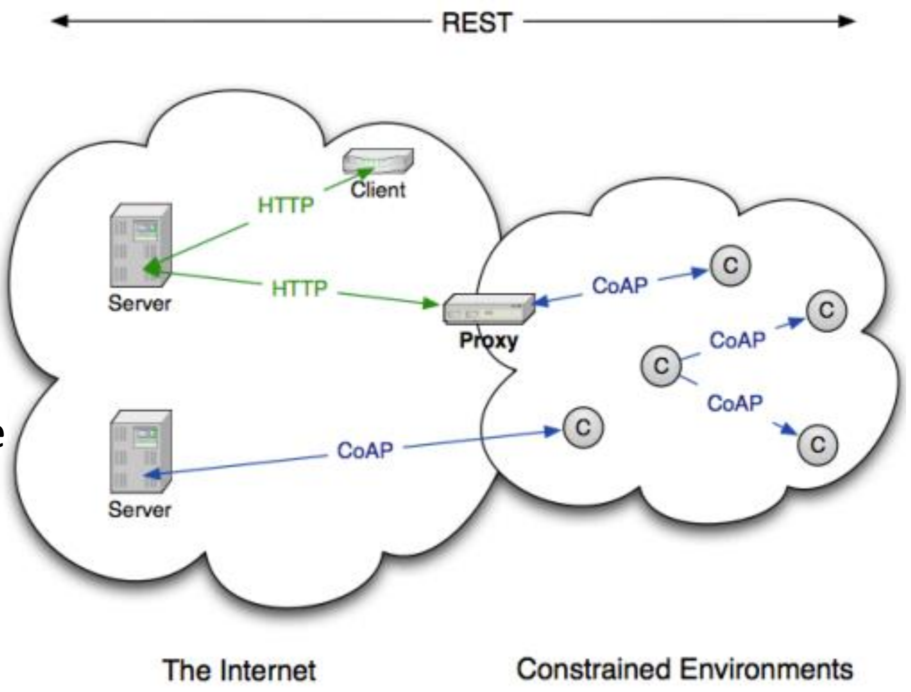
- CoAP is
  - A RESTful protocol
  - Both synchronous and asynchronous
  - For constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP

- CoAP is not
  - A replacement for HTTP
  - General HTTP compression
  - Separate from the web

CORE Architecture

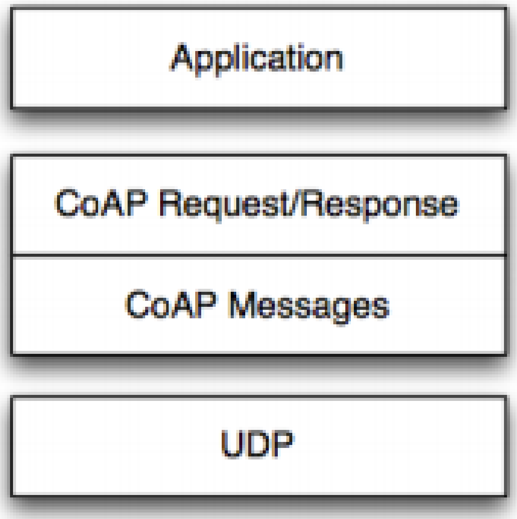
- Use of web services on the Internet depends on the Representational State Transfer (REST) architecture
- **RESTful** protocol to cater the special requirements of a constrained environment
  - Simple RESTful protocol transport
  - Create, Read, Update, Delete, Notify
    - Small, simple header
    - 4 byte base header
    - TLV options, typically 1-270 bytes per option
  - URI support: e.g. **coap://fec0::1:5683/temp**
  - Subset of content types
  - Subset of HTTP-compatible response codes
  - UDP bindings
  - default port = **5683**

C: Constrained node - sensor nodes
Non-constrained nodes - server, proxy, node, etc



The Transaction Model

- Transport
  - CoAP is defined for UDP
- Messaging
  - Simple message exchange between end points
  - CON, NON, ACK, RST
- REST
  - Request/Response piggybacked on messages
  - Method, Response Code and Options (URI, content-type etc.



CoAP Message Format

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 |  OC   |      Code     |          Message  ID          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- **Ver**: Version,
- **T**: Transaction Type (CON,NON,ACK…)
- **OC**: Number of options
- **Code**: Method or Response Code
- **Message ID**: A unique ID assigned by the originator

Option Header

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| option delta  |    length     | for 0..14
+---+---+---+---+---+---+---+---+

                                                for 15..270:
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| option delta  | 1   1   1   1 |          length - 15          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

- **Option Delta**- Difference between this option type and the previous!
- **Length** - Length of the option value (0-270)!
- **Value-** The value of Length bytes immediately follows Length

CoAP Code

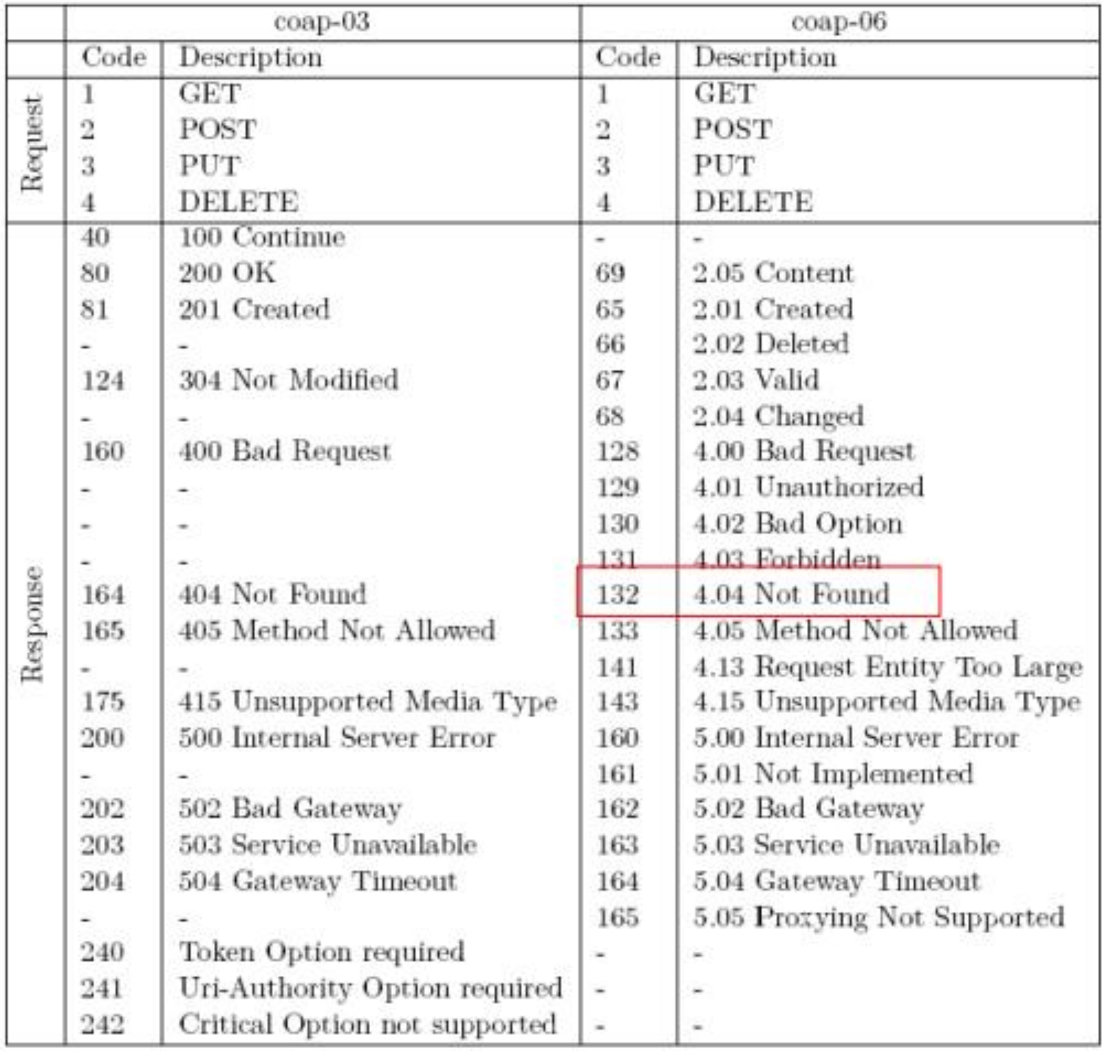
- Structure of the 8 bit Response code:

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|class|  detail |
+-+-+-+-+-+-+-+-+
```

- 3 classes:
  - 2 – Success
  - 4 – Client error
  - 5 – Server error

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|class|  detail |
+-+-+-+-+-+-+-+-+
|1 0 0|0 0 1 0 0|
+-+-+-+-+-+-+-+-+
     4  :  04
```

- Example: "Not Found" is written as 4.04 - indicating a value of decimal 132



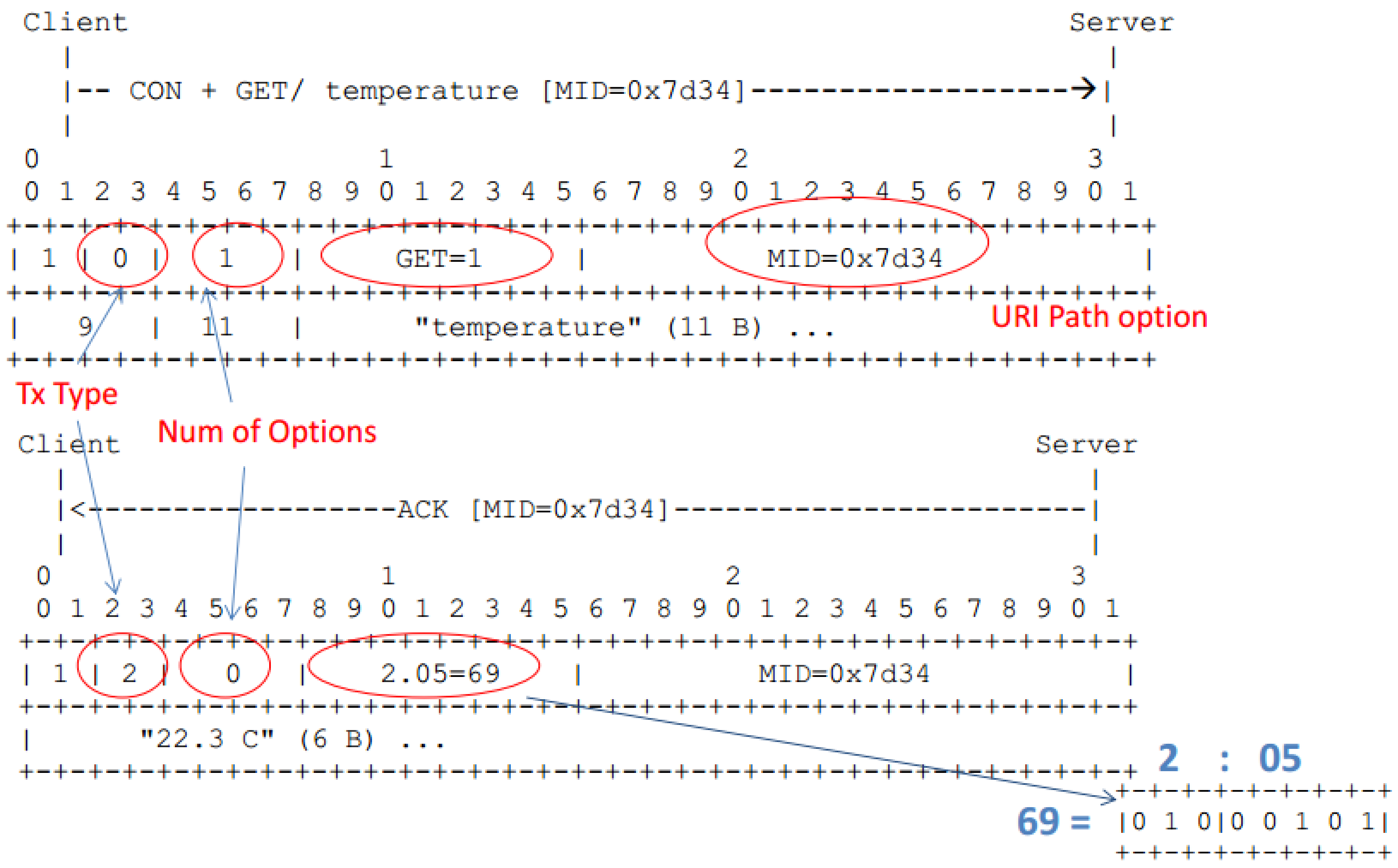
	coap-03		coap-06	
	Code	Description	Code	Description
Request	1	GET	1	GET
	2	POST	2	POST
	3	PUT	3	PUT
	4	DELETE	4	DELETE
Response	40	100 Continue	-	-
	80	200 OK	69	2.05 Content
	81	201 Created	65	2.01 Created
	-	-	66	2.02 Deleted
	124	304 Not Modified	67	2.03 Valid
	-	-	68	2.04 Changed
	160	400 Bad Request	128	4.00 Bad Request
	-	-	129	4.01 Unauthorized
	-	-	130	4.02 Bad Option
	-	-	131	4.03 Forbidden
	164	404 Not Found	132	4.04 Not Found
	165	405 Method Not Allowed	133	4.05 Method Not Allowed
	-	-	141	4.13 Request Entity Too Large
	175	415 Unsupported Media Type	143	4.15 Unsupported Media Type
	200	500 Internal Server Error	160	5.00 Internal Server Error
	-	-	161	5.01 Not Implemented
	202	502 Bad Gateway	162	5.02 Bad Gateway
	203	503 Service Unavailable	163	5.03 Service Unavailable
	204	504 Gateway Timeout	164	5.04 Gateway Timeout
	-	-	165	5.05 Proxying Not Supported
	240	Token Option required	-	-
	241	Uri-Authority Option required	-	-
	242	Critical Option not supported	-	-

Methods of CoAP

Method	Description
GET	Retrieves information of an identified resource
POST	Creates a new resource under the requested URI
PUT	Updates the resource identified by an URI
DELETE	Deletes the resource identified by an URI

- Resources are identified by URIs
- Methods are very similar to HTTP methods
- Response codes are a subset of HTTP response codes
- Options carry additional information (similar to HTTP header lines, but using a more compact encoding)

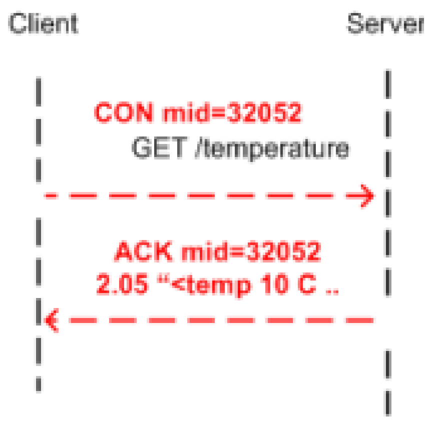
Example of CoAP Message

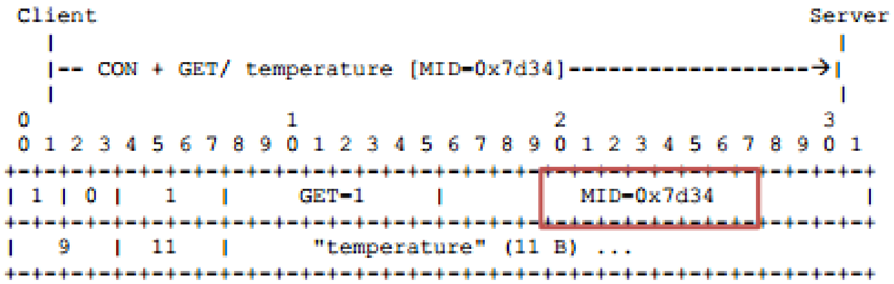


```
Client                                                      Server
   |                                                           |
   |-- CON + GET/ temperature [MID=0x7d34]-------------------->|
   |                                                           |
0                   1                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   1   |     GET=1     |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   9   |   11  |      "temperature" (11 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Client                                                    Server
   |                                                         |
   |<-----------------ACK [MID=0x7d34]-----------------------|
   |                                                         |
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 2 |   0   |    2.05=69    |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

       +-+-+-+-+-+-+-+-+
69 =   |0 1 0|0 0 1 0 1|
       +-+-+-+-+-+-+-+-+
```

