Application Layer Protocols for the IoT
COnstrained Application Protocol (COAP)
Background

☐ GOAL: to enable web-based services in constrained wireless networks
   • 8 bit micro-controllers
   • limited memory
   • low-power networks

☐ Problem: WEB solution are hardly applicable

☐ Solution: re-design web-based services for constrained networks -> COAP
How Does the Web Work?

- Resources in the Web are:
  - managed by servers
  - identified by URIs
  - accessed synchronously by clients through request/response paradigms

- In a word, Representational State Transfer (REST)
URL Resolution

Universal Resource Name (URN)
urn:Sensei:sensinode.com:NanoSensor:N740:3a-43-ff-12-01-01

Universal Resource Identifier (URI)

Universal Resource Locator (URL)
http://www.example.org:8080/sensors?id=light

Scheme  Authority  Port  Path  Query

Resource
HTTP
TCP
IP
Ethernet Link

http://www.example.org:8080/sensors?id=light

2001:dead:beef::1

DNS
The CoAP Architecture
CoAP Design Requirements

REQ1: Limited Flash/RAM
REQ2: Constrained networks
REQ3: Resource discovery
REQ4: Caching
REQ5: Resource manipulation
REQ6: Sub/Notify
REQ7: HTTP Mapping
REQ8: Sleeping nodes
REQ9: Multicast
REQ10: UDP Transport
REQ11: Reliability
REQ12: Low latency
REQ13: MIME Type
REQ14: Manageability

See draft-shelby-core-coap-req
CoAP At a Glance

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- 4 byte header
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer
COAP Messaging Basics

- Transport:
  - (mainly) UDP binding

- Message Exchange between Endpoints
  - Messages with 4 bytes header (shared by request and responses) containing a message ID (16 bits)
  - Reliable exchange through Confirmable Messages which must be acknowledged (through ACK or Reset Messages). Simple Stop-and-Wait retransmission with exponential back-off.
  - Unreliable exchange through Non-Confirmable Message
  - Duplicate detection for both confirmable and non-confirmable messages (through message ID)
COAP Messaging

Message ID
COAP Message Semantics

- REST Request/Response piggybacked on CoAP Messages
- Method, Response Code and Options (URI, content-type etc.)
COAP Request/Response Examples

Client

CON [0xbc90]
GET /temperature
(Token 0x71)

ACK [0xbc90]
2.05 Content
(Token 0x71)
"22.5 C"

Server

Message ID

Token

Client

CON [0xbc91]
GET /temperature
(Token 0x72)

ACK [0xbc91]
4.04 Not Found
(Token 0x72)
"Not found"

Server
COAP: Separate Response

Client

CON [0x7a10]
GET /temperature
(Token 0x73)

ACK [0x7a10]

... Time Passes ...

CON [0x23bb]
2.05 Content
(Token 0x73)
"22.5 C"

ACK [0x23bb]

Server
COAP: Non-confirmable Request

Client

NON [0x7a11]
GET /temperature
(Token 0x74)

Server

NON [0x23bc]
2.05 Content
(Token 0x74)
"22.5 C"
Message Header (4 bytes)

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

**Ver** - Version (1)

**T** - Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)

**TKL** - Token Length, if any, the number of Token bytes after this header

**Code** - Request Method (1-10) or Response Code (40-255)

**Message ID** - 16-bit identifier for matching responses

**Token** - Optional response matching token
**Option Format**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option Delta</th>
<th>Option Length</th>
<th>1 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>(extended)</td>
<td>(extended)</td>
<td></td>
</tr>
</tbody>
</table>

- **Option Delta** - Difference between this option type and the previous
- **Length** - Length of the option value
- **Value** - The value of Length bytes immediately follows Length
## Base Specification Options

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>If-Match</td>
<td>opaque</td>
<td>0-8</td>
<td>(none)</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Uri-Host</td>
<td>string</td>
<td>1-255</td>
<td>(see below)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>ETag</td>
<td>opaque</td>
<td>1-8</td>
<td>(none)</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>If-None-Match</td>
<td>empty</td>
<td>0</td>
<td>(none)</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Uri-Port</td>
<td>uint</td>
<td>0-2</td>
<td>(see below)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>Location-Path</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>Uri-Path</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Content-Format</td>
<td>uint</td>
<td>0-2</td>
<td>(none)</td>
</tr>
<tr>
<td>14</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Max-Age</td>
<td>uint</td>
<td>0-4</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>Uri-Query</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accept</td>
<td>uint</td>
<td>0-2</td>
<td>(none)</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>Location-Query</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>35</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Proxy-Uri</td>
<td>string</td>
<td>1-1034</td>
<td>(none)</td>
</tr>
<tr>
<td>39</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Proxy-Scheme</td>
<td>string</td>
<td>1-255</td>
<td>(none)</td>
</tr>
</tbody>
</table>