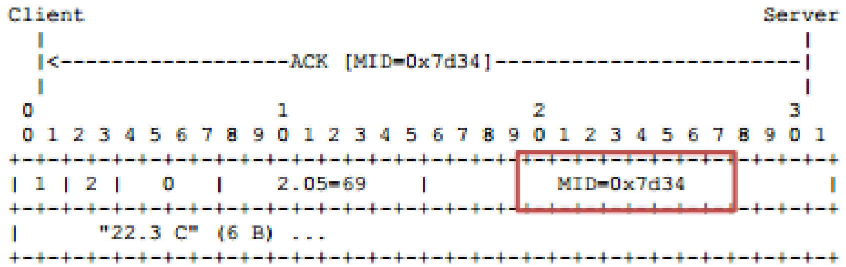
Transaction Messages of CoAP

- ❑ Confirmable (CON)
  - ❖ To know it did arrive or to deliver the reply
  - ❖ Return message type ACK/RST
- ❑ Acknowledgment (ACK)
  - ❖ ACK to the CON message (identified by a message ID)
  - ❖ Payload carries more data
- ❑ Reset (RST)
  - ❖ To let the client know the CON message is received, but processing failed
  - ❖ E.g. node has rebooted
- ❑ Non-Confirmable (NON)
  - ❖ Do not require an ACK
  - ❖ E.g. repeated readings from a sensor





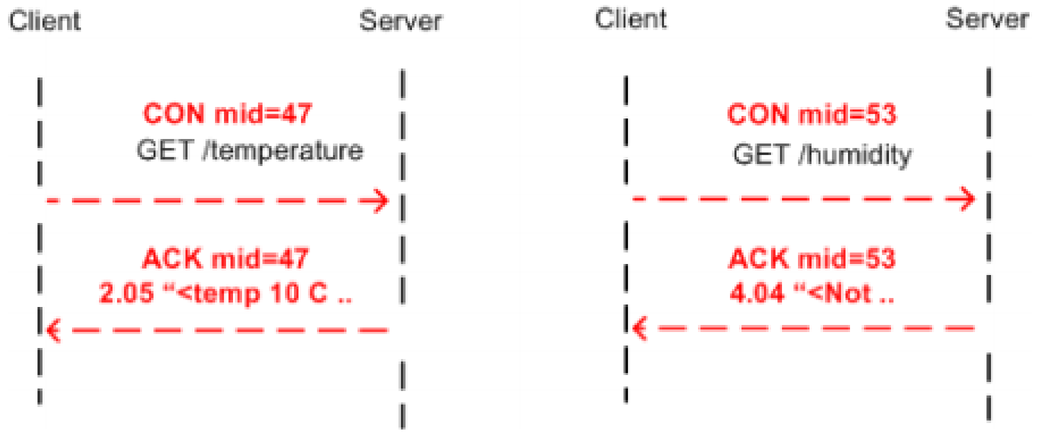
```
Client                                                        Server
   |                                                             |
   |-- CON + GET/ temperature [MID=0x7d34]--------------------->|
   |                                                             |
 0                   1                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   1   |     GET=1     |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   9   |  11   |      "temperature" (11 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



```
Client                                                        Server
   |                                                             |
   |<-----------------ACK [MID=0x7d34]--------------------------|
   |                                                             |
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 2 |   0   |    2.05=69    |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

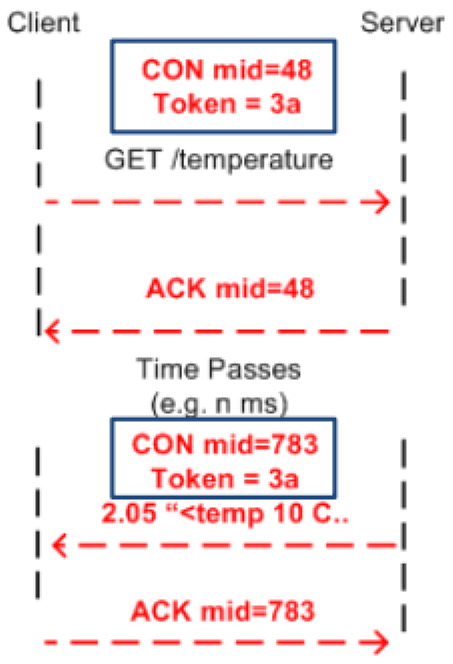
CoAP Operation

- ❑ CoAP interaction is similar to the client/server model (CoAP end points)
  - ❖ URI and Content-type support (like in HTTP)
- ❑ Message exchange is similar to HTTP
  - ❖ Request action is originated by a client (using a Method code)
  - ❖ Request is for a Resource (identified by a URI) on a server
  - ❖ Response is sent with a Response Code



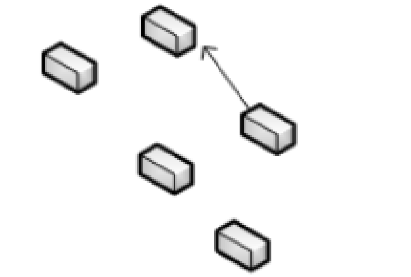
CoAP Operation

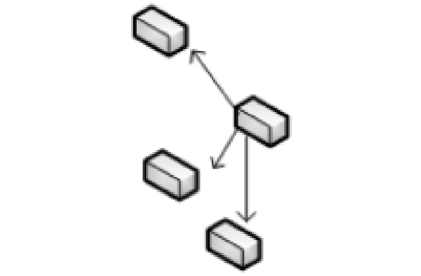
- ❑ Unlike HTTP, CoAP deals
  - ❖ Operates over UDP, with reliable unicast and multicast support
    - Retransmission (= reliable unicast) A CoAP end-point keeps track of open Confirmable messages it sent that are waiting for a response
    - Simple Congestion Control - exponential back-off mechanism
  - ❖ Deals with message interchanges separately
    - Server might need longer time to obtain the representation of resource. This results in the client retransmitting the request

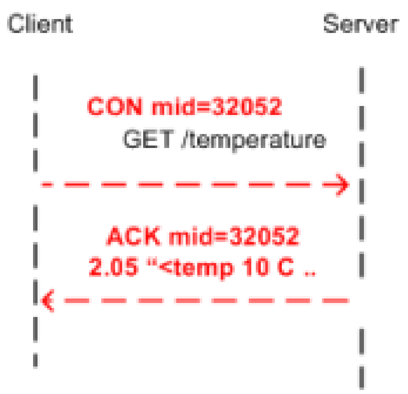


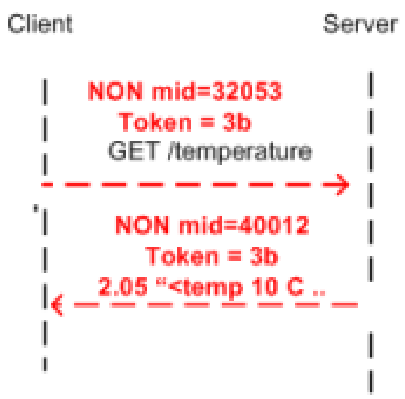
CoAP Operation

- ❑ Operates over UDP, with reliable unicast and best-effort multicast support
- ❑ CON vs NON messages?
- ❑ If CON/NON messages cannot be processed,
  - ❖ Server sends RST message
- ❑ Why MID & Token?


Retrieval of resources using Unicast GET


Retrieval of resources using Multicast GET





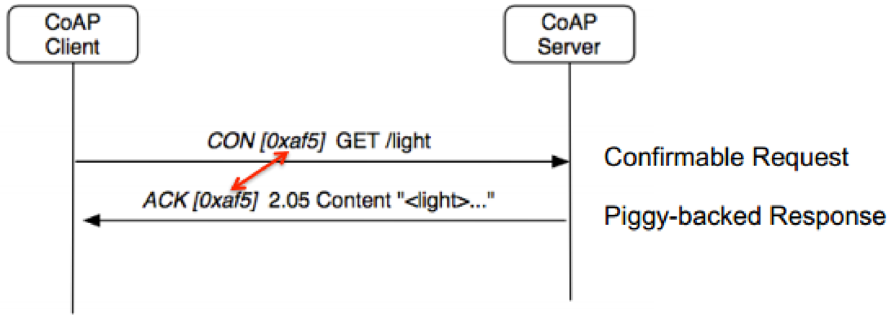
CoAP Options

- *Content-Type*: Specifies the format of the application payload
- *Max-Age*: Maximum age for a message to be cached
- *Proxy-Uri*: Defines an absolute URI to a proxy
- *Token*: Matches a separate response to a request
- *Uri-Host*: Specifies the Internet host of a resource (only added if not representing the destination IP address of the request)
- *Uri-Path*: Specifies the resource of the host
- *Uri-Port*: Specifies the port of the host (only added if differs from the destination UDP port request)
- *Uri-Query*: Indicates additional options for the request
- etc..

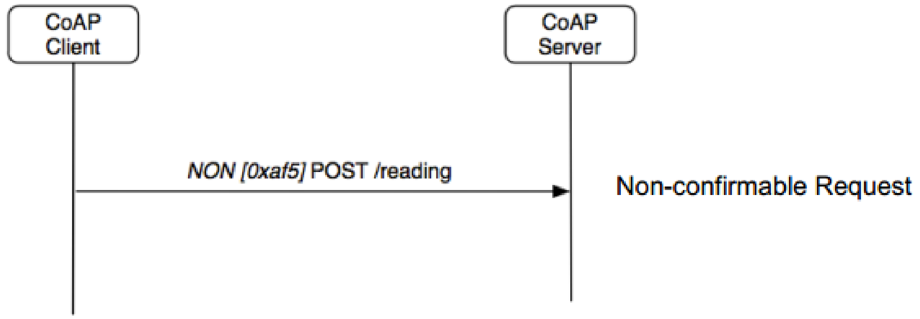
Request Example

❑ Request and Response

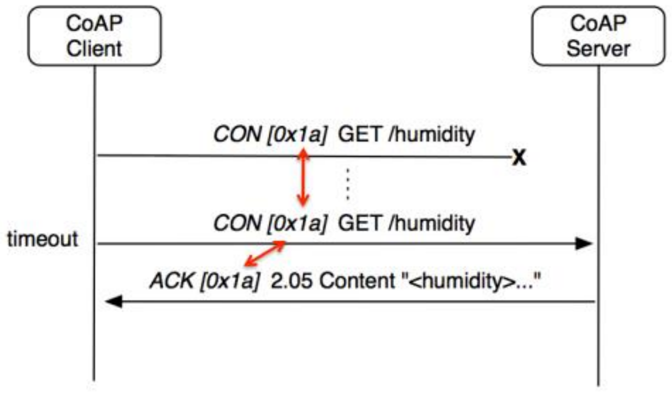
❖ CON message



❖ NON message

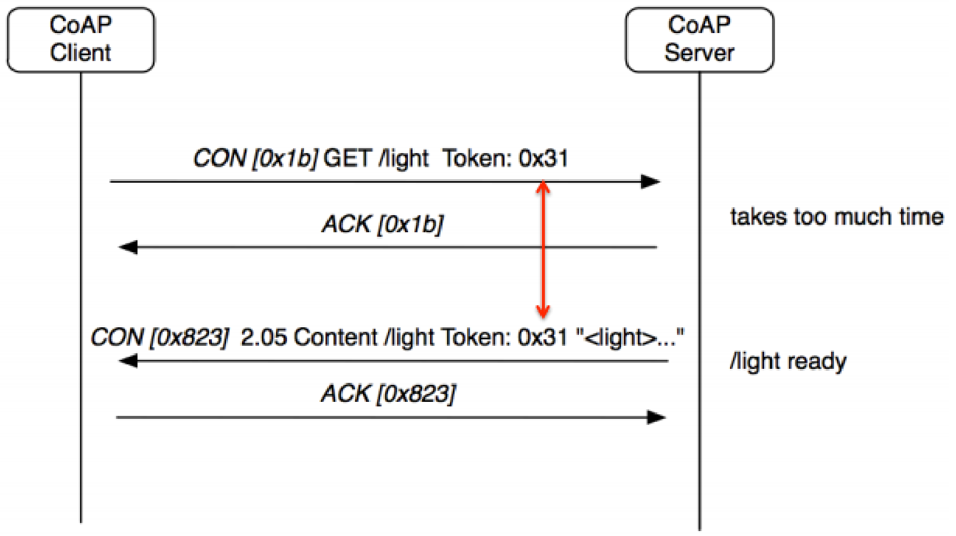


❑ Dealing with packet loss



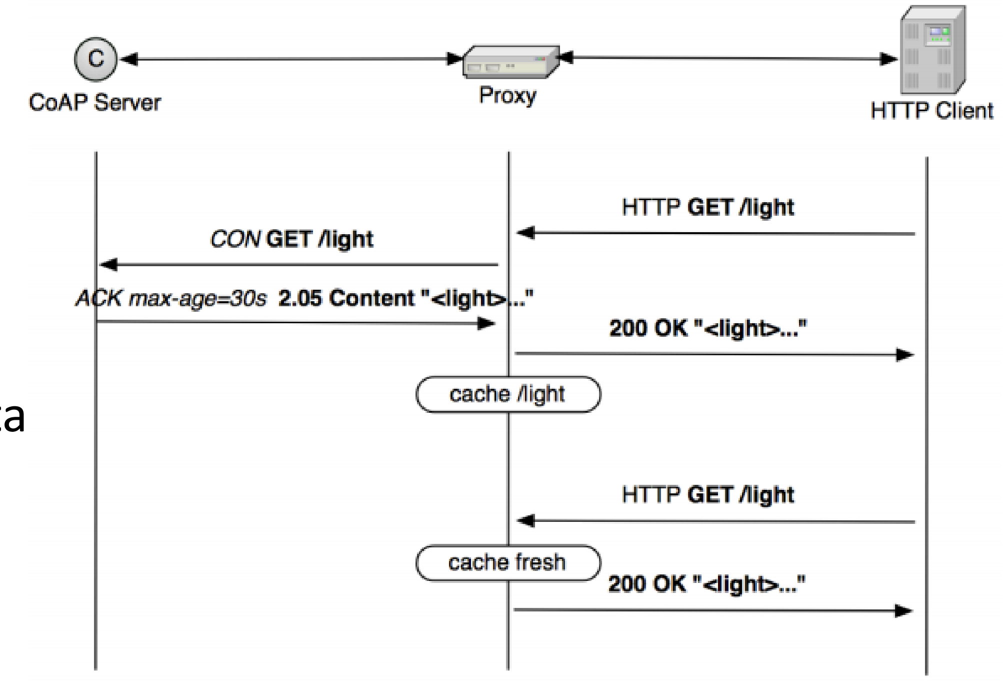
Request Example

- Normal response


CoAP Client
CoAP Server
CON [0x1b] GET /light Token: 0x31
ACK [0x1b]
takes too much time
CON [0x823] 2.05 Content /light Token: 0x31 "<light>..."
/light ready
ACK [0x823]

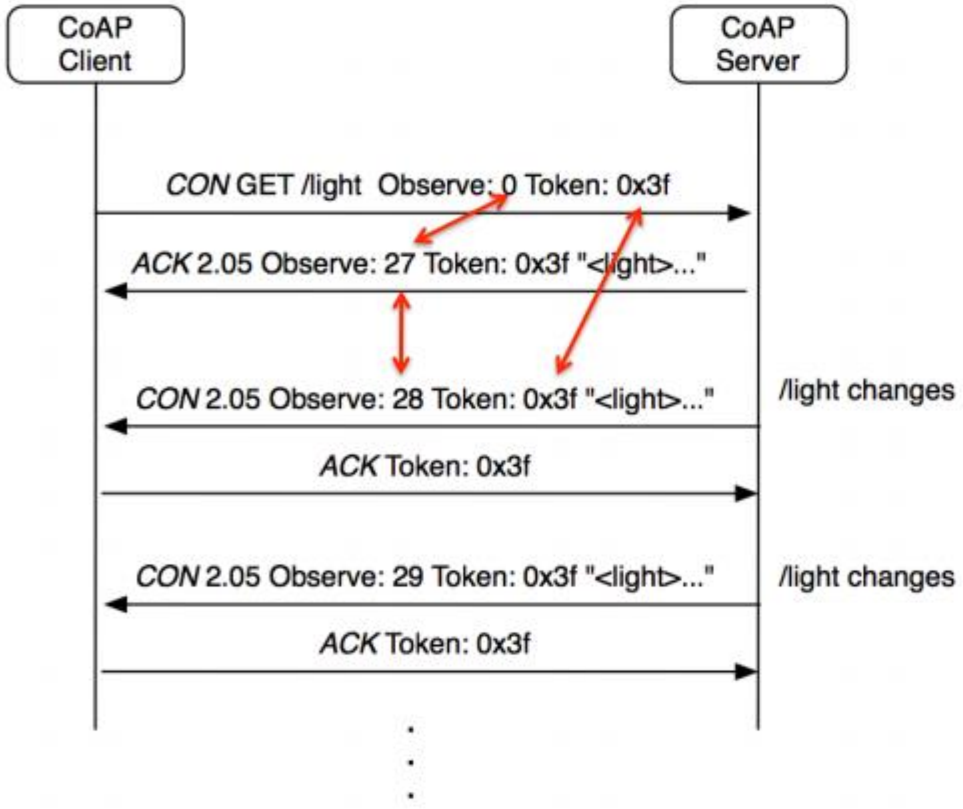
Caching

- ❑ CoAP includes a simple caching model
  - ❖ Cache ability determined by response code
- ❑ Freshness model
  - ❖ Max-Age option indicates cache lifetime
- ❑ Validation model
  - ❖ Validity checked using the Eta Option
- ❑ A proxy often supports caching
  - ❖ Usually on behalf of a sleeping node,
  - ❖ and to reduce network load



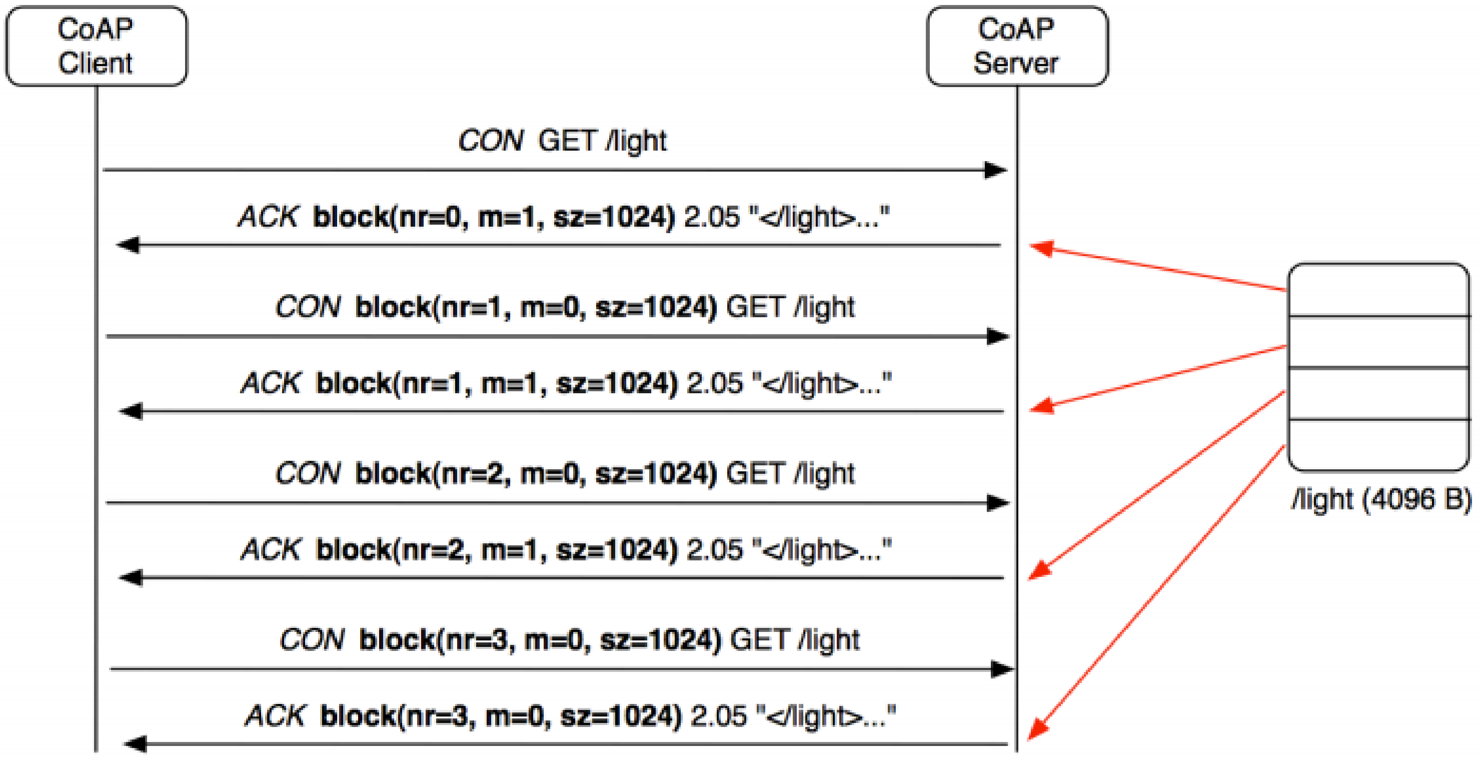
Observation

- The Observe Option, when present, modifies the GET method so it does not only retrieve a representation of the current state of the resource identified by the request URI, but also requests the server to add the client to the list of observers of the resource.
- In a response, the Observe Option identifies the message as a notification, which implies that the client has been added to the list of observers and that the server will notify the client of further changes to the resource state


CoAP Client
CoAP Server
CON GET /light Observe: 0 Token: 0x3f
ACK 2.05 Observe: 27 Token: 0x3f "<light>..."
CON 2.05 Observe: 28 Token: 0x3f "<light>..."
/light changes
ACK Token: 0x3f
CON 2.05 Observe: 29 Token: 0x3f "<light>..."
/light changes
ACK Token: 0x3f

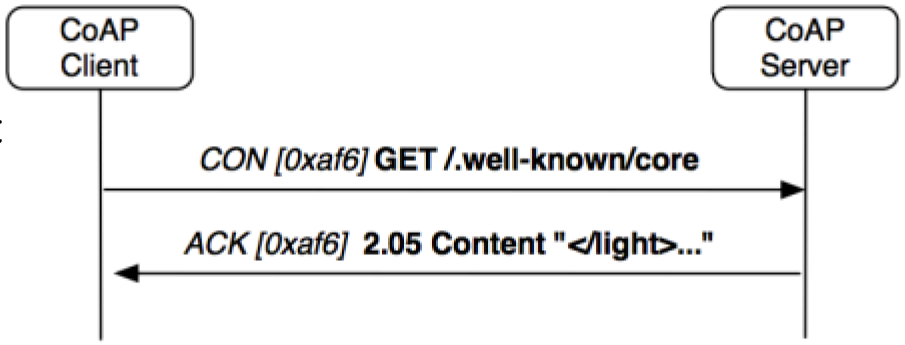
Block Transfer

- Transfers larger than can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.


CoAP Client
CoAP Server
CON GET /light
ACK block(nr=0, m=1, sz=1024) 2.05 "</light>..."
CON block(nr=1, m=0, sz=1024) GET /light
ACK block(nr=1, m=1, sz=1024) 2.05 "</light>..."
CON block(nr=2, m=0, sz=1024) GET /light
ACK block(nr=2, m=1, sz=1024) 2.05 "</light>..."
CON block(nr=3, m=0, sz=1024) GET /light
ACK block(nr=3, m=0, sz=1024) 2.05 "</light>..."
/light (4096 B)

Resource Discovery

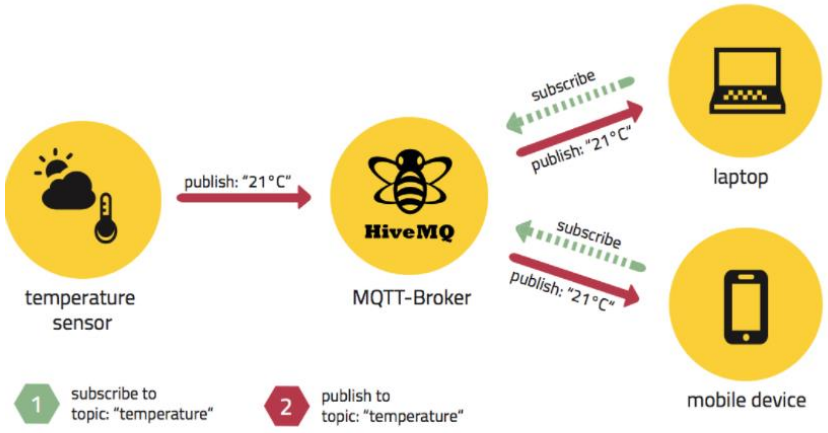
- ❑ Resource Discovery with CoRE Link Format
  - ❖ Web linking as per RFC5988
  - ❖ Discovering the links hosted by CoAP servers
  - ❖ **GET /.well-known/core**
  - ❖ Returns a link-header style format
- ❑ URL, relation, type, interface, content-type etc.

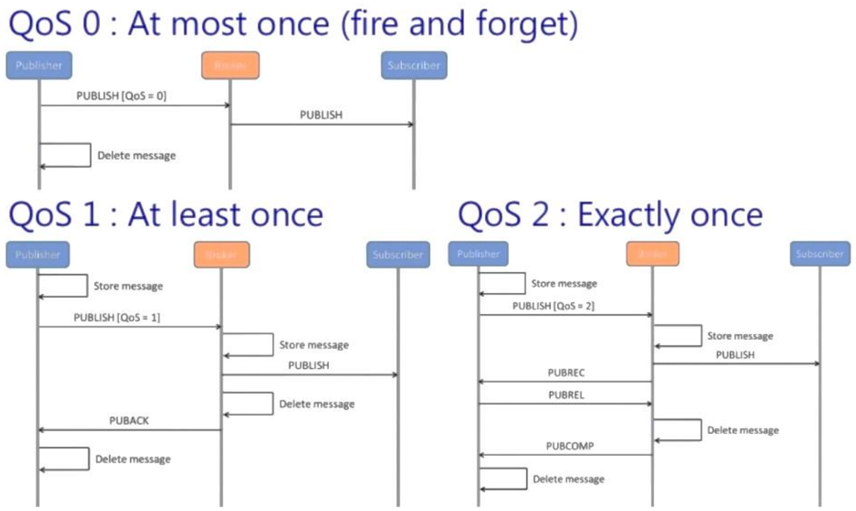

CoAP Client
CoAP Server
CON [0xaf6] **GET /.well-known/core**
ACK [0xaf6] **2.05 Content "</light>..."**

```
</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
```

MQTT

- ❑ **MQTT** (Message Queuing Telemetry Transport) is a machine-to-machine connectivity protocol that runs over TCP/IP.
- ❑ Lightweight, simple, MQTT is based on a publish- subscribe structure:
  - ❖ Publisher
  - ❖ Broker
  - ❖ Subscriber
- ❑ QoS Support
  - ❖ QoS 0, QoS 1, QoS 2



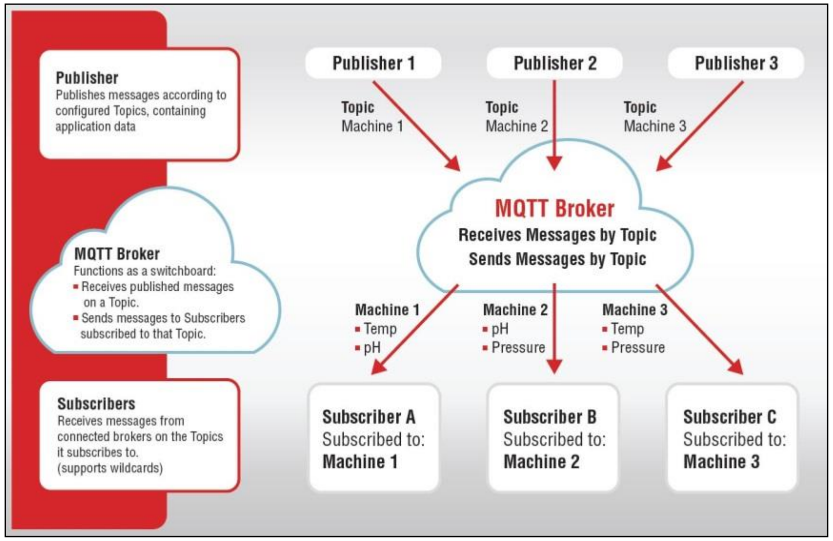


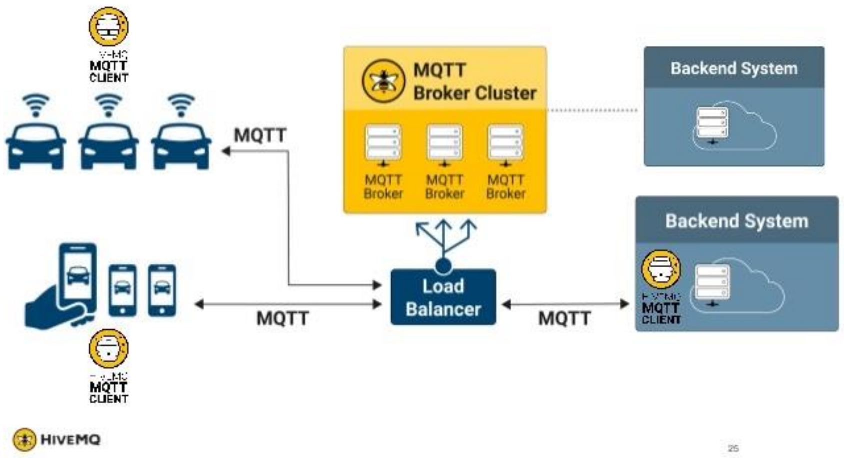
MQTT: Benefits

- **Lightweight and efficient**: MQTT reports by exception and message headers are very small, minimizing the resources required for the client and network bandwidth.
- **Bi-directional**: allows messaging from devices to the cloud and from the cloud to devices, which enables broadcasting messages to groups of things.
- **Supports unreliable networks**: support for persistent sessions reduces the reconnection time required over unreliable networks.
- **Scalable**: can scale to connect millions of IoT devices simultaneously.
- **Security enabled**: MQTT makes it easy to encrypt messages using TLS (transport layer security) and authenticate clients using modern authentication protocols, such as OAuth.
- **Reliable**: The publish/subscribe functionality and the ability to queue messages means no data loss through MQTT. MQTT also supports reliable message delivery through the three defined QoS