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable
Dealing with Packet Loss

- Stop and Wait approach
- Repeat a request after a time-out in case ACK (or RST) is not coming back
Back-Off Details

- Initial time-out set to:
  - Rand [ACK_TIMEOUT, ACK_TIMEOUT * ACK_RANDOM_FACTOR] ([2s, 3s])

- When time-out expires and the transmission counter is less than MAX_RETRANSMIT (4)
  - retransmit
  - Increase transmission counter
  - double the time-out value

- The procedure is repeated until
  - A ACK is received
  - A RST message is received
  - the transmission counter exceeds MAX_RETRANSMIT
  - the total attempt duration exceeds MAX_TRANSMIT_WAIT (93s)
COAP Observation

PROBLEM:
- REST paradigm is often “PULL” type, that is, data is obtained by issuing an explicit request
- Information/data in WSN is often periodic/triggered (e.g., get me a temperature sample every 2 seconds or get me a warning if temperature goes below 5°C)

SOLUTION: use Observation on COAP resources
Observation

See draft-ietf-core-observe
COAP Block Transfer

- PROBLEM: avoid segmentation in the lower layers (IPv6)
- SOLUTION: COAP Block Transfer Mode
  - brings up fragmentation at the application layer
Block transfer

- Block2 Option added to messages
  - nr=incremental block number within original data
  - m=more blocks flag
  - sz=block size
Discovery & Semantics

- Resource Discovery
  - GOAL: Discovering the links hosted by CoAP (or HTTP) servers
    
    ```
    GET /.well-known/core?optional_query_string
    ```
  
  - Returns a link-header style format
    
    - URL, relation, type, interface, content-type etc.
CoRE Resource Discovery

CoAP Client

CON [0xaf6] GET /.well-known/core

ACK [0xaf6] 2.05 Content "<light>..."

CoAP Server

</dev/bat>;obs;if="";rt="ipso:dev-bat";ct="0",
</dev/md1>;if="";rt="ipso:dev-md1";ct="0",
</dev/mfg>;if="";rt="ipso:dev-mfg";ct="0",
</pwr/0/rel>;obs;if="";rt="ipso:pwr-rel";ct="0",
</pwr/0/w>;obs;if="";rt="ipso:pwr-w";ct="0",
</sen/temp>;obs;if="";rt="ucum:Cel";ct="0"
Open source implementations:

- Java CoAP Library [California](#)
- C CoAP Library [Erbium](#)
- [libCoAP](#) C Library
- [jCoAP](#) Java Library
- [OpenCoAP](#) C Library
- TinyOS and Contiki include CoAP support

- Firefox has a CoAP [plugin called Copper](#)
- Wireshark has CoAP plugin
The Message Queuing Telemetry Transport (MQTT)

Invented in 1999 (IBM proprietary standard)
Released in 2010
Official OASIS standard since 2014
Current version MQTT 3.1.1  http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
MQTT in short

MQTT is a Client Server publish/subscribe messaging transport protocol.

☐ More features:
  - Simple to implement (especially at the sensor side)
  - QoS Support
  - Lightweight and bandwidth efficient
  - Data agnostic
  - Session awareness
Communication Pattern

- Publish/Subscribe paradigm
  - Clients don’t know each other
  - One-to-Many paradigm
  - Every client publishes & subscribes
  - PUSH information paradigm compared to PULL’s one in COAP

![Diagram showing MQTT communication pattern between Senosr node, Laptop, and Mobile device](image)
MQTT Components

- Publish/Subscribe paradigm
  - Clients don’t know each other
  - One-to-Many paradigm
  - Every client publishes & subscribes
  - PUSH information paradigm compared to PULL’s one in COAP

MQTT Broker

- Sensor node
- MQTT Client

Laptop
MQTT Client

Mobile device
MQTT Client

Publish: “30°C”

Subscribe: “?°C”

Publish: “30°C”

Subscribe: “?°C”

Publish: “30°C”
MQTT Topics

deib/antlab/room5/temperature

Wildcards allowed only when subscribing

deib/antlab/ + /temperature

Single level Wildcard

Plus sign can be used in multiple levels
Multi-level Wildcards

deib/antlab/room5/ #

Multi-level Wildcard
(can be used only at the end)
MQTT Connections

- Each MQTT client opens one connection to the MQTT Broker
- Push capabilities
- Works even through firewalls

MQTT

TCP

IP
CONNECT message fields:

- clientId: "clientMatteo"
- cleanSession: true
- username (opt): "matteo"
- password (opt): "1234"
- lastWillTopic (opt): "matteo/temp"
- lastWillQoS (opt): 1
- lastWillMessage (opt): "something wrong"
- keepAlive: 30
Open Connections

CONNACK message fields:

- sessionPresent: true
- returnCode: 0-4

- 0: everything ok
- 1: unacceptable version
- 2: id rejected
- 3: server unavailable
- 4: bad username and pwd
- 5: unauthorized
PUBLISH message fields

- **packeId**: 2
- **topicName**: “matteo/temp”
- **QoS**: 1
- **retainFlag**: false
- **Payload**: “temperature:30”
- **dupFlag**: false
QoS 0: “at most once”

- Best effort transfer (same reliability provided by the underlying transport protocol)
The MQTT client stores the message and keeps retransmitting it until it is acknowledged by the MQTT broker (message can be received multiple times).

- PUBACK message fields:
  - packetId  2
QoS 2: “exactly once”

- PUBREC, PUBREL, PUBCOMP message fields:
  - PacketId 2

- PUBLISH reception @ MQTT broker: process the packet accordingly, send PUBREC message back, store locally packetId to avoid duplicate processing
- PUBREC reception @ MQTT client: discard the initial packet and send PUBREL
- PUBREL reception @ MQTT client: clear any current state and send PUBCOMP
SUBSCRIBE message fields:
- packetId: 2
- QoS1: 0
- Topic1: "matteo/temp/1"
- QoS2: 1
- Topic2: "kitchen/temp/2"
- ............ ............

SUBACK message fields:
- packetId: 2
- returnCode1: 2
- returnCode2: 0

one QoS/topic couple for each subscription

one returnCode for each topic in the subscription
Unsubscribing

- **UNSUBSCRIBE message fields:**
  - `packetId`: 2
  - `Topic1`: “matteo/temp/1”
  - `Topic2`: “kitchen/temp/2”
  - ..........

- **UNSUBACK message fields:**
  - `packetId`: 2

- One `returnCode` for each topic in the subscription.
Persistent Sessions

☐ In default operation mode when the client disconnects, all the client-related status at the broker is flushed (list of subscription, QoS pending messages, etc.)

☐ In persistent sessions both client and broker maintain a session:
  ■ Broker:
    □ Existence of a session, even if there are no subscriptions
    □ All subscriptions
    □ All messages in QoS 1 or 2 flow, which are not confirmed by the client
    □ All new QoS 1 or 2 messages, which the client missed while it was offline
    □ All received QoS 2 messages, which are not yet confirmed to the client
  ■ Client
    □ All messages in a QoS 1 or 2 flow, which are not confirmed by the broker
    □ All received QoS 2 messages, which are not yet confirmed to the broker

☐ That means even if the client is offline all the above will be stored by the broker and are available right after the client reconnects.
Retained Messages

- Problem: publishing and subscribing are asynchronous processes
- A client subscribing to a topic pattern may not get any message on that topic until some other client publishes on it
- Retained messages are PUBLISH messages with the retainedFlag set to one
- The broker stores locally the retained message and send it to any other client which subscribes to a topic pattern matching that of the retained message
The Last Will and Testament (LWT) notifies other clients about an hard disconnection by a specific client.

Each client can specify its last will message when connecting to a broker.

The broker will store the message until it detects client hard disconnection.

The broker sends the message to all subscribed clients on the specific topic.

The stored LWT message will be discarded if a client disconnects gracefully by sending a DISCONNECT message.
LWT Set Up

CONNECT message fields:

- clientId: "clientMatteo"
- cleanSession: true
- username (opt): "matteo"
- password (opt): "1234"
- lastWillTopic (opt): "matteo/temp"
- lastWillQoS (opt): 1
- lastWillMessage (opt): "something wrong"
- keepAlive: 30
LWT Message is sent when..

- An I/O error or network failure is detected by the server.
- The client fails to communicate within the Keep Alive time.
- The client closes the network connection without sending a DISCONNECT packet first.
- The server closes the network connection because of a protocol error.
Keepalive

- It is responsibility of the client to keep the MQTT connection active
- Upon expiration of the keepalive, if no other interaction has happened with broker, the client ”pings” the broker which “pings it back”

PINGREQ and PINGRESP messages have null payload
Main differences

- Extended architecture with Gateways and Forwarders
- New gateway discovery procedures (and messages)
- Some messages are more “compressed”
- Extended keepalive procedures to support sleeping clients
Getting started

☐ Brokers list
  - Mosquitto
  - HiveMQ

☐ MQTT and NodeRed. Let’s use it