MQTT Example

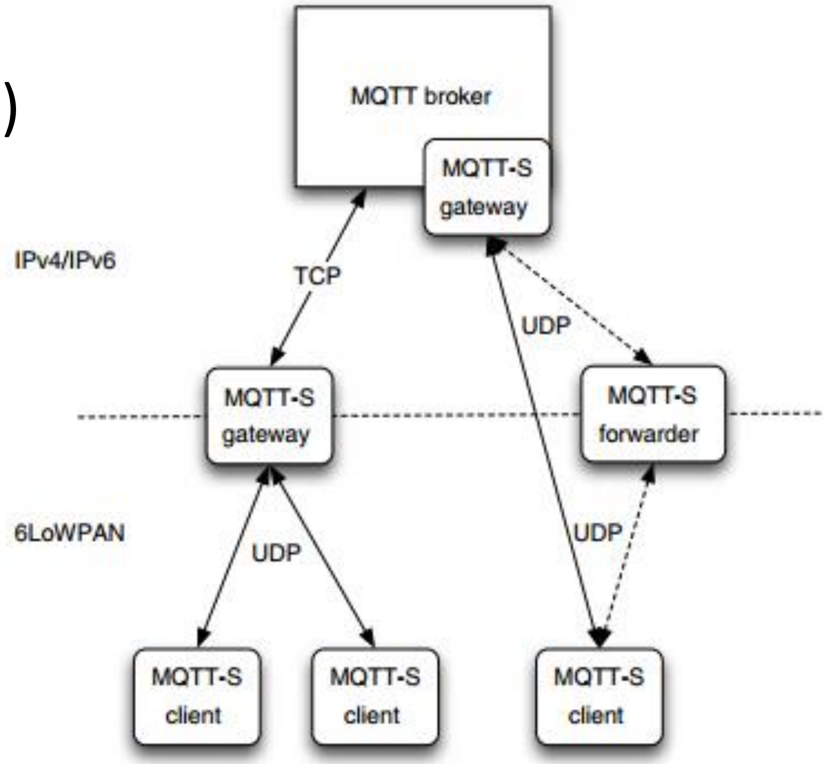
- Clients **do not have addresses** like in email systems, and messages are not sent to clients.
- Messages are **published to a broker on a topic**.
- The job of an MQTT broker is to **filter messages** based on topic, and then **distribute them to subscribers**.
- A client can receive these messages by subscribing to that topic on the same broker
- There is **no direct connection** between a publisher and subscriber.
- **All clients** can publish (broadcast) and subscribe (receive).
- MQTT brokers do not normally store messages.


Publisher
Publishes messages according to configured Topics, containing application data
MQTT Broker
Functions as a switchboard:
Receives published messages on a Topic.
Sends messages to Subscribers subscribed to that Topic.
Subscribers
Receives messages from connected brokers on the Topics it subscribes to. (supports wildcards)
Publisher 1
Publisher 2
Publisher 3
Topic Machine 1
Topic Machine 2
Topic Machine 3
MQTT Broker
Receives Messages by Topic
Sends Messages by Topic
Machine 1 Temp pH
Machine 2 pH Pressure
Machine 3 Temp Pressure
Subscriber A Subscribed to: Machine 1
Subscriber B Subscribed to: Machine 2
Subscriber C Subscribed to: Machine 3



MQTT-S Protocol

- ❑ MQ Telemetry transport for WSNs
- ❑ Publish/Subscribe protocol
- ❑ Optimized for low-bandwidth wireless sensor networks (MQTT-S)
- ❑ Integrated with Brokers (gateways)
- ❑ Can be used with UDP/6LoWPAN
- ❑ Model: client/broker
- ❑ Application: M2M



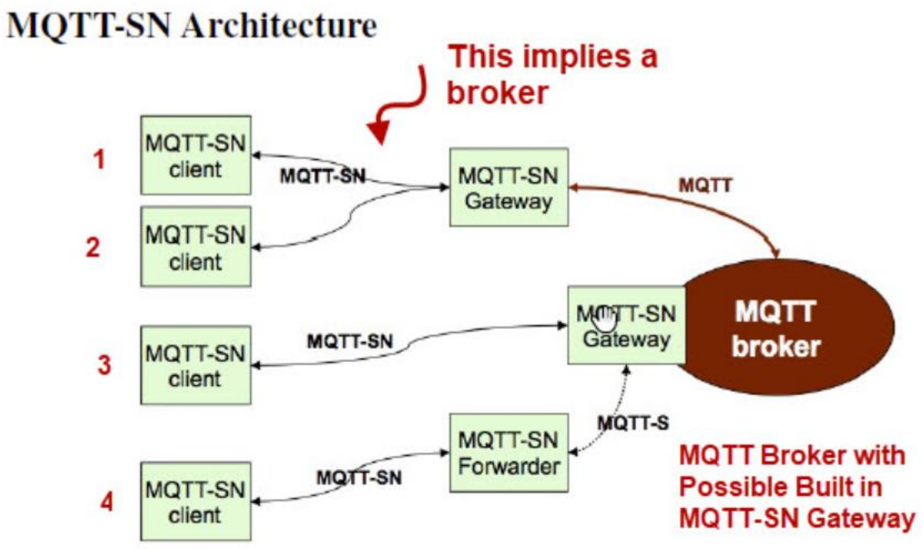
MQTT-S Protocol

- Connect message split into three messages, two are optional and are used for the will message
- Topic ID’s used in place of topic names.
- Short Topic names
- Pre-defined topics.
- Discovery process to let clients discover the Gateway
- Will Topic and messages can be changed during the session
- Offline keep alive procedure for **sleeping clients.**

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Length    |    Msg Type   |     Variable Message Part    ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

MQTT-SN Architecture

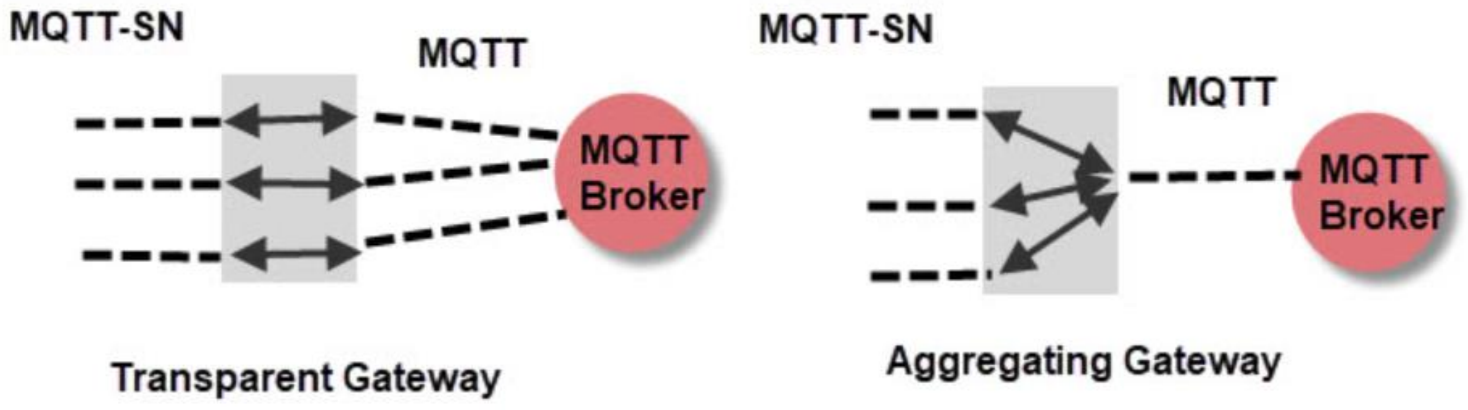
- When client 1 talks to client 2 does it require the MQTT broker to do so? Specifically in the diagram above for clients 3 and 4. Is it: MQTT-SN >MQTT-SN. In this case the gateway is also a MQTT-SN broker.
- or is it: MQTT-SN >MQTT>MQTT-SN. In this case the gateway is an MQTT-SN gateway and an MQTT broker.



http://www.steves-internet-guide.com/mqtt-sn/

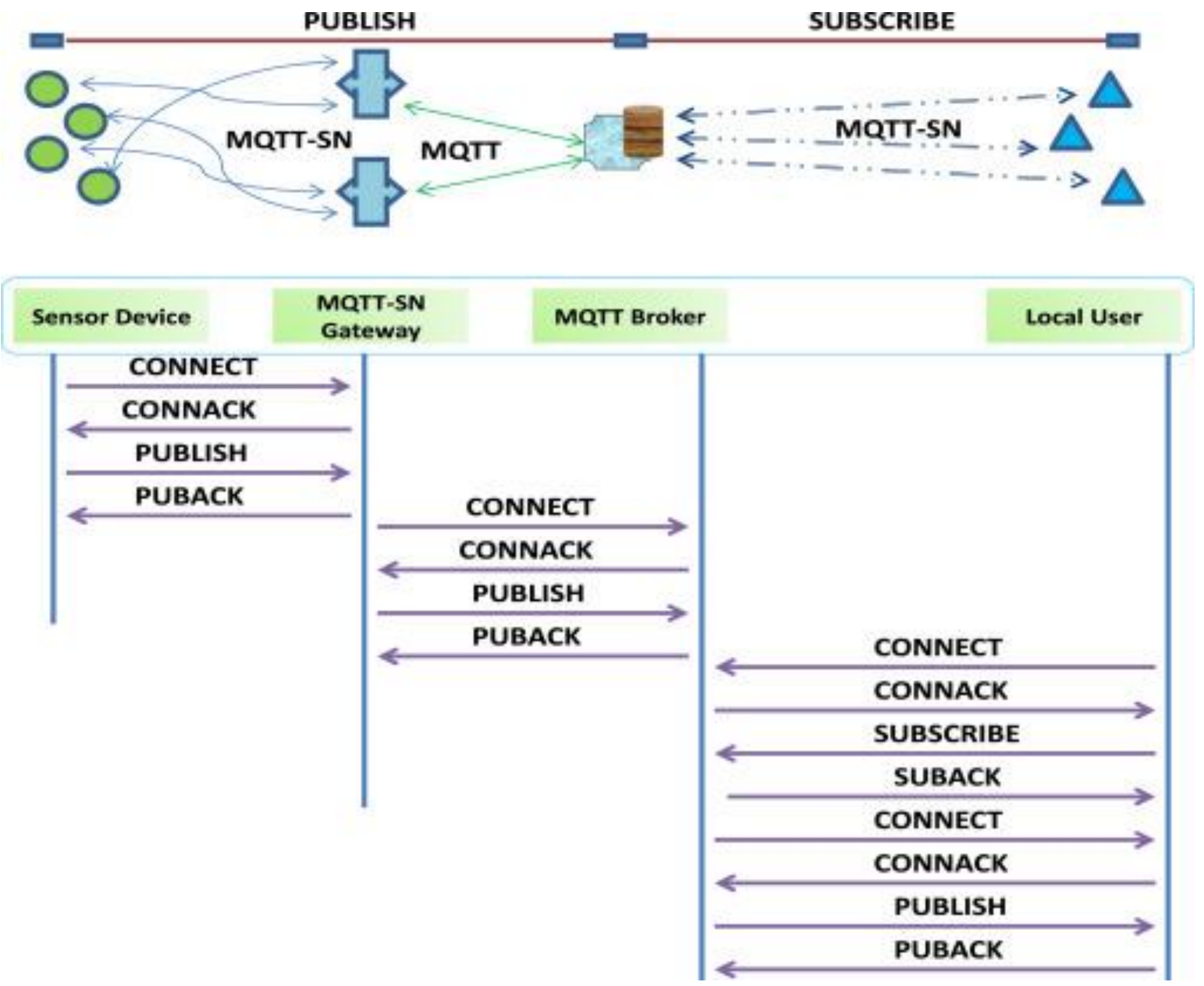
MQTT-SN in Contiki

- Gateway Type: The specification defines two gateway types:
  - A transparent gateway were each MQTT-SN connection has a corresponding MQTT connection. This is the easiest type to implement.
  - An aggregating gateway were multiple MQTT-SN connections share a single MQTT connection



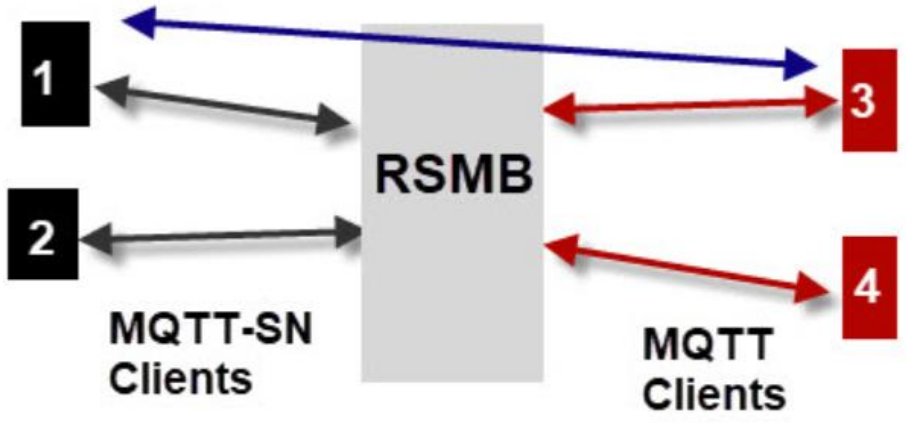
http://www.steves-internet-guide.com/mqtt-sn/

MQTT-SN: Data flow



Really Small Message Broker

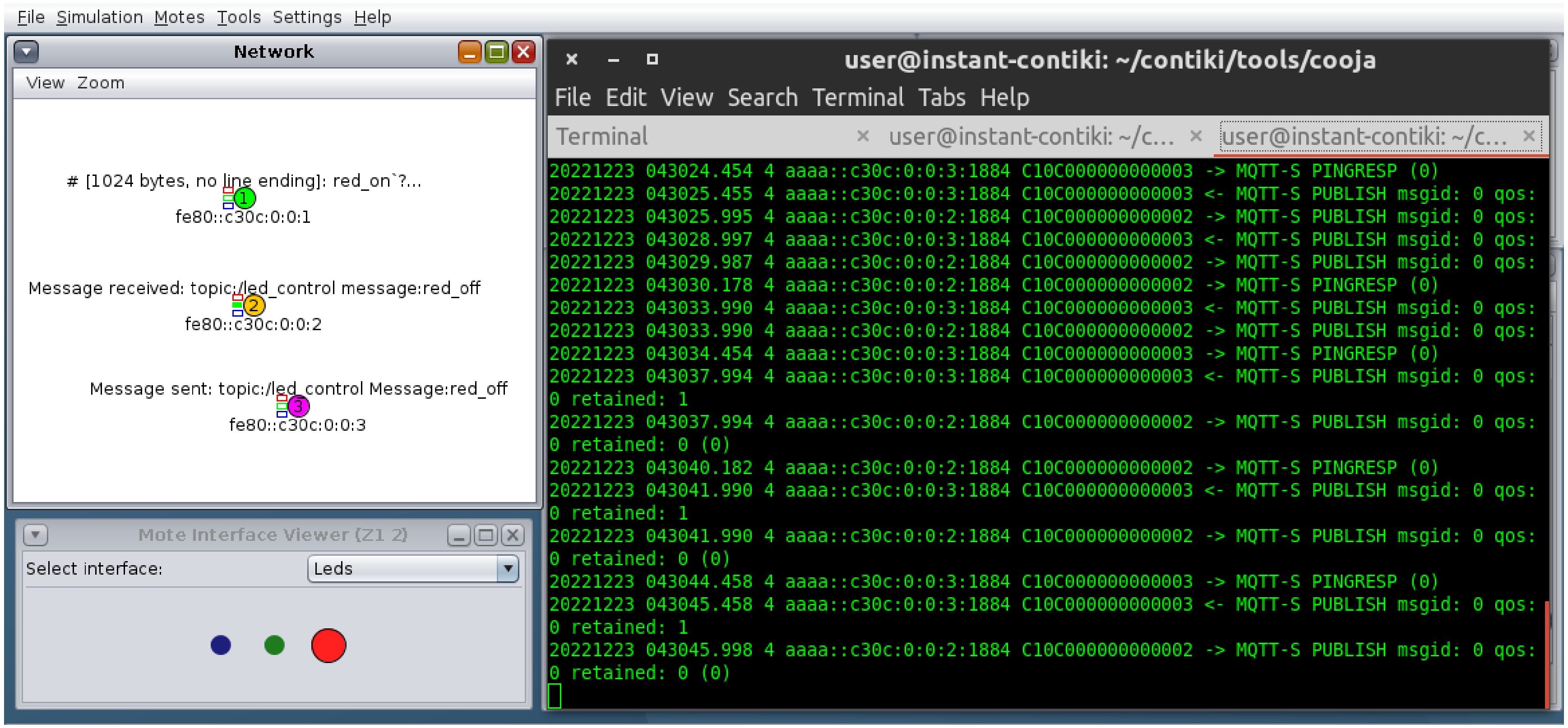
- Really Small Message Broker (RSMB)
  - Client 1 to Client 2 then **RSMB = MQTT-SN Broker**
  - Client 3 to Client 4 then **RSMB = MQTT Broker**
  - Client 1 to Client 3 then **RSMB = MQTT-SN-MQTT Gateway**
- The RSMB broker functions both as a MQTT-SN broker and also as an MQTT broker allowing messages between MQTT-SN clients and MQTT clients



http://www.steves-internet-guide.com/mqtt-sn/

MQTT-SN in Contiki

❑ MQTT-SN with RSMB in Contiki


File Simulation Motes Tools Settings Help
Network
View Zoom
Mote Interface Viewer (Z1 2)
Select interface:
Leds
user@instant-contiki: ~/contiki/tools/cooja
File Edit View Search Terminal Tabs Help
Terminal
user@instant-contiki: ~/c...
user@instant-contiki: ~/c...

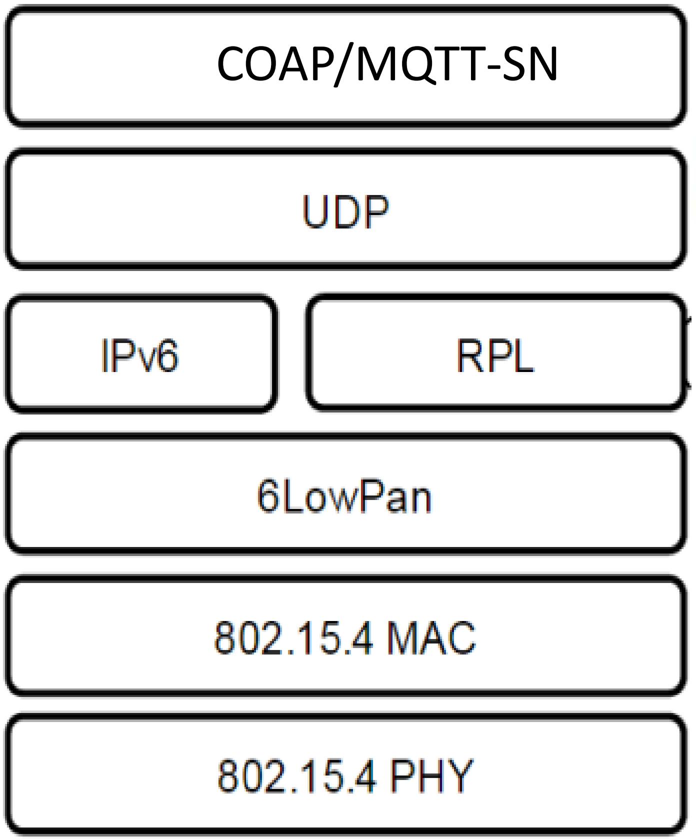
[1024 bytes, no line ending]: red_on`?...
fe80::c30c:0:0:1

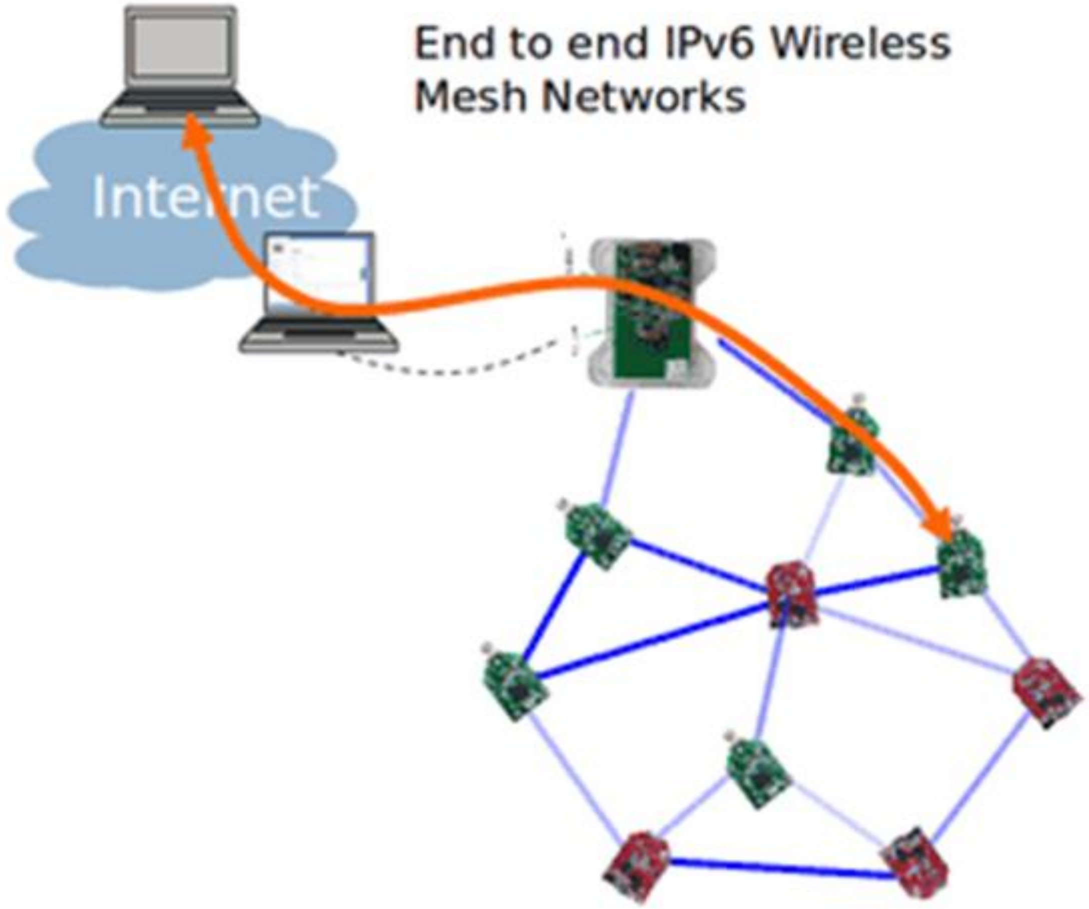
Message received: topic:/led_control message:red_off
fe80::c30c:0:0:2

Message sent: topic:/led_control Message:red_off
fe80::c30c:0:0:3

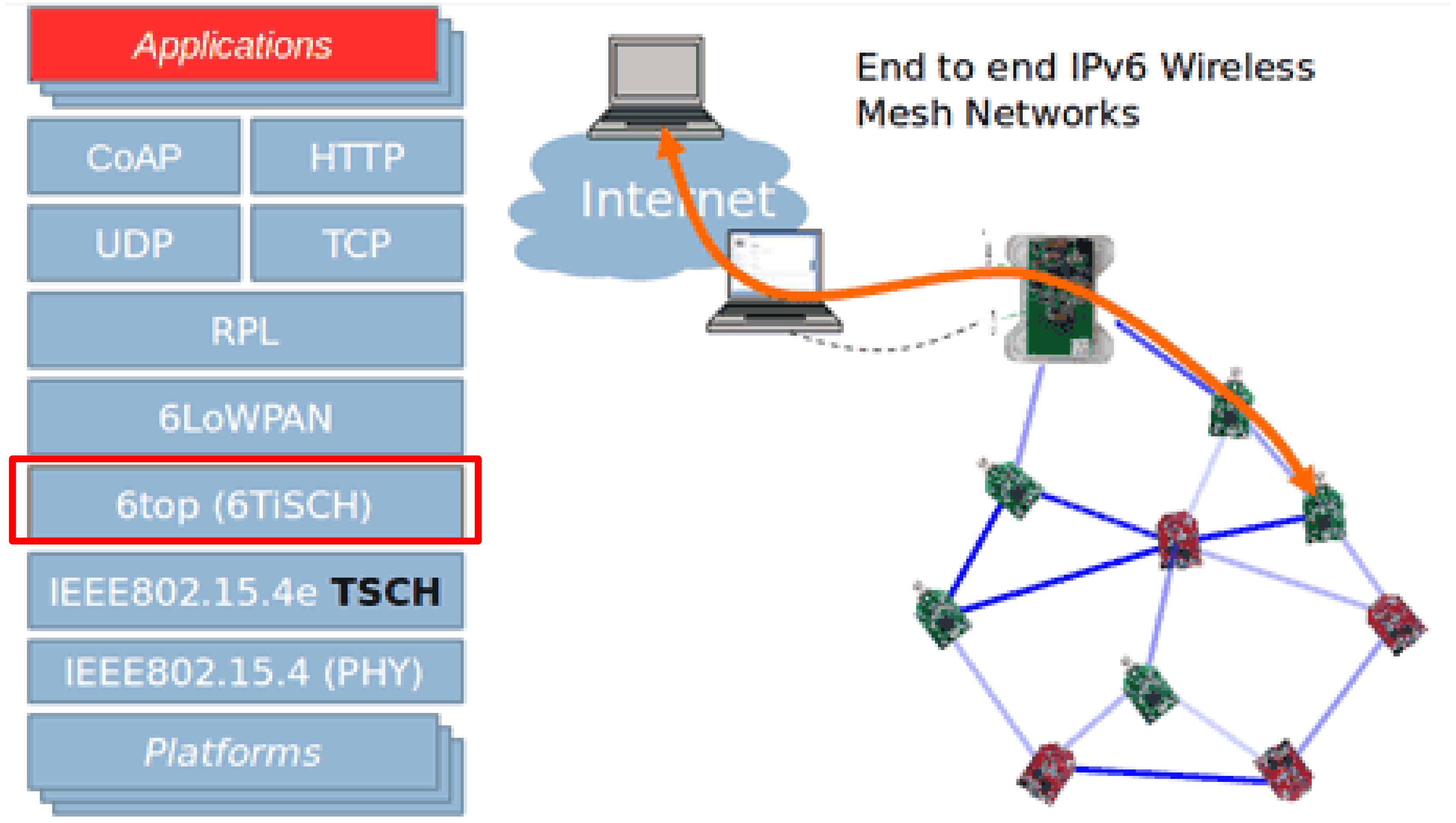
```
20221223 043024.454 4 aaaa::c30c:0:0:3:1884 C10C000000000003 -> MQTT-S PINGRESP (0)
20221223 043025.455 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
20221223 043025.995 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
20221223 043028.997 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
20221223 043029.987 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
20221223 043030.178 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PINGRESP (0)
20221223 043033.990 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
20221223 043033.990 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
20221223 043034.454 4 aaaa::c30c:0:0:3:1884 C10C000000000003 -> MQTT-S PINGRESP (0)
20221223 043037.994 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
0 retained: 1
20221223 043037.994 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
0 retained: 0 (0)
20221223 043040.182 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PINGRESP (0)
20221223 043041.990 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
0 retained: 1
20221223 043041.990 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
0 retained: 0 (0)
20221223 043044.458 4 aaaa::c30c:0:0:3:1884 C10C000000000003 -> MQTT-S PINGRESP (0)
20221223 043045.458 4 aaaa::c30c:0:0:3:1884 C10C000000000003 <- MQTT-S PUBLISH msgid: 0 qos:
0 retained: 1
20221223 043045.998 4 aaaa::c30c:0:0:2:1884 C10C000000000002 -> MQTT-S PUBLISH msgid: 0 qos:
0 retained: 0 (0)
```

Full stack with 802.15.4 MAC





Full stack with 802.15.4e MAC

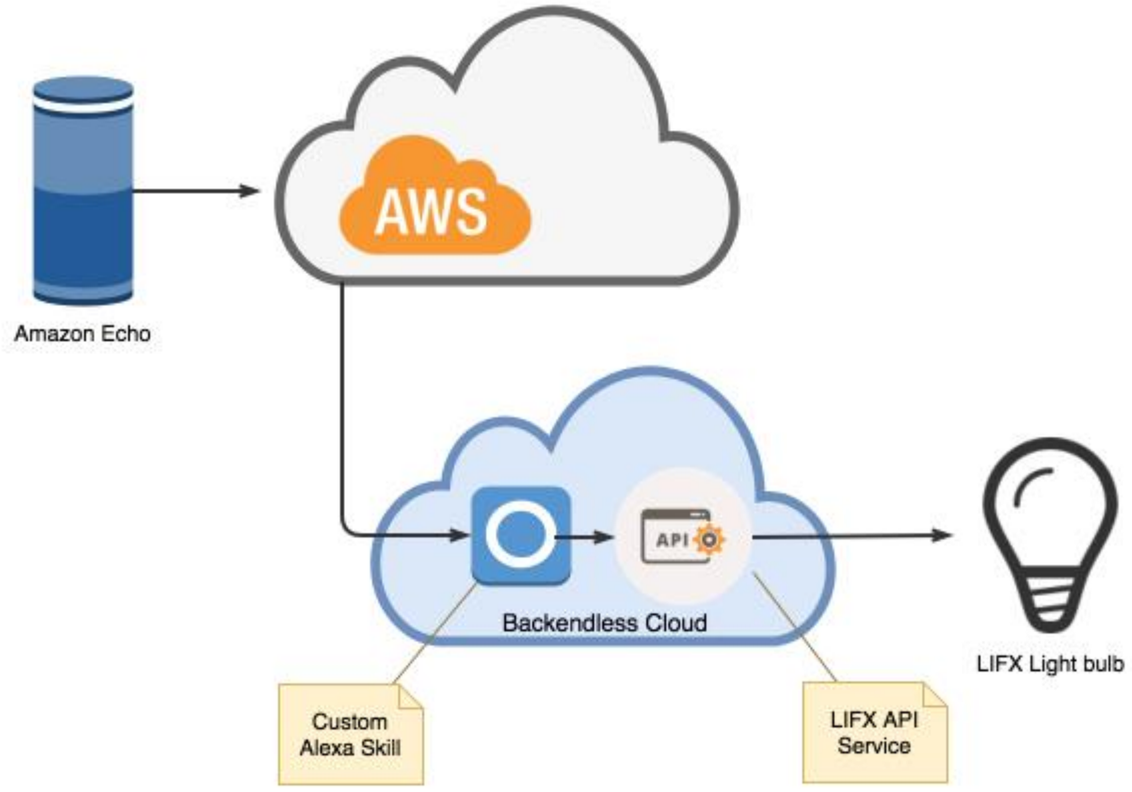


Human-Machine Interface

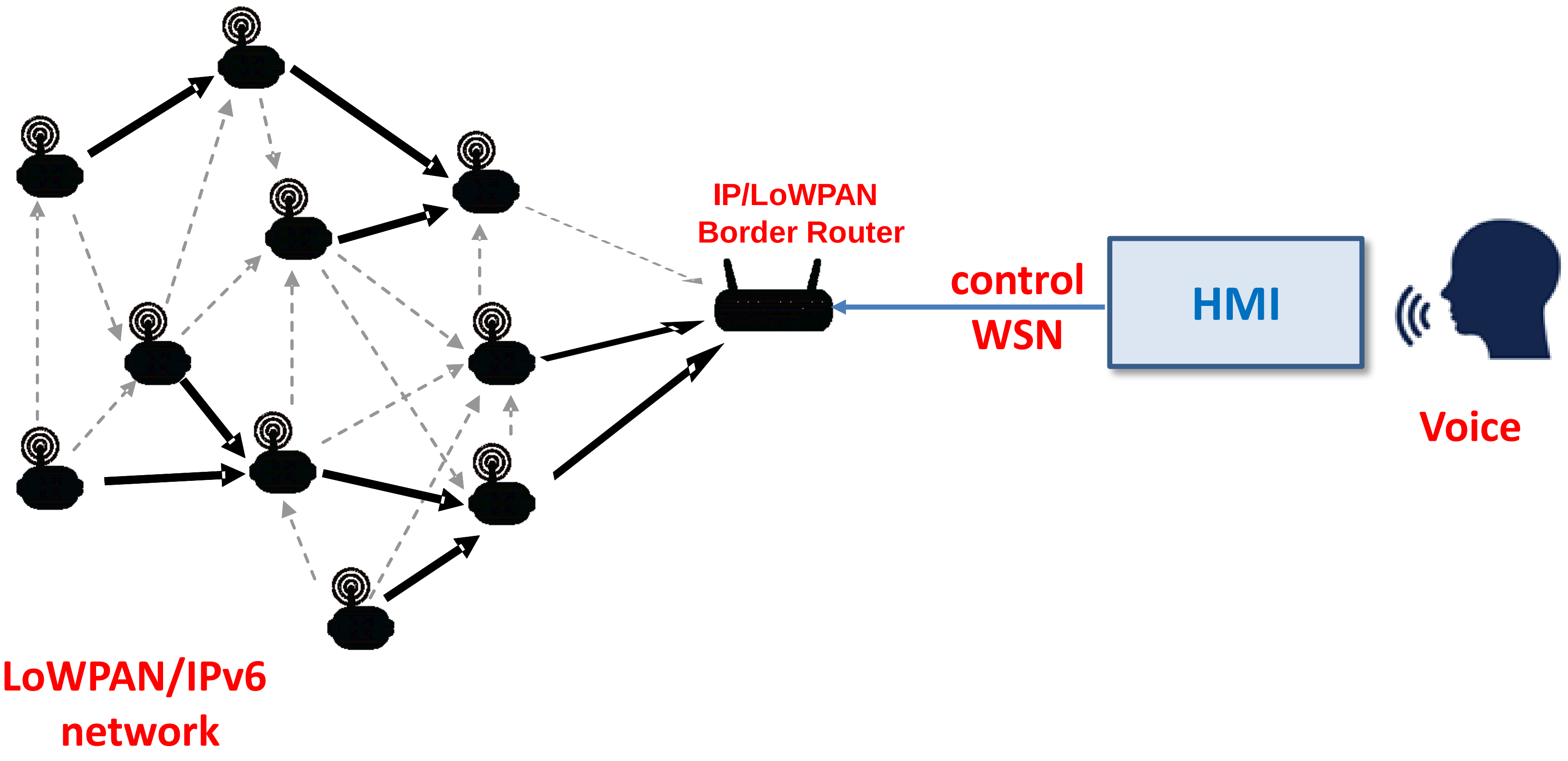
- Benefit:
  - Make the Smart Applications more than a Remote-control system
  - Natural interaction with humane (via voice, gesture, etc..)
- Technologies:
  - AI, Machine learning, Data Science
  - Cloud-based or local model (AI on-the-Edge)

Human-Machine Interface

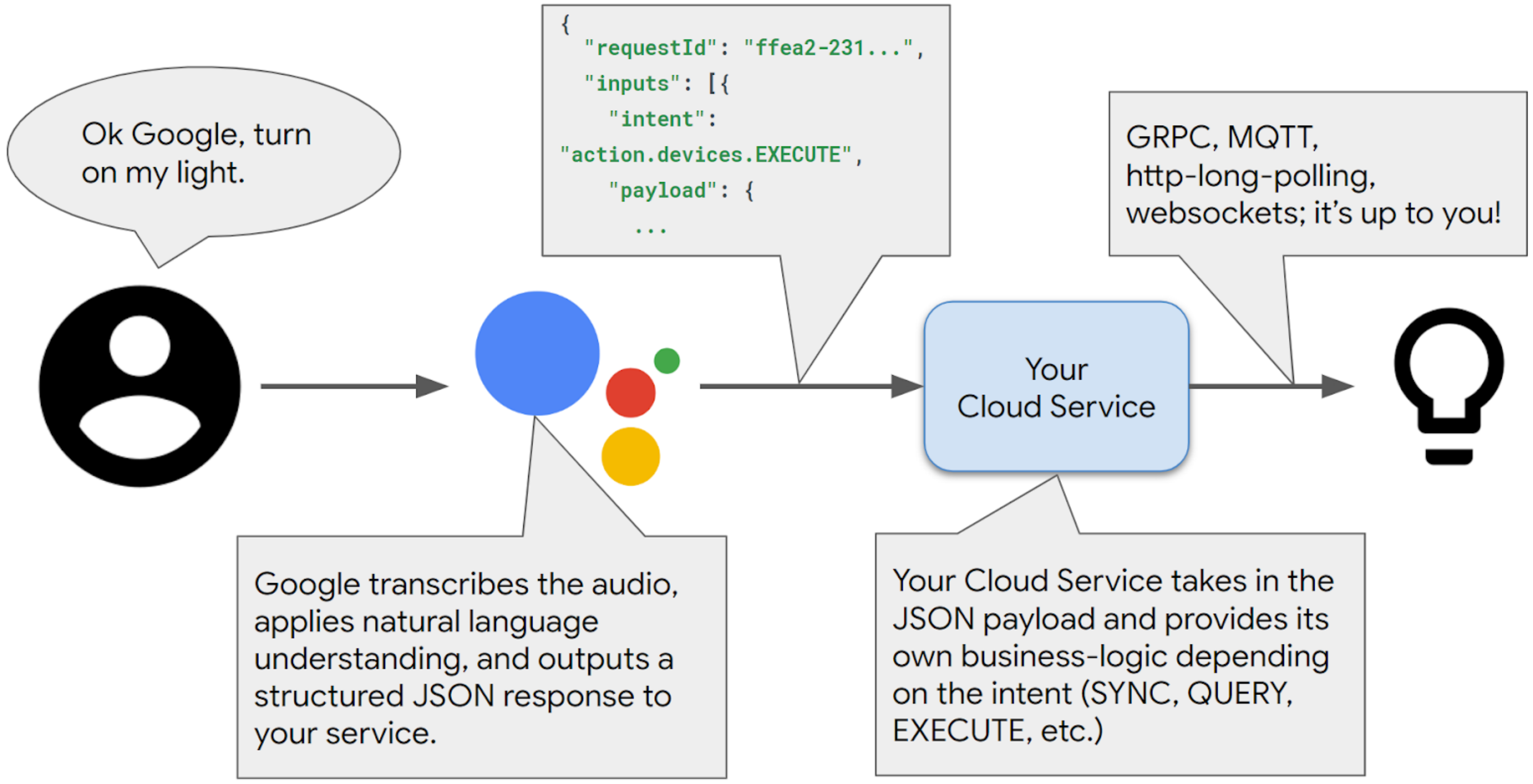
- ❑ Integrated Alexa, Google Voice



Human-Machine Interface



Human-Machine Interface

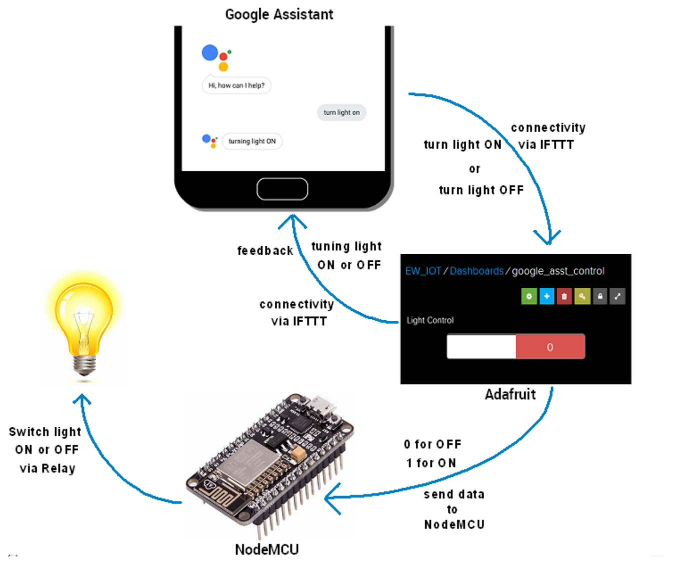

Ok Google, turn on my light.
Your Cloud Service
GRPC, MQTT, http-long-polling, websockets; it's up to you!

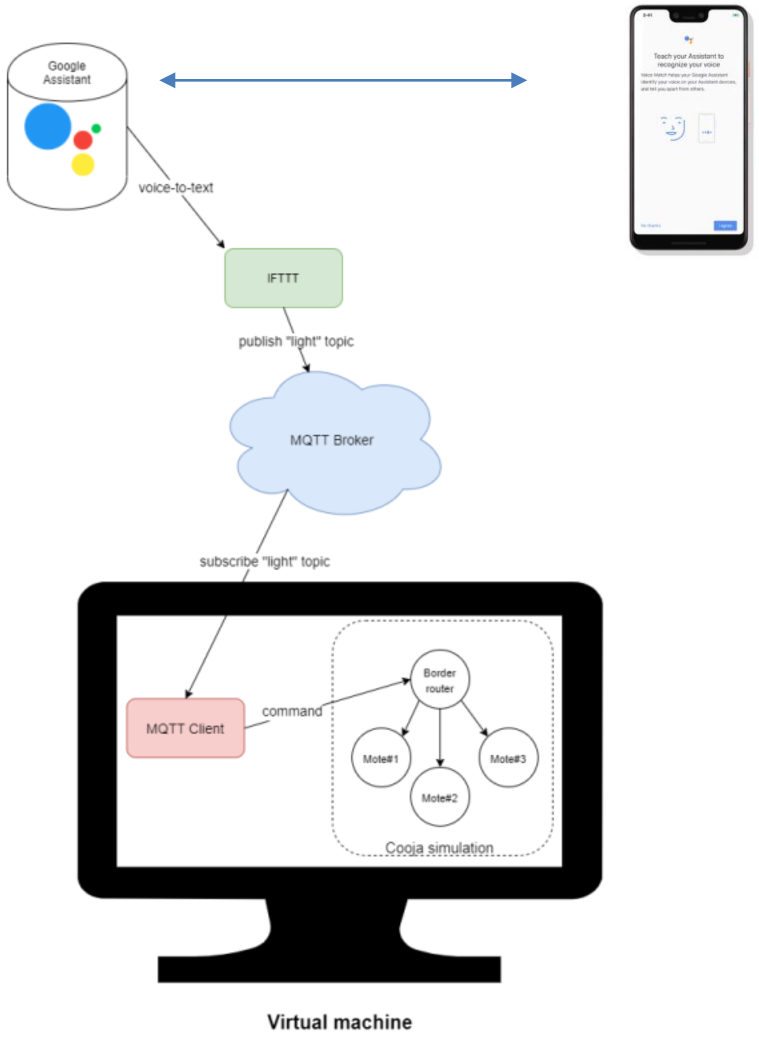
```
{
  "requestId": "ffea2-231...",
  "inputs": [{
    "intent":
"action.devices.EXECUTE",
    "payload": {
      ...
```

Google transcribes the audio, applies natural language understanding, and outputs a structured JSON response to your service.

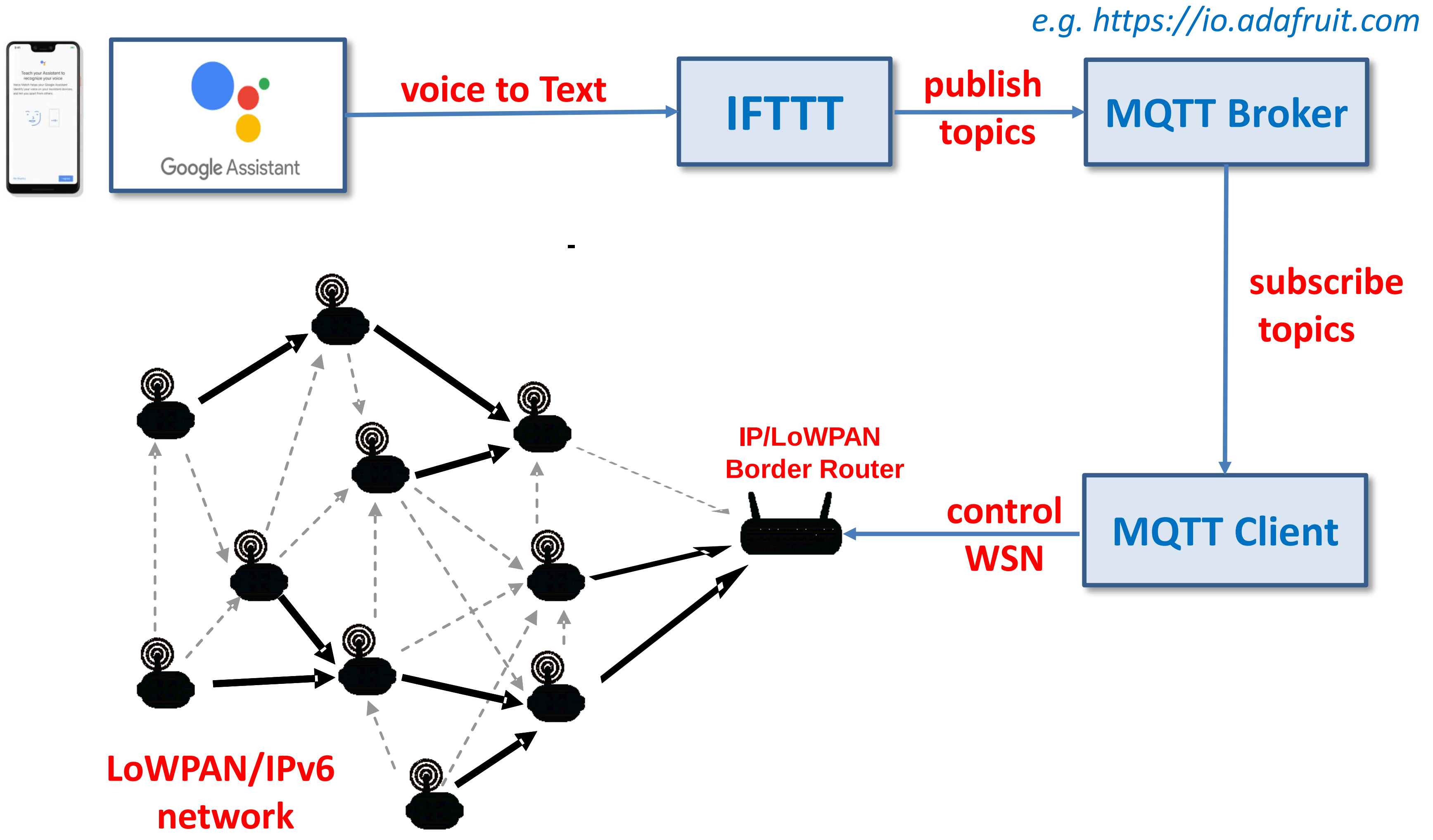
Your Cloud Service takes in the JSON payload and provides its own business-logic depending on the intent (SYNC, QUERY, EXECUTE, etc.)

HMI: using Google Assistant





HMI: example using Google Assistant

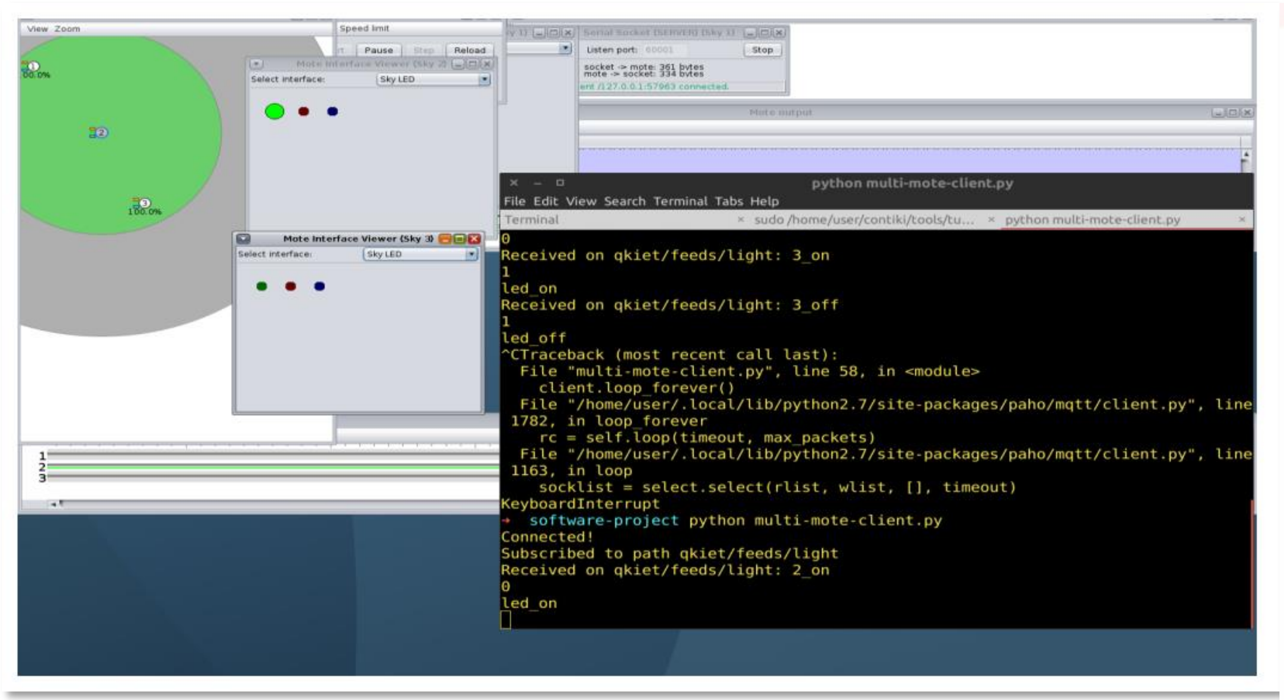


HMI: an example with Cooja

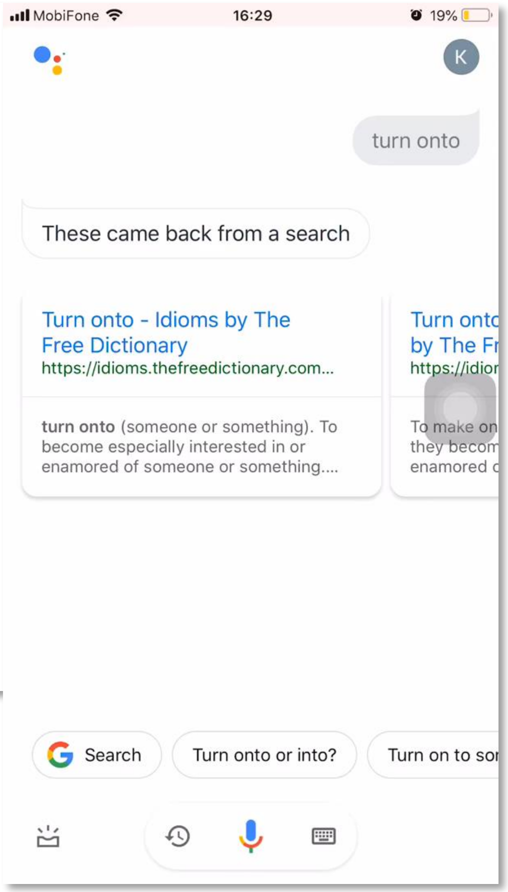
- **Google Assistant:** Convert voice to text, and transfer text to IFTTT platform
- **IFTTT:** a platform to connect devices and applications together.

In this scenario, it connect Google Assistant with MQTT Broker. When a voice-to-text is sent to IFTTT, it will publish this text to MQTT Broker.
- **MQTT Broker:** relay message to MQTT clients via topics. So it is the intermediate node to bridge the communication between IFTTT and MQTT client (running in virtual machine)
- **MQTT Client:** an application running in virtual machine to convert the text (from voice) to command to control a node in the Cooja simulated network. Note that in order to get the text from Google Assistant, the MQTT client and IFTTT must use the same topic (e.g. “light” in this scenario)
- **Wireless Sensor Networks (can be simulated or real):** is the end node in this system. This WSN includes some IPv6/6LoWPAN nodes with a Border Router, connecting nodes to the outside network. Hence the command from MQTT client can send to a specific node via UDP/IPv6 protocol.

HMI: an example with Cooja



```
0
Received on qkiet/feeds/light: 3_on
1
led_on
Received on qkiet/feeds/light: 3_off
1
led_off
^CTraceback (most recent call last):
  File "multi-mote-client.py", line 58, in <module>
    client.loop_forever()
  File "/home/user/.local/lib/python2.7/site-packages/paho/mqtt/client.py", line
 1782, in loop_forever
    rc = self.loop(timeout, max_packets)
  File "/home/user/.local/lib/python2.7/site-packages/paho/mqtt/client.py", line
 1163, in loop
    socklist = select.select(rlist, wlist, [], timeout)
KeyboardInterrupt
➜  software-project python multi-mote-client.py
Connected!
Subscribed to path qkiet/feeds/light
Received on qkiet/feeds/light: 2_on
0
led_on
```



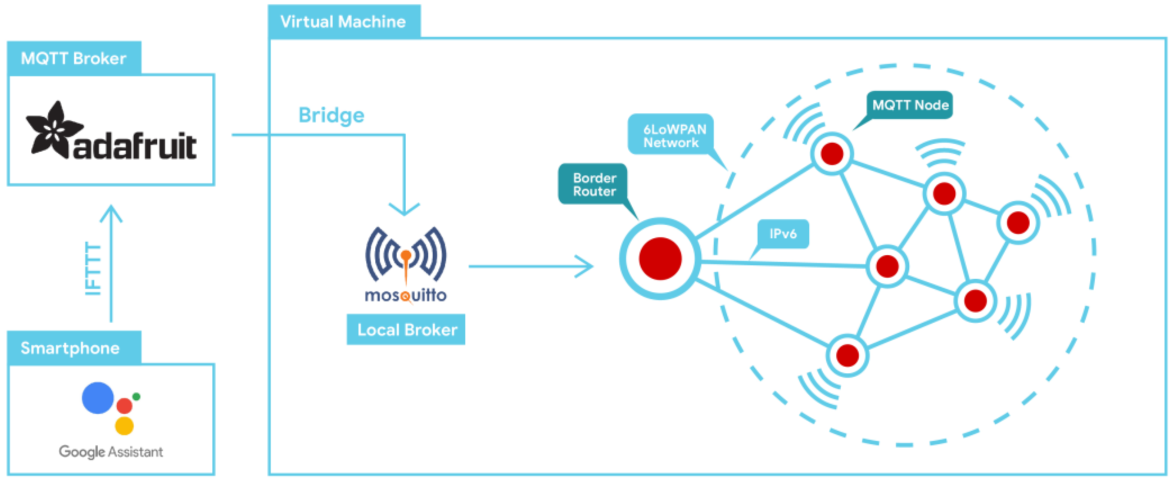
turn onto

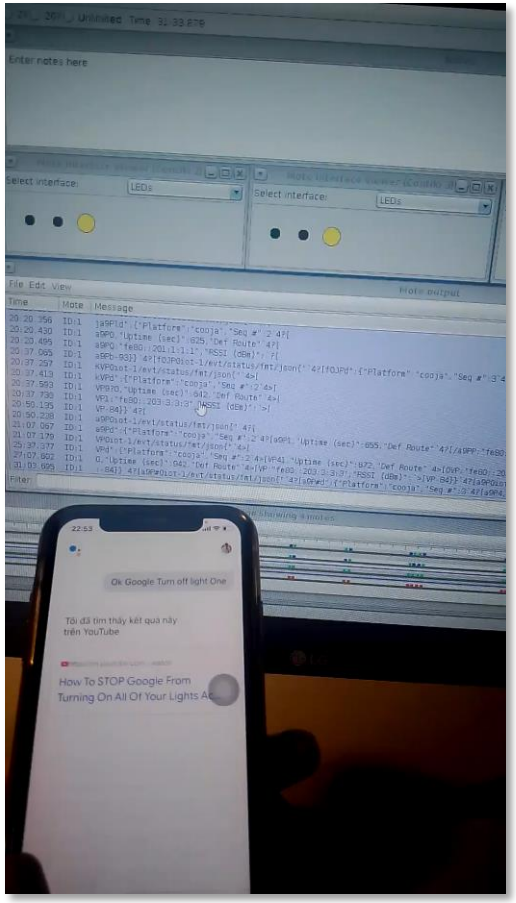
These came back from a search

Turn onto - Idioms by The Free Dictionary
https://idioms.thefreedictionary.com...
turn onto (someone or something). To become especially interested in or enamored of someone or something....

Search | Turn onto or into? | Turn on to so

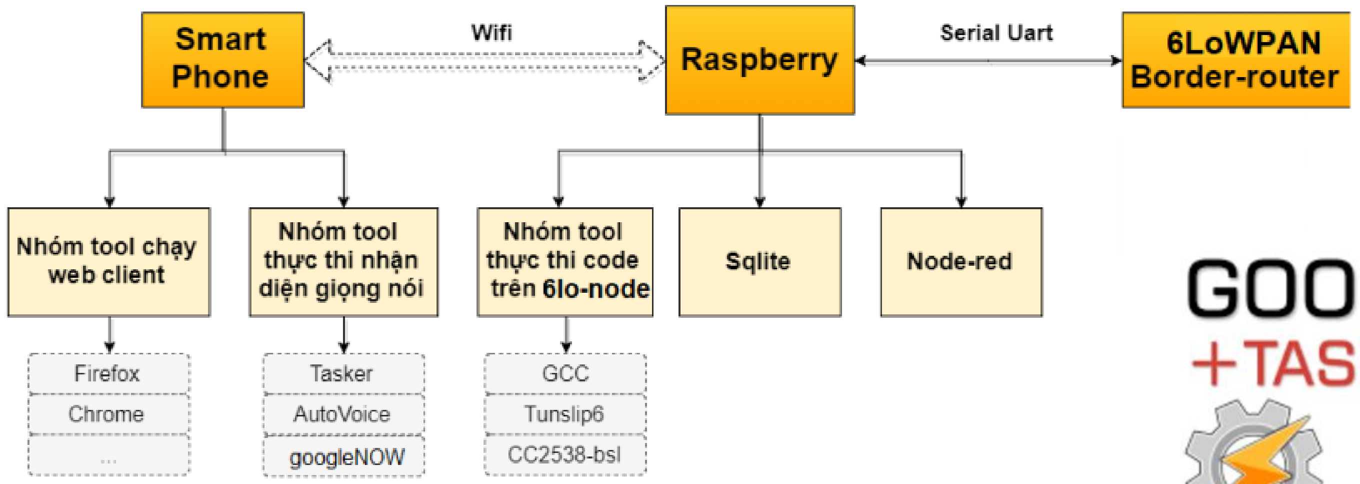
HMI: an example with Cooja

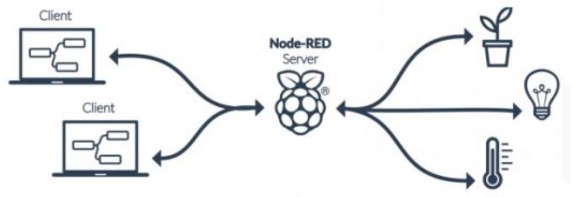




- ❑ Local Broker will bridge Cloud Broker
- ❑ All nodes will run MQTT protocol
- ❑ Subscribe topics from Local broker

HMI: a deployment example

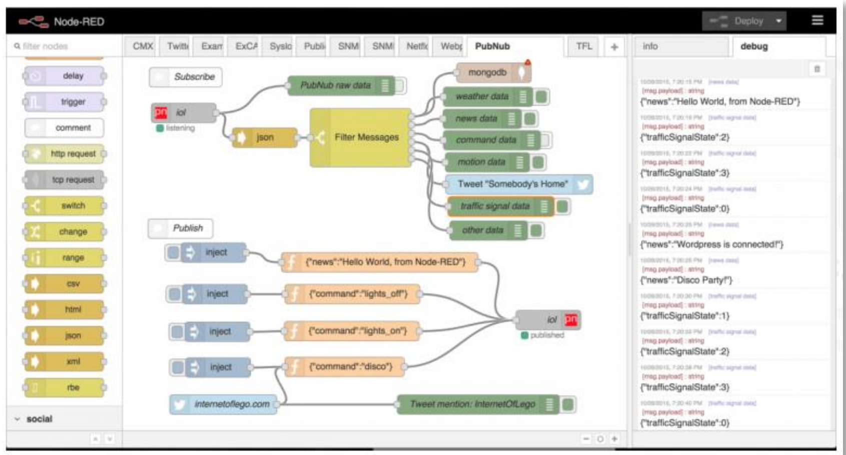




SQL-lite

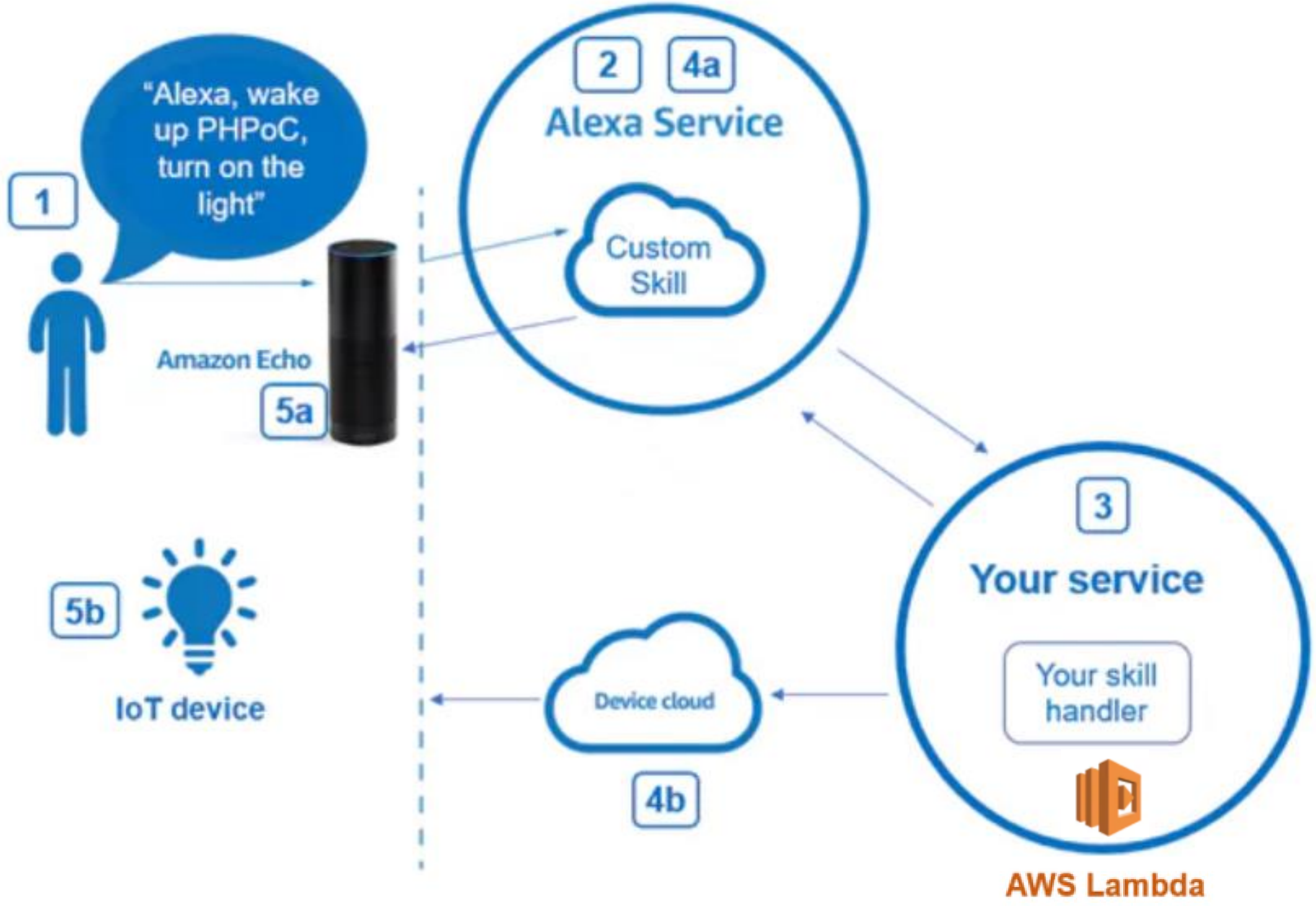
TASKER

AUTO-VOICE



```
{"news":"Hello World, from Node-RED"}
{"trafficSignalState":2}
{"trafficSignalState":3}
{"trafficSignalState":0}
{"news":"Wordpress is connected!"}
{"news":"Disco Party!"}
{"trafficSignalState":1}
{"trafficSignalState":2}
{"trafficSignalState":3}
{"trafficSignalState":0}
```

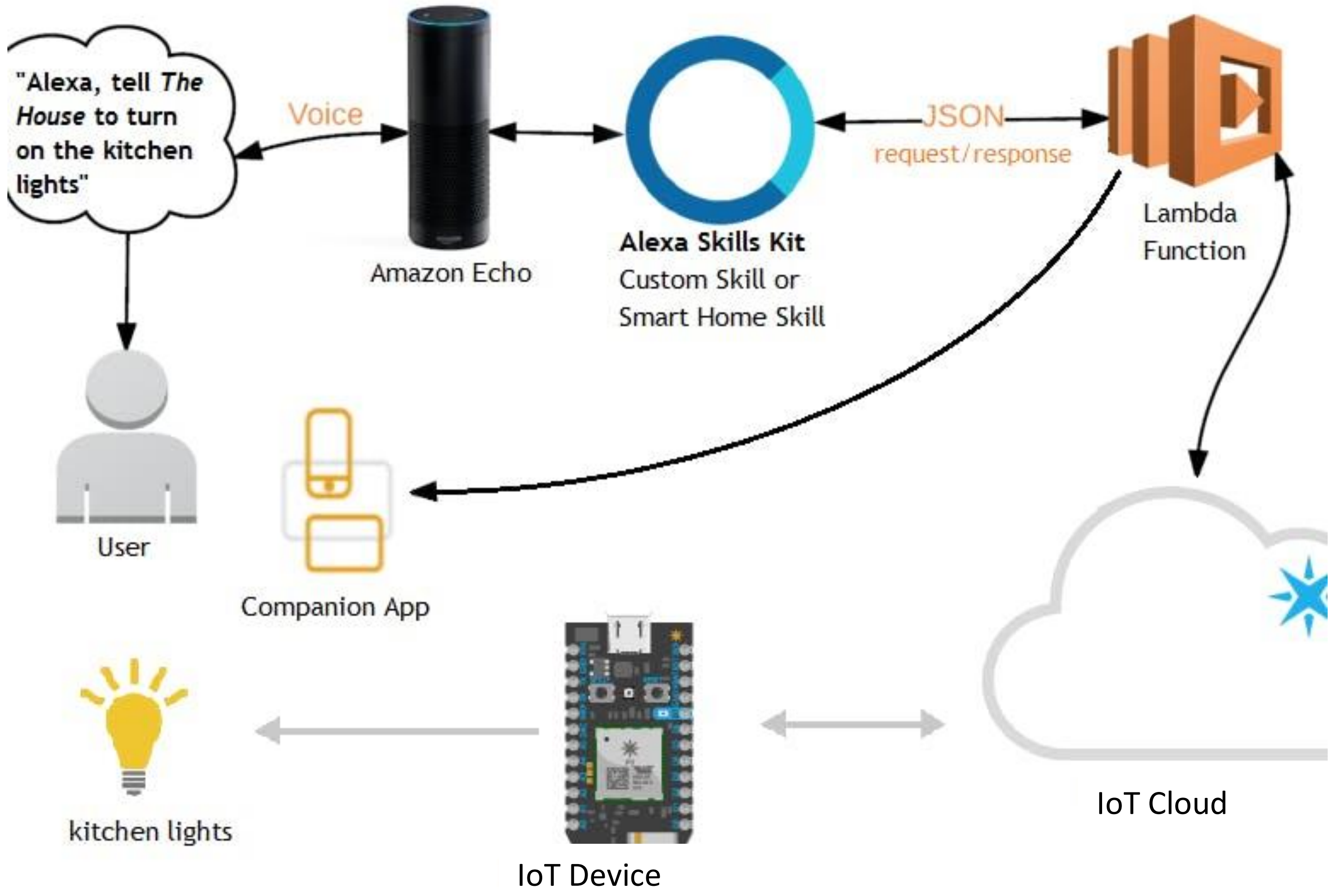
Human-Machine Interface: Alexa



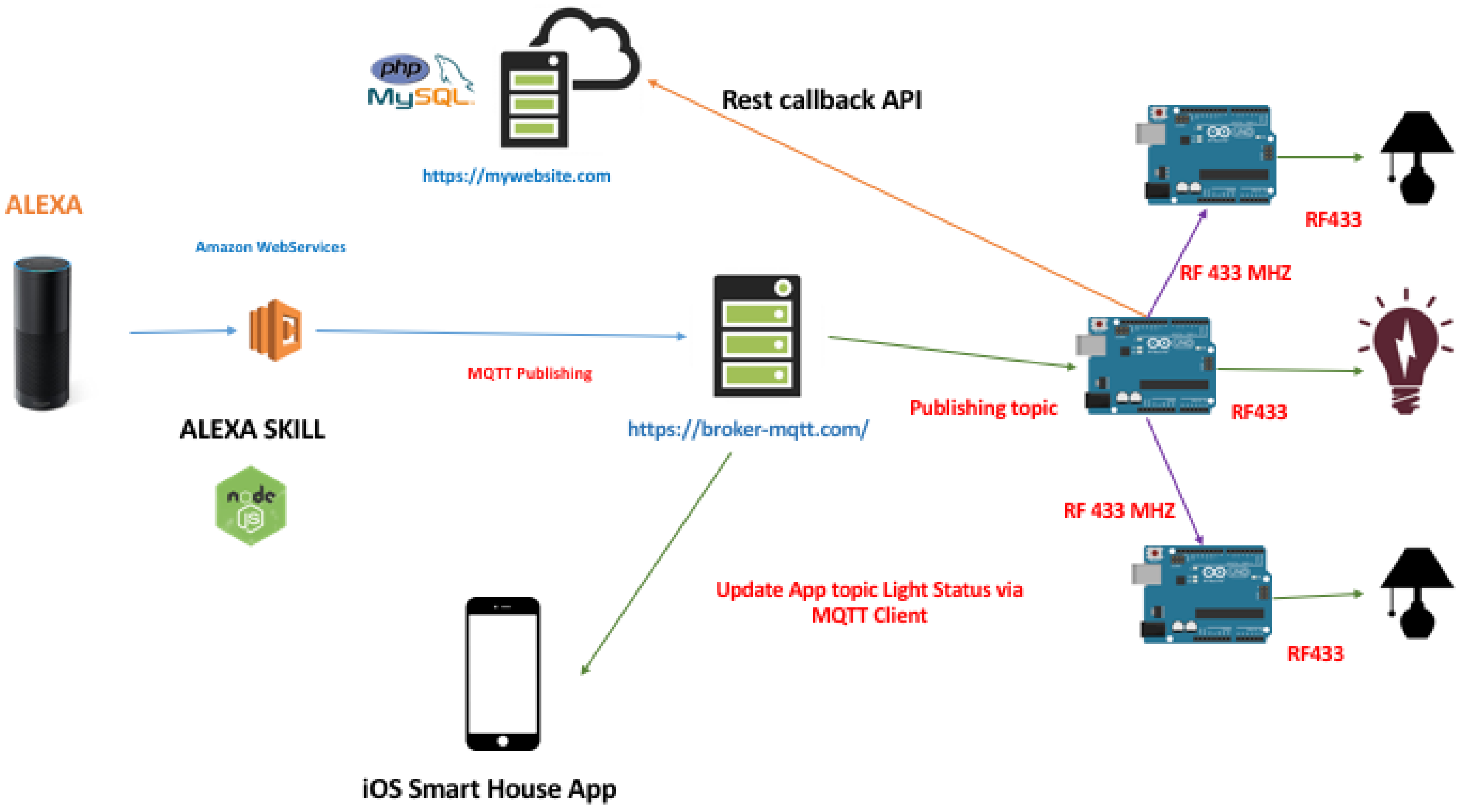
Human-Machine Interface: Alexa

- **(1)** Devices using Alexa Voice Service to record the user’s command and sends it to Alexa Service where developer create their Custom Skills.
- **(2) Alexa Service** converts the voice into text and analyze its, creates intent data then sends it to **Your Service**. In order to **Alexa Service** can send data to **Your Service** developer need to provide endpoint (Amazon Resource Names (ARNs) if use **Lambda** or developer URL host).
- **(3) Your Service** needs developer’s code to implement 2 tasks: sends command or data to **Device Cloud** and sends the response to **Alexa Service**. This place needs to know the endpoint of **Device Cloud**.
- **(4a) Alexa Service** receives response from **Your Service** then converts it to voice and sends to **Devices** using **Alexa Voice Service**. **(4b)** at the same time **Device Cloud** gets the command from **Your Service** and forwards it to **IOT devices**.
- **(5a)** Devices using **Alexa Voice Service** receives response and ***speaks*** to user. **(5b) IOT device** implements command

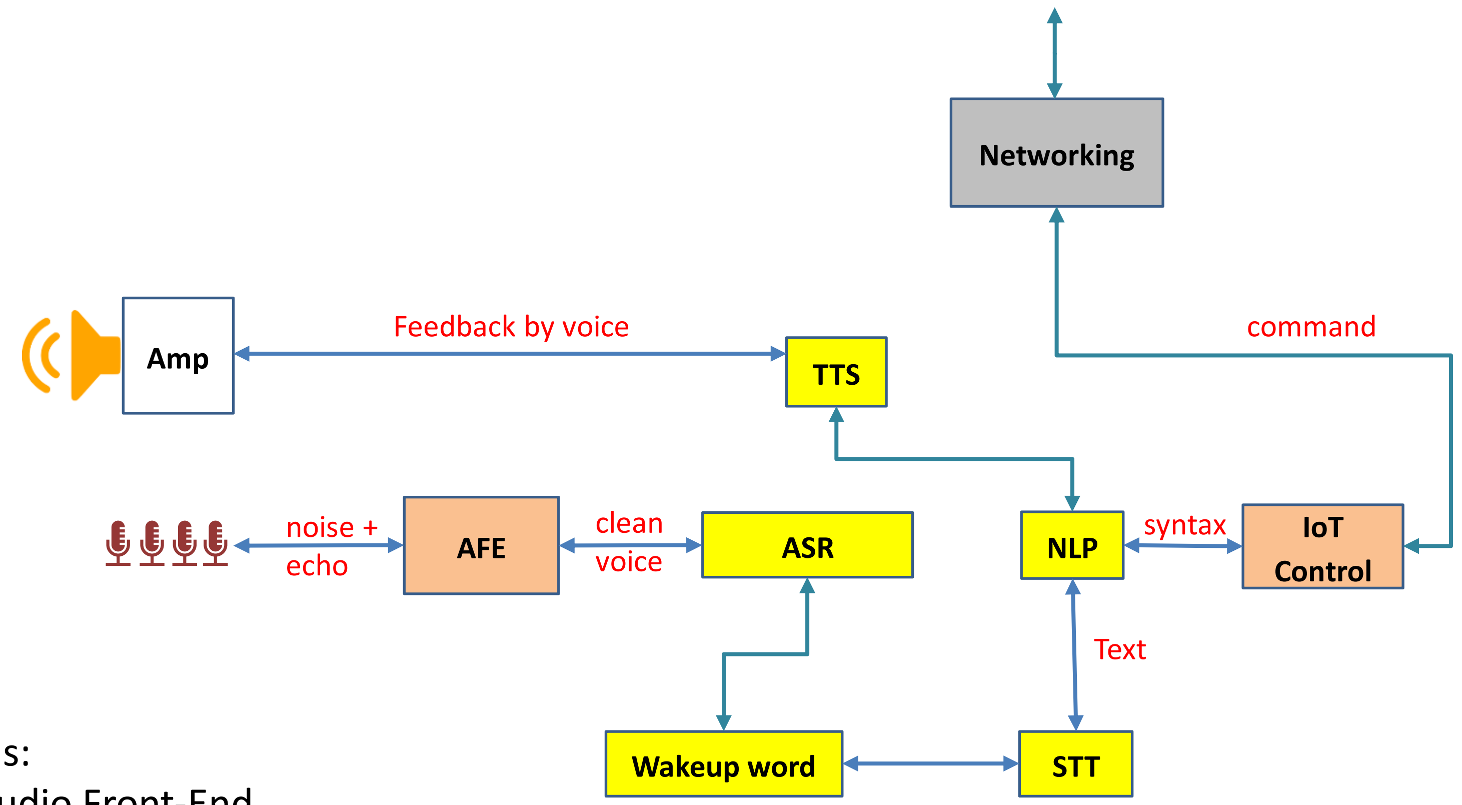
HMI: Example using Alexa



HMI: Example using Alexa + MQTT



Local Voice recognition



Acronyms:
- AFE: Audio Front-End
- ASR: Automatic Speech Recognition
- NLP: Natural Language Processing
- STT: Speech To Text
- TTS: Text To Speech